Доброго времени суток, дорогой друг

Я очень благодарен тебе, за то, что ты не обделил своим вниманием данный материал и искренне надеюсь, что он окажется для тебя полезным. Со своей стороны, я очень постарался сделать его максимально содержательным и доступным для начинающего пользователя.

Впрочем, ты всегда можешь обратиться ко мне в приватной ветке и задать свой вопрос. Даже если этот вопрос выходит за рамки материала, я всегда охотно иду на контакт и буду рад услышать свежие идеи и дать совет по смежным областям, а если конкретнее:

1. Программирование на языке C# (WPF/Selenium/Zennoposter)
2. Интернет-маркетинг (особенно таргетированная реклама ВК)
3. Разработка веб-сайтов, фреймворков, плагинов CMS и прочих решений
4. Дизайн (Photoshop/Illustrator)
5. Вёрстка (не pro-фронтендер, но пристойно сверстать могу многие вещи)
6. Разбираюсь в JS (в частности Vue.js)

...и всё, что связано с манимейкингом. Буду рад помочь :)

Прежде чем начать, обсудим процесс нашего совместного обучения. Активная фаза поддержки начнётся с даты выдачи материала, и прекратится спустя два (или три, если ты выполнил условия) месяца после даты завершения складчины. Впоследствии я буду отвечать на вопросы в ветке, но по мере своих возможностей.

Прошу учесть, что данный материал не претендует на звание справочника и не является исчерпывающим. Данный материал призван дать вам твёрдые основы и показать направление в котором стоит двигаться не только по части разработки на языке Python, но и по части парсинга.

Как и в прошлых курсах, я весьма серьёзно подошёл к разработке продукта. Старался создать максимально исчерпывающий, понятный и качественный материал.

Отдельно отмечу, что весьма открыто отношусь к критике и всегда стремлюсь вынести из неё конкретные идеи по улучшению своей работы. Поэтому буду вам признателен, если вы окажете мне услугу и изложите свои пожелания по поводу улучшения материала.

Удачи нам!

Введение в Python

Python – это скриптовый язык программирования общего назначения. На данный момент он нашёл распространение в сферах машинного обучения, серверной разработки, сбора данных, написания игр и многого другого.

Python имеет широкую область применения, так как не является исключительно серверным языком программирования, код на Python переносим между популярными ОС, а также имеет модуль работы с ОС. То есть, python-скрипты можно использовать в качестве прикладного софта непосредственно на ПК.

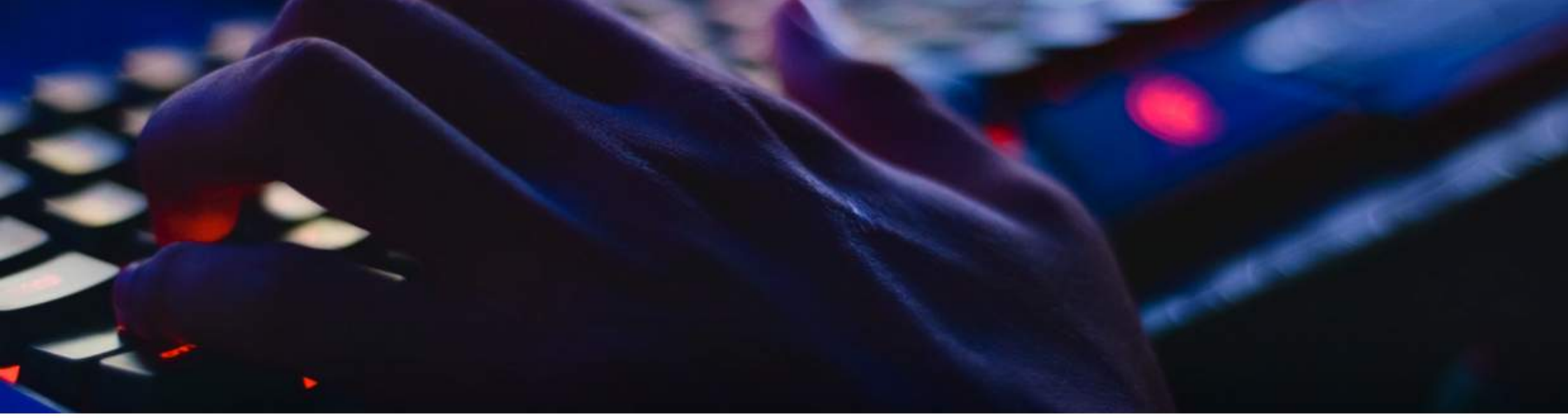
Например, можно управлять директориями или файлами, запускать программы и многое другое.

Следует отметить, что несмотря на свою мощность и функциональность, Python является одним из самых простых языков программирования, как по синтаксису, так и по архитектуре при написании приложений.

Именно поэтому, Python лидирует не только в сфере прикладной разработки, но и в образовательной сфере. В США, Python считается одним из самых предпочтительных языков программирования для обучения, как в школах, так и в высших учебных заведениях.

Python настолько перспективен, что его используют большинство крупных компаний и перспективных стартапов, например: Google, Amazon, YouTube, Spotify, Instagram, Яндекс и т.д.

Python так популярен потому что он сочетает в себе скорость, простоту и надёжность. В последующих уроках курса вы убедитесь в этом сами.



Установка Python на ПК

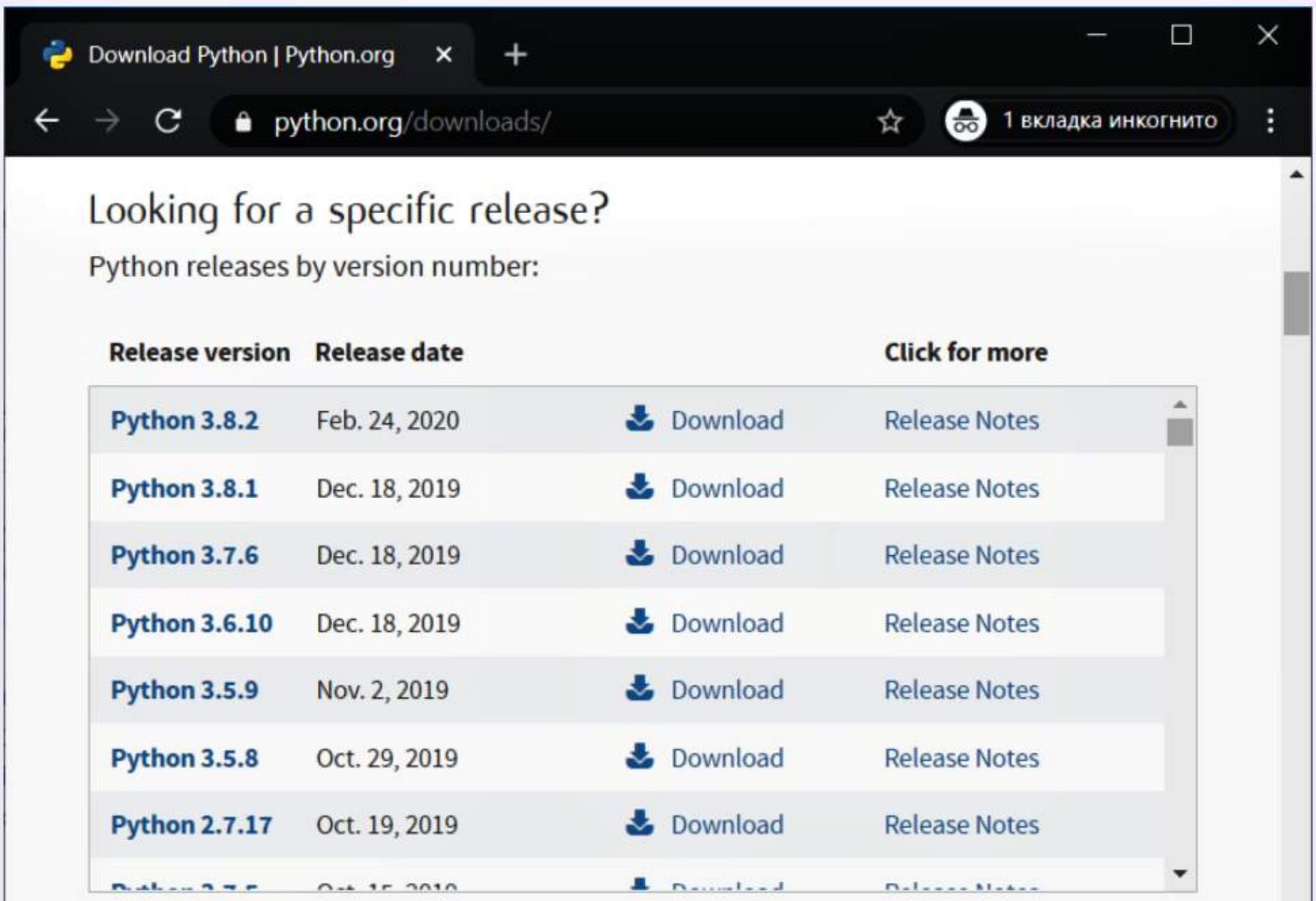
Содержание урока

Для работы с Python не нужно устанавливать и настраивать сложного окружения. Достаточно лишь скачать и установить интерпретатор, который будет выполнять написанный вами код.

Скачиваем Python

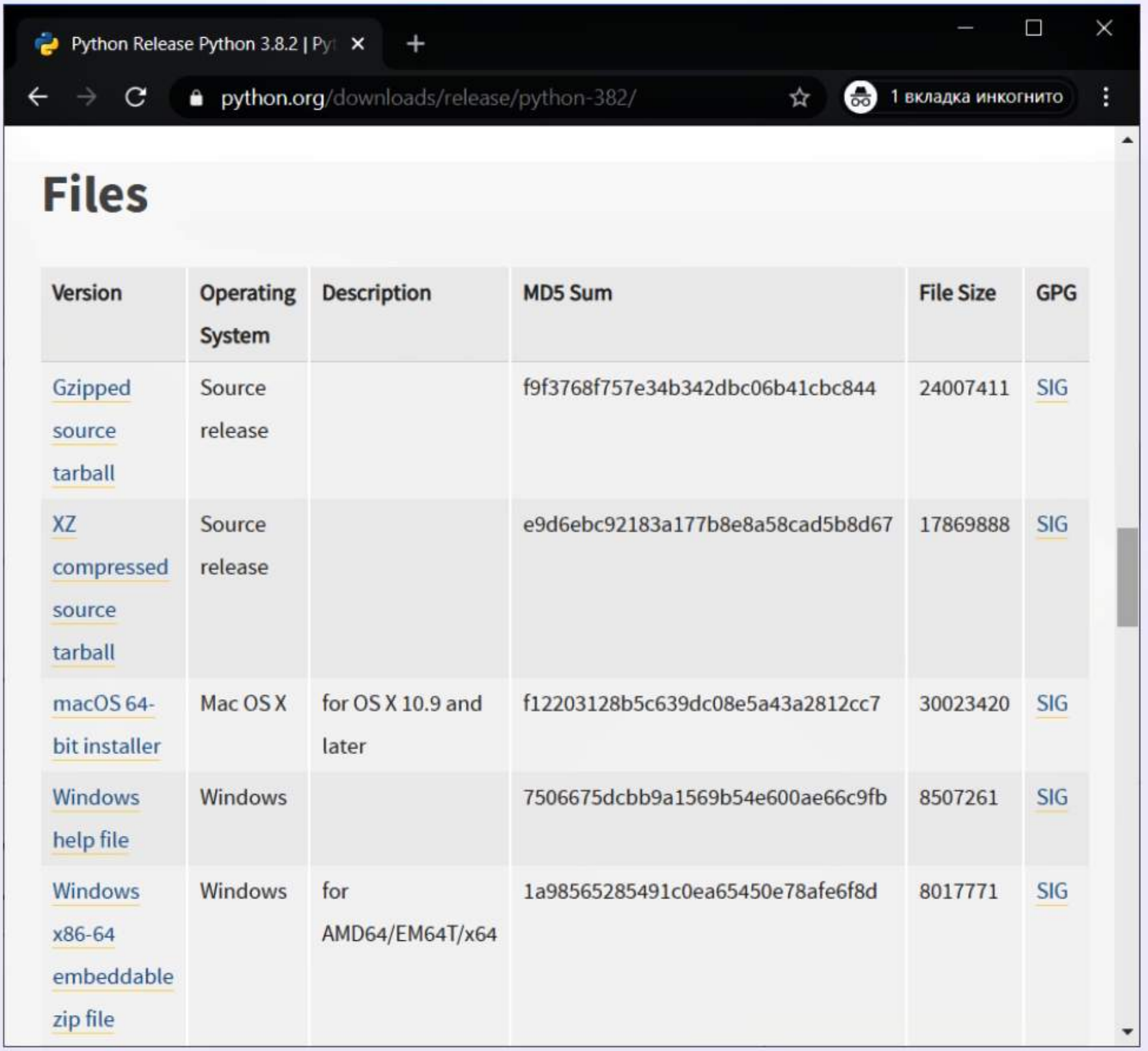
Для этого переходим по ссылке: https://www.python.org/downloads/

Выбираем последнюю актуальную версию интерпретатора. На момент написания урока это Python 3.8.2:



View older releases

Далее, в разделе Files выберите интерпретатор для вашей версии ОС:



Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		f9f3768f757e34b342dbc06b41cbc844	24007411	SIG
XZ compressed source tarball	Source release		e9d6ebc92183a177b8e8a58cad5b8d67	17869888	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	f12203128b5c639dc08e5a43a2812cc7	30023420	SIG
Windows help file	Windows		7506675dcbb9a1569b54e600ae66c9fb	8507261	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	1a98565285491c0ea65450e78afe6f8d	8017771	SIG

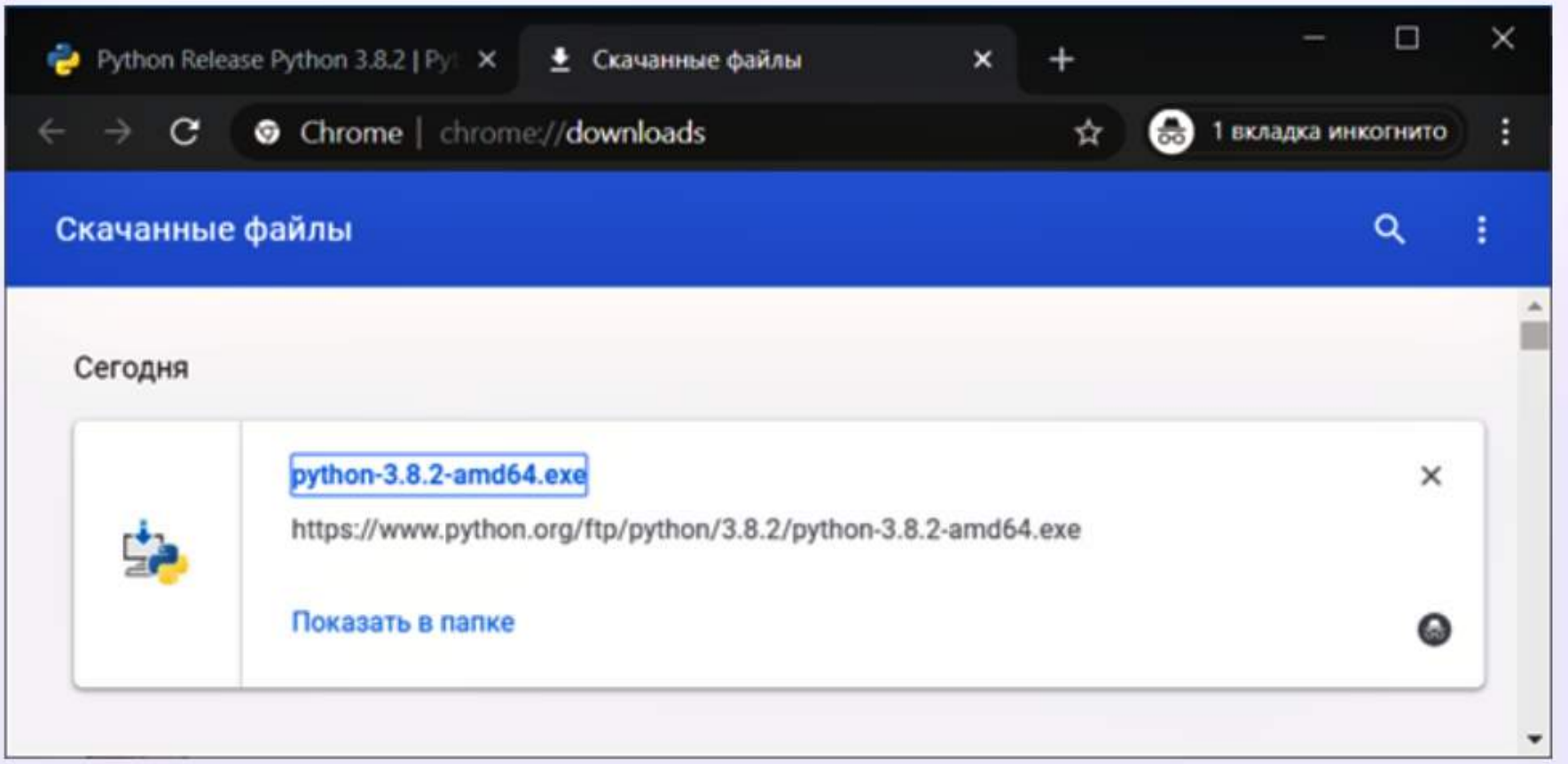
Разрядность ОС Windows можно узнать следующим образом:

Нажимаем сочетание клавиш «Win + I»
В открывшемся окне, выбираем раздел «Система»
В левом боковом меню выбираем пункт «О системе»
В открывшемся окне, в блоке «Характеристики устройства» видим тип системы:

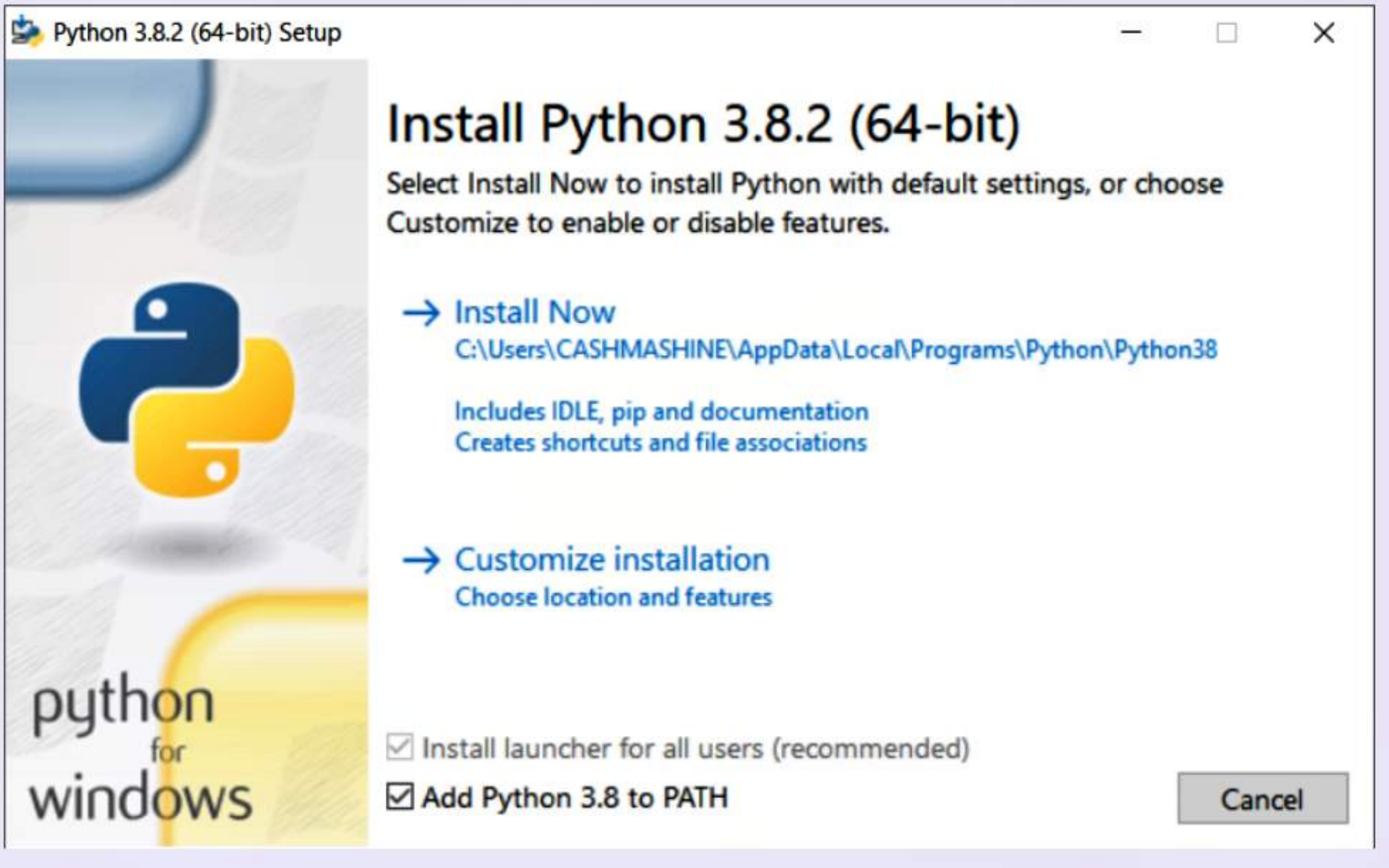
Тип системы	64-разрядная операционная система, процессор x64

Устанавливаем Python на ПК

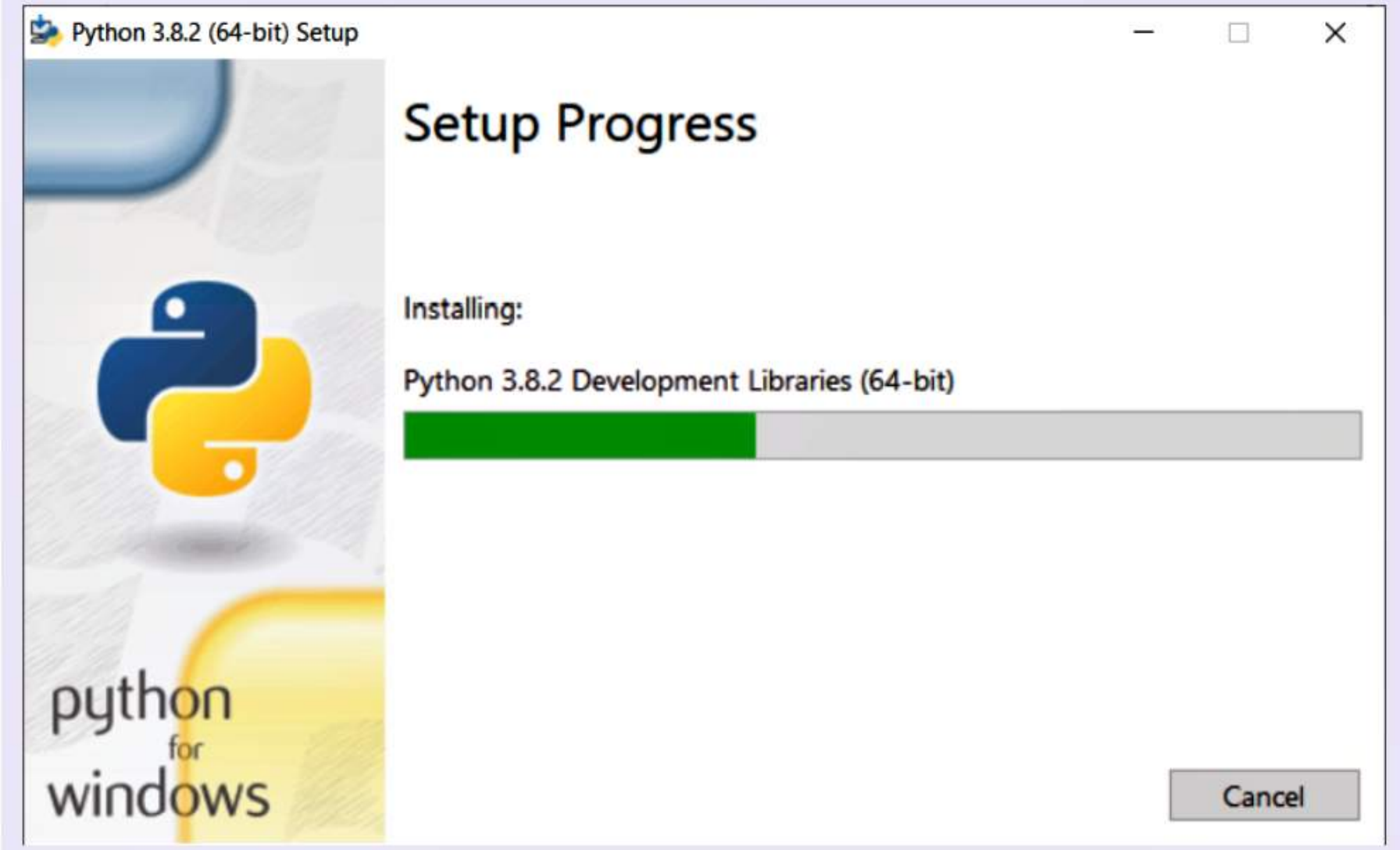
После скачивания интерпретатора Python, запускаем установочный файл:



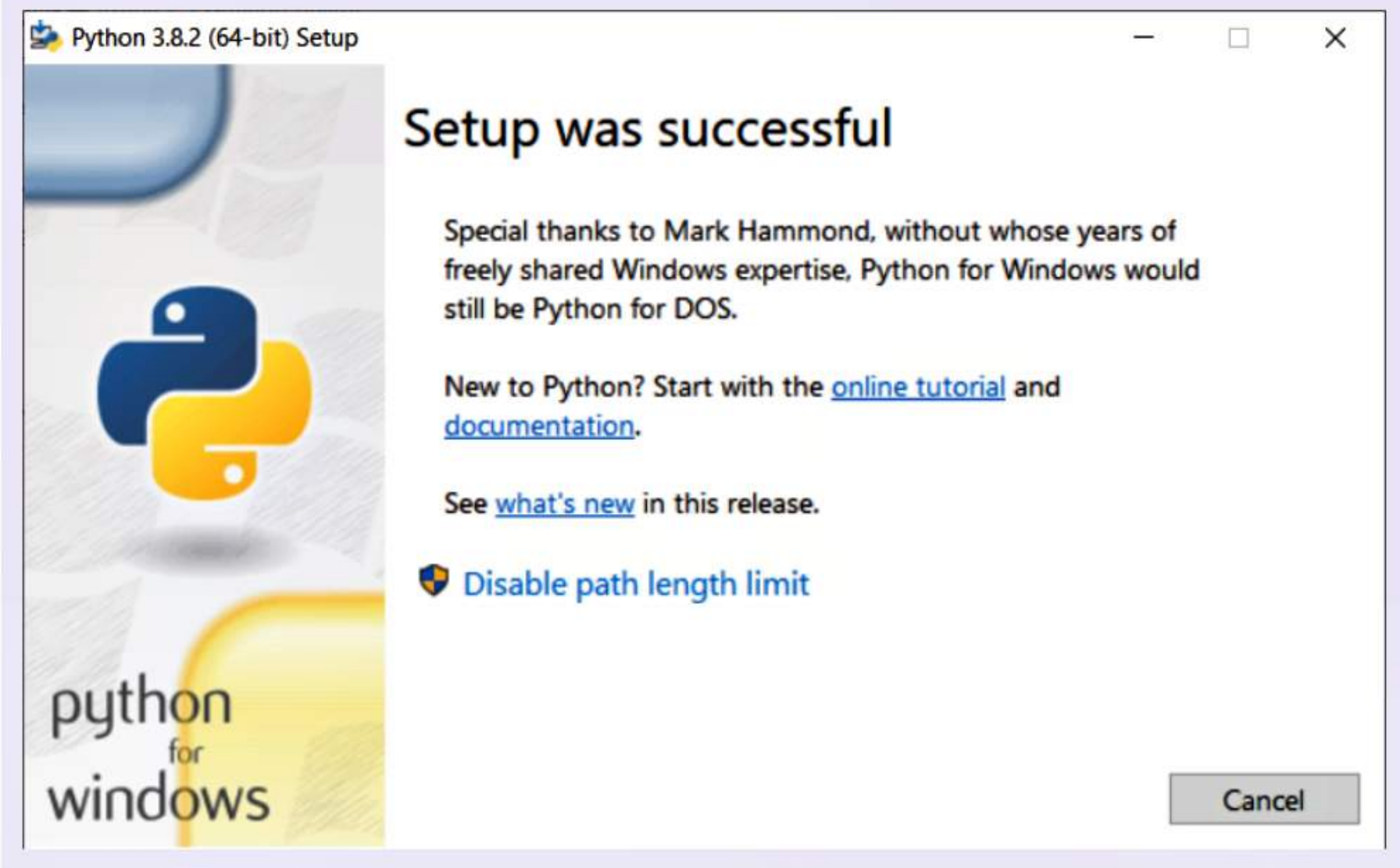
В открытом окне, обязательно ставим галочку «Add Python to Path»:



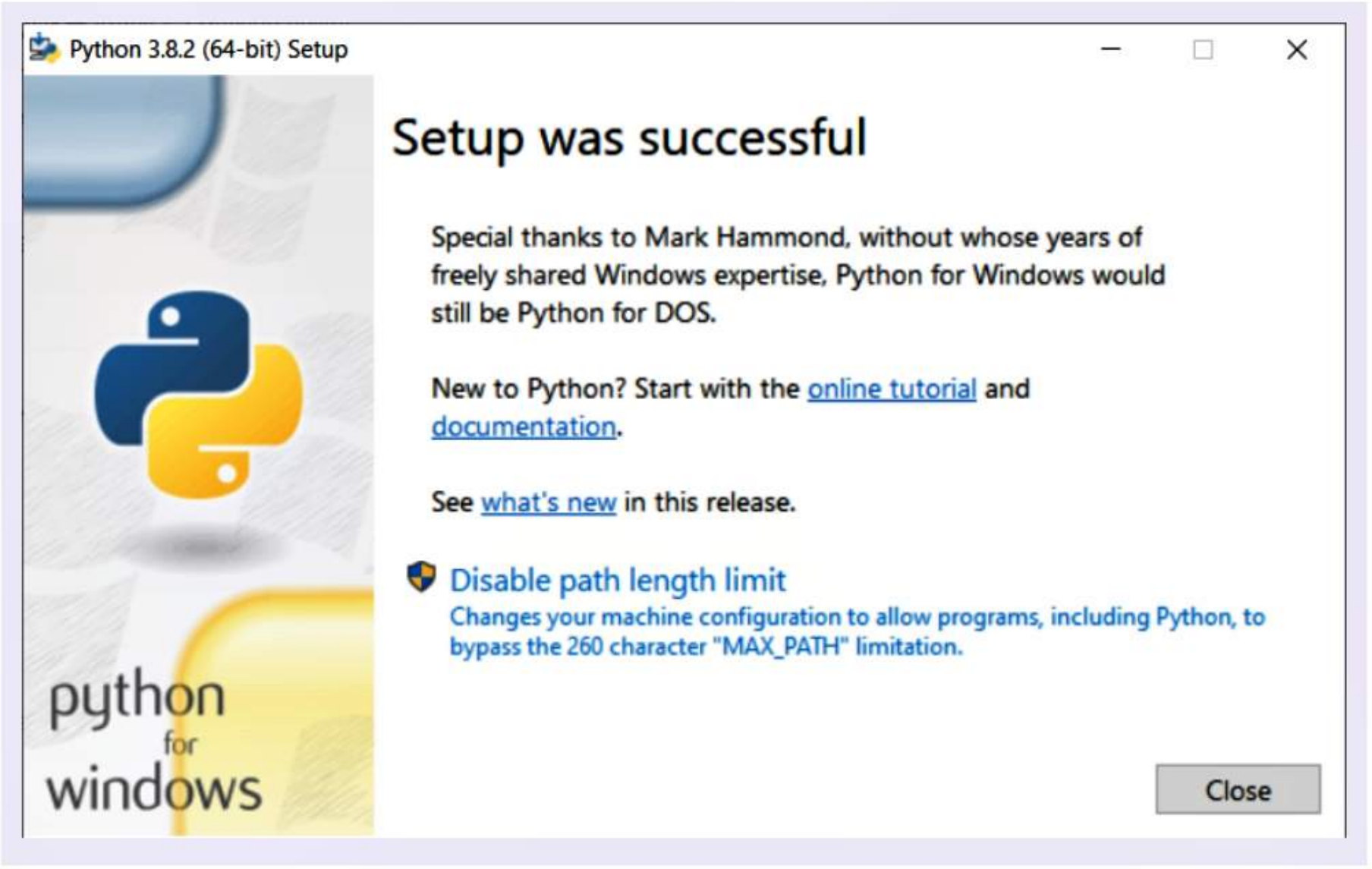
Затем, нажимаем кнопку Install Now. После чего, начнётся процесс установки:



После успешной установки, вы увидите следующее окно:



После успешной установки, вы увидите следующее окно:



Нажимаем кнопку «Close». Поздравляю, мы успешно установили интерпретатор Python.

Python-скрипт

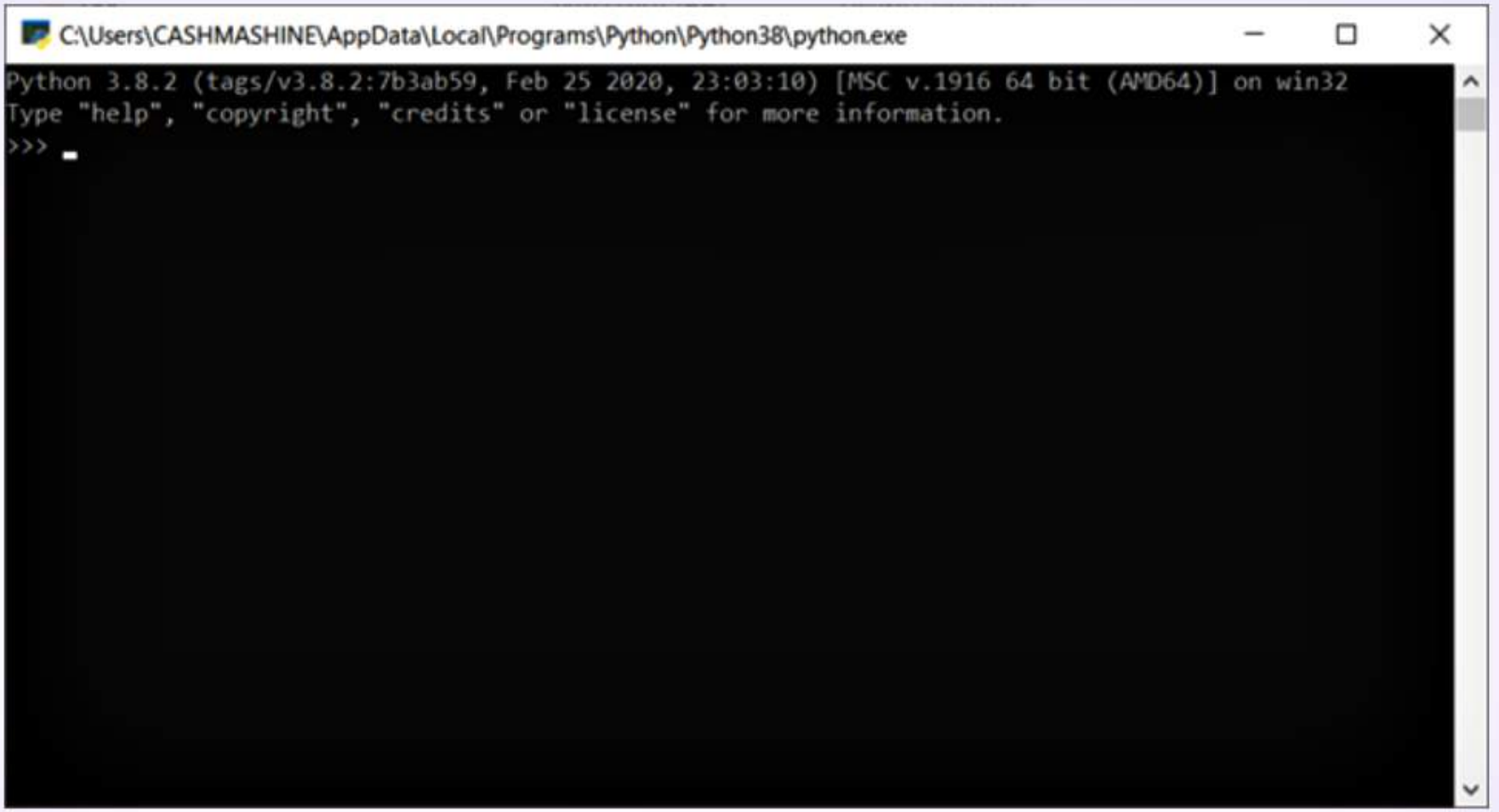
Содержание урока

Мы успешно установили интерпретатор Python и теперь можем запускать написанные нами Python-скрипты. Но сперва, нам необходимо написать свой первый скрипт. Приступим.

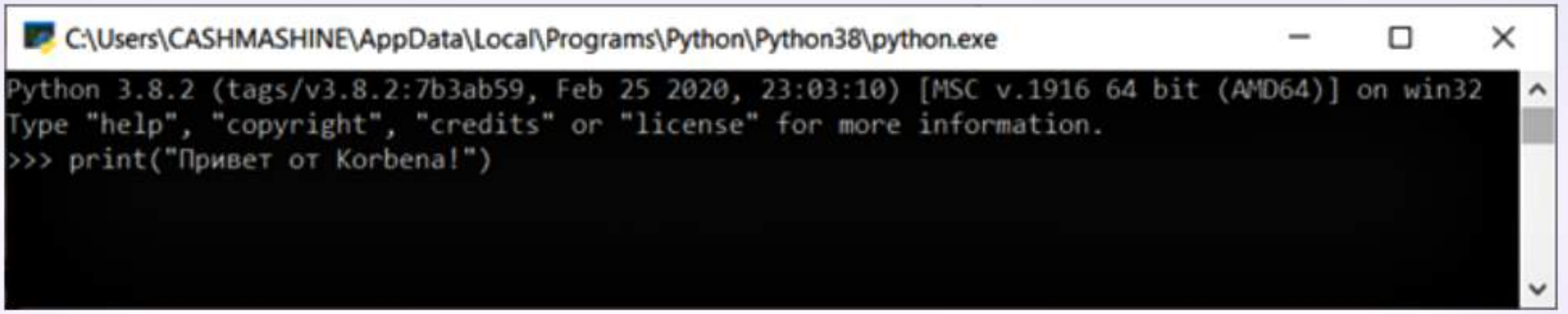
Запускаем скрипт из консоли

Если вы устанавливали Python в точности по прошлому уроку, то папка с файлами и инструментами интерпретатора лежит по адресу: `C:\Users\[имя_пользователя]\AppData\Local\Programs\Python\Python38`

В этой папке лежит наш интерпретатор (файлик с именем python.exe), запустим его:



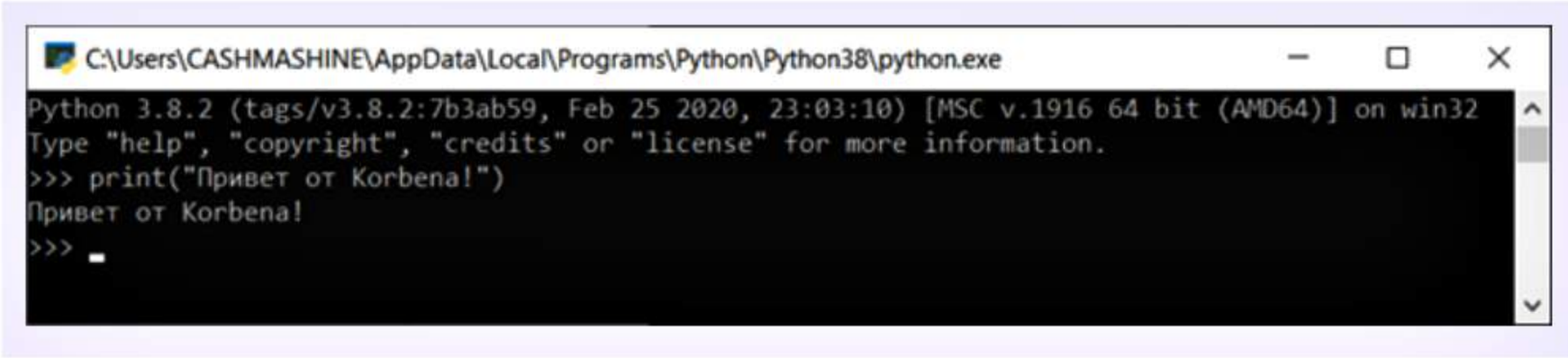
Это программа получает на вход исходный код python-скрипта, и выполняет его. Давайте проверим, и напишем простейшую инструкцию, которая выведет в эту же консоль приветствие:



Здесь я написал инструкцию: `print("Привет от Korbena!")`

Данная инструкция (print) выводит в консоль заданную строку, в моём случае: «Привет от Korbena». Советую, на текущий момент, не задумываться об этой конструкции, более детально мы разберём её в следующих уроках.

После того, как ввели данную инструкцию, нажимаем клавишу «Enter» и вот что получается:

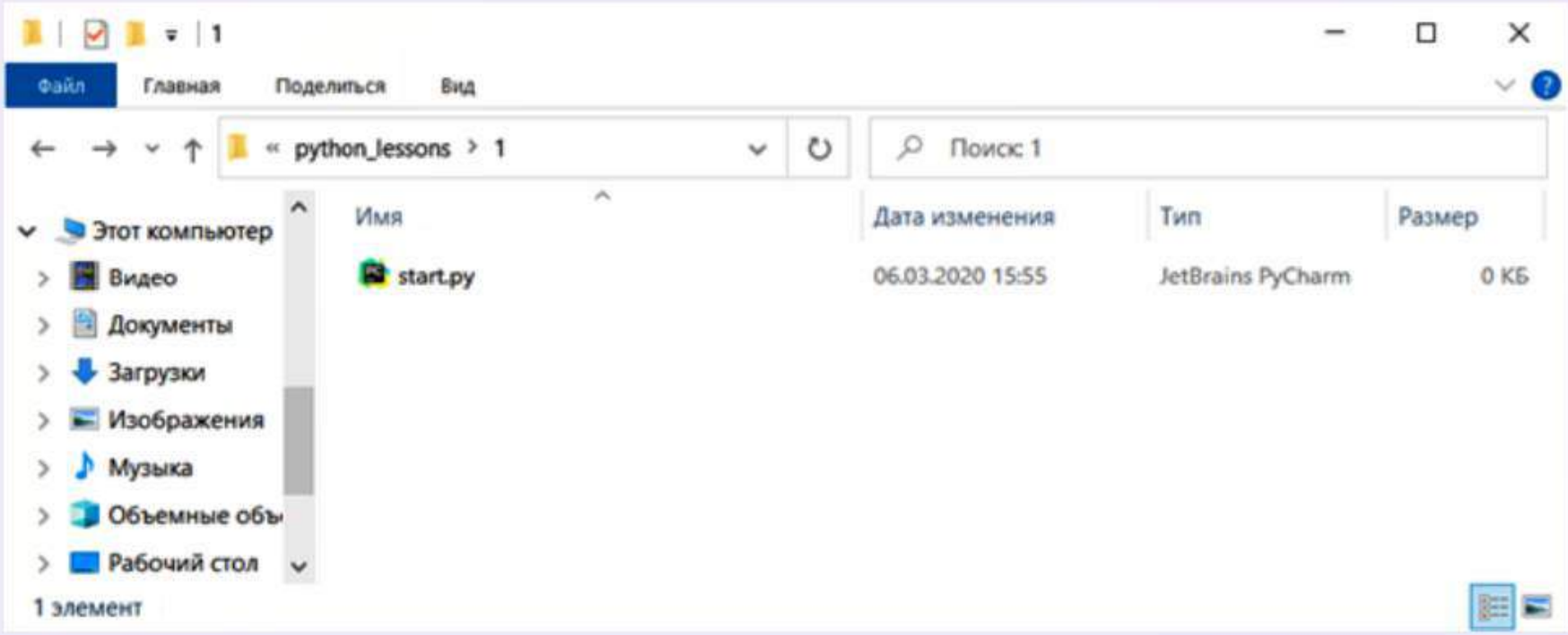


> Как видите, интерпретатор Python выполнил свою работу и вывел в консоль необходимое нам значение. Сейчас мы выполнили простейший Python-скрипт, однако это далеко не всё, на что способен данный язык.
>
> Как можно догадаться, язык Python это набор синтаксических конструкций (специальных слов и символов), на основе которых пишутся те или иные прикладные программы, затем эти программы загружаются в интерпретатор Python, который в свою очередь их исполняет.
>
> Если говорить проще, то мы просто пишем текст, на основе определённых правил языка Python, а затем загружаем его в специальную программу – интерпретатор Python.

Запускаем скрипт из файла

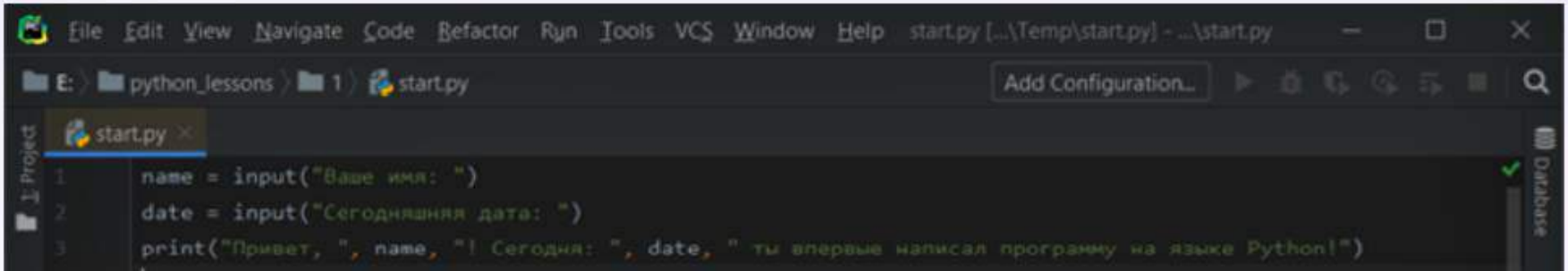
Более сложные и многочисленные инструкции, как правило вводят не в консоль, а в отдельный файл, с расширением `.py`, например: `account.py`, данный файл, может содержать логику работы с аккаунтами на сайте и т.д.

Практика. Создайте в удобном для вас месте, на вашем ПК папку `python_lessons`, внутри неё создайте папку под названием 1, в ней создадим файл `start.py`:



Откройте созданный файл в текстовом редакторе и введите следующий код:

```
name = input("Ваше имя: ")
date = input("Сегодняшняя дата: ")
print("Привет, ", name, "! Сегодня: ", date, " ты впервые написал программу на языке Python!")
```

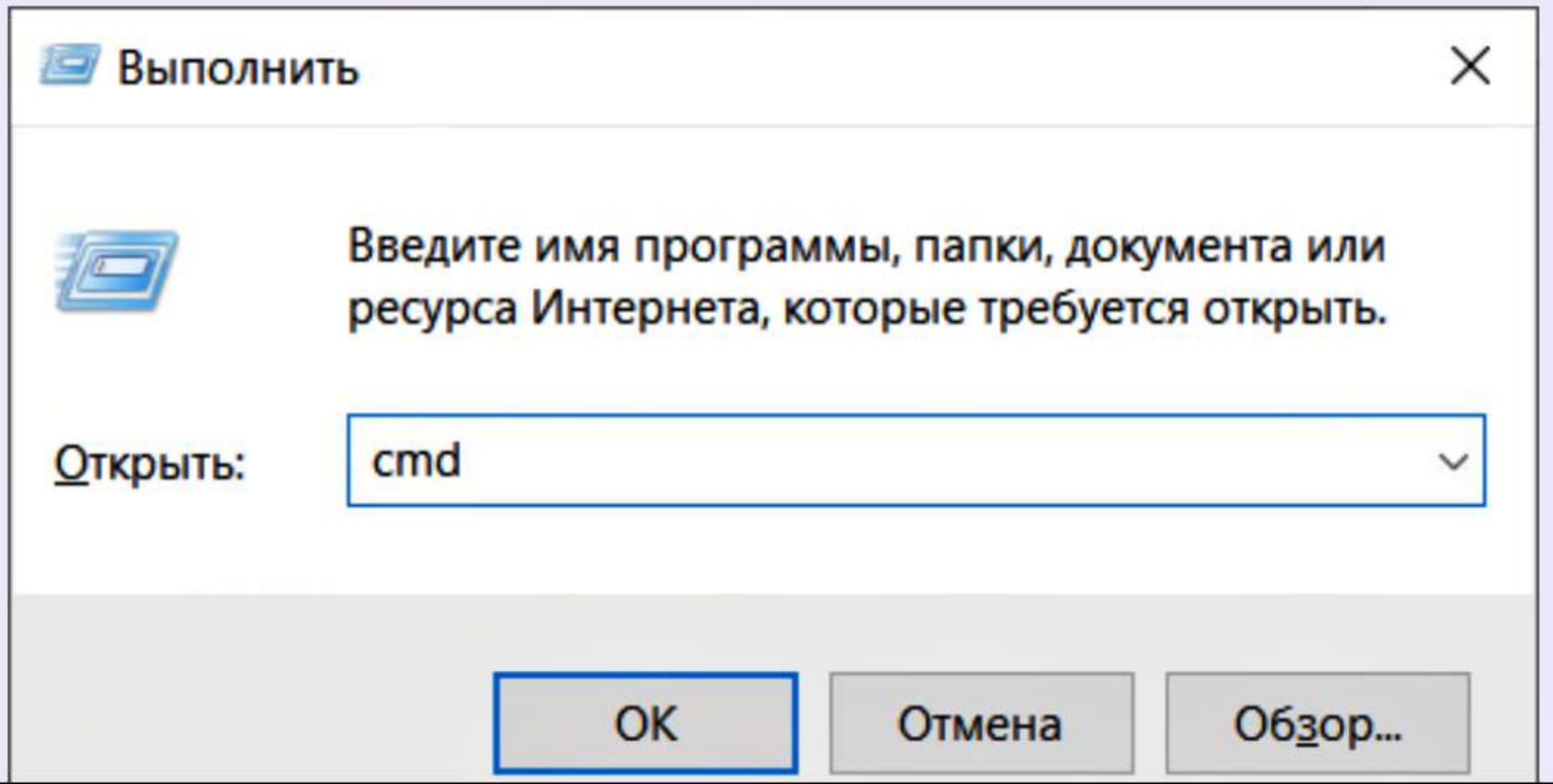


Конструкция input позволяет сохранить в переменную данные, которые вы вводите в консоли. В данном случае, мы сохраняем два значения из консоли. Сперва, мы вводим имя пользователя, а затем текущую дату.

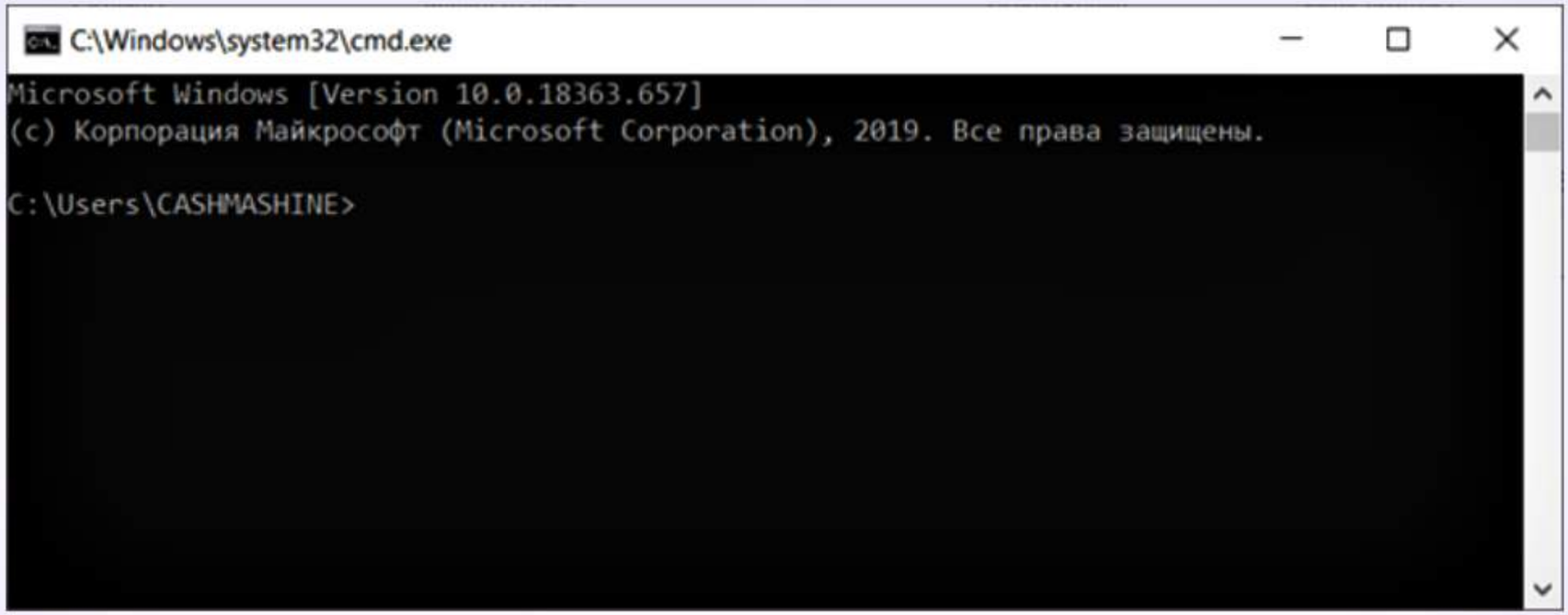
После этого, конструкция print выводит на основе этих данных модифицированный ответ.

В данном случае, эти команды бесполезны, но мы, например, можем вводить адрес сайта, который понадобится скрипту для парсинга, или логин с паролем для авторизации и т.д.

Сохраним файл, и затем запустим его. Для начала нам нужно запустить консоль, для этого нажмите сочетание клавиш «WIN + R» появится окно:



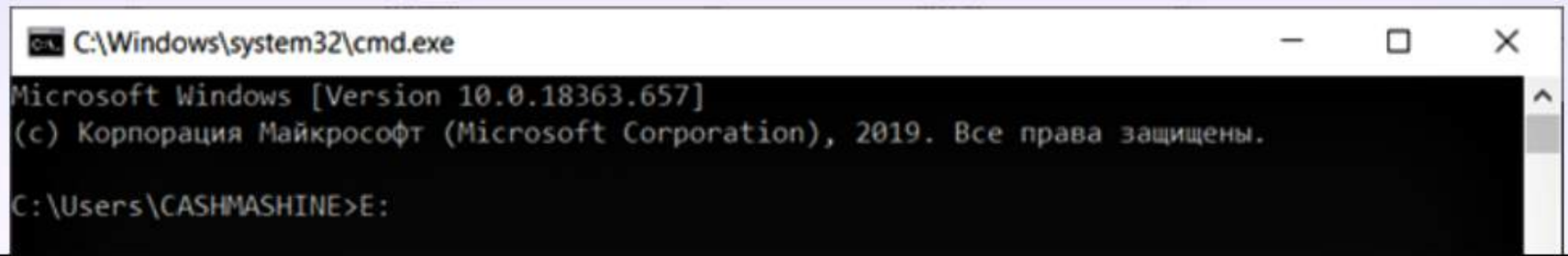
В этом окне, в поле ввода пропишите «cmd» и нажмите клавишу «ОК». Если вы всё сделали правильно, откроется терминал:



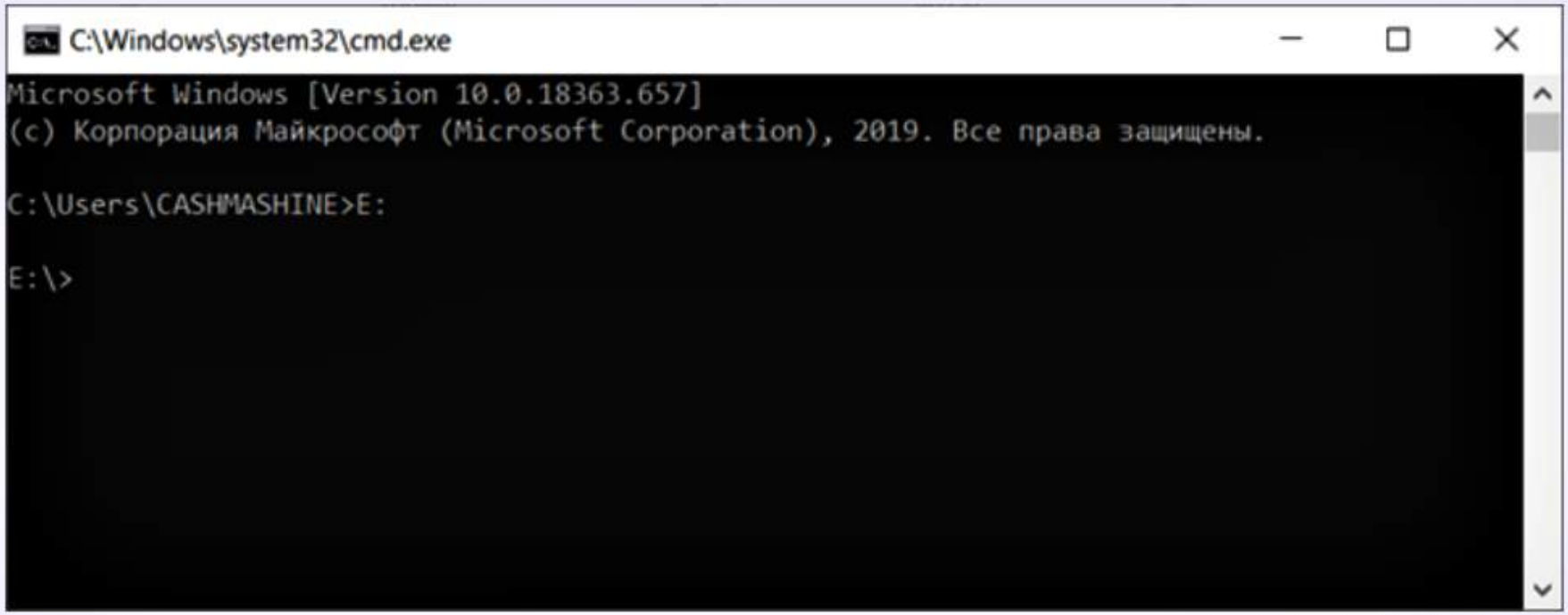
Теперь нам необходимо через консоль войти в папку, в которой лежит наш файл с python-скриптом.

Если ваш файл лежит на другом диске, или вам просто нужно перейти в самое начало диска, то достаточно ввести в консоль следующую команду: <Буква диска>:

Например, если ваш файл лежит на диске E, как у меня:

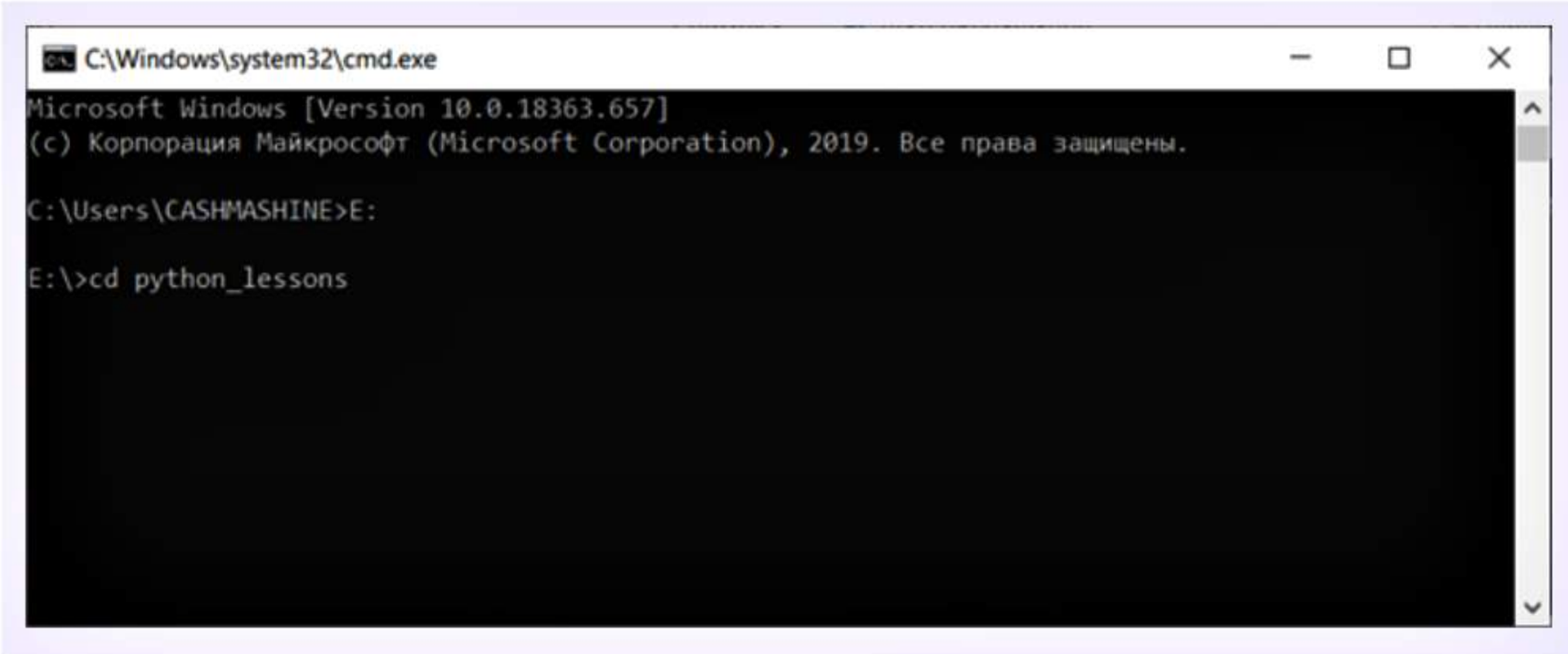


После чего, нажимаем клавишу «Enter» и перемещаемся в начало диска «E»:

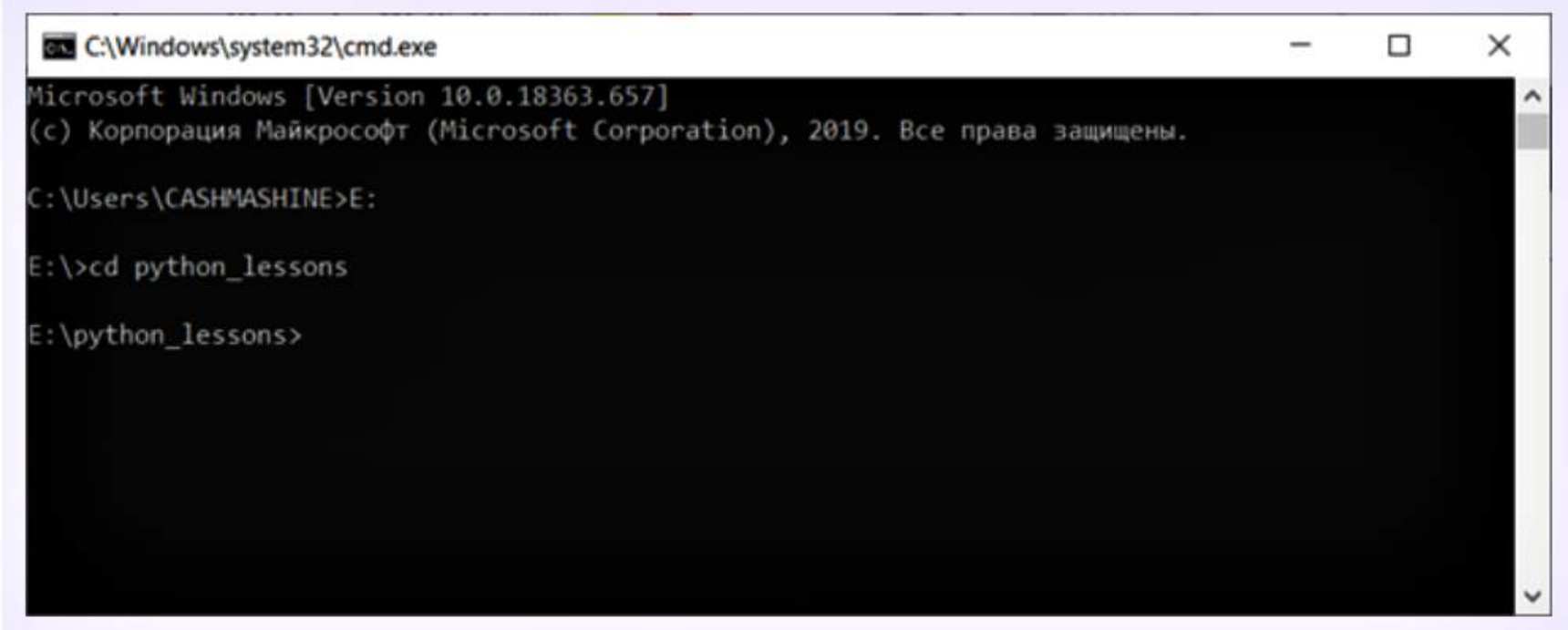


Теперь нам нужно перейти в рабочую папку. Мой файл находится по адресу:
`E:\python_lessons\1\start.py`

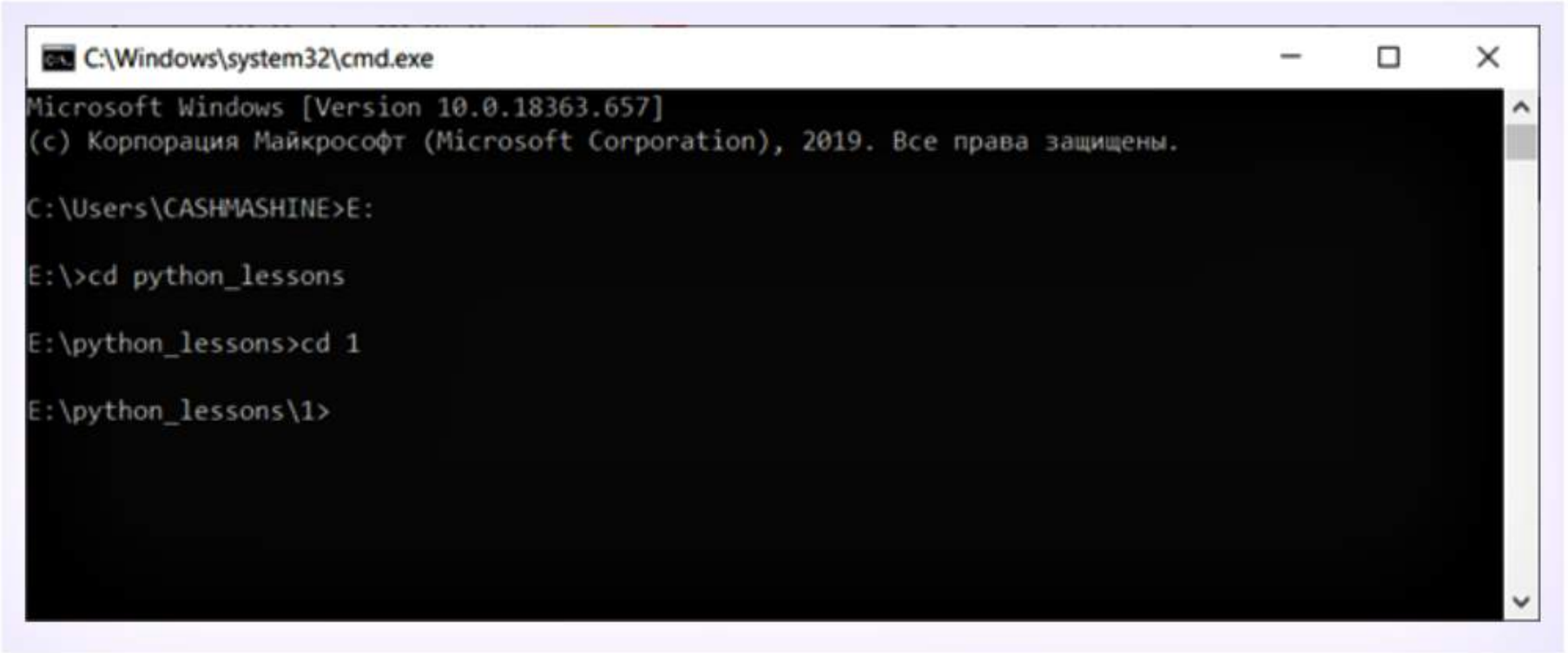
Для этого в консоли, я использую директиву cd, она позволяет перейти в нужную папку, если такая существует, например:



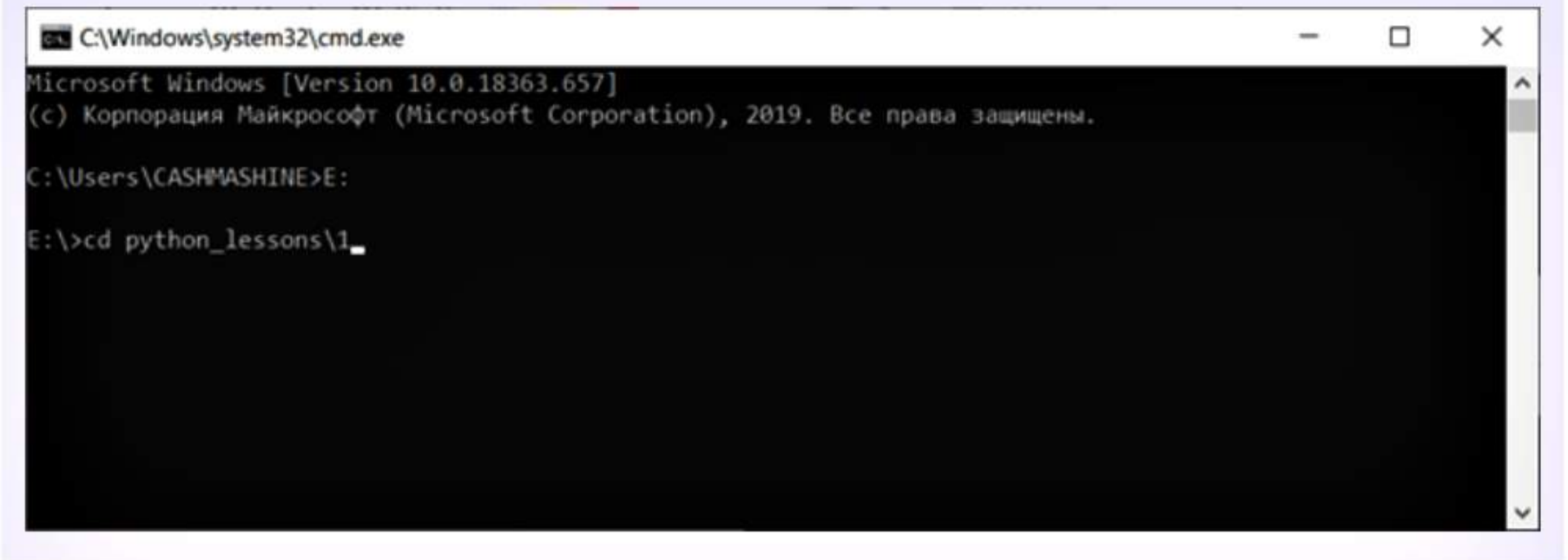
Нажимаем «Enter» и мы заходим в папку `python_lessons`:



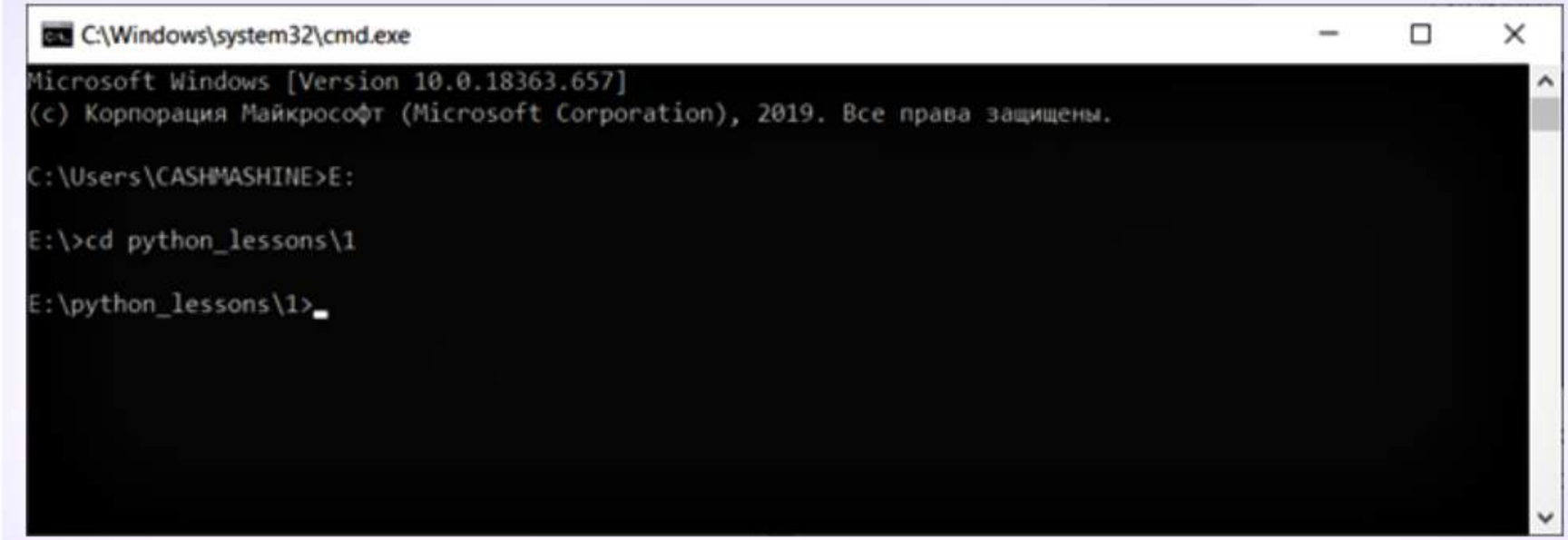
То же самое делаем для перехода в папку 1:



То есть, мы соблюдаем уровень вложенности. Сперва заходим в основную папку, затем в ту, что внутри неё и так далее. Хотя можно прописать сразу вот так:

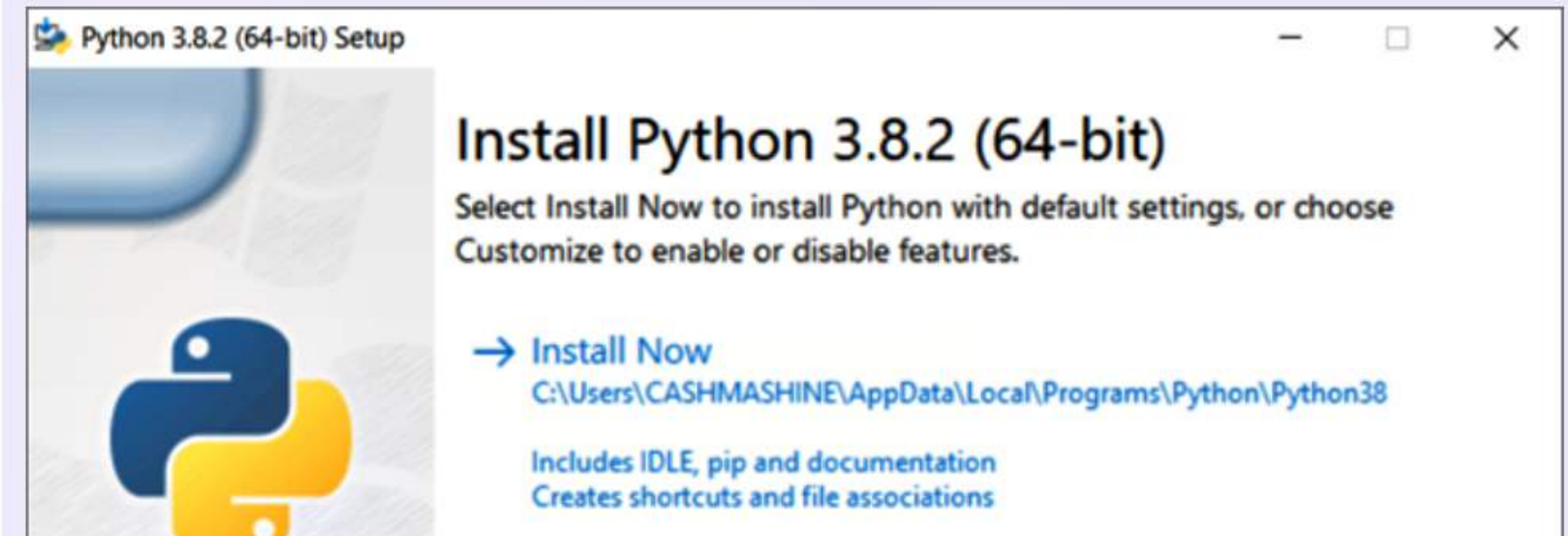


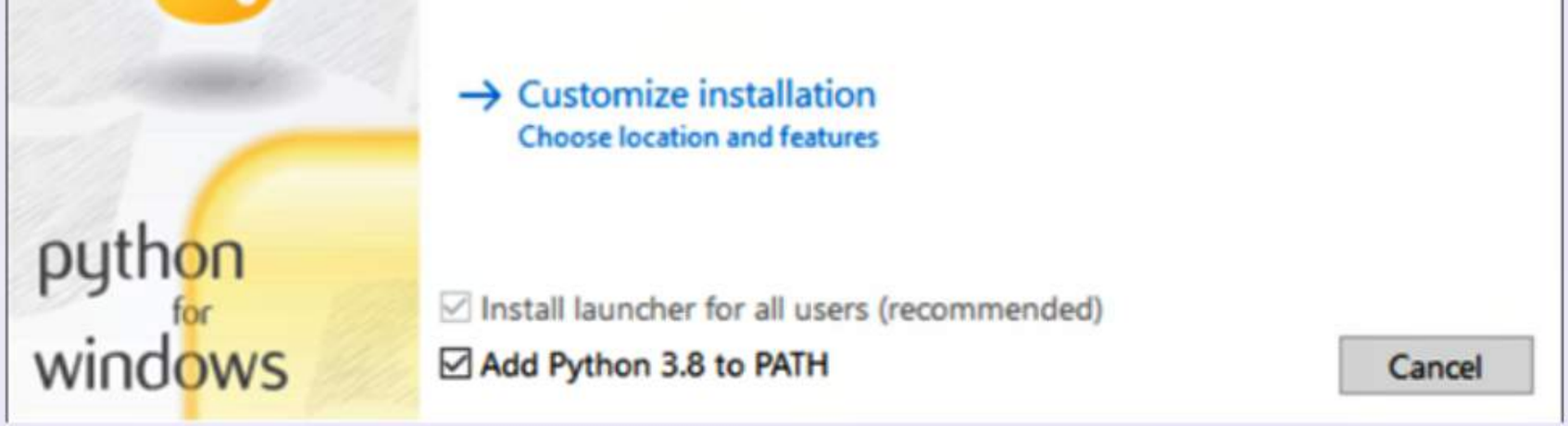
И тогда мы одной командой переходим в нужный нам раздел:



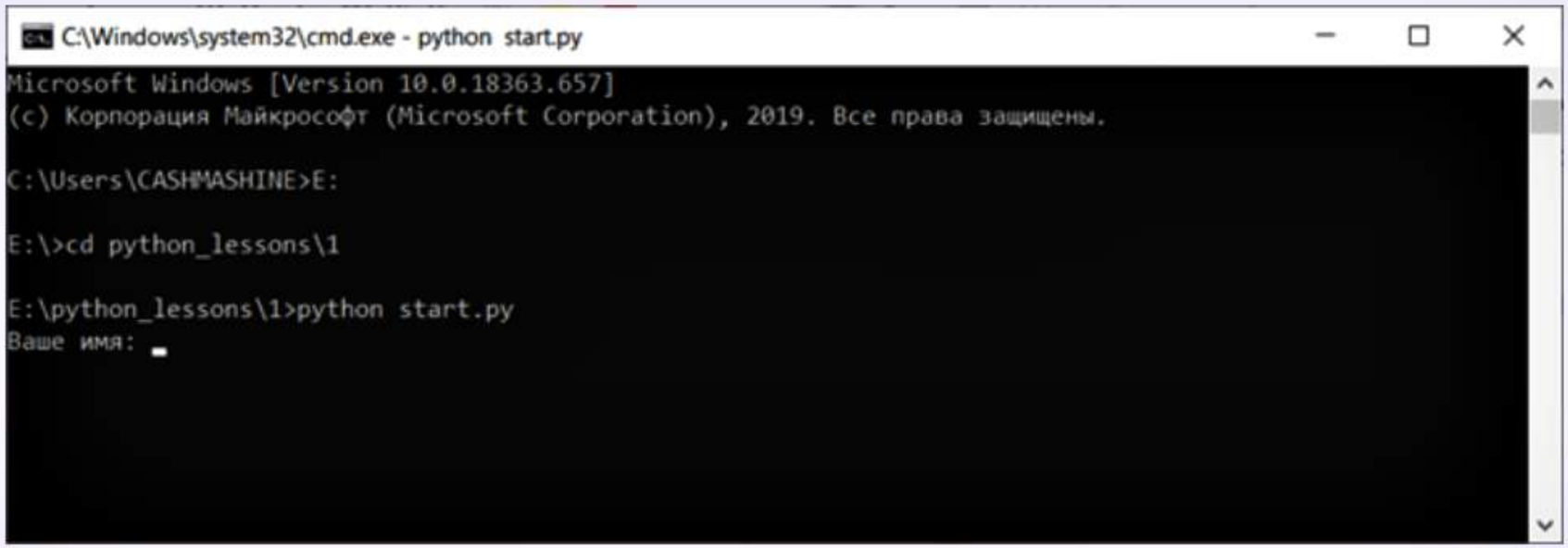
Итак, мы находимся в директории, в которой лежит наш файл. Для того, чтобы его запустить нам достаточно ввести команду вида: python `<имя_скрипта>.py`, например: `python start.py`

Важно понимать, что команда python будет работать, только если при установке интерпретатора вы указали флажок «Add Python to Path»:

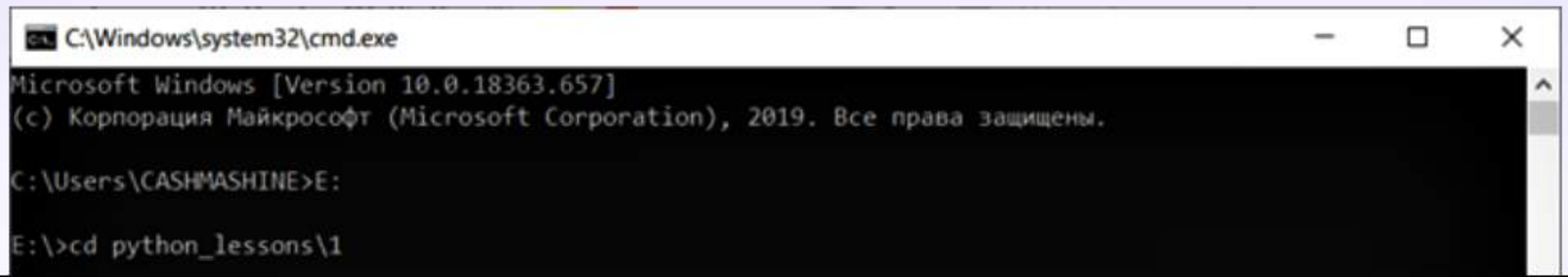




Запустив данную команду, мы можем увидеть что скрипт начал работу:

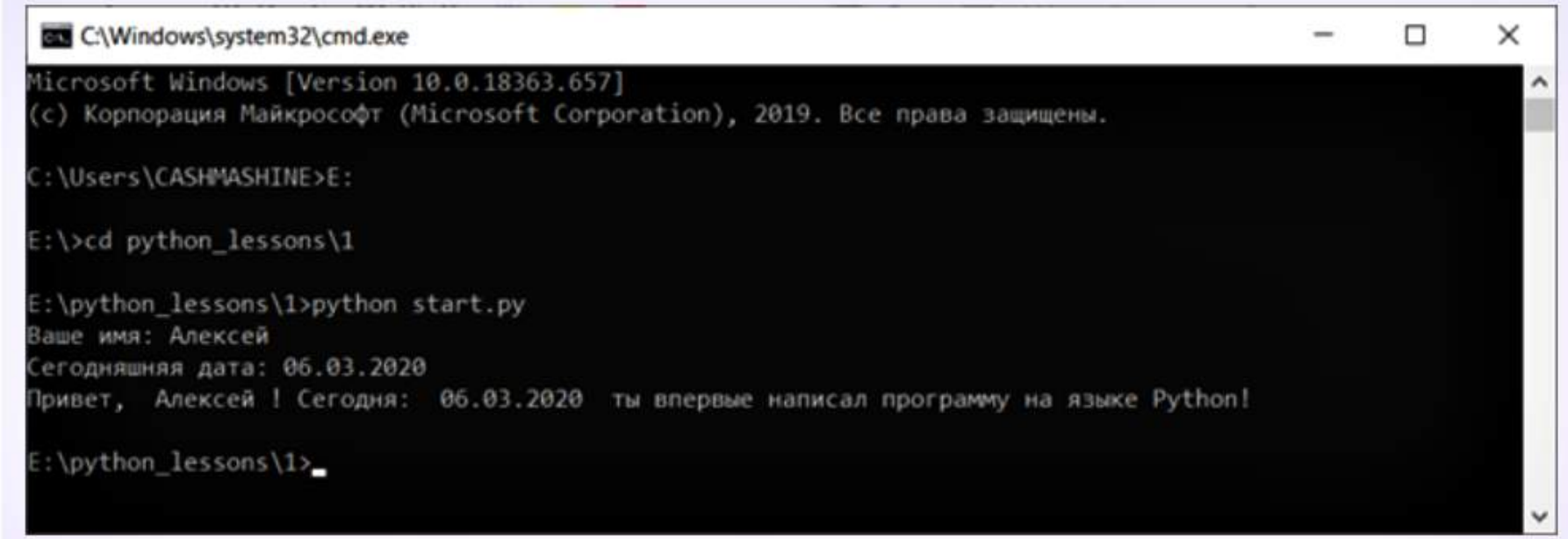


Заполняем данные для скрипта и получаем результат:



```
E:\python_lessons\1>python start.py
Ваше имя:
```

Заполняем данные для скрипта и получаем результат:



Всё, как и планировалось: вы вводите своё имя и текущую дату, а скрипт на основе этих данных формирует ответ. Напомню, что исходный код лежит в файле start.py, инструкции в котором мы передаём на выполнение интерпретатору, через консоль.

В дальнейшем, мы будем записывать в такие файлы инструкции для автоматизации рутинных задач в сфере заработка в интернете и таким же образом их исполнять, передавая написанный нами исходный код интерпретатору.

Среда разработки PyCharm

Содержание урока

Как мы говорили в предыдущих уроках, приложение на Python это совокупность текстовых файлов с кодом, имеющих расширение *.py

Код для прошлого урока можно было написать на любом текстовом редакторе по выбору. В данном уроке хочу посоветовать вам интегрированную среду разработки (IDE). IDE – это система инструментов, направленная на эффективное написание кода.

Если говорить проще, то это текстовой редактор, обладающий специальным функционалом для написания программ на том или ином языке программирования. В нашем случае, это IDE для Python.

То есть это такой набор программ, который позволяет писать программы быстрее, проще и удобнее, чем в обычном текстовом редакторе.

Из своего опыта написания кода на языке Python я вынес две наиболее приемлемые IDE, а именно:

PyCharm
Visual Studio

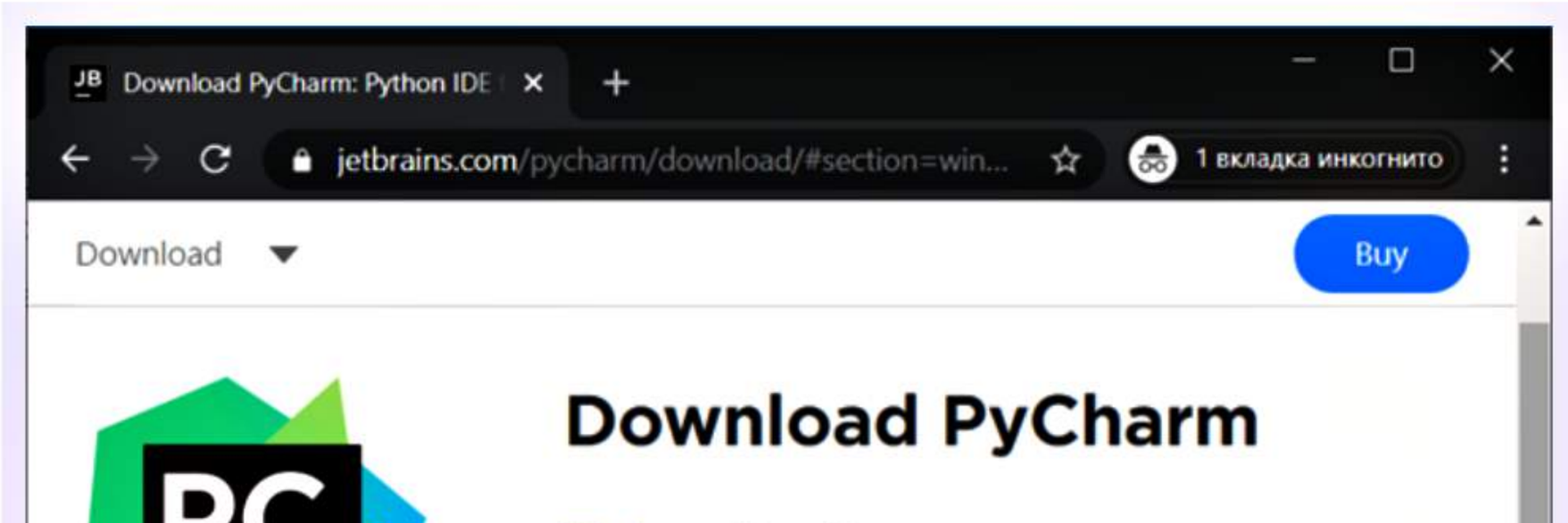
Обе IDE имеют бесплатную и бизнес-версию. Лично я предпочитаю PyCharm, потому что она специализирована только под Python и не несёт лишних зависимостей, к тому же у неё, на мой взгляд более дружелюбный интерфейс.

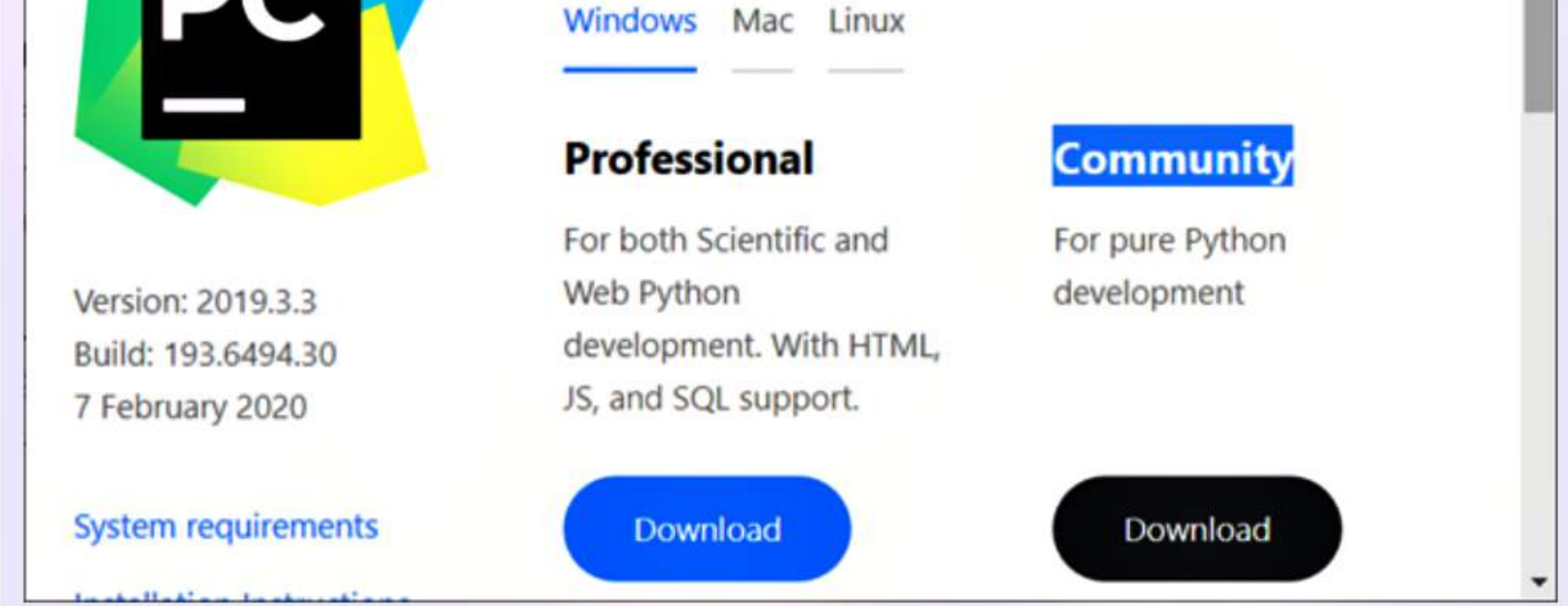
В связи с этим, настаиваю на том, чтобы в процессе прохождения курса вы использовали именно PyCharm, поскольку на основе данной IDE будут рассматриваться все примеры из курса.

Скачиваем PyCharm

Для начала скачаем и установим дистрибутив бесплатной версии PyCharm:
https://www.jetbrains.com/pycharm/download/

На странице загрузки выбираем версию для вашей операционной системе. Скачиваем Community-версию дистрибутива:

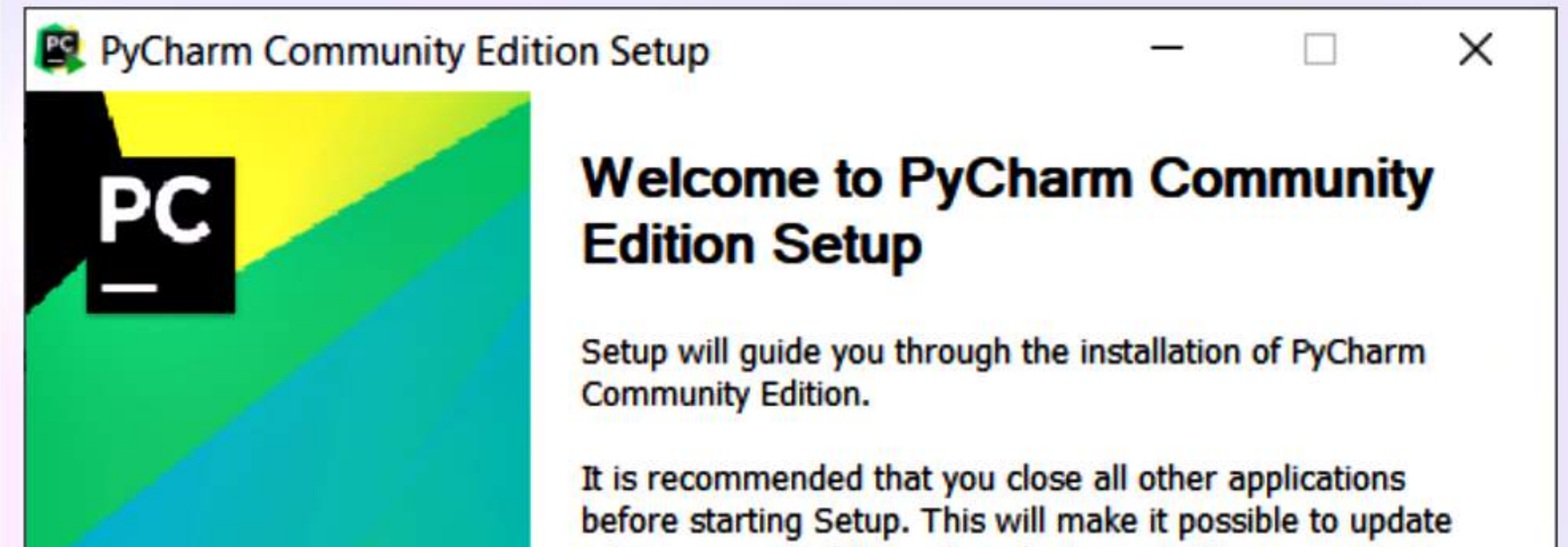


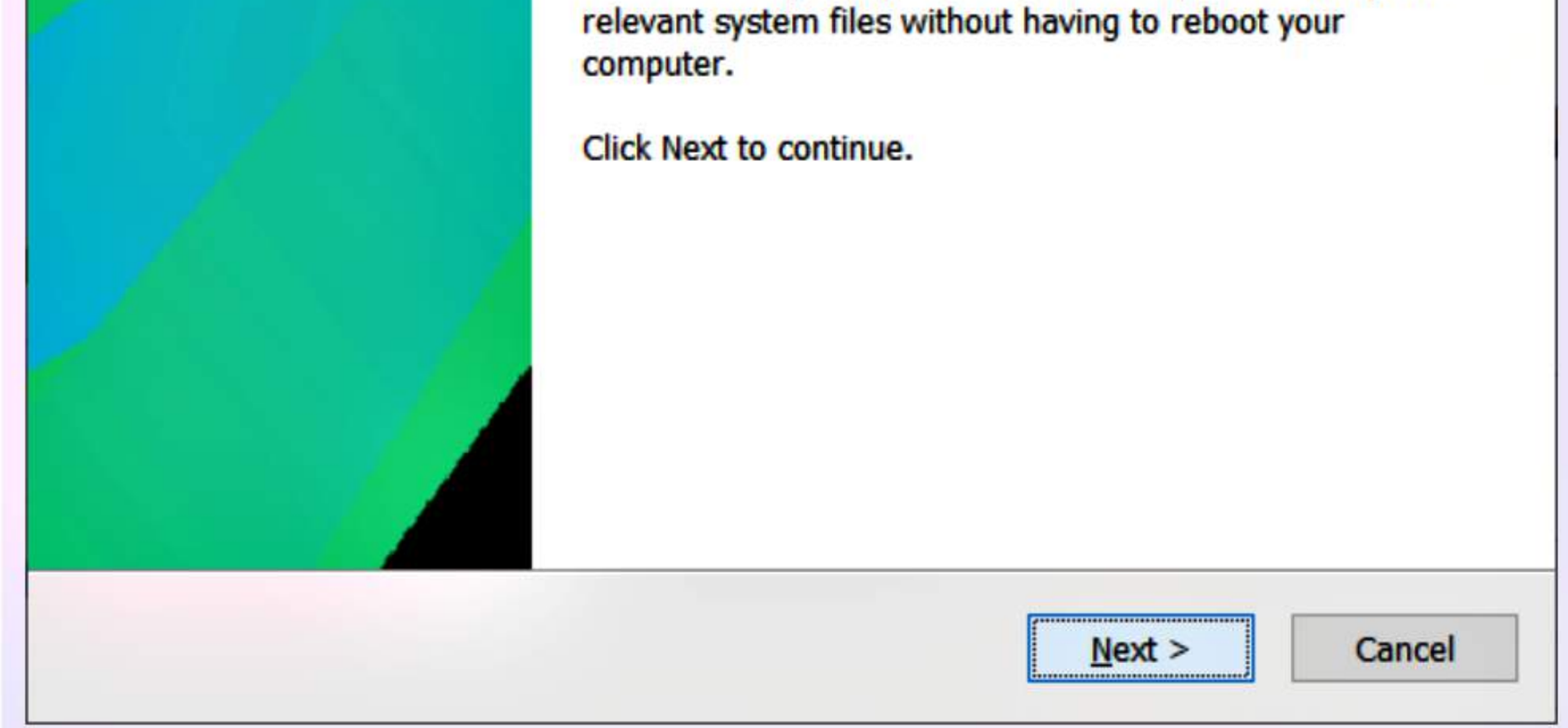


Эта бесплатная версия, которой вполне будет достаточно для обучения программированию на Python.

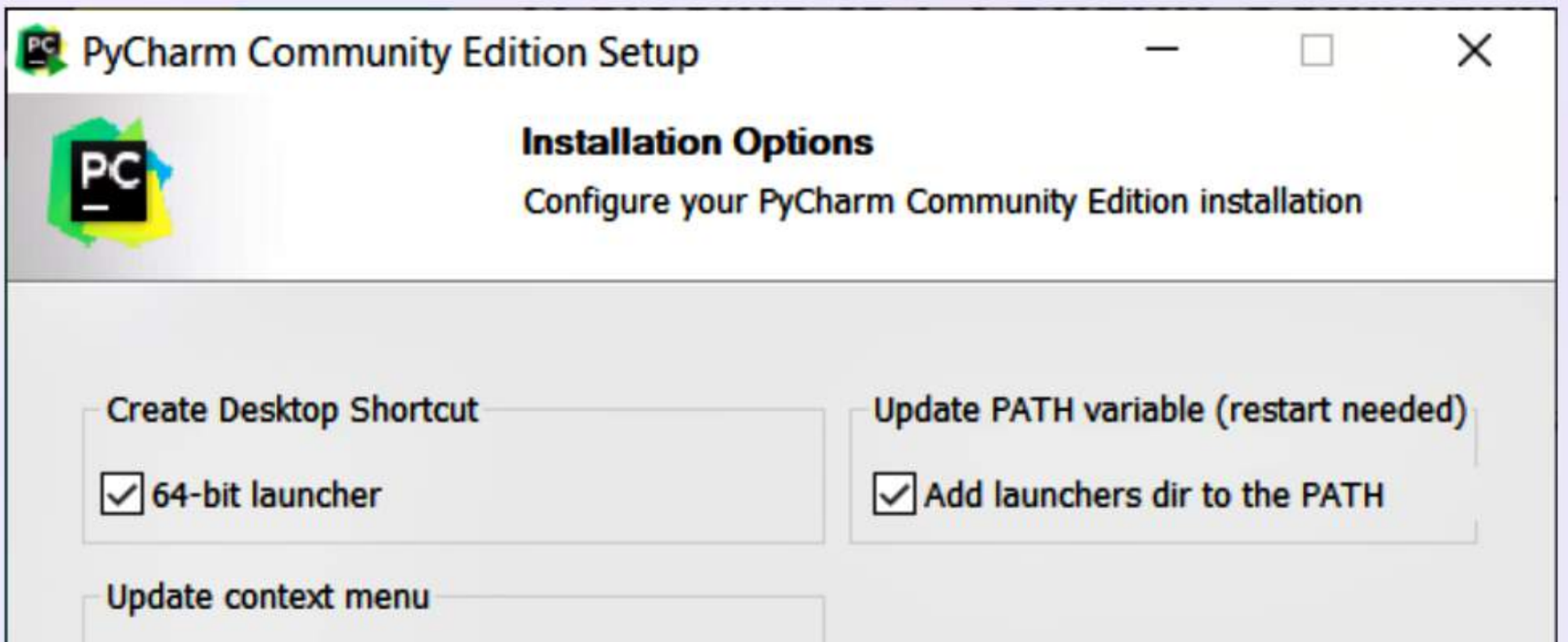
Устанавливаем PyCharm

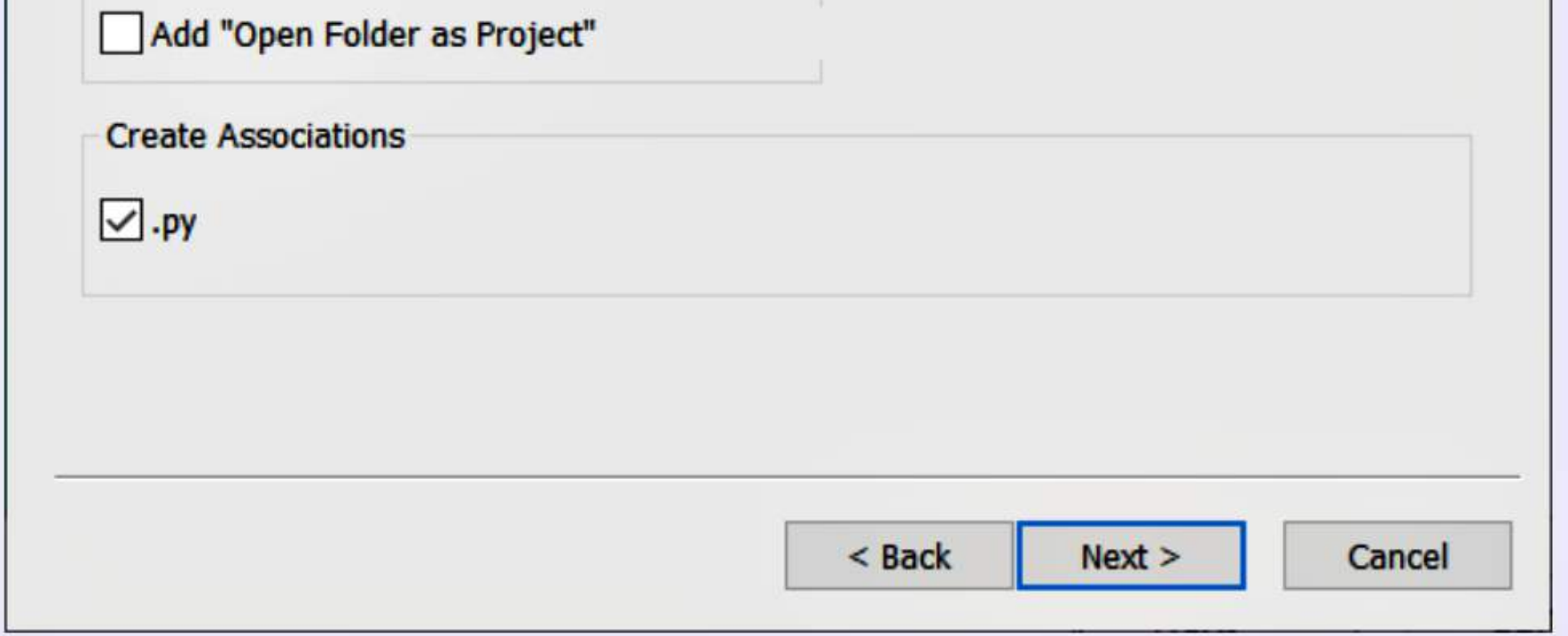
После скачивания запускаем дистрибутив:





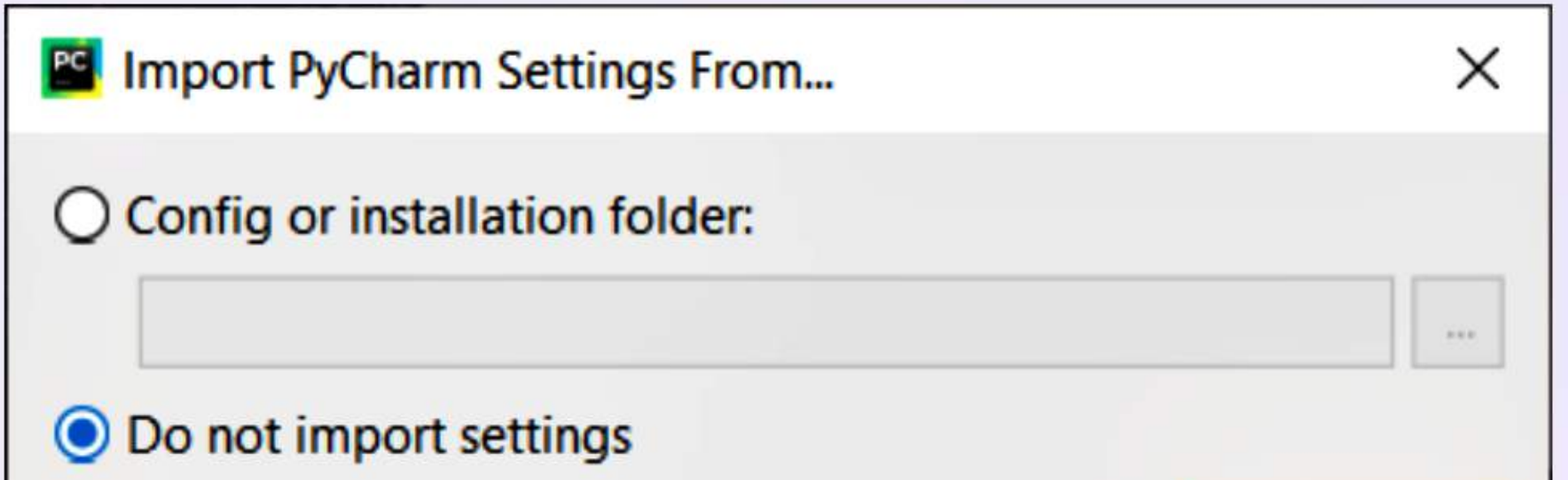
Нажимаем кнопку «Next» до тех пор, пока не перейдёте до этого окна:

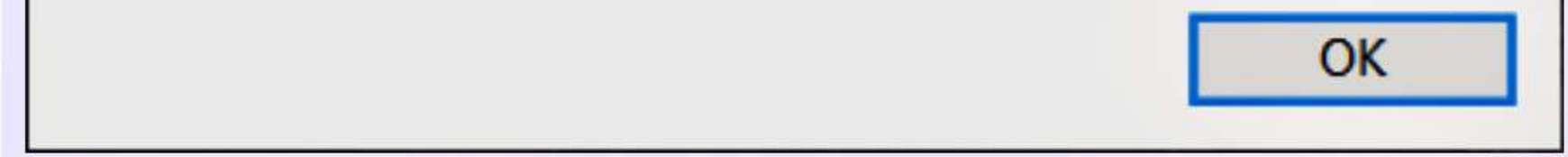




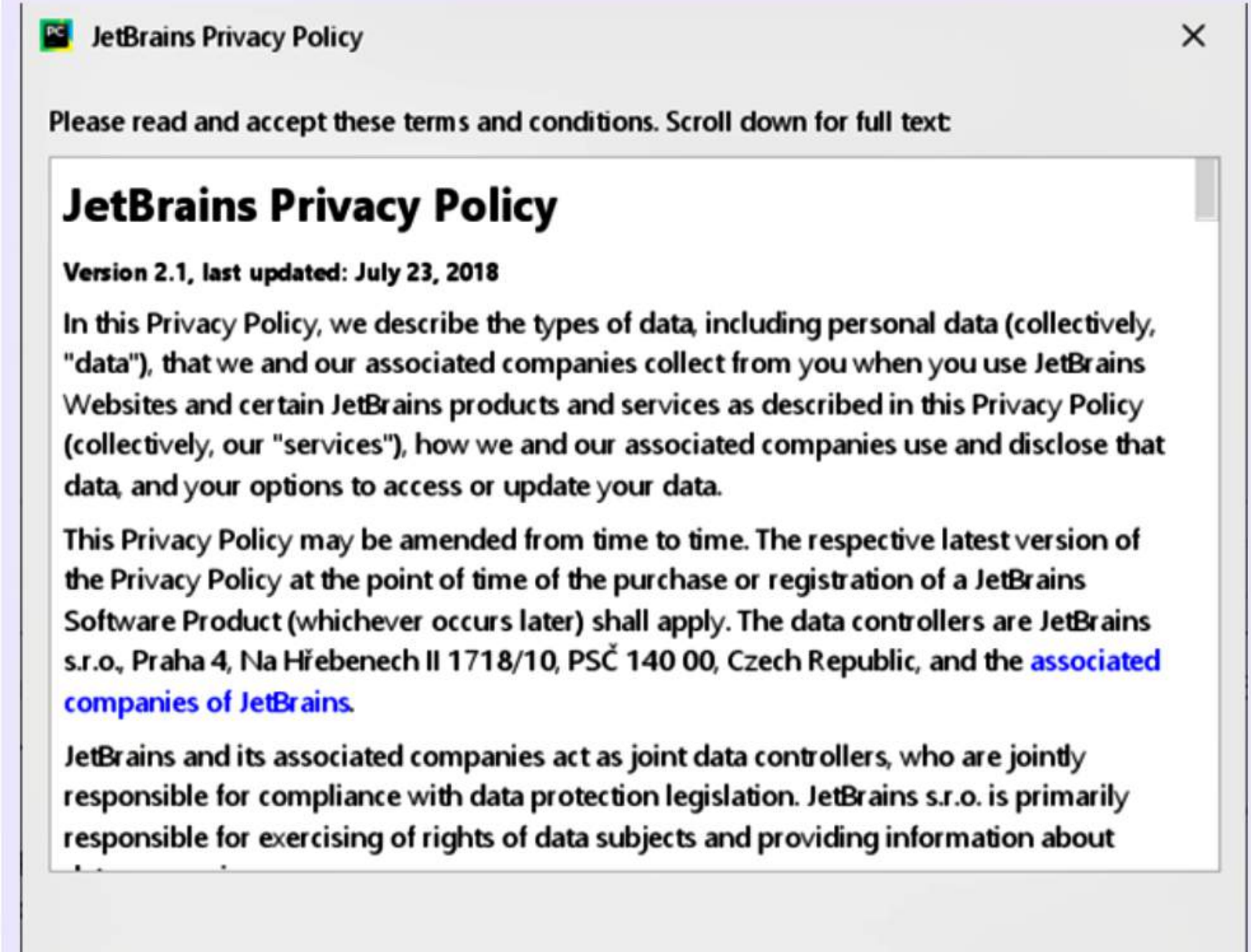
Выбираем соответствующие галочки (в том случае, если ваша система на 64 бита). Далее, ждём установку PyCharm.

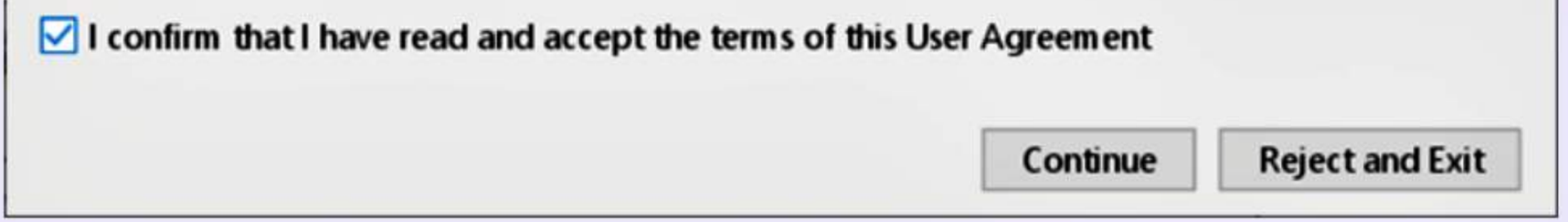
После того как IDE полностью установится на ваш компьютер, запустим её и создадим первый проект. Запускаем ярлык с рабочего стола, пока не импортируем никаких настроек:



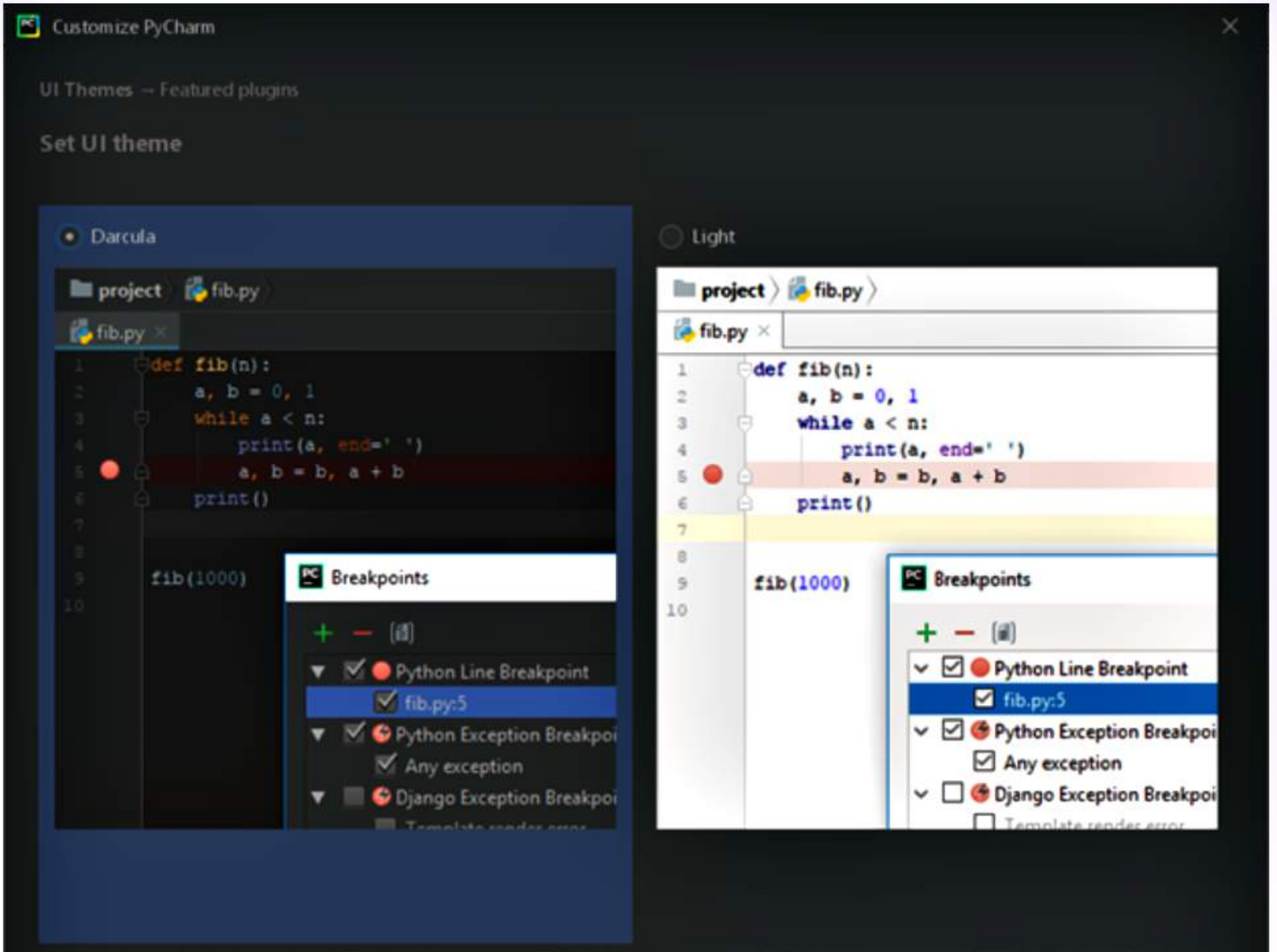


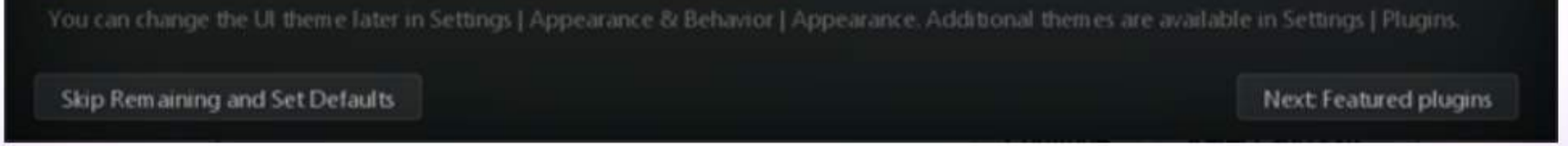
Соглашаемся с условиями:



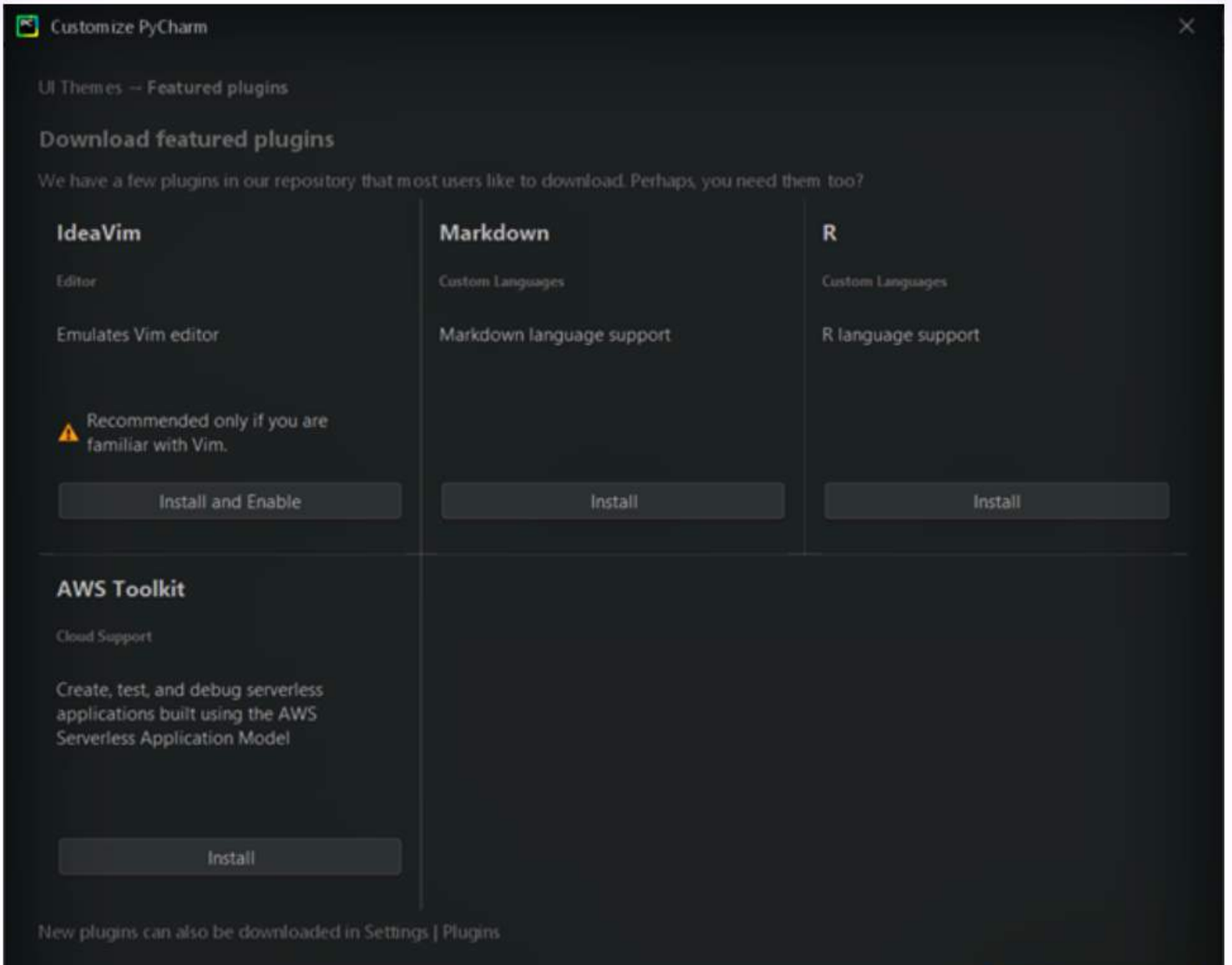


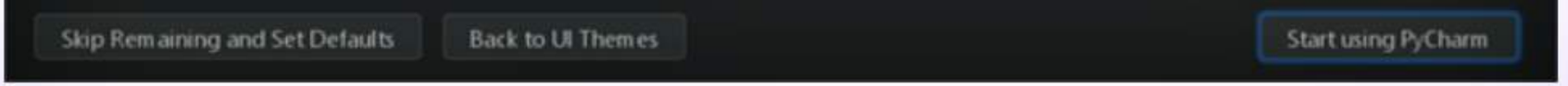
Выбираем тему оформления по собственному усмотрению:





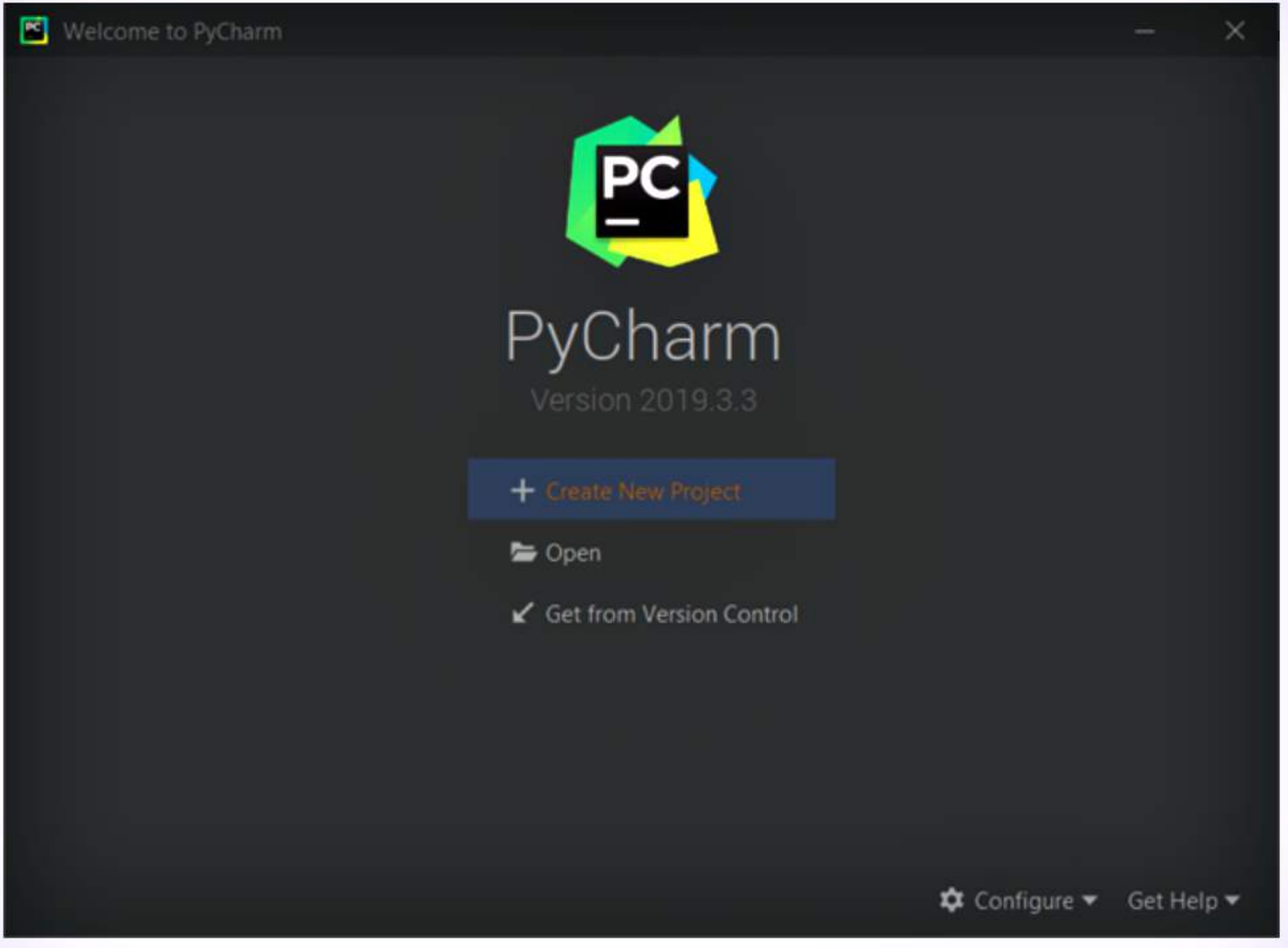
Пока не выбираем никаких дополнительных модулей:



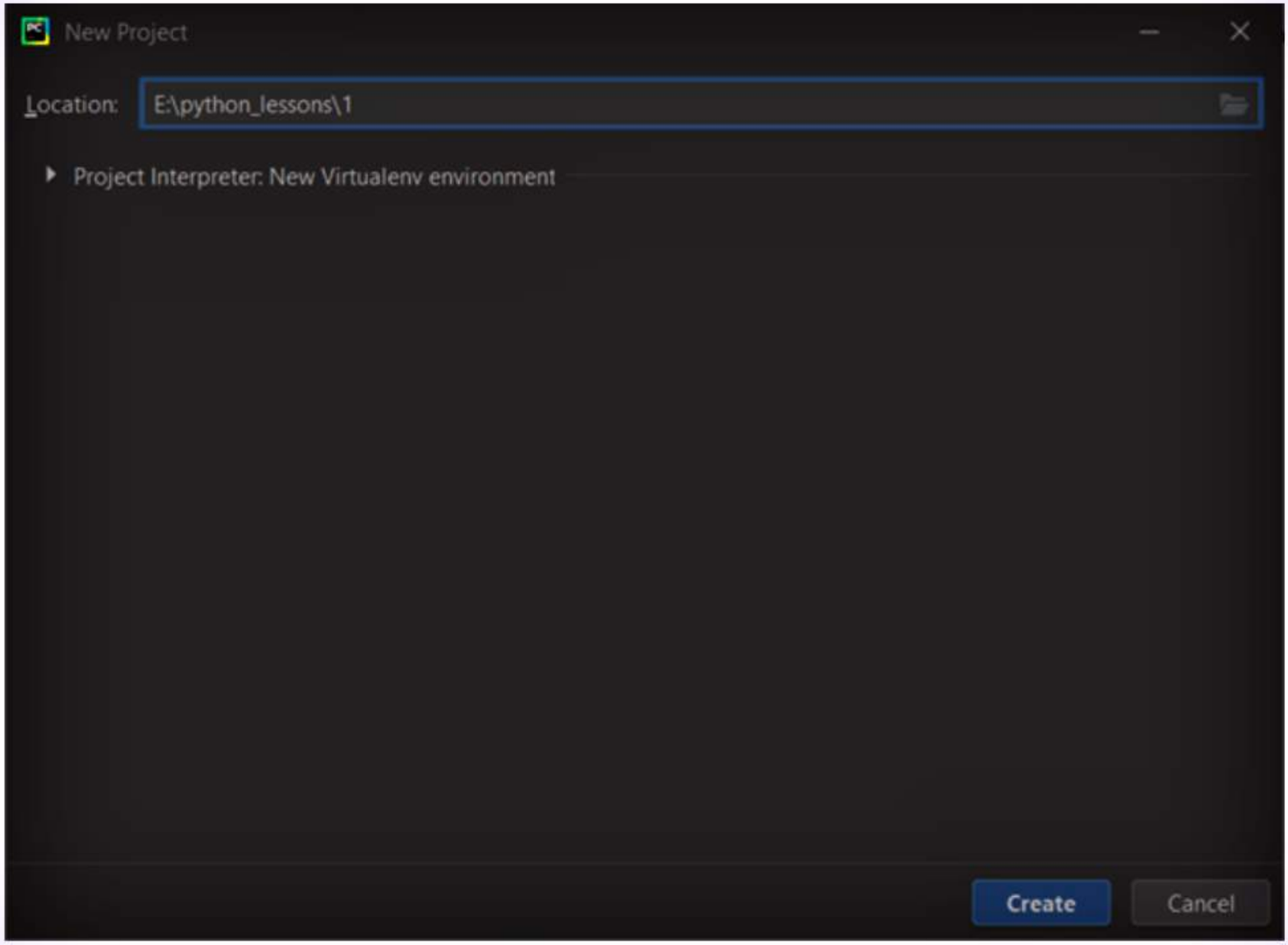


Создаём новый проект

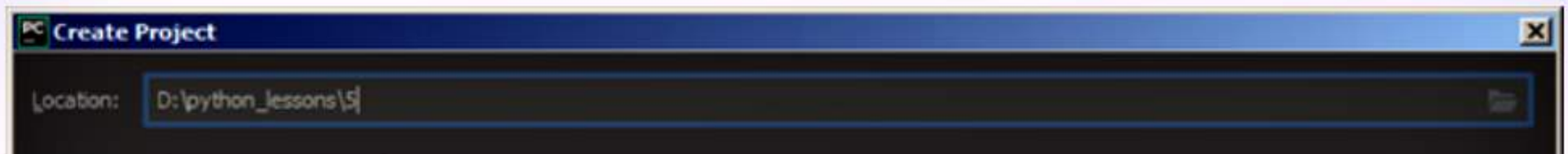
После 10000 шага мы наконец дошли до процесса создания проекта. Нажимаем кнопку `Create New Project`:

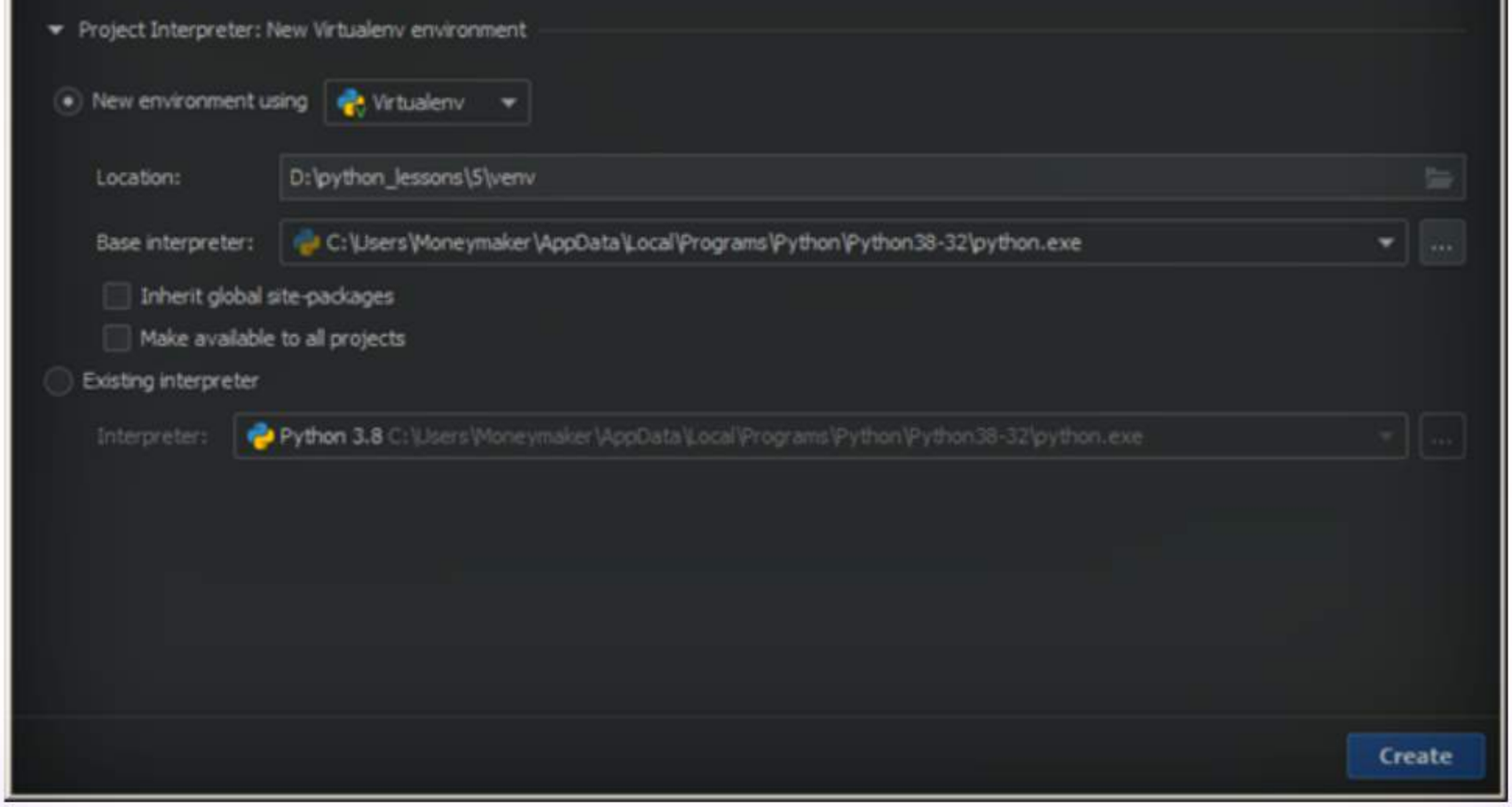


Указываем путь до папки, в которой мы будем создавать наши python-скрипты:



Советую нажать на ярлычок `Project Interpreter: New Virtualenv environment`:

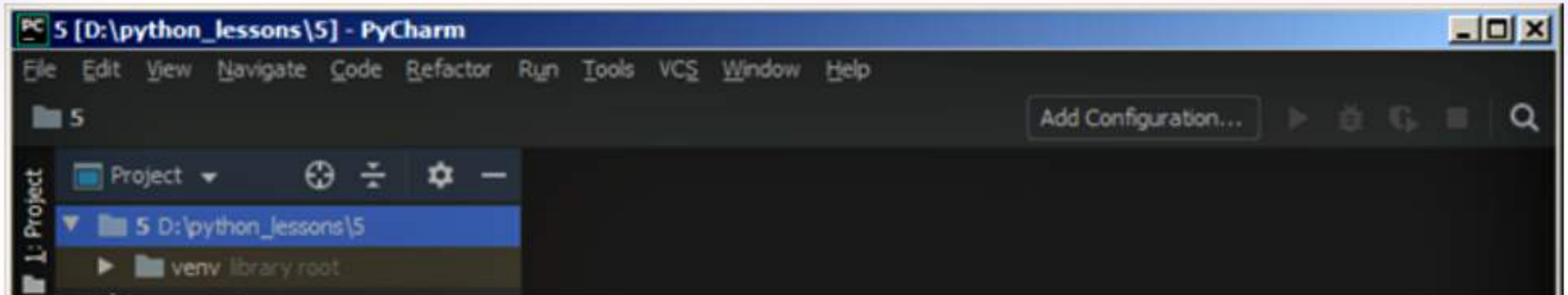


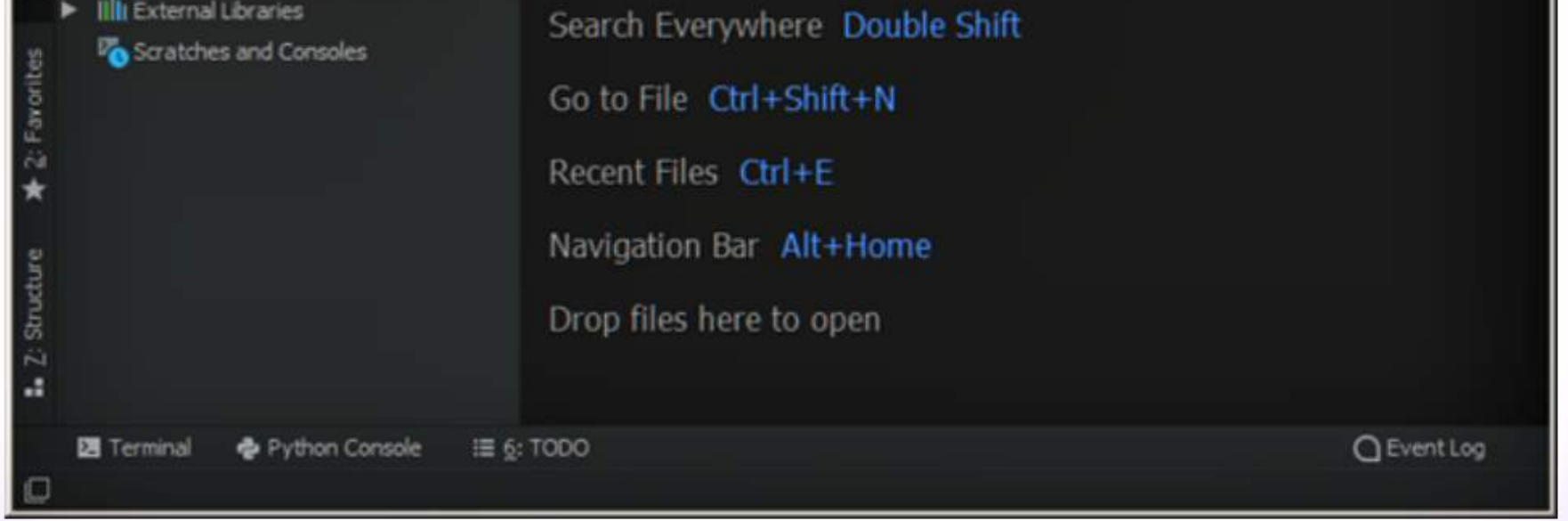


Кликните по пункту «New environment using», чтобы он стал активным. Данный пункт подключит к нашему проекту виртуальное окружение и интерпретатор Python.

Один из плюсов IDE сводится к тому, что для запуска скриптов вам не нужно записывать команды в терминал. Для этого к IDE подключается необходимый интерпретатор, а запуск скрипта осуществляется по кнопке.

Нажимаем кнопку «Create» и ждём запуска проекта:



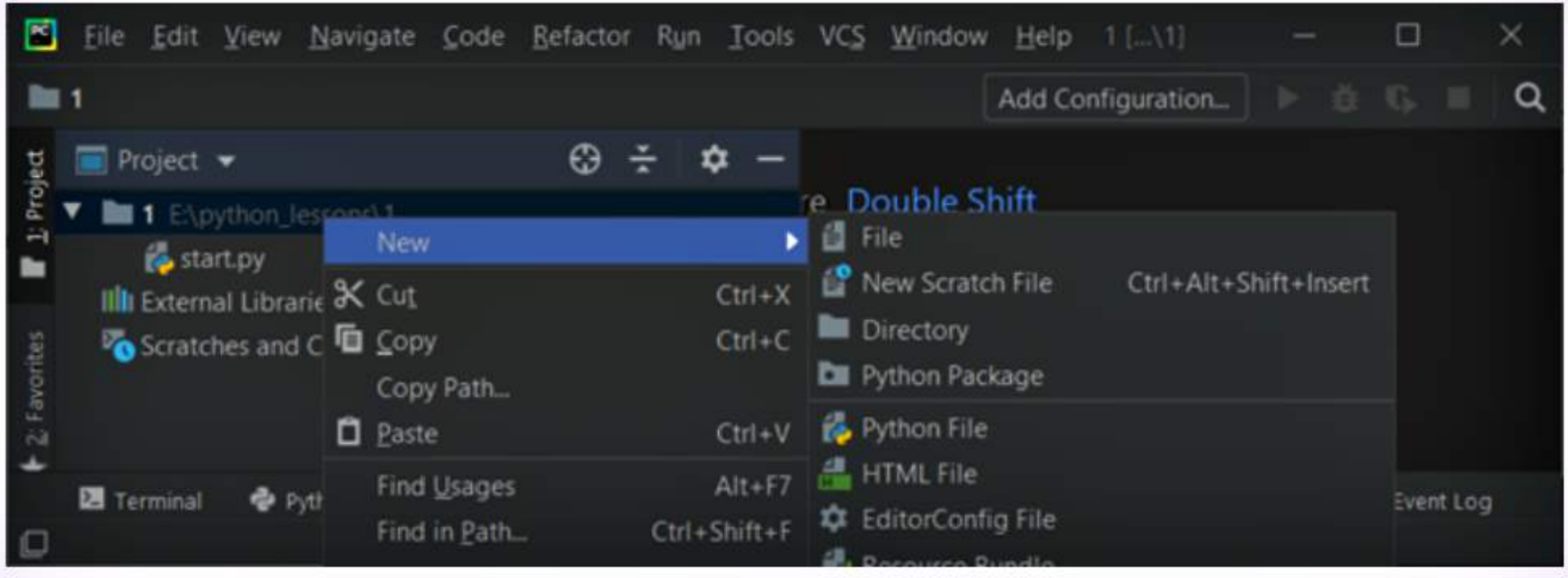


Поздравляю, ваш первый проект в PyCharm создан, теперь вы можете с удобством писать крутые и полезные скрипты на Python.

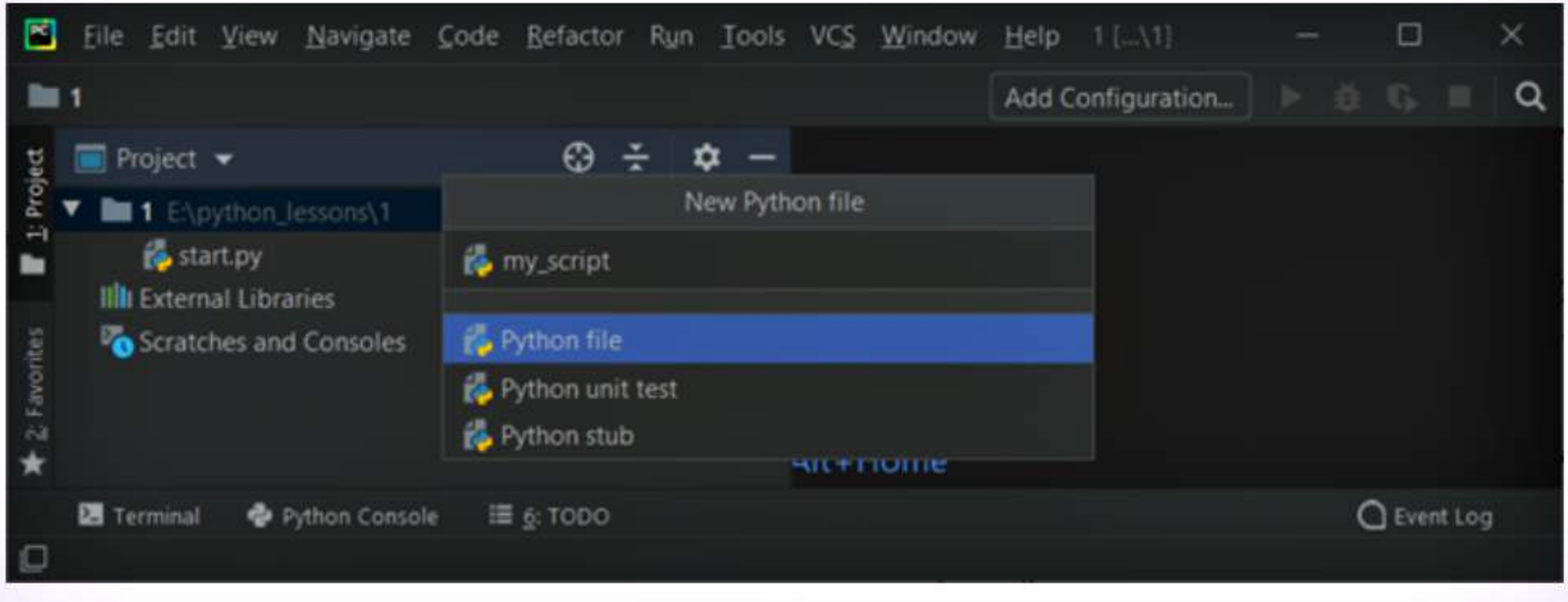
Создание файла с python-скриптом

Открываете рабочую папку в обозревателе проекта (панель слева) и нажимаете правой кнопкой по папке, в которой вы ходите создать файл скрипта. Появится соответствующее меню, где вы

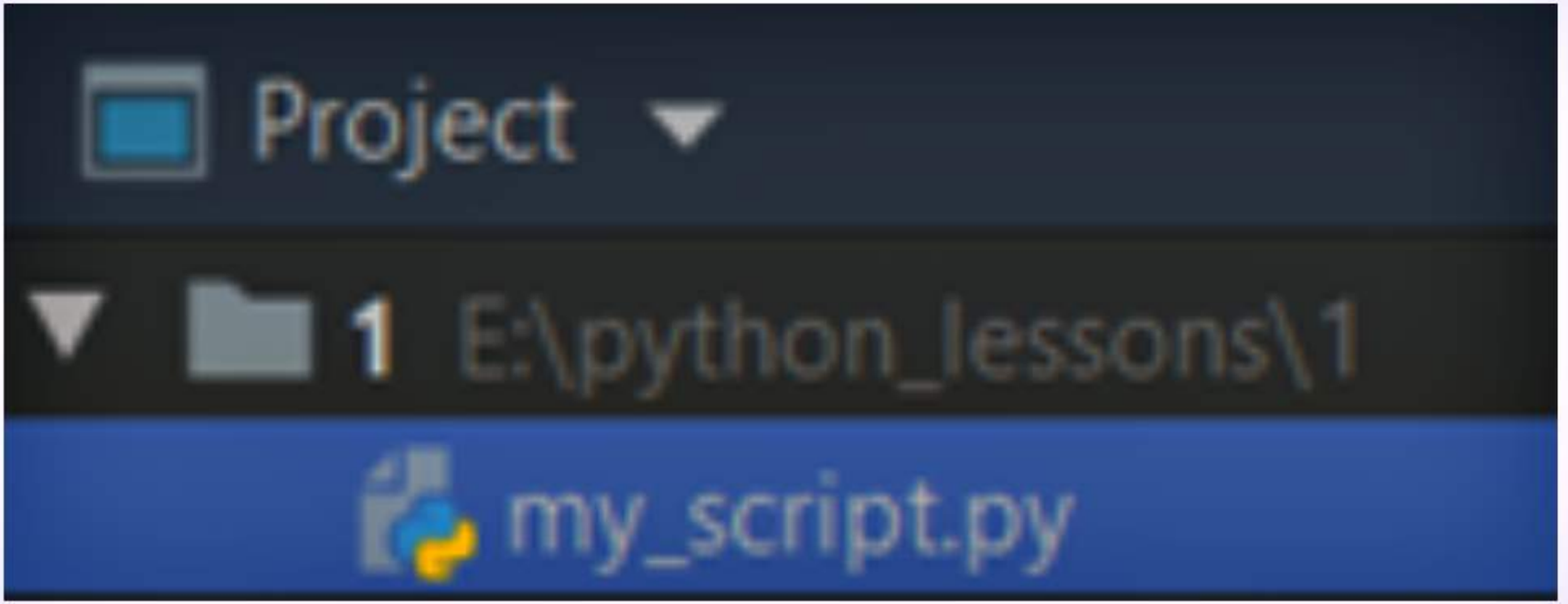
наводитесь на пункт `New`, затем, нажимаете пункт `Python File`.



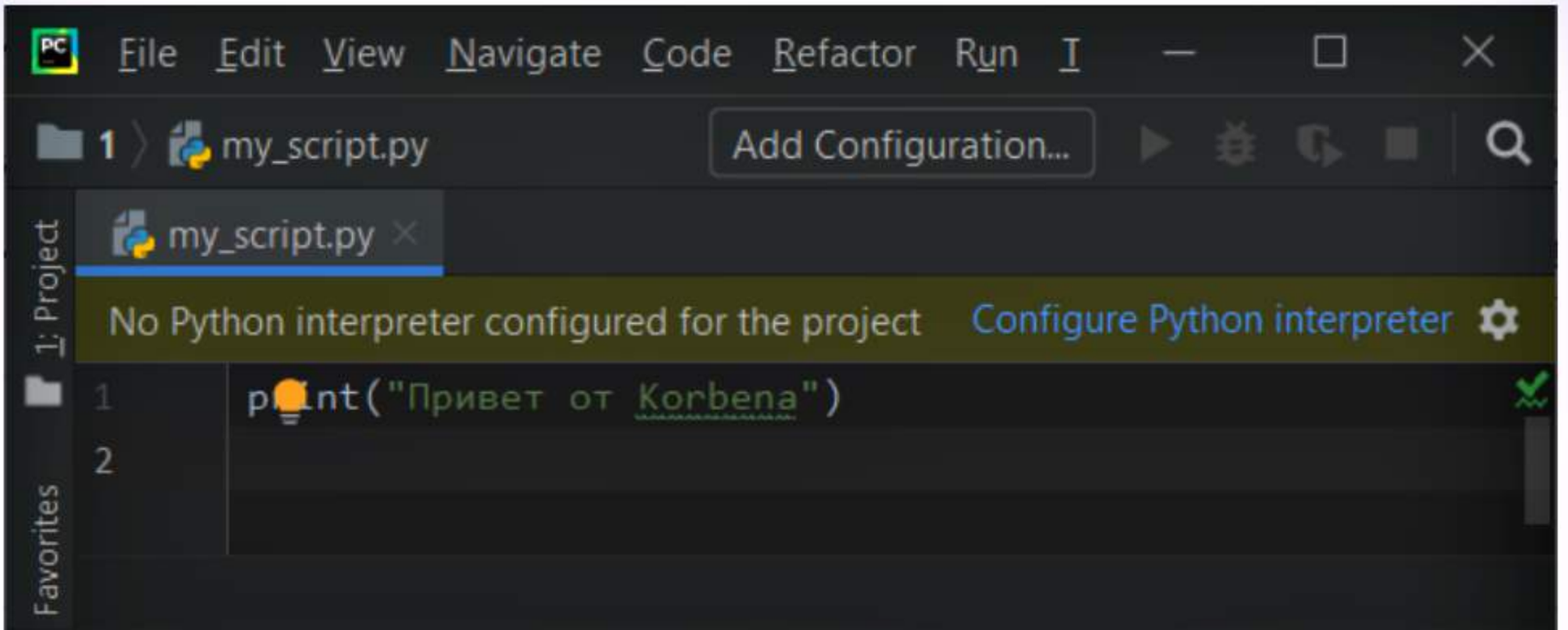
Появится окно, в котором необходимо указать имя для создаваемого файла:

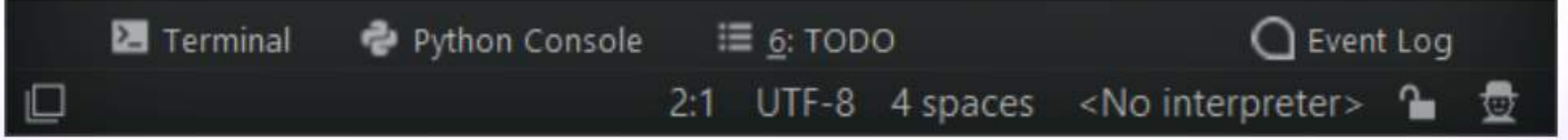


После ввода имени нажимаете клавишу «Enter» и IDE автоматически создаст новый файл для python-скрипта в указанной вами директории проекта.

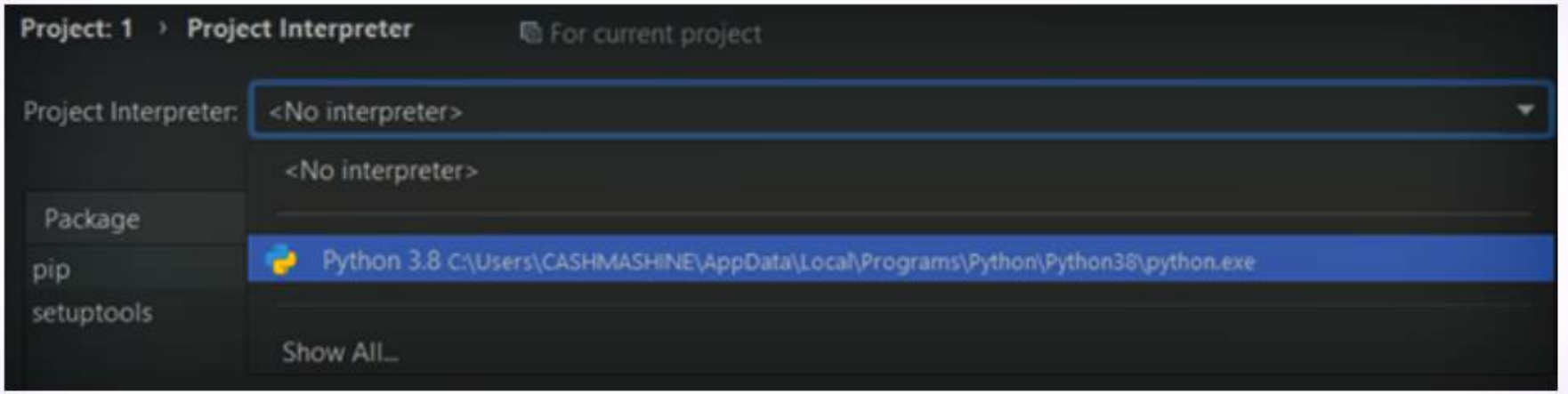


При работе с новым проектом возможно появление такого предупреждения:

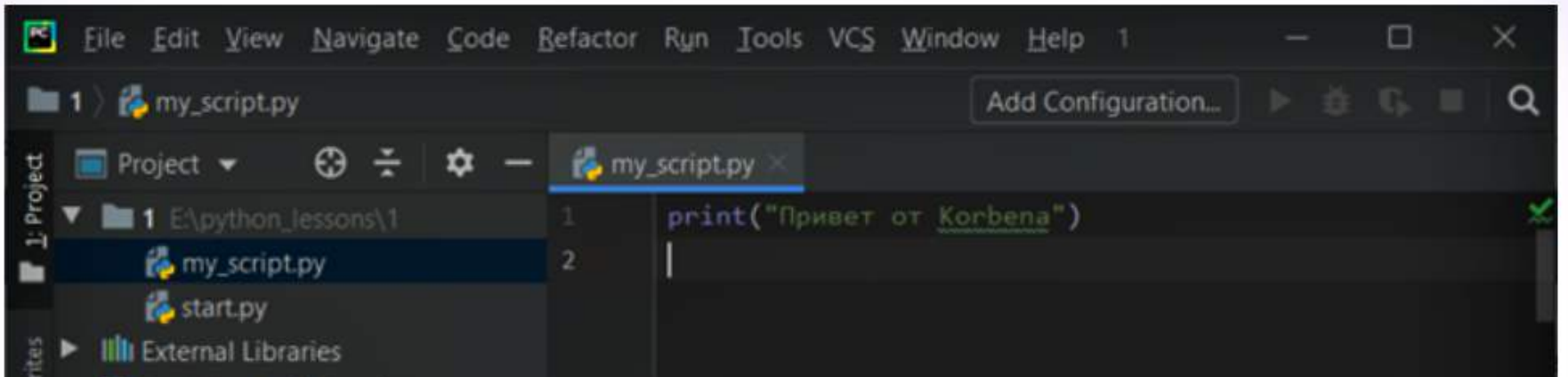


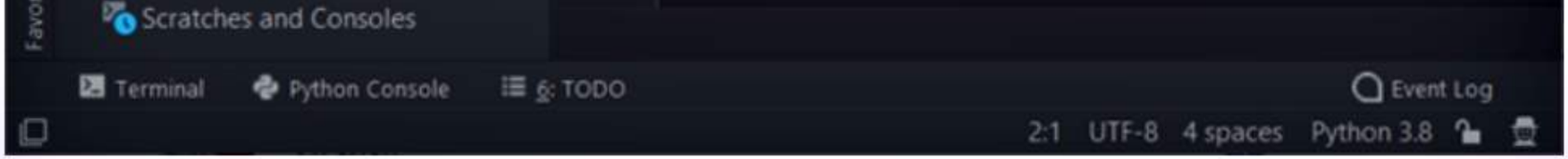


Чтобы пофиксить эту проблему, нажимаем на кнопку `Configure Python interpreter`, и в появившемся окне нажимаем на выпадающее меню `Project interpreter`, где выбираем установленный нами при первом запуске проекта интерпретатор:

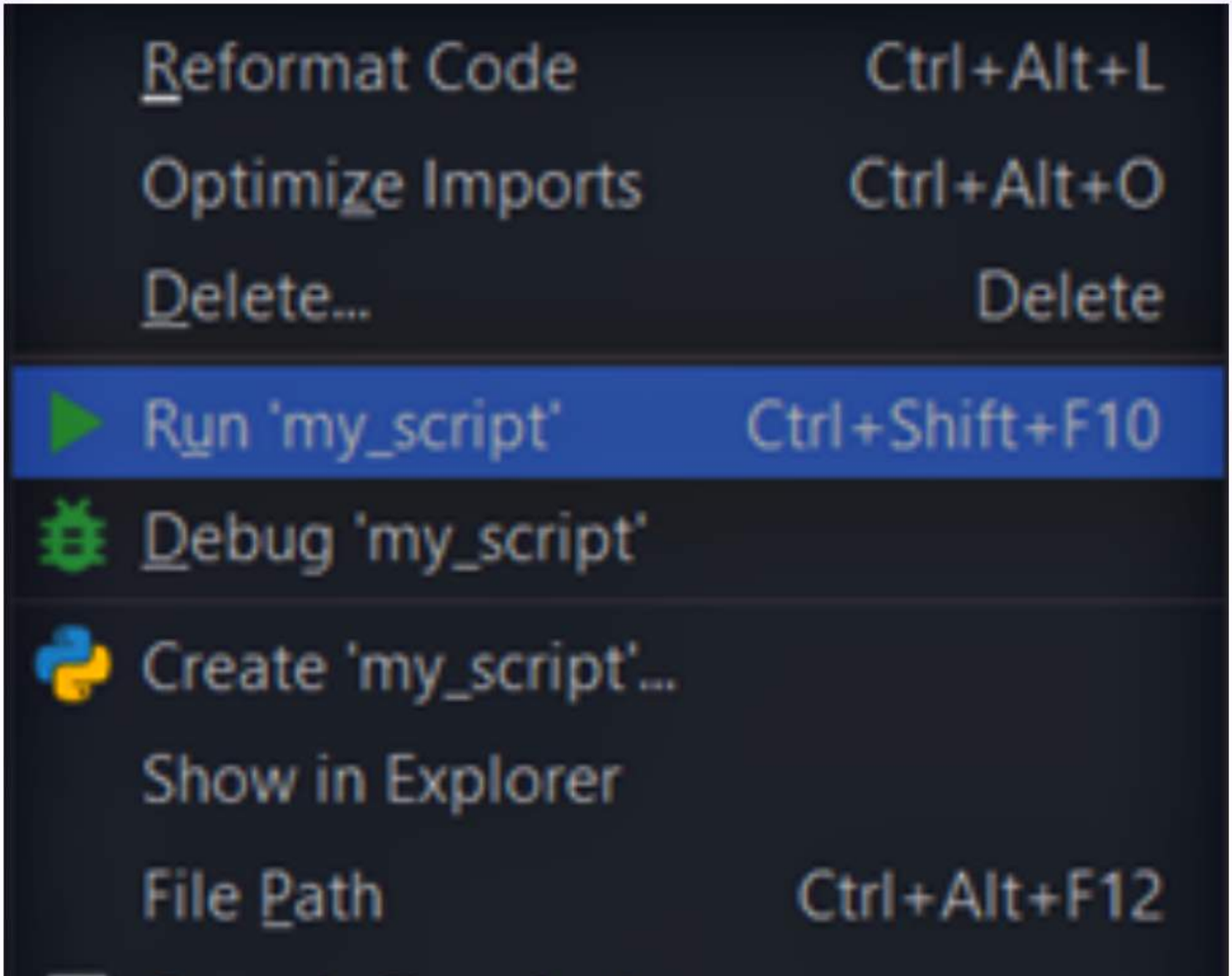


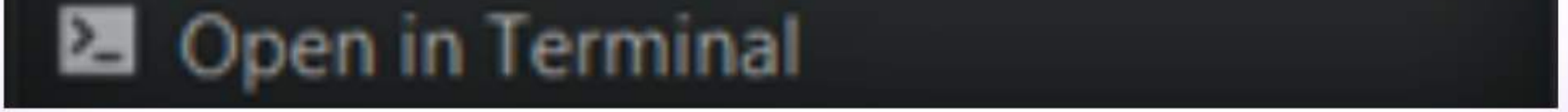
Поздравляю, теперь мы можем написать и запустить скрипт прямо из IDE. Для этого дважды нажимаем левой кнопкой мыши на имя файла, затем введём туда какой-нибудь простой код из прошлых уроков, например:



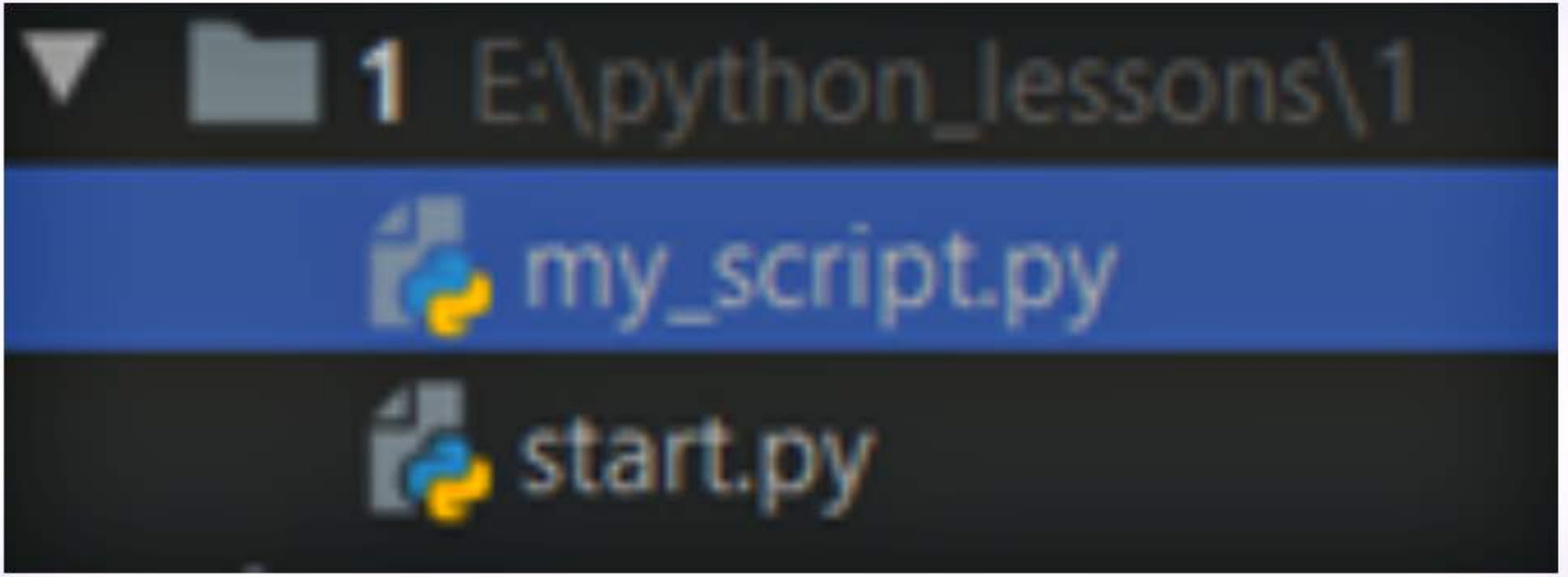


Чтобы запустить скрипт, нам достаточно нажать на файл нашего скрипта правой кнопкой, появится меню и в нём выберем пункт `Run`:

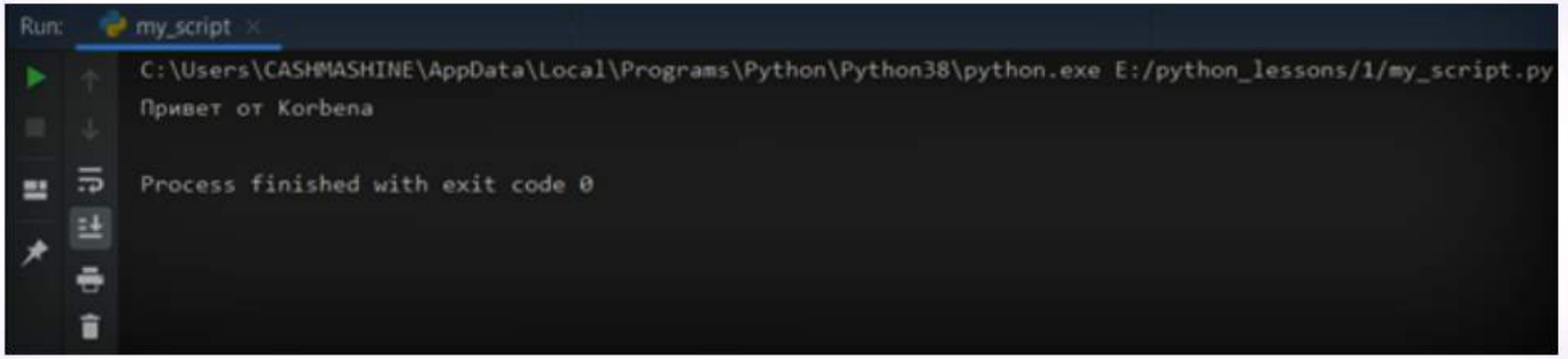




Либо сочетание клавиш `Shift + Ctrl + F10`, но для того, чтобы запустить нужный скрипт необходимо чтобы он был выделен в панели проекта:



После запуска скрипта, его результат отобразится в консоли:



Как видите, в консоль вывелось наше приветствие.

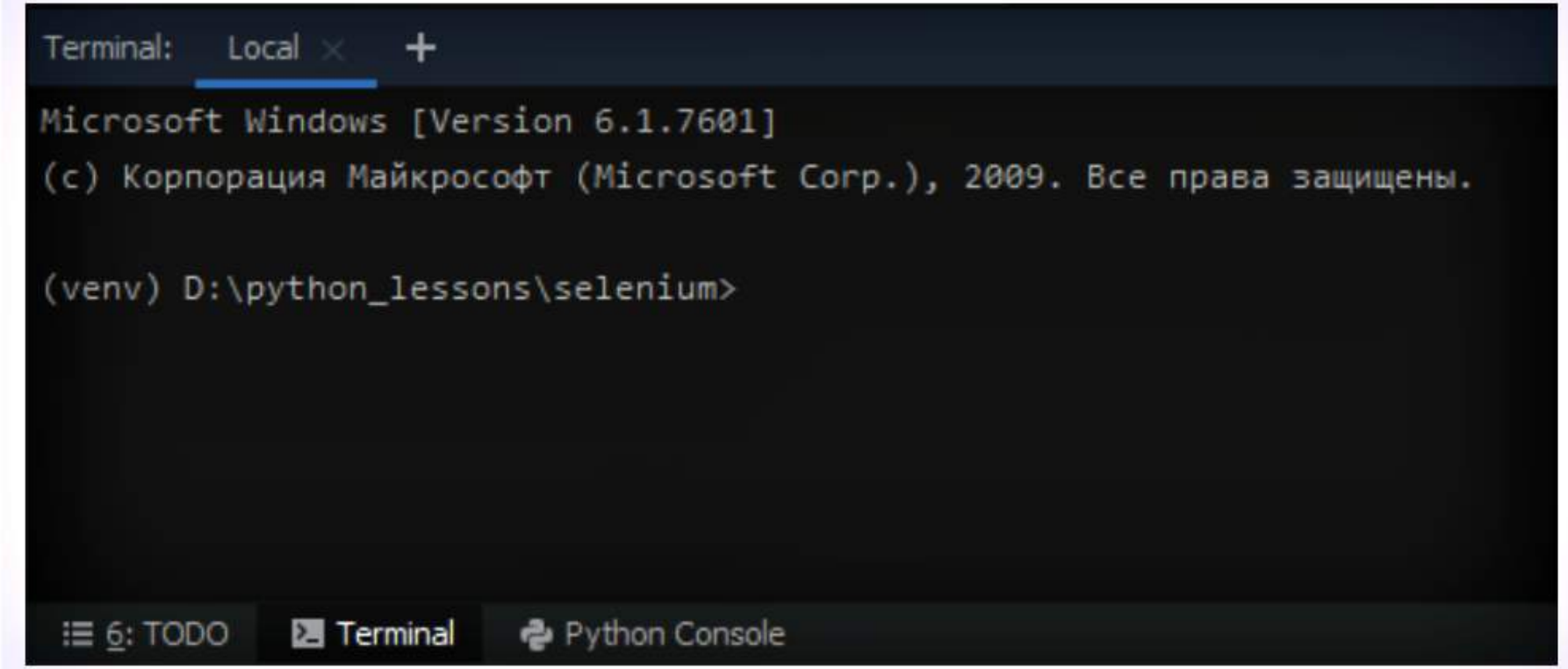
Встроенный терминал

Раньше, команды для интерпретатора Python мы запускали в командной строке, теперь, после установки Pycharm запускать команды для интерпретатора Python можно и нужно во встроенном терминале, а не в обычной консоли, это поможет избежать лишних ошибок и путаницы, к тому же это просто удобнее, когда терминал находится в том же окне.

Отмечу, если вы создали проект и работаете в нём в PyCharm, то вы **всегда должны** запускать консольные команды из встроенного терминала, иначе они будут работать некорректно.

Для того, чтобы открыть встроенный терминал необходимо обратить внимание на кнопку с названием `Terminal` слева внизу окна `PyCharm`:

Нажимаете на него и открывается окно терминала:



Спустя пять уроков, мы разобрались с тем, что из себя представляет язык программирования Python узнали, как пишется простейший код python-скриптов и как эти скрипты запускать.

Впоследствии мы изучили полезный инструмент для разработки скриптов на Python IDE PyCharm, научились более быстро удобно создавать и запускать скрипты.

Содержание python-скрипта

Содержание урока

Форматирование и отступы

Python по части синтаксиса и принципа составления инструкций является нетипичным языком. Он не относится к языкам с Си-подобным синтаксисом, и в отличии от большинства языков, при написании кода огромное значение имеют отступы и переводы строк.

Python-скрипт состоит из набора инструкций. Точнее, из набора команд, которые впоследствии исполняются интерпретатором. Каждая инструкция начинается с новой строки.

Для примера, рассмотрим несколько инструкций:

```
# Инструкция сложения
sum = 3 + 3

# Инструкция умножения
mul = 2 * 2
```

Как видите, здесь каждая инструкция начинается с начала строки, без отступов. А теперь поговорим об отступах. Отступы в python выступают в качестве разграничителей блоков, то есть, указывают, к какому блоку кода относятся те или иные инструкции.

Давайте обратимся для примера к предыдущему листингу:

```
# Инструкция сложения
sum = 3 + 3
# Инструкция умножения
mul = 2 * 2
```

В данном случае, наши инструкции, определены, скажем, на нулевом уровне. Они не входят ни в какой блок.

Но существует также и инструкции, которые содержат в себе другие инструкции. И для того, чтобы интерпретатор понял, что эти инструкции вложенные и находятся внутри инструкции более высокого уровня, используют отступы. Например, условная конструкция if. Она проверяет определённое условие, и если оно верно выполняет какой-либо вложенный код. И чтобы код был «вложен» в эту конструкцию, он отмечается отступом от начала самой конструкции:

```
# Инструкция сложения
sum = 3 + 3
# Инструкция умножения
mul = 2 * 2
if mul < sum:
    print("mul меньше чем sum")
```

Здесь можно заметить, что условно, на нулевом уровне находятся инструкции сложения, умножения и условной конструкции. А на первом уровне находится вложенный внутрь конструкции if код, который в свою очередь «привязан» к условной конструкции отступом.

Если бы мы не поставили пробел после объявления условной конструкции, то исполнение данного кода вызвало бы ошибку:

```
# Инструкция сложения
sum = 3 + 3
# Инструкция умножения
mul = 2 * 2
if mul < sum:
print("mul меньше чем sum")
```

Так как конструкция была объявлена, но никакой код в неё «вложен» не был.

Подытожим. Если нам необходимо вложить некоторый код, внутрь соответствующей конструкции, мы должны определить это через отступ, относительно самой конструкции.

По стандартам языка, отступы должны быть кратны четырём (то есть, 4-8-12 и т.д.), хотя это условие и не обязательно.

В среде разработки PyCharm при нажатии на клавишу «Tab» в месте установки курсора вставляется 4 отступа, что очень удобно.

Регистр

В языке Python присутствует зависимость от регистра, то есть, два одинаковых слова, но определённых в разном регистре для Python будут разными словами. Мы с вами уже знакомы с функцией print, которая выводит в консоль определённое строковое значение. Если вы вызовите инструкцию Print с большой буквы, то получите ошибку. Т.к. для python print и Print это два разных слова:

```
print("mul меньше чем sum") # mul меньше чем sum
Print("mul меньше чем sum") # NameError: name 'Print' is not defined
```

Комментирование кода

Как уже было сказано, весь текст, написанный в файле с расширением *.py будет учитываться интерпретатором. Однако, нередко возникает необходимость написать какую-то заметку в той или иной части кода. Для этого есть механизм комментирования.

Данный механизм позволяет скрыть от интерпретатора любую строку. Таким образом, как будто её не существует в исходном файле.

Для того, чтобы закомментировать определённый участок кода, необходимо перед ним поставить символ #.

```
# Этот текст интерпретатор проигнорирует
def name():
    print()  # И этот тоже
    print()
    print()
```

При этом, всё что находится перед символом `#` интерпретируется без изменение. Таким образом, инструкция `print`, после которой размещён комментарий будет работать без изменений.

Для того, чтобы закомментировать одну или несколько строк в среде разработки PyCharm нужно выделить одну/несколько строк и зажать комбинацию клавиш `Ctrl + /`.

Переменные, данные и типы данных

Содержание урока

Любая программа, на любом языке программирования занимается в общем смысле модификацией, изменением данных. То есть, в программу вводятся некоторые данные, они изменяются так или иначе, а затем, в рамках работы программы возвращаются новые данные.

Для того, чтобы манипулировать данными, сохранять их, читать или модифицировать в большинстве языков программирования присутствует такая сущность, как переменные.

Переменные Python

Переменные - это такие конструкции, которые позволяют ограниченно управлять памятью ПК. При помощи переменных можно сохранять определённые данные в память ПК, а затем, так или иначе ими манипулировать.

Таким образом, некий абстрактный участок с данными, получает уникальное читаемое имя. И когда вам нужно будет получить или изменить эти данные, вы обращаетесь к ним по установленному вами имени.

Переменные в python объявляются по следующему принципу: `<имя_переменной> = <значение переменной>`

Пример:

```
len = 45
name = "Mark"
weight = 88
height = 180.2
```

Создание имени переменной в python подчиняется определённым правилам. Имя переменной должно начинаться с алфавитного символа или с символа нижнего подчёркивания. Но не может начинаться с цифры.

В имени переменной не должно быть отступов. Если имя переменной состоит из нескольких слов, то их нужно писать слитно. Чтобы не нарушалась читаемость переменных как правило каждое новое слово пишут либо с большой буквы, либо через символ нижнего подчёркивания:

```
# Snake case
my_name = "Mark"

# CamelCase
myName = "Mark"
```

Как видите, переменные могут хранить разные значения. Одна – число, другая – строку, третья – число с дробным значением (с плавающей точкой). На самом деле, переменные в python хранят множество различных типов данных.

Типы данных

Тип данных – это некая категория, отражающая специфические характеристики совокупности значений (например, 5, 10, 187 – это категория (тип) int – то есть, целое число). А также набор операций, которые мы можем производить над этими значениями (int мы можем сложить, умножить, разделить, вычесть).

Пример. Все автомобили с выхлопной трубой имеют двигатель внутреннего сгорания. Их тип – «машина с ДВС». Метод, который поддерживает тип «машина с ДВС» - залить топливо.

Пример. Все целые числа в python могут быть отрицательными и положительными, но не могут иметь остаток. Их тип – `int`. Метод который поддерживает тип `int` - умножить.

Типов данных в python не так уж и мало. В рамках данного урока вскользь опишу основные типы данных, а затем, в дальнейших уроках разберём их более детально.

1. Строки – строковые символьные значения. Пример: "Mark"
   str - строка
2. Неопределённое значение переменной.
   None – отсутствие значения
3. Логическое значение переменной. True или False.
   Boolean
4. Числовое значение переменной.
   int – целое число
   float – число с плавающей точкой
   complex – комплексное число
5. Списочные значения. Когда переменная хранит некий список данных.
   list – список
   tuple – кортеж
   range – диапазон
6. Словари.
   dict – словарь
7. Множества.
   set – множество
   frozenset – неизменяемое множество
8. Бинарные списки.
   bytes – байты
   bytearray – массивы байт

Все страшные слова и формы выбросьте из головы. Эта информация приведена в качестве справочной, со временем вы поймёте её назначение и по новой к ней вернётесь.

Главное, что нужно сейчас понять это то, что переменные в python могут хранить разные по свойствам и структуре данные. В соответствии с типом данных хранящихся в переменной, над этой переменной можно производить действия характерные для типа данных, находящегося в этой переменной.

Например, переменные хранящие в себе тип данных int можно умножить их друг на друга и получить новое число, а вот если в переменных будут, например, данные типа str, то есть строки, то умножить их друг на друг на друга не получится.

Определение типа

Если те или иные данные приходят к вам динамически, и вы не можете проконтролировать их тип, вы можете узнать к какому типу относятся эти данные при помощи функции `type()`.

Для этого внутрь функции нужно передать значение или переменную содержащую некое значение.

```
type("Mark")  # <class 'str'>
type(myName)  # <class 'str'>
```

Операции над числами. Модификация числовых значений

Содержание урока

Как говорилось ранее, каждый тип данных имеет определённый набор операций, который специфическим образом модифицирует данные этого типа. В частности, для числовых типов, данных есть арифметические операции.

Они выполняют абсолютно те же функции, что и арифметические операторы в математике, хотя присутствует и несколько удобных специфических операторов.

Простые арифметические операторы

Сложение. Оператор `+`, складывает два и более числа между собой. Пример:

```
print(2 + 2)      # 4
print(2 + 2 + 2)  # 6
print(2.5 + 2.5)  # 5.0
print(3.5 + 3.5)  # 7.0
```

Вычитание. Оператор `-`, вычитает из левой части выражения правую, и все последующие. Пример:

```
print(2 - 2)      # 0
print(5 - 2 - 2)  # 1
print(3.5 - 2.5)  # 1.00
print(7.5 - 3.5)  # 4.00
```

Умножение. Оператор `*`, перемножает операнды выражения между собой. Пример:

```
print(2 * 2)      # 4
print(5 * 2 * 2)  # 20
print(3.5 * 2.5)  # 8.75
print(7.5 * 3.5)  # 26.25
```

Деление. Оператор `/`, делит левый левую часть выражения на правую. Пример:

```
print(2 / 2)      # 1.0
print(5 / 2 / 2)  # 1.25
print(3.5 / 2.5)  # 1.4
print(7.5 / 3.5)  # 2.142857142857143
```

Нужно отметить, что если мы делим друг на друга, два целых числа, результат деления которых является целым (например, 2 / 2 = 1), то мы всё равно получаем дробное значение (с плавающей точкой).

Целочисленное деление. Оператор `//`, делит левую часть выражения на правую отбрасывая остаток. Пример:

```
print(5 / 2)         # 2.5
print(5 // 2)        # 2
print(5.00 // 2.00)  # 2.00
```

Обратите внимание, что деление двух целых чисел возвращает целочисленное значение. Деление двух чисел с плавающей точкой – возвращают значение с плавающей точкой.

Остаток от деления. Оператор %, возвращает остаток от деления двух чисел. Пример:

```
print(5 % 2)       # 1
print(5 % 4)       # 1
print(2 % 2)       # 0
print(2.00 % 2.00) # 0
```

5 на 2 нацело делится 2 раза (2 + 2 = 4), единица и есть остаток. Его данная функция и выводит.

Возведение в степень. Оператор **, возводит левую часть выражения в степень правой части выражения. Пример:

```
print(2 ** 2)        # 4
print(2.00 ** 2.00)  # 4.0
```

Если в выражении присутствует несколько разных операторов, например, `5 + 6 * 4 - 4`, то вычисления производятся в порядке приоритетности операторов (самый высокий по приоритетности оператор выполняется первым) и слева направо.

Таблица приоритетности операторов: https://tirinox.ru/python-op-precedence/

По данным таблицы, приведённый пример выполнится в следующем порядке: `(5 + (6 * 4)) - 4 = 25`

```
print(5 + 6 * 4 - 4)  # 25
```

То есть, сперва произойдёт умножение, и выражение превратится в: 5 + 24 – 4, а далее, т.к. приоритет у операторов + и – одинаковые, вычисления пойдёт слева направо.

На мой взгляд, если вы производите расчёты, то самым разумным подходом будет управление ходом вычислений явно, через оператор «()» тогда вы точно будете знать, какая часть выражения в каком месте сработает. Пример:

```
print((5 + 4) + (6 * 4) - 4)  # 29
```

Арифметические операции с присвоением

Идентичны обычным арифметическим операциям, с оной особенностью, они присваивают левому операнду значение самого операнда и побочного значения. Например, у вас есть переменная со значением 5, и вы хотите прибавить к ней какое-то дополнительное значение, с обычными арифметическими операторами вы сожете сделать так:

```
num = 5        # 5
num = num + 5  # 10
```

То есть, вы вашу переменную перезаписываете значением выражения между вашей переменной и новым значением.

Арифметические операции с присвоением позволяют сделать вот так:

```
num = 5   # 5
num += 5  # 10
```

Как видите, это выражение короче и удобнее. Такой же синтаксис работает и со всеми остальными выражениями:

```
num = 5    # 5
num += 5   # 10

num = 5    # 5
num -= 5   # 0

num = 5    # 5
num *= 5   # 25

num = 5    # 5
num /= 5   # 1.0
```

```
num = 5    # 5
num //= 5  # 1

num = 5    # 5
num %= 5   # 0
print(num)

num = 5    # 5
num **= 5  # 3125
```

Округление чисел

Иногда, числа с плавающей точкой имеют слишком много знаков после запятой. И если вам нужно округлить значение до целого числа, или до определённого числа после запятой, то необходимо использовать функцию `round(<число>, <знаков после запятой>)`:

```
f = 5.154544
round(f)     # 5
round(f, 2)  # 5.15
```

Приведение чисел

Порой, числа приходят не в том формате, в котором мы бы хотели их видеть. Например, когда на сервер приходит строковое значение, представляющее число, например: `5`

Хоть мы и понимаем, что это число, а интерпретатор python будет считать, что это строка. И как только мы попробуем сложить эту строку с числом:

```
print(5 + "5")
```

Интерпретатор python выдаст ошибку: TypeError: unsupported operand type(s) for +: 'int' and 'str'

Но мы можем указать, что данная строка является числом, при помощи функций: `int()` и `float()`.

Функция `int` приводит строку к типу `int`, или значение типа `float` к типу `int`. Функция `float` приводит строку к типу `float` или значение типа `int` к типу `float`:

```
n = 5
f = 5.00
s = "5"

print(type(int(s)))    #
print(type(int(f)))    #

print(type(float(s)))  #
print(type(float(n)))  #
```

Как видите, функция int привела значения типа `string` и `float` к типу `int`. Функция `float` привела значения типа `string` и `int` к типу `float`.

Функции

Содержание урока

У вас возникла задача, которую нужно решить, и вы пишете отличный, качественный код, который заданную задачу решает. Здесь и сейчас. Но что, если эту задачу нужно будет решить в другом месте? Легко, наш код уже написан. Копируем его в другое место, где он решает данную задачу. А затем ещё в одно, и в следующие 20 мест тоже. Удобно? Вполне.

И вот у нас уже 23 места, где мы вставили наш код, который решает задачу. Но вдруг, наши условия изменились, задача внезапно усложнилась и нам нужно переписывать заданный код. Мы его переписали, а теперь вынуждены заменять его в 23 местах. Удобно? Ничуть.

Понятие функции

Чтобы избежать данную проблему, в языке Python существует такой инструмент, как функция. Функция — это специальный, именованный блок кода, который позволяет вызывать заключённый в него код из любого места программы, при том, что код объявлен единожды и в одном известном месте.

Таким образом, объявив функцию с кодом, который решает наши задачи единожды, её можно вызывать и обращаться к ней в сотнях и тысячах мест, а редактировать эту функцию, тем не менее, лишь в одном месте, там, где функция была объявлена.

Объявленная функция выглядит следующим образом:

```
def hello():
    print("Привет")
```

А если подробнее, то:

```
def имя_функции ([параметры]):
    инструкции
```

Объявление функции состоит из:

`def` – ключевое слово, оказывающее интерпретатору на то, что объявляется функция.
`имя_функции` – устанавливает уникальную последовательность символов, по которым можно обратиться и вызвать именно эту функцию.
`[параметры]` – необязательная последовательность переменных, которые можно передать в функцию. Смысл параметров, заключается в том, что благодаря ним, мы можем передавать внутрь функции некоторые данные и команды, необходимые для работы функции.
`инструкции` – это непосредственно строки вашего кода, выполняющие определённые задачи.
`:` - символ двоеточия, обязательный символ предваряющий начало кода, который вы определяете для функции.

Отдельно отмечу, что после того как вы поставите символ двоеточия, вам необходимо перейти на новую строку, а затем сделать 2-4 пробела. В IDE PyCharm после перехода на новую строку, будет достаточно нажатия клавиши «tab», при нажатии на эту клавишу IDE автоматически вставит 4 пробела.

Зачем нужны пробелы? Когда вы вставляете код внутрь какого-либо другого блока кода, вам необходимо явно указать, что этот код относится к конкретному блоку и что запустить этот код можно только в том случае, если будет запущен сам блок кода.

Давайте создадим файл, в директории нашего проекта, созданного в PyCharm, откроем его и пропишем следующий код:

```
def learn():
    print("Привет! Я учу Python!")
```

Для того, чтобы запустить этот код, необходимо обратиться к нему по имени:

```
def learn():
    print("Привет! Я учу Python!")

learn()
```

Если мы запустим этот код в консоли, то мы увидим следующее сообщение:

> Привет! Я учу Python!

Мы можем запустить эту функцию сколько угодно раз:

```
def learn():
    print("Привет! Я учу Python!")

learn()
learn()
learn()
```

```
learn()
learn()
```

Результат:

> Привет! Я учу Python!
> Привет! Я учу Python!
> Привет! Я учу Python!
> Привет! Я учу Python!
> Привет! Я учу Python!

Функция с параметрами

В функцию также можно передавать и данные, при помощи параметров:

```
def learn(how_much):
    print("Привет! Я учу Python уже " + str(how_much) + " дней!")


learn(1)
learn(2)
learn(3)
learn(4)
learn(5)
```

Результат:

> Привет! Я учу Python уже 1 дней!
> Привет! Я учу Python уже 2 дней!
> Привет! Я учу Python уже 3 дней!
> Привет! Я учу Python уже 4 дней!
> Привет! Я учу Python уже 5 дней!

Как это работает. При объявлении функции: `def learn(имя_переменной):` мы указываем, что внутри функции будет объявлена переменная с заданным именем, например: `how_much` как в нашем случае.

Таким образом, мы можем обращаться к этой переменной внутри функции, как при работе с любой другой переменной:
print("Привет! Я учу Python уже " + str(`how_much`) + " дней!")

Чтобы "поместить" некоторое значение в эту функцию, достаточно при вызове этой функции указать в скобочках это значение: learn(`2`)

Следует обратить внимание на то, что внутри функции можно вызывать и другие функции, например, в данном случае мы вызвали функцию `print` и функцию `str`.

Также можно передать любое количество переменных в функцию, для этого при объявлении функции просто достаточно указать это в скобках: `def learn(имя_переменной1, имя_переменной2):`

Соответственно внутри функции появится две переменных: `имя_переменной1`, `имя_переменной12`

Необязательные значения

Порой случается так, что значения указывать не всегда необходимо, или есть необходимость установить какое-то общее, наиболее часто встречающееся значение, которое при желании можно изменить. Для этого существуют значения по умолчанию. И выглядят они следующим образом:

```python
def learn(how_much, name="Пользователь"):
    print("Привет, " + name +"! Ты учишь Python уже " + str(how_much) + " дней!")

learn(5)
learn(19, "Марк")
```

Как видите, в момент объявления параметра, мы сразу присваиваем ему значение. Впоследствии, мы можем указать своё значение или не указывать ничего, в таком случае будет подставлено значение по умолчанию.

Параметры с именем

Если у функции имеется два параметра, то мы передаём их в той последовательности, в которой они объявлены в функции. Но при помощи именованных параметров, можно изменить порядок аргументов, которые мы передаём в функцию. На примере прошлой написанной нами функции `learn`, вспомним, что сперва мы указываем количество дней, которое прошлой с начала обучения, а затем имя ученика. Но если при вызове функции мы явно укажем имя аргумента, которому передаём параметр, то можно использовать любую последовательность:

```python
def learn(how_much, name="Пользователь"):
    print("Привет, " + name +"! Ты учишь Python уже " + str(how_much) + " дней!")

learn(name="Марк", how_much=19)
```

Результат:

> Привет, Марк! Ты учишь Python уже 19 дней!

Как видите, несмотря на то, что первый параметр при определении функции — это параметр `how_much`, мы смогли переместить его на второе место просто явно задав имя параметров.

Произвольное количество параметров

При необходимости, мы можем указать разное, не фиксированное количество параметров в нашу функцию, для этого достаточно лишь указать перед параметром символ звёздочка `*`.

Тогда внутрь функции мы сможем передать различное количество параметров, как в приведённом выше примере. Указывая ряд значений, они записываются в виде массива данных, условно, как некоторая последовательность отдельных значений, записанных в одну переменную. И при помощи цикла `for` (о нём мы поговорим в следующих уроках) мы можем получить доступ ко всем значениям переданным в функцию.

Пример функции с произвольным количеством параметров:

```python
def summ(*nums):
    res = 0
    for num in nums:
        res += num
    print(res)

summ(5,4,7,9)
summ(2,7)
```

Результат:

> 25
> 9

Данная функция просто суммирует все переданные в неё числа и затем, выводит значение суммы на экран. Как видите ошибки нет. Функция успешно принимает различное количество значений и успешно с ними работает.

Инструкция return. Возвращение результата работы функции

В предыдущих примерах функции которые мы создавали не возвращали никакого значения, они просто выводили результат своей работы на экран. Это полезно, когда вам нужно вычислить какое-то значение, и затем его куда-то отправить.

Перепишем функцию таким образом, чтобы она возвращала значение, а не выводила его на экран:

```
def summ(*nums):
    res = 0
    for num in nums:
        res += num
    return res

sum_result = summ(1, 2, 3, 4, 99)
```

Как видите, здесь мы вместо функции `print` написали инструкцию return, которая возвращает значение переменной `res`.

Если мы запустим этот код, то в консоль ничего выведено не будет, т.к. функция просто вернула значение и теперь оно находится в переменной `sum_result`. Над которой мы, в свою очередь, можем выполнять определённые действия, например, вывести это значение на экран:

```
def summ(*nums):
    res = 0
    for num in nums:
        res += num
    return res

summ = summ(1, 2, 3, 4, 99)
print(summ)
```

В консоль будет выведена сумма введенных чисел: `109`

Всем советую придерживаться именно такого подхода, когда функция порождающая или вычисляющая данные их обязательно возвращает, а не распечатывает на экран или отправляет куда-то по сети, т.к. всегда есть вероятность, что нам понадобится эти данные так или иначе модифицировать, а для этого их обязательно нужно получить.

Область видимости переменных

Содержание урока

Область видимости переменной — это конкретный блок кода, в котором мы можем получить доступ к некоторой переменной. Это та «площадь» кода, в которой мы можем «увидеть» некоторую переменную и произвести над ней те или иные действия, например, получить её значение, удалить или изменить.

Существует три контекст области видимости:

Глобальная
Локальная
Нелокальная

Глобальная область видимости

Глобальная область видимости — это область видимости переменной, при которой переменная объявлена как бы в самом верху иерархии и «видна» всем бокам кода, например:

```
car_name = "Tesla Model X"

def start():
    print(car_name + " начала движение!")

def stop():
    print(car_name + " прекратила движение!")

start()
stop()
```

Выведет:

> Tesla Model X начала движение!
> Tesla Model X прекратила движение!

Несложно догадаться, что переменная объявленная вне контекста конкретной функции, объявленная как бы «выше» этих функций является глобальной, то есть доступна для всех остальных функций.

Глобальную переменную также можно объявить и внутри функции, при помощи ключевого слова `global`:

```
def start():
    global car_name
    car_name = "Tesla Model X"
    print(car_name + " начала движение!")

def stop():
    print(car_name + " прекратила движение!")

start()
stop()
```

Выведет:

> Tesla Model X начала движение!
> Tesla Model X прекратила движение!

Переменная была объявлена внутри другой функции, но в связи с тем, что мы сделали её глобальной, она стала доступна во всех других функциях.

Локальная область видимости – это область видимости ограниченная некоторым блоком кода, к этой переменной можно получить только в том блоке кода, в котором она была объявлена. Обращение к локальной переменной за пределами её области видимости вызовет ошибку, т.к. данной переменной там не существует:

```
def start():
    car_name = "Tesla Model X"  # Это локальная переменная
    print(car_name + " начала движение!")

def stop():
    print(car_name + " прекратила движение!")

start()  # Tesla Model X начала движение!
stop()  # Ошибка: NameError: name 'car_name' is not defined
```

В данном случае, переменная `car_name` была определена внутри функции `start` и существовала только в этой функции. Область видимости переменной `car_name` является функция `start`, поэтому попытка получить значение

этой переменной в функции `stop` привело к ошибке, т.к. в этой функции переменная с именем `car_name` не была объявлена.

Конфликт локальной и глобальной области видимости

Рассмотрим пример:

```
car_name = "BMW"

def start():
    car_name = "Tesla Model X"
    print(car_name + " начала движение!")

def stop():
    print(car_name + " прекратила движение!")

start()
stop()
```

Подумайте, что выведет функция `start`, а что, функция `stop`. С одной стороны, кажется очевидным, что мы определили глобальную переменную и задали ей значение `BMW`, затем в функции `start` мы изменили значение глобальной переменной `car_name` на `Tesla Model X` и теперь при вызове функций start и stop мы получим вывод именно модели `Tesla Model X`:

> Tesla Model X начала движение!
> Tesla Model X прекратила движение!
> Давайте запустим данный код и посмотрим, что он выведет на самом деле:
> Tesla Model X начала движение!
> BMW прекратила движение!

Но почему так? Всё просто, в функции `start` мы присвоили значение `Tesla Model X` не для глобальной переменной, нет, мы создали новую локальную переменную с тем же именем и присвоили ей значение. Таким образом работает конфликт областей видимости в Python. Если в локальной области видимости, например, в функции определяется переменная с таким же именем что и у глобальной переменной, то в контексте этой области видимости значение глобальной переменной будет перезаписано локальным значением.

Но при этом вне локальной области видимости значение глобальной переменной останется неизменным, как бы оно не перезаписывалось в локальной области видимости.

Чтобы изменить значение глобальной переменной в локальной области видимости, нужно её также «глобализовать» при помощи ключевого слова `global`:

```
car_name = "BMW"

def start():
    global car_name
    car_name = "Tesla Model X"
    print(car_name + " начала движение!")

def stop():
    print(car_name + " прекратила движение!")

start()
stop()
```

Выведет:

> Tesla Model X начала движение!
> Tesla Model X прекратила движение!

Значение переменной `car_name` было изменено на глобальном уровне.

Нелокальная область видимости

В Python также есть возможность создавать вложенные функции. У этих вложенных функций также существует локальная область видимости и иногда возникает проблема при передаче значения переменной, объявленной в одной из вложенных функций, например:

```
def car():
    # Счётчик количества поездок
    trip = 0
    def start():
        trip += 1
        car_name = "Tesla Model X"
        print(car_name + " начала движение!")
        print("Номер поездки: " + str(trip))

    return start

car = car()
car()
car()
car()
car()
```

Выведет:

> UnboundLocalError: local variable 'trip' referenced before assignment

Дело в том, что переменная `trip` к которой мы обращаемся во вложенной функции `start` – не объявлена в этой функции, она объявлена выше, в функции `car` и поэтому мы прибавляем единицу к несуществующей в данном контексте переменной.

Ведь переменная `trip` не является глобальной. Чтобы решить эту проблему, нам нужно как-бы принять локальную переменную `trip` из функции `car` в функцию `start`, для этого используют ключевое слово `nonlocal`:

```
def car():
    # Счётчик количества поездок
    trip = 0
    def start():
        nonlocal trip
        trip += 1
        car_name = "Tesla Model X"
        print(car_name + " начала движение!")
        print("Номер поездки: " + str(trip))

    return start

car = car()
car()
car()
car()
car()
```

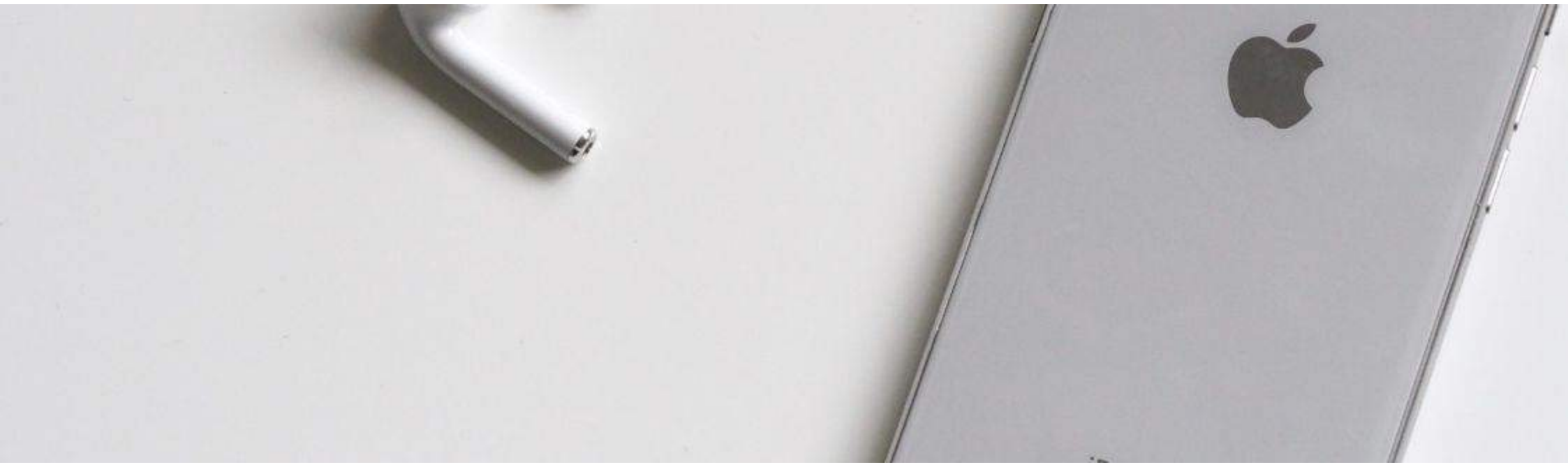
Выведет:

```
Tesla Model X начала движение!
Номер поездки: 1
Tesla Model X начала движение!
Номер поездки: 2
Tesla Model X начала движение!
Номер поездки: 3
Tesla Model X начала движение!
Номер поездки: 4
```

Внутри функции, область видимости которой не относится к области видимости заданной переменной мы объявили переменную trip нелокальной и теперь у нас появился локальный доступ, за пределами локальной области видимости данной переменной.

Это достаточно сложная концепция и в данном случае она вам, скорее всего не пригодится. Поэтому советую лишь поверхностно с ней ознакомиться и в последствии вернуться лишь тогда, когда вам в действительности понадобится работать с нелокальной областью видимости.

Для прохождения дальнейших уроков и практических заданий в курсе, эти знания вам не пригодятся.



Ветвление. Условные конструкции и выражения

Содержание урока

Иногда нашу программу нужно научить «думать», то есть, принимать решения в зависимости от внешних условий, по типу: «Если условие равно А, то выполняем инструкцию АВС». Это есть ни что иное, как логическая операция. В обычной жизни подобные операции производятся в два этапа. Сперва определяется, удовлетворяет ли действительность заданным условиям, то есть равны ли обстоятельства `True` или `False`. Если обстоятельства равны True, то мы выполняем одни действия. В противном случае - другие.

Операции сравнения

Операции сравнения реализуют процесс сравнения двух операндов между собой по определённому условию. В реальном мире это можно выразить следующим образом: «А, больше Б?» или так «5 = 2».

То есть мы буквально задаём вопрос и получаем на него ответ в виде «Да» или «Нет». В Python всё выглядит таким же образом. Вы просто сравниваем два значение и если это значение действительно, то оператор сравнения возвращает нам значение типа Boolean, то есть, `True` или `False`.

```
# Два равно двум? Если да, то возвращаем True
# иначе - False
answer = 2 == 2
print(answer)  # True
```

Предположим, что мы пишем модуль для регистрации пользователя на сайте с контентом для совершеннолетних. И в том случае, если пользователь при регистрации указывает возраст меньше 18 лет, мы должны прервать его регистрацию и оповестить о том, что содержимое сайта доступно только лицам чей возраст равен, либо больше 18 лет.

В таком случае, нам и пригодятся операции сравнения:

Оператор больше «>»

Возвращает `True`, если левый операнд больше правого, иначе – `False`.

Используется так:

```
<первый_операнд> > <второй_операнд>
```

В коде:

```
result = 5 > 2
```

Оператор меньше «<»

Возвращает `True`, если левый операнд меньше правого, иначе – `Falses`.

```
<первый_операнд> < <второй_операнд>
```

В коде:

```
result = 5 < 2
```

Оператор больше либо равно «>=»

Возвращает `True`, если левый операнд больше или равен правому, иначе `False`.

```
<первый_операнд> >= <второй_операнд>
```

В коде:s

```
result = 5 >= 2
```

Оператор меньше либо равно «<=»

Возвращает `True`, если левый операнд меньше или равен правому, иначе `False`.

```
<первый_операнд> <= <второй_операнд>
```

В коде:

```
result = 5 <= 2
```

Оператор равно «==»

Возвращает `True`, если левый операнд равен правому, иначе `False`.

```
<первый_операнд> == <второй_операнд>
```

В коде:

```
result = 5 == 2
```

Оператор не равно «!=»

Возвращает `True`, если левый операнд не равен правому, иначе `False`.

```
<первый_операнд> != <второй_операнд>
```

В коде:

```
result = 5 != 2
```

На примере проверки возраста пользователя:

```
# Возраст пользователя
age = 25
# Минимальный возраст
minimum_age = 18

res = age > minimum_age   # True возраст больше чем минимальный возраст
res = age < minimum_age   # False возраст не меньше чем минимальный возраст
res = age >= minimum_age  # True возраст больше или равен минимальному возрасту
res = age <= minimum_age  # False возраст не меньше и не равен минимальному возрасту
res = age == minimum_age  # False возраст не равен минимальному возрасту
res = age != minimum_age  # True возраст не равен минимальному возрасту
```

Логические операторы

Нередко возникает потребность в создании более сложных логических выражений, например, учитывать несколько логических условий по типу: «Если пользователю больше 18 лет и он женского пола, то…» или «Если пользователь младше 18 лет или старше 70, то…».

Реализуется это при помощи логических операторов. Логические операторы, это в сущности такие конструкции которые позволяют объединять отдельные операции в одну более сложную и составную.

Оператор «and» (Логическое и)

Данный оператор возвращает `True`, если все части выражения возвращают `True`.

```
age = 25
sex = "male"

res = age > 18 and sex == "male"  # True
```

То есть, если одновременно и выражение `age > 18` и выражение `sex == "male"` будут возвращать `True`, то всё выражение вернёт `True`.

Для наглядности, обернём каждое выражение в скобки:

```
res = (age > 18) and (sex == "male")  # True
```

В данном случае, в переменную `res` будет установлено значение `True` только если левое и правое выражение в скобках вернёт `True`.

Если один из операндов в выражении с `and` возвращает `False` – то второй уже не оценивается.

Оператор «or» (Логическое или)

Данный оператор возвращает `True`, если хотя бы одна часть выражения возвращает `True`.

```
age = 25
sex = "male"

res = age > 150 or sex == "male"  # True
```

То есть, если хотя бы одно выражение `age > 150` (это False) или выражение `sex == "male"` (это True) будет возвращать True, то всё выражение вернёт True.

Для наглядности, обернём каждое выражение в скобки:

```
res = (age > 150) or (sex == "male")  # True
```

В данном случае, в переменную `res` будет установлено значение True только если хотя бы одно выражение в скобках (левое или правое) вернёт `True`.

Оператор «not» (Логическое не)

Данный оператор возвращает `True`, если выражение равно `False`. Например, при регистрации пользователя, нам необходимо определить, есть ему 18 или нет, если ему нет 18, то регистрация на сайте для него запрещена.

```
age = 15

res = not age >= 18 # True
```

Выражение `age >= 18` возвращает `False`, т.к. пользователю меньше 18 лет, но из-за того, что перед выражением стоит `not`, всё выражение в целом – возвращает `True`. Проще говоря, оператор `not` – переворачивает значение, если вы ставите его перед выражением возвращающим `True` – он возвращает `False`, если выражение возвращает `False` – он возвращает `True`.

Условная конструкция if

Допустим, при помощи логических выражений мы научились определять, истинно то или иное условие, или ложно. Но как же в зависимости от значений этих условий влиять на алгоритм управления программой?

Для этого в языке Python существует условная конструкция `if`. В условную конструкцию `if` помещается логическое выражение, и если оно возвращает `True` – срабатывает блок `if`, иначе ничего не происходит.

Как выглядит условная конструкция `if`:

```
if логическое_выражение:
    # Инструкции, которые
    # будут выполняться
    # если условное выражение
    # вернёт True
```

Давайте посмотрим на примере:

```
age = 18

if age >= 18:
print("Вы можете получить права")
```

Если мы запустим этот код, то в консоль будет выведено:

> Вы можете получить права

Но если мы изменим значение переменной age на 15:

```
age = 15

if age >= 18:
    print("Вы можете получить права")
```

И запустим код, то он ничего не выведет, так как логическая конструкция «не сработала», потому что логическое выражение вернуло False, ведь переменная age не больше 18.

Блок else

А что, если в случае, когда условное выражение возвращает `False`, мы должны произвести некоторое действие. Для этого у `if`, есть необязательный блок `else`, он срабатывает только в том случае, когда условное выражение в `if` вернулся `False`:

```
age = 15

if age >= 18:
    print("Вы можете получить права")
else:
    print("Вам ещё нет 18 лет")
```

Выводит:

> Вам ещё нет 18 лет

А если мы поменяем значение переменной age на 22:

```
age = 22

if age >= 18:
    print("Вы можете получить права")
else:
    print("Вам ещё нет 18 лет")
```

Выведет:

> Вы можете получить права

То есть, если выражение `age >= 18` возвращает `True`, то будут исполнятся инструкции из блока `if`, если же выражение вернёт `False`, то выполняются инструкции из блока `else`.

Блок elif

Но что, если в рамках одного блока необходимо ввести сразу несколько условий? Например, мы пишем программу, которая в зависимости от возраста человека прогнозирует вероятность получения им прав и нам нужно выводить результаты в зависимости от диапазона.

Для этого нужен блок `elif`. Данный блок, подобен блоку `else`, он срабатывает только в том случае, если предшествующий ему блок `if` или `elif` не сработал. Разница в том, что блок `elif` тоже принимает условное выражение, и если оно возвращает `False`, то этот блок также не срабатывает.

Например:

```
animal = "Кот"

if animal == "Кот":
    print("В переменной animal сидит Кот")
elif animal == "Пёс":
    print("В переменной animal сидит Пёс")
elif animal == "Голубь":
    print("В переменной animal сидит Голубь")
```

```
else:
    print("В переменной animal сидит неизвестное существо")
```

Выводит:

> В переменной animal сидит Кот

Все остальные инструкции не сработали. Попробуем так:

```
animal = "Пёс"

if animal == "Кот":
    print("В переменной animal сидит Кот")
elif animal == "Пёс":
    print("В переменной animal сидит Пёс")
elif animal == "Голубь":
    print("В переменной animal сидит Голубь")
else:
    print("В переменной animal сидит неизвестное существо")
```

Выведет:

> В переменной animal сидит Пёс

Сработала только вторая инструкция. Или так:

```
animal = "Голубь"

if animal == "Кот":
    print("В переменной animal сидит Кот")
elif animal == "Пёс":
    print("В переменной animal сидит Пёс")
elif animal == "Голубь":
    print("В переменной animal сидит Голубь")
else:
    print("В переменной animal сидит неизвестное существо")
```

Выводит:

> В переменной animal сидит Голубь.

Сработала только третья инструкция.

И наконец, давайте посадим в переменную animal – верблюда:

```
animal = "Верблюд"

if animal == "Кот":
    print("В переменной animal сидит Кот")
elif animal == "Пёс":
```

```
    print("В переменной animal сидит Пёс")
elif animal == "Голубь":
    print("В переменной animal сидит Голубь")
else:
    print("В переменной animal сидит неизвестное существо")
```

Выведет:

> В переменной animal сидит неизвестное существо

Если ни один из блоков `if` или `elif` не сработал, выводится выводится блок `else`.

Операции над строками

Содержание урока

Строковые значения в Python, да и в большинстве других языков, являются одними из самых распространённых. Нередко для работы с пользователями, сбора, обработки и вывода информации мы используем строковые значения, поэтому важно знать основные операции, используемые при работе со строками.

Строковое значение в Python (далее «строка», «строки»), это набор отдельных символов Unicode, при объявлении строки в коде используется следующая форма:

```
str_1 = "Mark"
str_2 = 'Mark'
```

То есть, переменной присваивается строка, её обязательно оборачивать в кавычки. Либо в двойные, либо в одинарные. Особенной разницы между ними нет.

Объединение строк

Если вы хотите объединить (конкатенировать) значение двух строчных переменных, то вам необходимо использовать символ `+`:

```
name = "Mark "
surname = 'Johnson'

result = name + surname  # Mark Johnson
```

Также при конкатенации можно добавить произвольное строчное значение:

```
name = "Mark "
surname = 'Johnson'

result = name + surname + ": mjohnson@gmail.com"  # Mark Johnson: mjohnson@gmail.com
```

Экранированные последовательности

Представим ситуацию. Вам нужно написать текст, в котором присутствует цитата, например:

```
quote = "И он сказал: "Встань и иди""
```

Это вызовет ошибку интерпретатора, т.к. кавычки в данном случае являются синтаксическим инструментом, определяющим строку, но интерпретатор увидит строку только в этой области:

```
quote = "И он сказал: "
```

Ведь он ждёт открывающих и закрывающих кавычек. А последующие кавычки считает отдельной последовательностью, т.е. интерпретатор, условно увидит эту инструкцию вот так:

```
quote = "И он сказал: "
Встань и иди""
```

Первая строка без ошибок. Вторая – одна сплошная ошибка. Чтобы исправить эту ситуацию нужно экранировать кавычки в тексте при помощи символа `\`:

```
quote = "И он сказал: \"Встань и иди\""
```

Или так:

```
quote = 'И он сказал: \'Встань и иди\''
```

Тогда эти кавычки будут интерпретированы как текст.

Тогда эти кавычки будут интерпретированы как текст.

Если вы не хотите экранировать кавычки, можете использовать разные типы кавычек в одном выражении.

В двойных кавычках можно не экранировать одинарные:

```
quote = "И он сказал: 'Встань и иди'"
```

В одинарных кавычках можно не экранировать двойные:

```
quote = 'И он сказал: "Встань и иди"'
```

Помимо символа обратного слеша `\` есть ещё ряд подобных символов, основные из них: «\\» - экранирование самого символа, остаётся один «\».

```
print("Один слеш: \\")
```

Выведет:

> Один слеш: \

Хотя в тексте мы явно указали два обратных слеша. «\n» - перевод строки. Переводит часть строчного значения на новую строку.

```
print("- Назови моё имя..\n- Хайзенберг")
```

Выведет:

> - Назови моё имя..
> - Хайзенберг

Как видите, строку «разбило» надвое, переносом.

«\t» - вставляет символ табуляции.

```
print("Один таб\tВторой таб\tТретий таб")
```

Выводит:

> Один таб Второй таб Третий таб

Описание других Escape-последовательностей: https://docs.microsoft.com/ru-ru/cpp/c-language/escape-sequences?view=vs-2019

Сравниваем строки

Иногда, в наши приложения могут приходить строчные данные, которые необходимо проверить на соответствие определённому шаблону. Допустим, мы отправляем запрос на сайт, проверяя статус заказа. Он может быть: `failed` или `success`, если статус `success`, то выполняем определённые действия, иначе – ничего не делаем.

Для этого нам достаточно провести простое логическое сравнение двух строк:

```
status = get_status()
true_status = status == "success"  # boolean

if true_status:
    print("That's good")
```

Если строки равны друг другу, то возвращается `True`, иначе – `False`.

Вхождение в строку

Иногда, нам необходимо проверить, есть ли в конкретном фрагменте текста слово или словосочетание, это сделать также несложно, при помощи ключевого слова `in`:

```
string = "Мама мыла раму"
if "раму" in string:
    print("Success!")
```

При работе со сбором данных и написанием всяких чат-ботов – незаменимое и изящное решение языка. Но мы не всегда можем знать, в каком регистре находится искомое слово, например, оно начинается с большой буквы, а мы ищем с маленькой. Де-факто, слово в тексте присутствует, но найдено оно не будет.

Например, при таких условиях поиска в консоль ничего не выведется:

```
string = "Мама мыла Раму"
if "раму" in string:
    print("Success!")
```

Для того чтобы исправить эту проблему мы можем перевести рассматриваемую строку и слово, которое мы ищем в один общий регистр. Для этого есть специальные функции: `lower()` и `upper()`.

Функция `lower()` переводит строку в нижний регистр, а `upper()` в верхний:

```
string = "Мама мыла Раму"
print(string.lower())
print(string.upper())
```

В консоль выведет:

> мама мыла раму МАМА МЫЛА РАМУ

Используя данные функции мы значительно улучшили и обезопасили нашу программу:

```
string = "Мама мыла Раму"
find = "РаМу"

if find.lower() in string.lower():
    print("Success!")
```

В консоль выведет:

> Success!

Теперь слова и буквы могут быть в совершенно любом регистре, но вхождение подстроки в строку будет найдено (если искомое слово присутствует в строке в любом регистре).

Длина строки

Длина строки определяется при помощи функции `len`:

```
word = "Привет, юный питонист!"
print(len(word))
```

Выводит:

> 22

Обращение по индексу

Строка – это последовательность символов Unicode, поэтому её можно интерпретировать как некоторую структуру данных, по типу списка, в котором каждый элемент имеет свою позицию. К каждому элементу строки можно обращаться как к элементу списка:

```
word = "Привет, юный питонист!"
```

```
print(word[0])
print(word[1])
print(word[2])
print(word[3])
```

Выводит:

> П р и в

Подобно элементам списка, символы строки можно получать и с конца, при помощи отрицательного индекса:

```
word = "Привет, юный питонист!"

print(word[-1])
print(word[-2])
print(word[-3])
print(word[-4])
```

Выводит:

> ! т с и

Получить часть строки. Слайсы

Часто необходимо обрезать длинную строку, сделав её меньше. Например, получить только первые 10 символов. Замечательная новость в том, что мы можем обрезать строку на основе механизма слайсов:

```
word = "Привет, юный питонист!"
print(word[:10:])
```

Выведет:

> Привет, юн

Массивы. Списки

Содержание урока

Понятие списка

Что такое список и зачем он нужен? Начнём с определения, список — это нечто вроде обычной переменной, которая содержит внутри себя много значений. У каждого значения, внутри переменной-списка есть адрес, например:

```
my_list = ["Петя", "Оля", "Аня", "Игорь"]
print(my_list[2])
```

Если мы выведем в консоль значение списка по адресу 2, нам вернёт значение:

> Аня

Так как нумерация значений в списке происходит с нуля, то 0 элемент – это Петя, 1 элемент – Оля, 2 элемент – Аня.

Получается, список – это множество (набор) значений, сгруппированных в рамках одной именованной области памяти (переменной). Да, определение не совсем профессиональное, но оно максимально простое, и, как мне кажется, отражает основную суть списков.

Зачем нужны списки? Как правило, списки нужны для того, чтобы хранить и управлять коллекциями связанных и однотипных данных. Например, у нас в таблице excel хранятся записи о работниках организации. Нам нужно прочитать этот файл и извлечь каждую строку с данными о работниках в программу.

Именно для этого и нужны списки (а в целом и массивы), когда нужно разместить некоторое, заранее неизвестное количество однородных данных.

Создание списков

Для того, чтобы создать список есть 2 конструкции: list() и []:

```
first = list()
second = []
```

Логика создания такова:

```
имя_переменной = [механизм создания списка]
```

Выше представлен процесс определения списка. Но также, в момент определения списка его можно проинициализировать значениями:

```
first = list([1, 2, 3, 4, 5, 6])
second = [1, 2, 3, 4, 5, 6]
```

Обращение к элементам списка

Для обращения к элементам списка существует следующая конструкция: `имя_переменной[индекс_элемента]`

```
first = list([1, 2, 3, 4, 5, 6])
second = [1, 2, 3, 4, 5, 6]

print(first[0])
```

```
print(first[1])
print(first[2])
```

Выводит:

> 1 2 3

Индекс элемента, это номер значения, находящегося внутри списка, «первое значение, второе, третье и т.д.». Нужно учитывать, что нумерация элементов списка начинается с 0, то есть, первое значение в списке имеет номер 0.

Вы можете обращаться к индексам с конца, используя символ минус `-`:

```
first = list([1, 2, 3, 4, 5, 6])

print(first[-1])
print(first[-2])
```

Выводит:

> 6 5

При индексе `[-1]` выводит последний элемент из списка, при индексе `[-2]` – предпоследний, и так далее.

Операции над списками

Добавляем новый элемент в конец списка

Функция `append(элемент_списка)`:

```
first = list([1, 2, 3, 4, 5, 6])
first.append(7)

print(first)
```

Выводит:

> [1, 2, 3, 4, 5, 6, 7]

Добавился новый элемент.

Добавить новый элемент по индексу

Функция `insert(индекс, элемент_списка)`:

```
first = list([1, 2, 3, 4, 5, 6])
first.insert(0, 100)

print(first)
```

Выводит:

> [100, 1, 2, 3, 4, 5, 6]

Функция вставляет элемент на заданный индекс, при этом, сдвигает элемент который был на этом индексе до этого. Функция удалила все значения в списке.

Получение индекса по значению элемента

Функция `index(элемент)`:

```
first = list([1, 2, 3, 4, 5, 6, "Марк"])
print(first.index("Марк"))
```

Выводит:

> 6

Удалить элемент по значению

Функция `remove(item)`:

```
first = list([1, 2, 3, 4, 5, 6, "Марк"])
first.remove("Марк")

print(first)
```

Выводит:

> [1, 2, 3, 4, 5, 6]

Элемент «Марк» был удалён. Удаляет только первое вхождение элемента, то есть, если в списке два элемента со значением «Марк», то удалит только один из них.

Очистка всего списка.

Функция `clear()`:

```
first = list([1, 2, 3, 4, 5, 6])
first.clear()

print(first)
```

Выводит:

> []

Перевернуть элементы списка задом-наперёд

Функция `reverse()`:

```
first = list([1, 2, 3, 4, 5, 6, "Марк"])
first.reverse()
print(first)
```

Выводит:

> ['Марк', 6, 5, 4, 3, 2, 1]

Сортируем список

Функция `sort()`:

```
first = [5, 7, 9, 74, 1, 99, 2]
first.sort()
print(first)
```

Выводит:

> [1, 2, 5, 7, 9, 74, 99]

По умолчанию сортирует по увеличению, но также, в функцию `sort()`, можно передавать функцию для сортировки.

Удалить и вернуть значение удалённого элемента

Функция `pop(индекс):`

```
first = [5, 7, 9, 74, 1, 99, 2]
del_index = first.pop(3)
print(del_index)
print(first)
```

Выводит:

> 74 [5, 7, 9, 1, 99, 2]

Удалит четвёртый элемент (с индексом 3), выведет массив без удалённого индекса.

Проверка вхождения элемента в список

Ключевое слово in:

```
first = [5, 7, 9, 74, 1, 99, 2, "Mark"]
print("Mark" in first)
print("mark" in first)
print(5 in first)
```

Выводит:

> True False True

Возвращает логическое значение, часто используется в конструкции `if-else`.

Слияние списков

Если у вас есть два списка, условно `a_list` и `b_list`, то их можно объединить в один при помощи символа `+`:

```
first = [5, 7, 9, 74, 1, 99, 2, "Mark"]
second = [15, 17, 19, 174, 11, 9, 12, "Alex"]
new_list = first + second
print(new_list)
```

Выведет:

> [5, 7, 9, 74, 1, 99, 2, 'Mark', 15, 17, 19, 174, 11, 9, 12, 'Alex']

Обрезка списка. Слайсы

Порой мы не можем контролировать количество элементов, которое нам нужно хранить в списке, допустим из интернета пришёл массив на 200 элементов, а нам нужно только 100. При помощи слайсов мы можем обрезать наш список до нужной длинны.

Обрезка списка имеет следующий синтаксис:

```
имя_списка[<от какого элемента обрезать > : <до какого элемента обрезать > : <длина шага>] или: my_list[0:15:1]
```

```
first = [5, 7, 9, 74, 1, 99, 2, "Mark"]
print(first[0:3:1])
```

Выведет:

> [5, 7, 9]

Если вам нужно, например, получить N элементов от начала массива, то можно указать только среднее значение:

```
first = [5, 7, 9, 74, 1, 99, 2, "Mark"]
print(first[:3:])
```

Выведет то же самое, что и в прошлый раз:

> [5, 7, 9]

Количество элементов списка

Для того, чтобы определить сколько элементов хранится в нашем списке мы можем воспользоваться функцией `len(список)`:

```
first = [5, 7, 9, 74, 1, 99, 2, "Mark"]
count = len(first)
print(count)
```

Выведет:

> 8

Вложенные списки

Списки могут содержать в себе другие списки, например:

```
cpu = [
    ["AMD Threadripper 2990WX", "3.0"],
    ["Intel Core i7-7820X", "3.6"],
    ["Intel Core i3-9350KF", "4.0"],
    ["AMD Threadripper 2950X", "3.5"]
]
```

Список CPU, внутри которого вложены другие списки с данными по названию и частоте процессора.

Обратиться ко внутреннему элементу списка легко, например мы хотим получить частоту работы первого процессора из списка, для этого сперва обращается к самому элементу:

```
    ["AMD Threadripper 2990WX", "3.0"],
    ["Intel Core i7-7820X", "3.6"],
    ["Intel Core i3-9350KF", "4.0"],
    ["AMD Threadripper 2950X", "3.5"]
]

print(cpu[0])
```

Выведет:

```
['AMD Threadripper 2990WX', '3.0']
```

А теперь, ко внутреннему элементу заданного списка:

```
cpu = [
    ["AMD Threadripper 2990WX", "3.0"],
    ["Intel Core i7-7820X", "3.6"],
    ["Intel Core i3-9350KF", "4.0"],
```

```
    ["AMD Threadripper 2950X", "3.5"]
]

print(cpu[0])
print(cpu[0][0])
print(cpu[0][1])
```

Мы обратились к первому элементу списка CPU, внутри которого мы обратились к элементу номер 2: `cpu[0][1]` и номер один: `cpu[0][0]`

Выведет:

```
['AMD Threadripper 2990WX', '3.0']
AMD Threadripper 2990WX
3.0
```

Динамическое добавление вложенных элементов. Можно добавить вложенный массив при помощи всё той же функции `append()`:

```
cpu = [
    ["AMD Threadripper 2990WX", "3.0"],
    ["Intel Core i7-7820X", "3.6"],
    ["Intel Core i3-9350KF", "4.0"],
    ["AMD Threadripper 2950X", "3.5"]
]

cpu.append(["AMD Ryzen 9 3900X", "3.8"])
print(cpu)
```

Выведет:

```
[
    ['AMD Threadripper 2990WX', '3.0'],
    ['Intel Core i7-7820X', '3.6'],
    ['Intel Core i3-9350KF', '4.0'],
    ['AMD Threadripper 2950X', '3.5'],
    ['AMD Ryzen 9 3900X', '3.8']
]
```

Последний элемент как раз и является добавленным нами.

Массивы. Кортежи

Содержание урока

Понятие веб-сервера

Кортеж — это, в сущности, тот же самый список, за исключением одного обстоятельства - он не изменяемый. То есть, его можно определить единожды и после этого нельзя изменить. Нельзя добавить, удалить или отредактировать элемент кортежа.

Кортеж создаётся при помощи оператора «круглые скобки»:

```
user = ("Oleg", +79517896514)
ids = (12, 345, 556, 213, 44, 3, 576, 345)
```

При определении кортежа можно не использовать скобки вовсе, следующий синтаксис будет эквивалентен прошлому примеру:

```
user = "Oleg", +79517896514
ids = 12, 345, 556, 213, 44, 3, 576, 345
```

Работа с кортежем

Получение значение элемента кортежа происходит идентично значению словаря, по индексу:

```
ids = 12, 345, 556, 213, 44, 3, 576, 345
print(ids[1])
print(ids[2])
print(ids[3])
```

Выведет:

> 345 556 213

Вложенные кортежи. Кортежи, подобно спискам можно вкладывать друг в друга:

```
users = (
    ("Oleg", +79507894665),
    ("Nikita", +79507894665),
    ("Viktor", +79507894665),
    ("Anna", +79507894222),
)

print(users)
```

Выведет:

```
(
    ('Oleg', 79507894665),
    ('Nikita', 79507894665),
    ('Viktor', 79507894665),
    ('Anna', 79507894222)
)
```

Обращение к элементам сложных кортежей происходит также, как и у списков, для примера, получим номер Анны:

```
users = (
    ("Oleg", +79507894665),
    ("Nikita", +79507894665),
    ("Viktor", +79507894665),
    ("Anna", +79507894222),
)

print(users[3][1])
```

Выведет:

> 79507894222

Можно разбить кортеж на переменные, для этого нужно через запятую объявить переменные, в которых будут размещены значения элементов кортежа, а затем присвоить им сам кортеж:

```
user = ("Oleg", +79507894665)
name, phone = user

print(name, phone)
```

Слайсы для кортежей работают также, как и для списков:

```
first = [5, 7, 9, 74, 1, 99, 2, "Mark"]
second = first[0:4:1]
print(second)
```

Выведет:

> [5, 7, 9, 74]

Массивы. Словарь

Содержание урока

Понятие словаря

Словарь - структура, подобно списку хранящая коллекцию элементов, но если в списке или кортеже элементы выстроены один за другим, то в словаре у каждого значения есть его уникальный ключ, т.е., условно говоря, для каждого значения в словаре есть уникальное название, который пользователь задаёт сам. Давайте сравним словарь и список:

```
list = ["Oleg", +79514789865]
dict = {"name": "Oleg", "phone": +79514789865}

print(list[0], list[1])
print(dict["name"], dict["phone"])
```

Выведет:

> Oleg 79514789865 Oleg 79514789865

Вывод произошёл идентичный, однако можно заметить, что в словаре каждое значение имеет свой ключ, благодаря чему очень удобно хранить и получать данные из словаря.

Создание словаря

Словарь определяется по следующему принципу:

```
имя_словаря = {ключ: значение, ключ: значение, ключ: значение}
dict = {"name": "Oleg", "phone": +79514789865, "height": 182, "weight": 91}
```

Обращение к элементу словаря производится также, как и к элементу списка, через ключ, с той лишь разницей, что ключ – не всегда числовой:

```
dict = {"name": "Oleg", "phone": +79514789865, "height": 182, "weight": 91}
print(dict["name"], dict["phone"], dict["height"])
```

Выводит:

```
Oleg 79514789865 182
```

Работа со словарём

Изменение элемента словаря производится тем же образом, что и изменение элемента списка, через обращение к элементу по ключу:

```
dict = {"name": "Oleg", "phone": +79514789865, "height": 182, "weight": 91}
dict["name"] = "Mark"
print(dict["name"])
```

Выводит:

> Mark

Если мы пытаемся обратиться к несуществующему элементу словаря, изменить его или прочитать, то мы получим ошибку:

```
dict = {"name": "Oleg", "phone": +79514789865, "height": 182, "weight": 91}
print(dict["names"])
```

Выведет:

> KeyError: 'names'

Ведь у нас в словаре нет элемента с ключом names. Чтобы не сталкиваться с подобной ошибкой, мы можем обратиться к ключевому слову `in`, пройденному нами в уроке по спискам:

```
dict = {"name": "Oleg", "phone": +79514789865, "height": 182, "weight": 91}

if "names" in dict:
    print(dict["names"])
```

В таком случае, ошибки мы не получим.

Удаление элемента в словаре. Чтобы удалить элемент словаря достаточно воспользоваться ключевым словом `del`:

```
dict = {"name": "Oleg", "phone": +79514789865, "height": 182, "weight": 91}
del dict["name"]

print(dict)
```

Выведет:

```
{'phone': 79514789865, 'height': 182, 'weight': 91}
```

Указывая ключевое слово `del` и передавая ему элемент, по ключу который мы хотим удалить, мы получаем очищенный от заданного элемента список.

Второй способ удаления элемента, функция `pop()`. Работает идентично функции `pop()` у списков, а именно – удаляет элемент по ключу и возвращает удалённое значение:

```
dict = {"name": "Oleg", "phone": +79514789865, "height": 182, "weight": 91}
deleted = dict.pop("name")

print(deleted)
print(dict)
```

Выведет:

> Oleg {'phone': 79514789865, 'height': 182, 'weight': 91}

Полная очистка словаря. Метод `clear()`:

```
dict = {"name": "Oleg", "phone": +79514789865, "height": 182, "weight": 91}
dict.clear()
print(dict)
```

Выведет:

> {}

Объединение словарей. Если вам нужно добавить новый словарь к уже существующему, вы можете воспользоваться методом `update()`:

```
dict = {"name": "Oleg", "phone": +79514789865, "height": 182, "weight": 91}
bio = {"age": 23, "snils": "123-456-789-01"}
dict.update(bio)
print(dict)
```

Выведет:

```
{'name': 'Oleg', 'phone': 79514789865, 'height': 182, 'weight': 91, 'age': 23, 'snils': '123-456-789-01'}
```

Копирование словарей. Если вам нужно создать новый словарь из уже существующего, то в вашем распоряжении метод `copy()`:

```
dict = {"name": "Oleg", "phone": +79514789865, "height": 182, "weight": 91}
new_dict = dict.copy()
new_dict["name"] = "Mark"
print(dict)
print(new_dict)
```

Выведет:

> {'name': 'Oleg', 'phone': 79514789865, 'height': 182, 'weight': 91} {'name': 'Mark', 'phone': 79514789865, 'height': 182, 'weight': 91}

Вложенные словари. Подобно спискам и картежам, внутрь словаря можно вкладывать другие словари, например:

```
users = {
    "Oleg": {"phone": +79514789865, "height": 182, "weight": 91},
    "Mark": {"phone": +79514789865, "height": 182, "weight": 91},
```

```
    "Lena": {"phone": +79514789865, "height": 182, "weight": 91}
}
```

Обращение к элементам, как и в списках:

```
users = {
    "Oleg": {"phone": +79514789865, "height": 182, "weight": 91},
    "Mark": {"phone": +79514789865, "height": 182, "weight": 91},
    "Lena": {"phone": +79514789865, "height": 182, "weight": 91}
}

print(users["Lena"]["phone"])
```

Выводит:

> 79514789865

Как и ожидалось.

Список словарей

Словарь, это вещь удобная, но относительно статичная. Она позволяет создавать очень мощные структуры данных, со вложенностями и богатым описанием. Но динамически добавлять в словарь однородные элементы достаточно сложно, да и не нужно, на мой взгляд.

Поэтому, если нам нужно создать удобно управляемую структуру с однородными словарями, например, списком пользователей, при том, у каждого пользователя должно быть несколько параметров с человекочитаемыми ключами, то разумнее создать список из словарей, вот так:

```
users = []

user_1 = {"name": "Oleg", "phone": +79514789861, "height": 182, "weight": 91}
user_2 = {"name": "Mark", "phone": +79514789862, "height": 179, "weight": 84}
user_3 = {"name": "Lena", "phone": +79514789863, "height": 167, "weight": 59}

users.append(user_1)
users.append(user_2)
users.append(user_3)

print(users)
```

Выведет:

```
 [ {'name': 'Oleg', 'phone': 79514789861, 'height': 182, 'weight': 91}, {'name': 'Mark', 'phone': 79514789862, 'height': 179, 'weight': 84}, {'name': 'Lena', 'phone': 79514789863, 'height': 167, 'weight': 59} ]
```

Мы создаём список, а затем добавляем в этот список по очереди 3 словаря. В итоге получился список словарей. Зачем это нужно? Например, в список мы можем легко добавить сколько угодно новых словарей с пользователями. А что, если вы получаете список пользователей из базы данных по двадцать пользователей за раз, а у вас их, скажем, сто? Например, для этого удобно использовать список словарей, при каждом запросе мы можем дописать в список по двадцать словарей с пользователями, пока их не станет сто, а затем можем удобно использовать весь список этих словарей.

Массивы. Множества

Содержание урока

Понятие множества

Множества — это структура данных также сходная со списками, с рядом оговорок. Во-первых, множество неупорядоченно, то есть, в отличии от списков, где каждый элемент имеет своё фиксированное место (индекс/ключ) - у множеств таких мест (индексов/ключей) нет. Также, элементы множеств являются уникальными, то есть, даже если вы объявите множество с двумя одинаковыми элементами, один из них будет выброшен из множества автоматически.

Множество может содержать строки, числа, кортежи, то есть, любой неизменяемый тип данных, например, кортеж или другое множество, но не может содержать, например, список.

Зачем нужно множество? Из-за его неупорядоченной природы процесс проверки на вхождение в него того или иного элемента/элементов происходит значительно быстрее. Проще говоря, если вам нужно проверить, содержит ли структура данных конкретный элемент или набор элементов, то гораздо быстрее произвести проверку в том случае, если этой структурой данных будет множество, а не список или кортеж.

Создание множества

Создать множество можно при помощи «фигурных скобок»:

```
one_set = {}
two_set = {1, 2, 3, 4, 5, 6, 7, 8}
```

А ещё, при помощи функции `set()`:

```
three_set = set([1, 2, 3, 4, 5, 6, 7, 8])
```

Которая принимает обычный список. Множество не содержит дубликатов:

```
two_set = {1, 1, 2, 3, 4, 5, 6, 7, 8}
print(two_set)
```

Выводит:

> {1, 2, 3, 4, 5, 6, 7, 8}

Как видите, повторяющаяся единица была отброшена.

Работа с множествами

Добавление элемента во множество

Функция `add(element)`:

```
two_set = {1, 1, 2, 3, 4, 5, 6, 7, 8}
two_set.add(9)
print(two_set)
```

Выводит:

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Удаление элементов из множества

Для удаления конкретного элемента используется функция `remove(element)`, в которую передаётся элемент который необходимо удалить:

```
two_set = {1, 1, 2, 3, 4, 5, 6, 7, 8}
two_set.remove(8)
print(two_set)
```

Выводит:

> {1, 2, 3, 4, 5, 6, 7}

Удалили элемент 9.

Чтобы полностью очистить множество, используется метод `clear()`:

```
two_set = {1, 1, 2, 3, 4, 5, 6, 7, 8}
two_set.clear()
print(two_set)
```

Выводит:

> set()

То есть, пустое множество.

Доступ к элементам множества

Так как множество неупорядоченно и у его элементов нет индексов и ключей и напрямую обратиться к конкретному элементу невозможно, однако, обратиться к значению элемента множество можно при помощи перебора его значений через цикл `for`:

```
two_set = {1, 1, 2, 3, 4, 5, 6, 7, 8}
for item in two_set:
    print(item)
```

При переборе через цикл for каждый элемент целиком помещается в переменную `item`. О переборе массивов мы поговорим в следующем уроке посвящённому циклам.

Объединить два множества

Метод `union(множество)`:

```
cars = {"BMW", "Toyota", "Honda"}
new_cars = {"Tesla", "Smart"}
all_cars = cars.union(new_cars)

print(all_cars)
```

Выводит:

> {'BMW', 'Toyota', 'Tesla', 'Honda', 'Smart'}

Проверка на пересечение

Функция `intersection(множество)` сравнивает два множества и возвращает новое множество, содержащее только те элементы, которые присутствуют одновременно в двух сравниваемых множествах:

```
cars = {"BMW", "Toyota", "Honda"}
new_cars = {"Toyota", "Tesla", "Smart"}
intersection = cars.intersection(new_cars)

print(intersection)s
```

Выводит:

> {'Toyota'}

Именно значение Toyota содержится в обоих множествах.

Получить список элементов, не входящих во второе множество

Функция `difference(множество)`:

```
cars = {"BMW", "Toyota", "Honda"}
new_cars = {"Toyota", "Tesla", "Smart"}
difference = cars.difference(new_cars)

print(difference)
```

Выводит:

```
{'BMW', 'Honda'}
```

Этих значений нет во втором множестве.

Проверка на вхождение текущего множества в другое множество

Функция `issubset` позволяет определить, входит ли заданное множество внутрь другого, более крупного множества:

```
cars = {"BMW", "Toyota", "Honda"}
new_cars = {"BMW", "Toyota", "Honda", "Tesla", "Smart"}

print(cars.issubset(new_cars))
```

Выводит:

> True

То есть, множество `cars` входит (является подмножеством) в множество `new_cars`

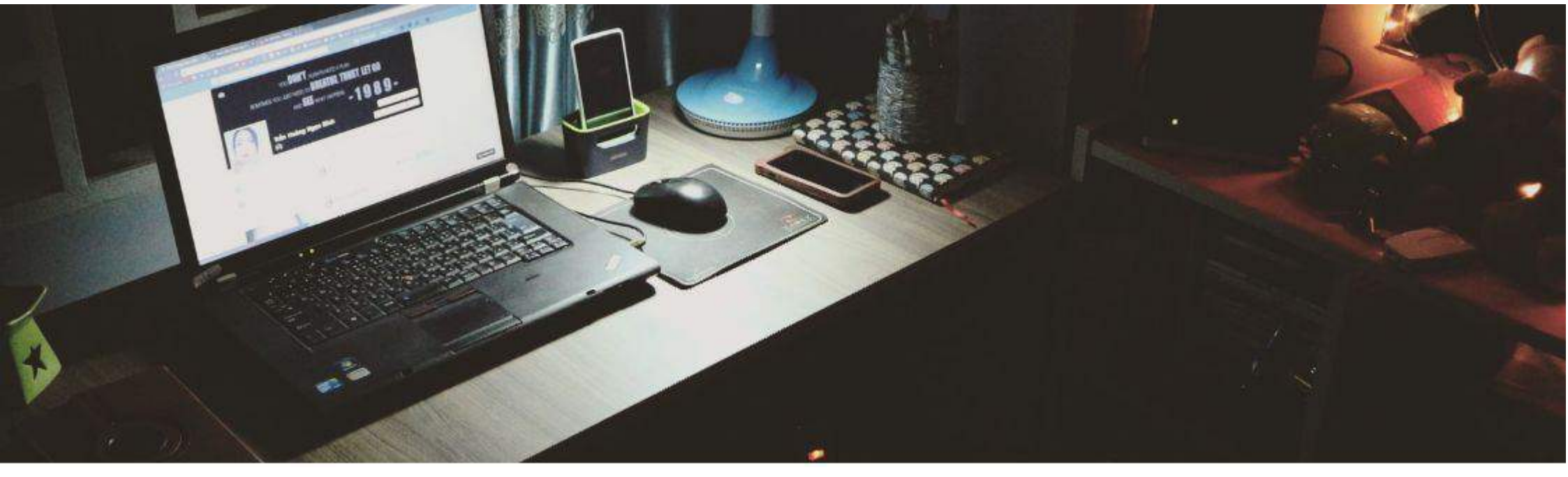
Проверка на вхождение в текущее множество другого множества. Функция `issuperset` возвращает `True`, если текущее множество является «общим» для другого, частного множества:

```
cars = {"BMW", "Toyota", "Honda"}
new_cars = {"BMW", "Toyota", "Honda", "Tesla", "Smart"}

print(new_cars.issuperset(cars))
```

Выводит:

> True

То есть, множество `new_cars` является надмножеством для множества `cars`.



Циклы

Содержание урока

Понятие цикла

В повседневной практике нередко приходится запускать определённый участок кода несколько раз. Например, у вас есть файл со ссылками на профили ВК, и вам нужно узнать, закрытые это профили или открытые. Вы написали код, который отправляет запрос к странице пользователя, если в ответ на запрос возвращается страница, которая содержит текст, указывающий на то, что она открыта, он добавляет этот профиль в таблицу excel, иначе - ничего не делает.

Но не будем же мы, этот код прописывать столько раз, сколько проверок нужно совершить. Конечно, нет. Мы просто вложим этот код в цикл и укажем, сколько раз необходимо данный код повторить. Для этого циклы и существуют. В общем смысле цикл — это такая конструкция, которая позволяет помещённый в неё код, запустить несколько раз подряд несмотря на то, что сам код приведён лишь один раз.

Python включает в себя два вида циклов: `while` и `for`

Цикл while

Цикл while — это цикл, который работает до тех пор, пока заданное условие возвращает True. Например, можно указать условие проверки на наличие строк в файле, пока строки в файле есть (возвращает True) - цикл while работает, если строк в файле больше нет (возвращает False) - цикл перестаёт работать.

Формально, объявление цикла while выглядит следующим образом:

```
while условие:
    #код, который нужно выполнить
```

Например, так (Данный код не работает, а служит в качестве формального примера):

```
lines_exists = True
lines = []

# Если переменная lines_exists содержит True - продолжаем цикл
# Если переменная lines_exists содержит False - станавливаем цикл
while lines_exists:
    # Проверяем, есть ли ещё строки в файле
    # если есть - возвращаем True, иначе - False
    lines_exists = LINE_EXIST("path/to/file.txt")
    # Добавляем строку в список (например)
    lines.append(get_line("path/to/file.txt"))
```

Во время работы цикла мы проверяем, есть ли ещё в файле строки, если их нет – присваиваем переменной `lines_exists` значение `False` и следующий повтор цикла уже не состоится.

Пример попроще:

```
count = 15

while count != 0:
    count -= 1
    print("Итерация номер: " + str(count))
```

Здесь цикл работает до тех пор, пока переменная count не равна 0. Каждую итерацию цикла мы отнимаем по 1 от заданного счётчика до тех пор, пока он не будет равен нулю.

Выведет:

> Итерация номер: 14 Итерация номер: 13 Итерация номер: 12 Итерация номер: 11 Итерация номер: 10 Итерация номер: 9 Итерация номер: 8 Итерация номер: 7 Итерация номер: 6 Итерация номер: 5 Итерация номер: 4 Итерация номер: 3 Итерация номер: 2 Итерация номер: 1 Итерация номер: 0

Цикл сделал 15 «оборотов» или корректнее говоря – итераций и закончил выполнение работы, т.к. значение счётчика стало рано 0.

Цикл for

Цикл for используется для перебора некоторых коллекций, например, списков, словарей, кортежей и т.д.:

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

for num in nums:
    print("Число: " + str(num))
```

Выведет:

> Число: 1 Число: 2 Число: 3 Число: 4 Число: 5 Число: 6 Число: 7 Число: 8 Число: 9 Число: 10 Число: 11

Цикл for, позволяет «пройтись» по всем значениям некоего множество, в данном случае – списка и получить в каждое отдельной итерации одно из значений множество.

Если вам просто нужно запустить цикл на некоторое количество повторений, можете воспользоваться функцией `range()`. Данная функция создаёт множество состоящее из последовательности чисел, которую вы задаёте сами. Range может принимать три параметра, подобно механизму слайсов у списков: range(от, до, шаг).

`range(до)` – создаёт список с числами от нуля и до значения, которое вы укажите (не включая это значение), например, если вы вызовете функцию range(5) она создаст множество (если проще, список) с числами от 0 до 4:

```
for num in range(5):
    print("Число: " + str(num))
```

Выведет:

> Число: 0 Число: 1 Число: 2 Число: 3 Число: 4

`range(от, до)` - создаёт список с числами в диапазоне от и до (не включая значение «до»), например, если вы вызовете функцию range(1, 5) она создаст множество (если проще, список) с числами от 1 до 4:

```
for num in range(1, 5):
    print("Число: " + str(num))
```

Выведет:

> Число: 1 Число: 2 Число: 3 Число: 4

`range(от, до, шаг)` - создаёт список с числами в диапазоне от (включая) и до (не включая), с определённым заданным шагом, например, если вы вызовете функцию range(1, 10, 2) она создаст множество (если проще, список) с числами: 1, 3, 5, 7, 9

```
for num in range(1, 10, 2):
    print("Число: " + str(num))
```

Выведет:

> Число: 1 Число: 3 Число: 5 Число: 7 Число: 9

Выход из цикла. Оператор break.

Например, у нас есть множество и нам нужно найти некоторый элемент, если мы его нашли – выполняем определённое действие, иначе – ничего не делаем. Но что, если данный элемент находится в середине списка, мы нашли этот элемент, выполнили с ним определённое действие, неужели нам необходимо продолжать работу цикла пока он не закончится?

Разумеется, нет, для этого случая существует оператор выхода из цикла break. Данный оператор прекращает выполнение цикла и выходит из него:

```
users = ["Mark", "Anna", "Maks", "Alex", "Roma"]

for user in users:
    print("Найден пользователь: " + user)
    if user == "Anna":
        print("Привет, " + user + "!")
        break
```

Выведет:

> Найден пользователь: Mark Найден пользователь: Anna Привет, Anna!

После того, как был найден пользователь Anna, цикл прекратил выполнение.

Выход из цикла. Оператор return

Также, если ваш цикл работает в рамках функции, можно прервать его выполнение вызовом оператора `return`, таким образом вы можете прервать выполнение цикла возвратом значения из функции.

Пропуск итерации цикла. Оператор continue

Оператор `continue` пропускает текущую итерацию, например, если перебирая некоторое множество нам нужно отбросить значения определённого типа и ничего с ними не делать, то мы можем воспользоваться этим оператором:

```
users = ["Mark", "Anna", "Maks", "Alex", "Roma"]

for user in users:
    if user == "Anna":
        continue
    print("Найден пользователь: " + user)
```

Выведет:

> Найден пользователь: Mark Найден пользователь: Maks Найден пользователь: Alex Найден пользователь: Roma

Операции над элементом со значением Anna не были произведены. Итерация с её значением был пропущена.

Перебор массивов

Перебор элементов списка

Элементы списка через цикл `for` перебираются следующим образом:

```
for значение in имя_списка:
    # Операции над значением
```

Значение – это любая корректная переменная, в которую помещается элемент из последовательности. Имя списка – переменная в которую помещается список:

```python
users = ["Mark", "Anna", "Maks", "Alex", "Roma"]

for user in users:
    print("Привет, " + user + "!")
```

Выводит:

> Привет, Mark! Привет, Anna! Привет, Maks! Привет, Alex! Привет, Roma!

Перебор элементов кортежа

Перебор кортежа производится по тому же принципу, что и список:

```python
users = ("Mark", "Anna", "Maks", "Alex", "Roma")

for user in users:
    print("Привет, " + user + "!")
```

Выведет:

> Привет, Mark! Привет, Anna! Привет, Maks! Привет, Alex! Привет, Roma!

Перебор элементов словаря

Перебор словаря также производится при помощи цикла `for`, но вместо значения ему возвращается ключ:

```python
user = {"name": "Mark", "phone": "+79518746317", "age": 23, "height": 182, "weight": 87}

for key in user:
    print(key)
```

Выведет:

> name phone age height weight

Цикл вывел не значения, как это было у списков и кортежей, а ключ. Чтобы получить значение, необходимо обратиться к элементу по его ключу:

```python
user = {"name": "Mark", "phone": "+79518746317", "age": 23, "height": 182, "weight": 87}

for key in user:
    print(user[key])
```

Выводит:

> Mark +79518746317 23 182 87

Или вот так:

```
user = {"name": "Mark", "phone": "+79518746317", "age": 23, "height": 182, "weight": 87}

for key in user:
    print(key + ": " + str(user[key]))
```

Выводит:

name: Mark phone: +79518746317 age: 23 height: 182 weight: 87

Перебор значений через функцию `values()`, синтаксис:

```
for значение in словарь.values():
    # Код для работы со значением
```

Данная функция возвращает исключительно значения словаря:

```
user = {"name": "Mark", "phone": "+79518746317", "age": 23, "height": 182, "weight": 87}

for value in user.values():
    print(value)
```

Выводит:

> Mark +79518746317 23 182 87

Перебор вложенных словарей

Перебор вложенных словарей происходит также, как и обращение к элементам вложенных словарей:

```
users = {
    "Mark": {"phone": "+79518746317", "age": 23, "height": 182, "weight": 87},
    "Anna": {"phone": "+79518746317", "age": 23, "height": 182, "weight": 87},
    "Oleg": {"phone": "+79518746317", "age": 23, "height": 182, "weight": 87},
    "Alex": {"phone": "+79518746317", "age": 23, "height": 182, "weight": 87},
}

for user_key in users:
    print()
    print("name: " + user_key)
    for user_attribute in users[user_key]:
        print(user_attribute + ": " + str(users[user_key][user_attribute]))
```

Выводит:

> name: Mark phone: +79518746317 age: 23 height: 182 weight: 87 name: Anna phone: +79518746317 age: 23 height: 182 weight: 87 name: Oleg phone: +79518746317 age: 23 height: 182 weight: 87 name: Alex phone: +79518746317 age: 23 height: 182 weight: 87

В первом цикле мы получаем доступ к каждому вложенному элементу (словарю):

```
for user_key in users:
```

Внутри итерации по каждому словарю, запускаем цикл, который перебирает все элементы вложенного словаря:

```
for user_attribute in users[user_key]:
    print(user_attribute + ": " + str(users[user_key][user_attribute]))
```

`users[user_key]` – это непосредственно вложенный словарь. `user_attribute` – это ключ элемента вложенного словаря. `users[user_key][ user_attribute]` – это значение элемента вложенного словаря.

По идентичному принципу работает процесс перебора вложенных списков и кортежей.

Перебор множеств

Идентичен процессу перебора списков и кортежей через цикл for:

```
users = {"Mark", "Anna", "Oleg", "Alex"}

for item in users:
    print(item)
```

Выведет:

> Alex Anna Mark Oleg

Номер итерации в цикле for

Порой, полезно узнать, сколько уже итераций было сделано в цикле. Для этого существует функция `enumerate(массив)`. Синтаксис:

```
for номер_итерации, значение in enumerate(массив):
    # Операции над номером итерации и значением
```

Пример:

```
users = ["Mark", "Anna", "Oleg", "Alex"]

for index, user in enumerate(users):
    print(str(index) +": " + user)
```

Выведет:

```
0: Anna 1: Oleg 2: Alex 3: Mark
```

Синтаксис при работе с этой функцией достаточно прост, вместо того чтобы передать в for сам массив, мы передаём в него функцию `enumerate ()`, в которую, в свою очередь, передаём наш массив.

Далее, перед переменной в которую будет записываться ключ или значение массива, указываем переменную, в которую будет записываться номер итерации (index) и разделяем их запятой.

Далее, в самом цикле можем использовать переменную с номером итерации так, как нам нужно.

Модули и пакеты

Содержание урока

До сих пор, код из примеров мы писали в отдельных, никак не связанных между собой файлах. Но что делать, когда мы начнём писать рабочее прикладное приложение, которое реализует сложные многофакторные задачи и имеет разветвлённую логику?

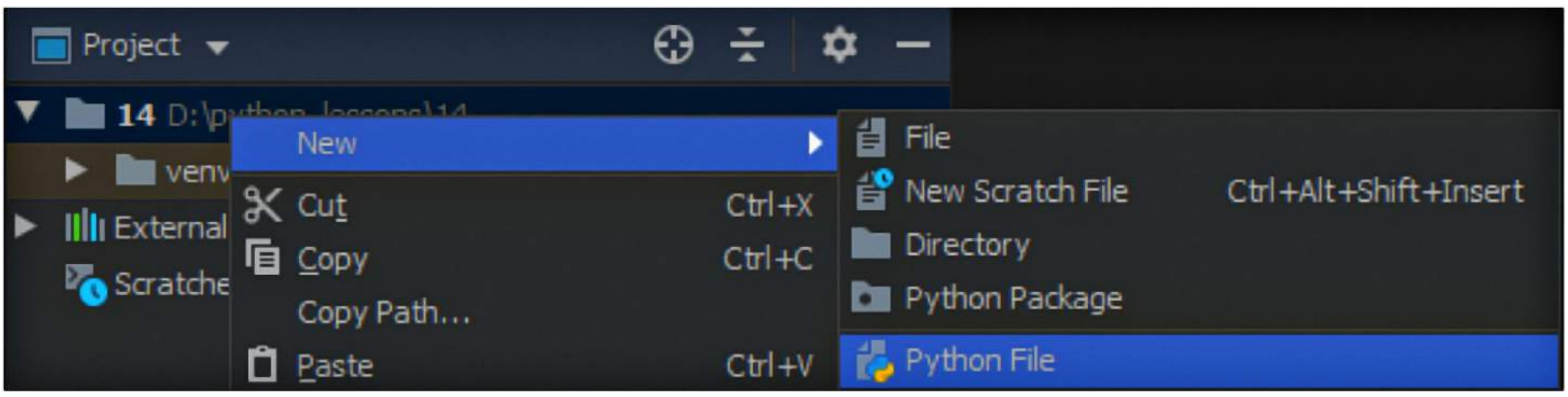
Понятие модуля

Разумеется, нам придётся разделять код на блоки и функции. И размещать наш код мы будем конечно же в отдельных файлах. Каждый файл будет отвечать за определённую задачу или модуль, в нём будут находится разные функции, решающие набор конкретных подзадач.

Затем, созданные нами файлы с функционалом будут подключаться друг к другу и использоваться в процессе работы над проектом. Так вот, в языке Python эти файлы именуются модулями. В общем смысле модуль в Python - это файл, содержащий некоторый код, который можно включать в другие модули и повторно использовать в различных частях приложения.

Модуль всегда определяется как файл с расширением *.py Название файла будет одновременно и названием модуля. Допустим, мы пишем собственный блог и нам нужен модуль для работы с пользователями. Например, авторизация, регистрация или блокировка пользователя. Для этого разумно создать модуль user.py и разместить в нём функции для работы с пользователем.

Для начала, в корневой директории нашего проекта (который мы создали через IDE PyCharm) создадим файл user.py:

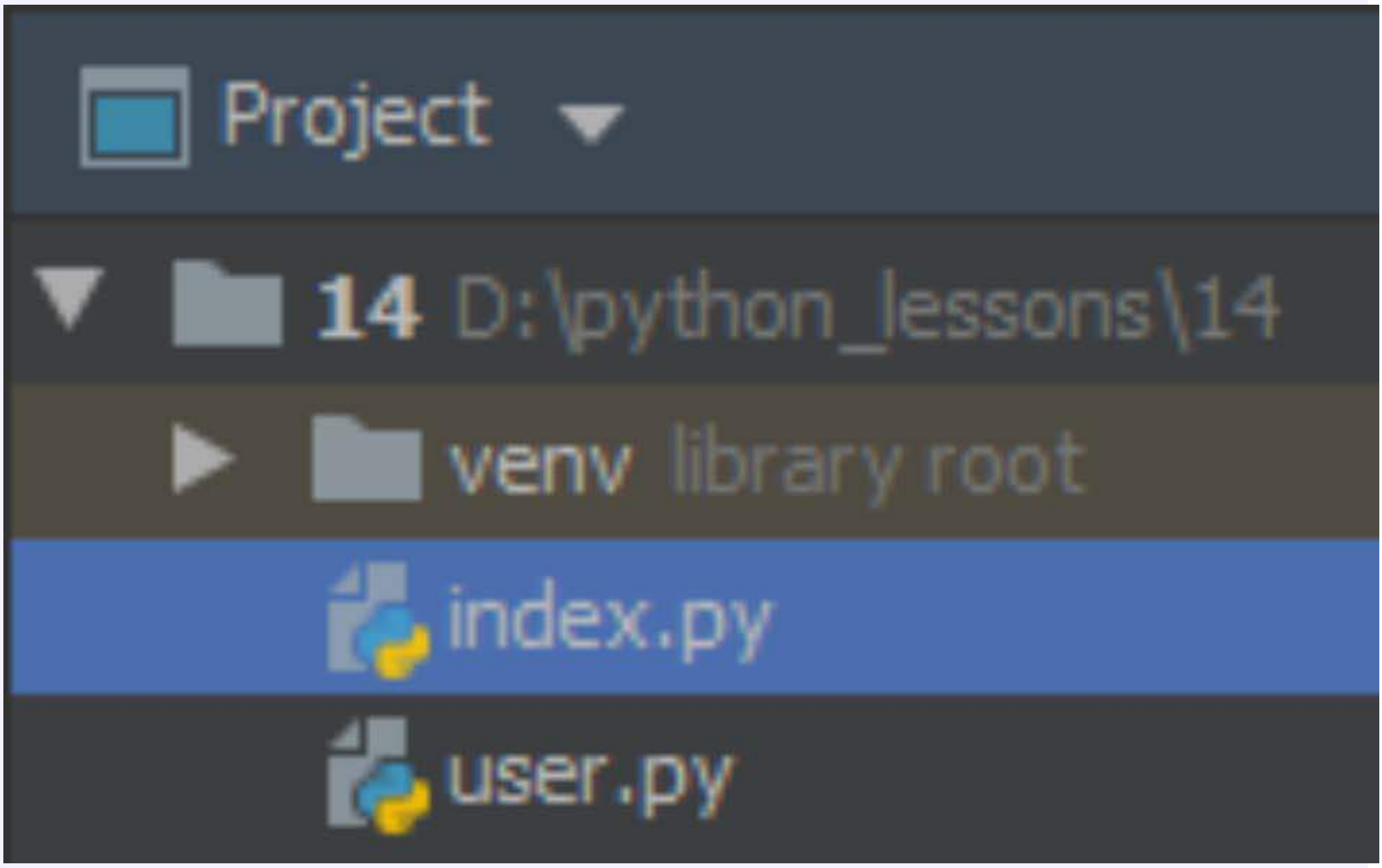


После создания файла можно считать, что мы создали новый модуль. Теперь давайте объявим в нём несколько функций:

```
def new(login, password):
    return "Пользователь успешно создан!"

def auth(login, password):
    return "Вы успешно авторизовались!"

def block(login):
    return "Пользователь заблокирован"
```

Теперь мы имеем готовый модуль с функционалом и его можно использовать в других модулях. Повторим, для того, чтобы создать модуль, достаточно всего лишь создать файл с корректным названием и расширением `*.py` и разместить в нём код, в нашем случае – функции.

Воспользуемся созданным модулем, подключив его к другому модулю. Для этого в той же корневой директории нашего проекта создадим новый файл, например, с названием `index.py`:



Чтобы подключить модуль, необходимо воспользоваться ключевым словом import, после него указывая название нашего файла с модулем, но без расширения `*.py`:

```
import user
```

Далее, мы можем получить доступ к функциям этого модуля при помощи синтаксиса: `название_модуля.название_функции()`:

```
# file: index.py

import user

user.new("Mark", "dsferiof23hf")
user.auth("Mark", "dsferiof23hf")
user.block("Mark")
```

Псевдонимы для модулей

Когда мы импортируем модуль, обращение к его членам производится через имя файла этого модуля (в нашем случае `user`), но можно поменять это имя при помощи ключевого слова `as`:

```
import user as user_account

user_account.new("Mark", "dsferiof23hf")
user_account.auth("Mark", "dsferiof23hf")
user_account.block("Mark")
```

Псевдоним для функций

Можно также объявить псевдоним для конкретной функции при помощи ключевого слова `from`:

```
import user as user_account
from user import new

new("Mark", "dsferiof23hf")
user_account.auth("Mark", "dsferiof23hf")
user_account.block("Mark")
```

Теперь для того, чтобы вызвать функцию new модуля `user` мы можем не указывать имя модуля, а вызывать напрямую функцию по её имени.

Импортировать все функции по имени можно при помощи символа `*` :

```
from user import *

new("Mark", "dsferiof23hf")
auth("Mark", "dsferiof23hf")
block("Mark")
```

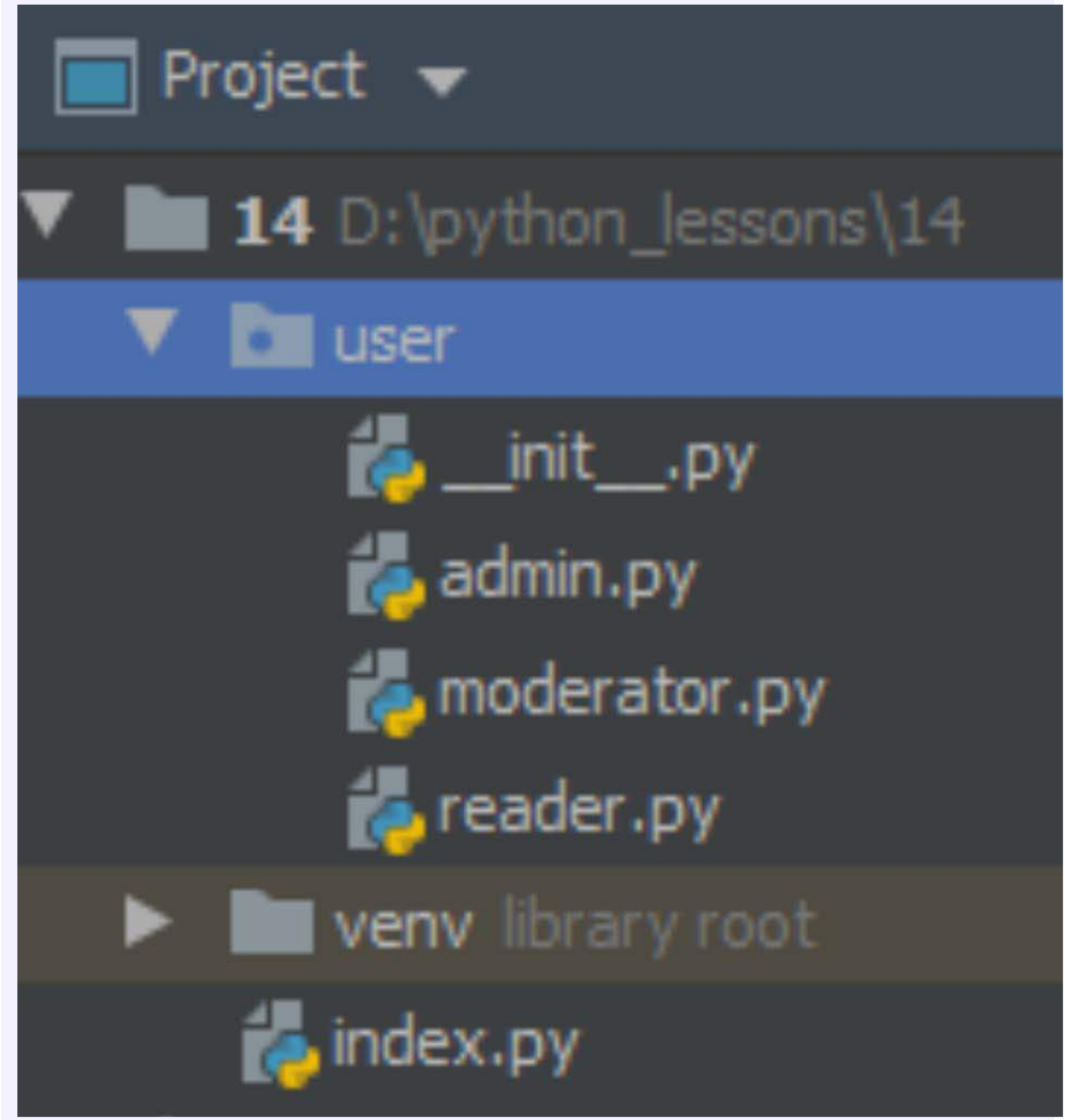
Теперь можно вызывать функции без имени модуля.

Пакеты

До текущего момента мы включали функции и модули из корневой директории проекта, но что делать если мы хотим разместить наши модули по папкам? Для этого необходимо создать пакеты модулей.

Пакет модулей – это специальным образом оформленная папка в директории проекта. Для того, чтобы создать пакет, необходимо:

Создать новую папку
Внутри неё разместить пустой файл с именем `__init__.py`
Разместить файлы с модулями внутри этой папки



Подключение модуля из пакета к другому модулю. Перейдём в файл index.py и импортируем новый модуль из нашего пакета:

```
import user.admin as admin

admin.new("Mark", "dsferiof23hf")
```

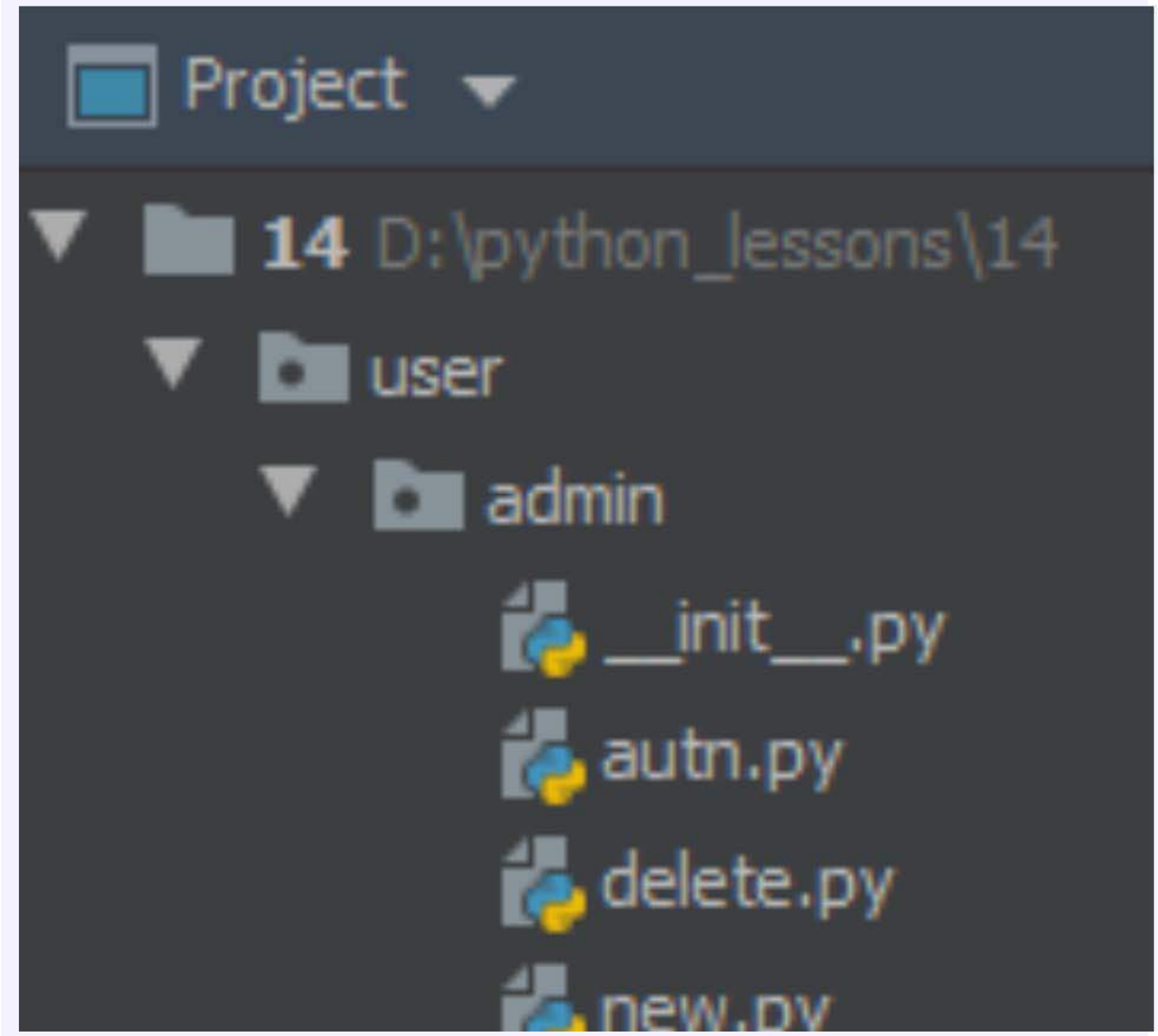
```
admin.auth("Mark", "dsferiof23hf")
admin.block("Mark")
```

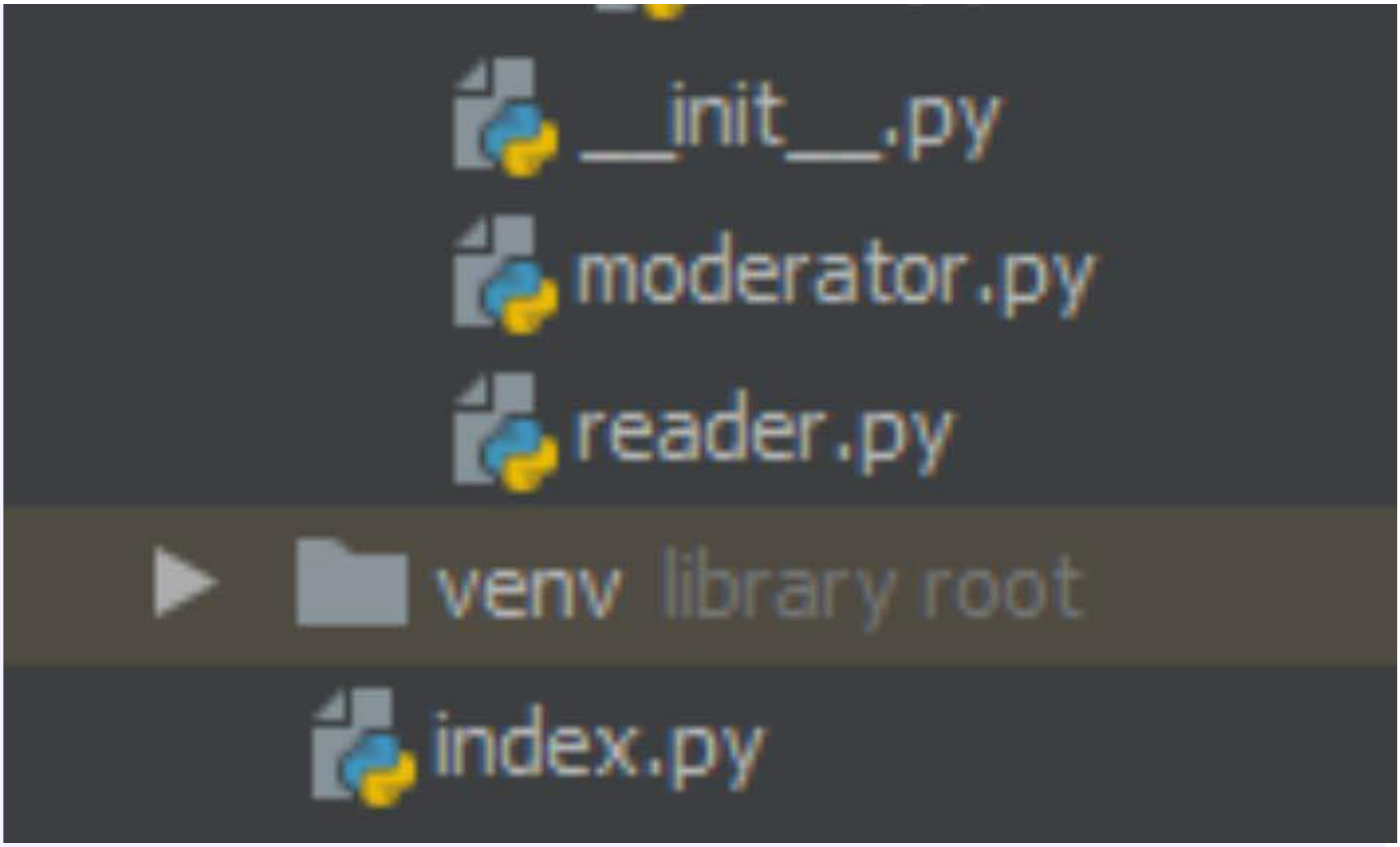
При помощи ключевого слова `import` указываем вложенный модуль, сперва указываем пакет в котором размещены наши модули, и через точку указываем необходимый нам модуль. И чтобы не писать каждый раз полный путь в виде пакет.модуль мы сразу указываем для него псевдоним через ключевое слово as.

Альтернативным решением может послужить подобная конструкция:

```
from user import admin

admin.new("Mark", "dsferiof23hf")
admin.auth("Mark", "dsferiof23hf")
admin.block("Mark")
```

Также пакет может иметь сколько угодно вложенных пакетов, для этого внутри директории пакета, размещается вложенная директория с файлом `__init__.py` и файлами модулей:





Подключение в файл происходит аналогичным образом, через символ «точка»:

```
import user.admin.new as new
new.make_account("Mark", "sr345tr4f54d4")
```

Или:

```
from user.admin import new
new.make_account("Mark", "sr345tr4f54d4")
```

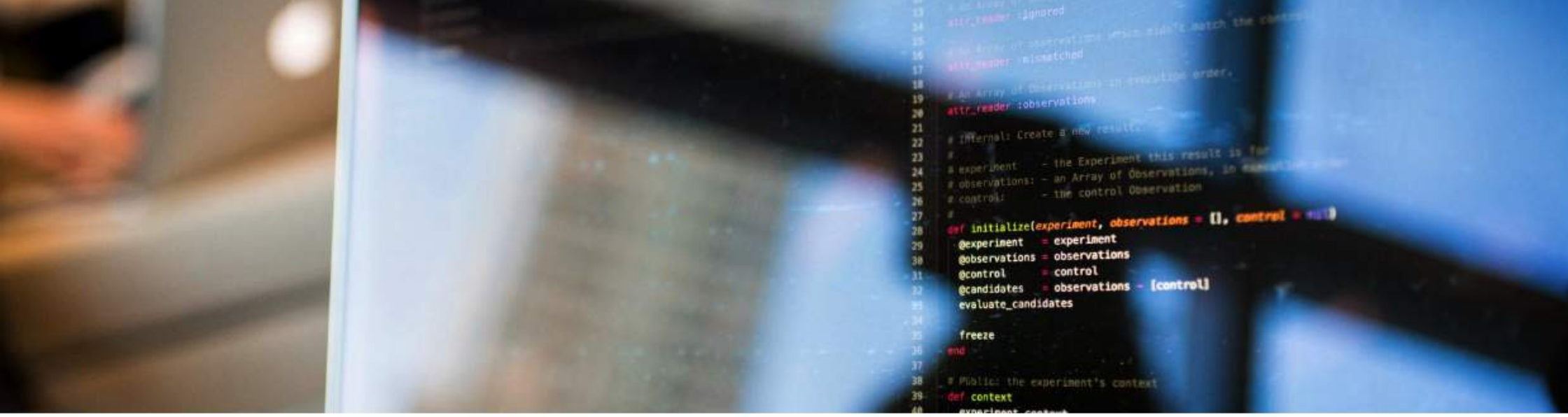
Встроенные модули

Существуют также и встроенные модули, которые поставляются вместе с самим интерпретатором Python. Они включаются также, как самостоятельно созданные модули, через конструкции `import` И `from`.

Со списком встроенных модулей Python можно ознакомиться:

Wiki (на русском): https://ru.wikipedia.org/wiki/Стандартная_библиотека_Python
Документация Python (на английском): https://docs.python.org/3/library/index.html

В следующих уроках мы затронем некоторые встроенные модули.



Установка сторонних модулей

Содержание урока

Что такое сторонние модули? Это уже готовые файлы с кодом, которые мы можем импортировать в наш проект и вызывать их функционал. Например, модуль `Requests` позволяет отправлять запросы к другим сайтам и серверам при помощи заранее написанных методов для этой библиотеки.

Вместо того, чтобы вам с нуля писать удобную обёртку над низкоуровневым интерфейсом для работы с сетью, вы можете просто установить модуль `requests`. Ну или любой другой, из тысяч модулей (на github более 130 000 публичных репозиториев), которые выполняют задачи всех цветов и расцветок.

Подытожим. Внешние библиотеки, это уже написанные кем-то модули, которые решают разного рода задачи.

Установка сторонних модулей

Преимущественно, установка пакетов производится при помощи встроенного в Python (с версии 3.4) менеджера пакетов `pip`.

Для того, чтобы установить пакет, нам нужно:

Перейти в терминале в папку с нашим Python-проектом
Прописать команду: pip install имя_модуля
Нажать Enter

Модуль, имя которого вы ввели будет автоматически скачан со стороннего сервера и загружен в ваш проект, после этого при помощи конструкции `import имя_пакета`, вы сможете его включить. Какие модули и пакеты включать через инструкцию `import` обычно пишут в документации библиотеки.

Удаление сторонних модулей

Для того, чтобы удалить пакет, нам нужно:

Перейти в терминале в папку с нашим Python-проектом
Прописать команду: pip uninstall имя_модуля
Нажать Enter

Модуль, имя которого вы ввели будет удалён из вашего проекта.

Список установленных модулей

Для того, чтобы узнать какие модули были установлены через `pip`, достаточно ввести в терминал команду:

```
pip list
```

И она выведет список с именами модулей которые были установлены в вашем проекте.

Сервера со сторонними модулями

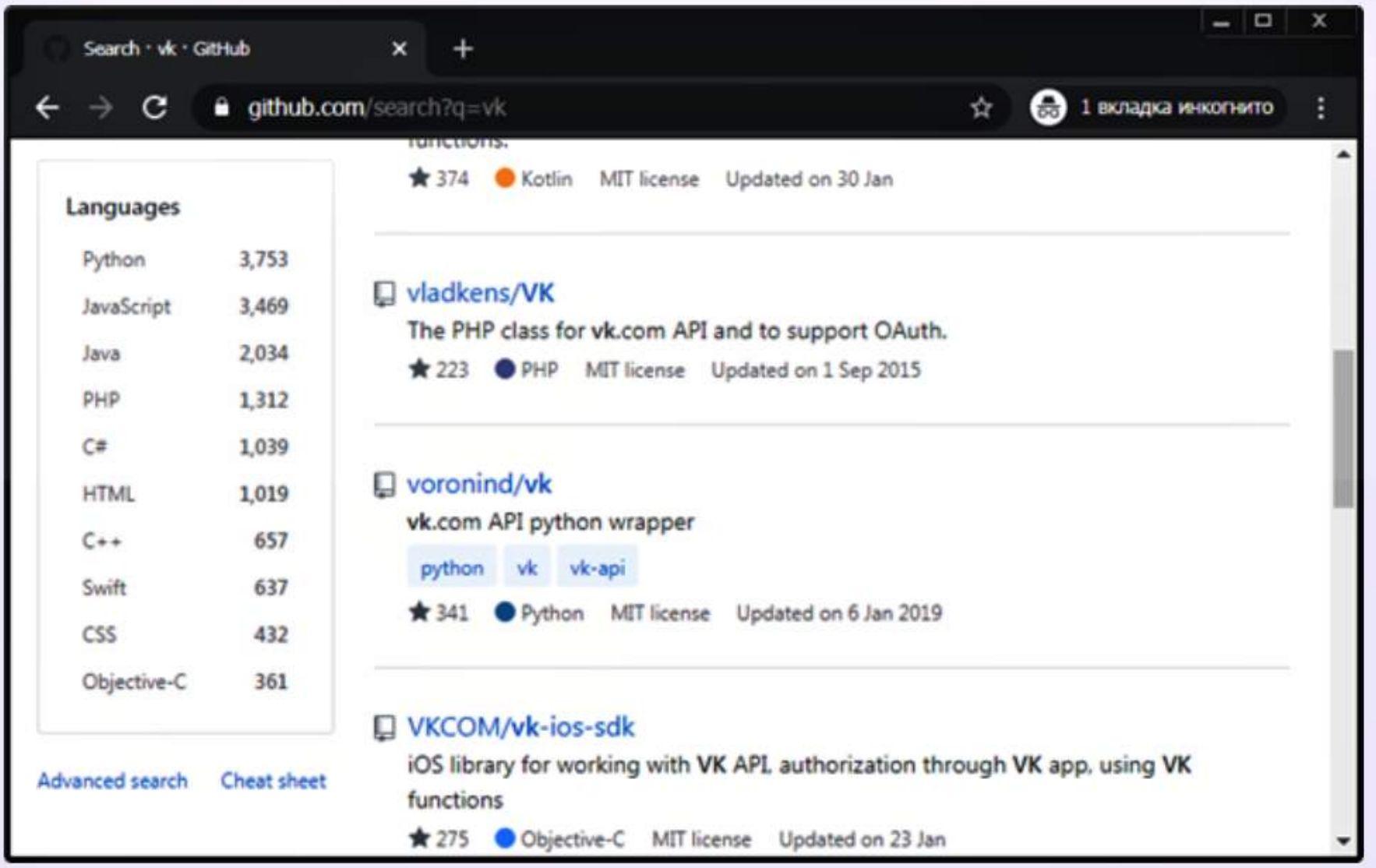
Хорошо, мы поняли, что такое сторонние модули, как их установить и удалить. Но, где их искать и откуда их скачивать? Обычно я использую старый добрый `github.com`. Но также есть сайт репзитория python package index: `pypi.org`

github.com

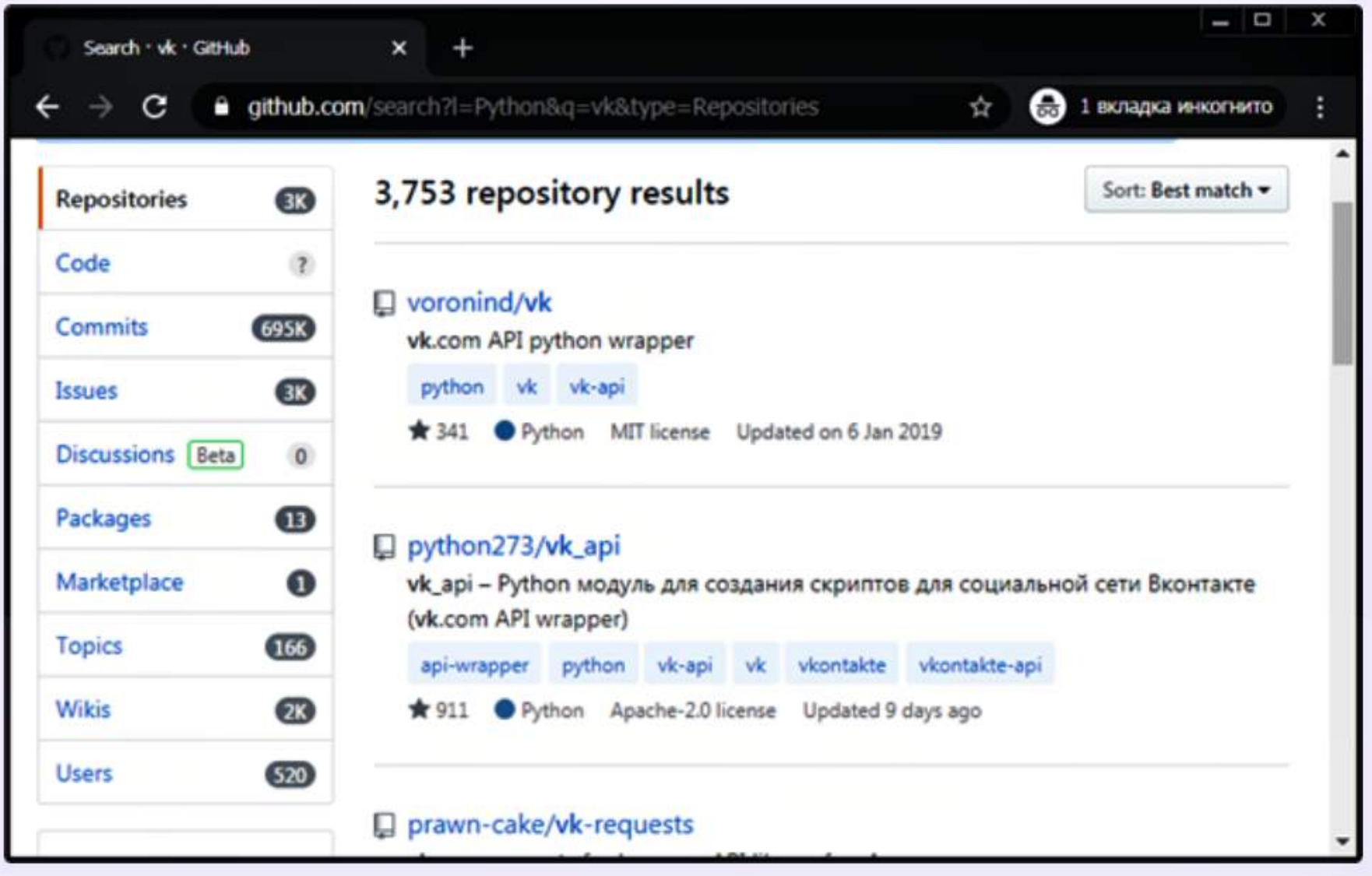
Допустим, мы хотим найти библиотеку для работы с ВК. Для этого переходим на сайт: https://github.com/

В поле для поиска (справа, вверху) вводим запрос: vk

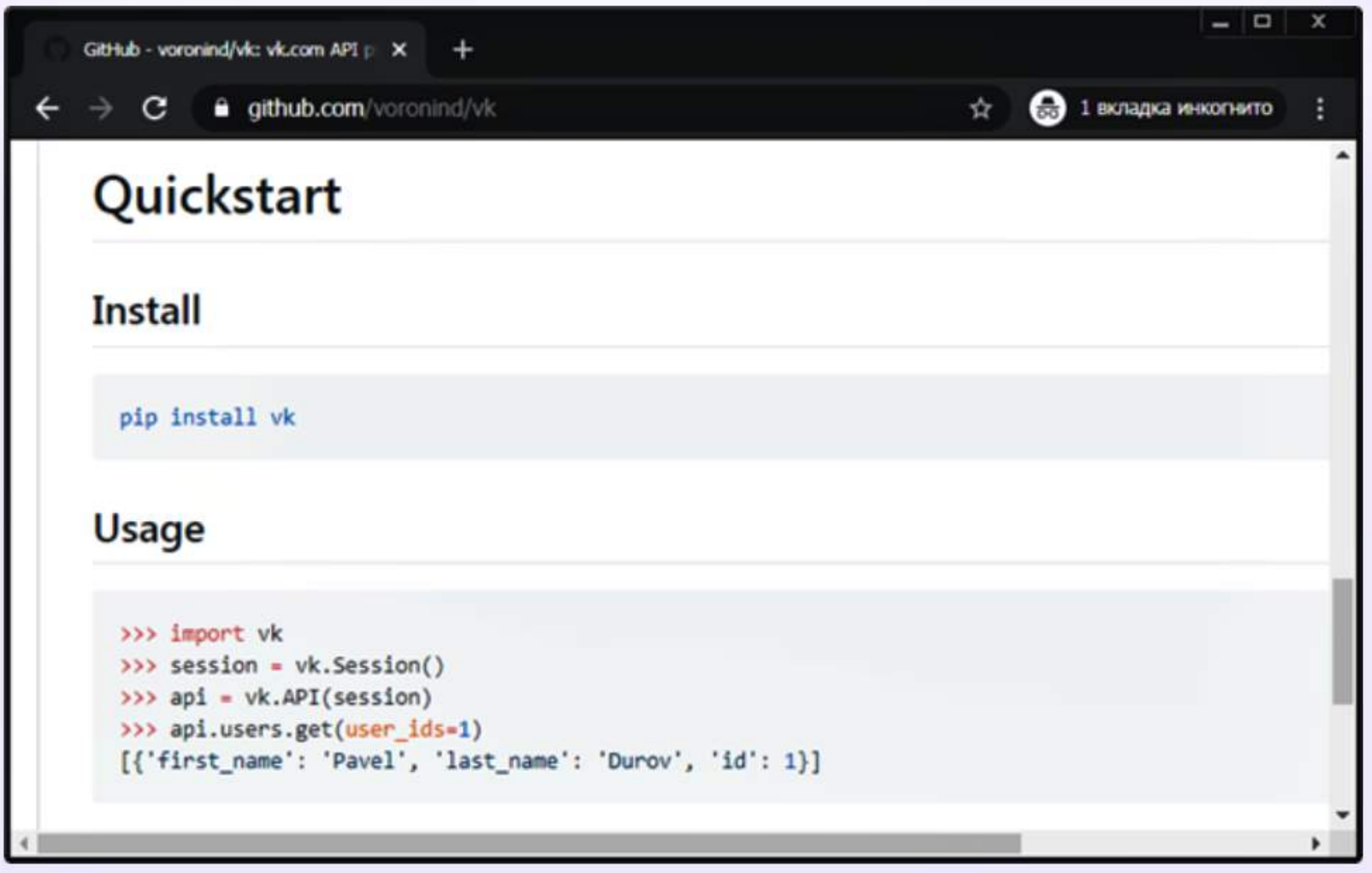
Открывается страница:



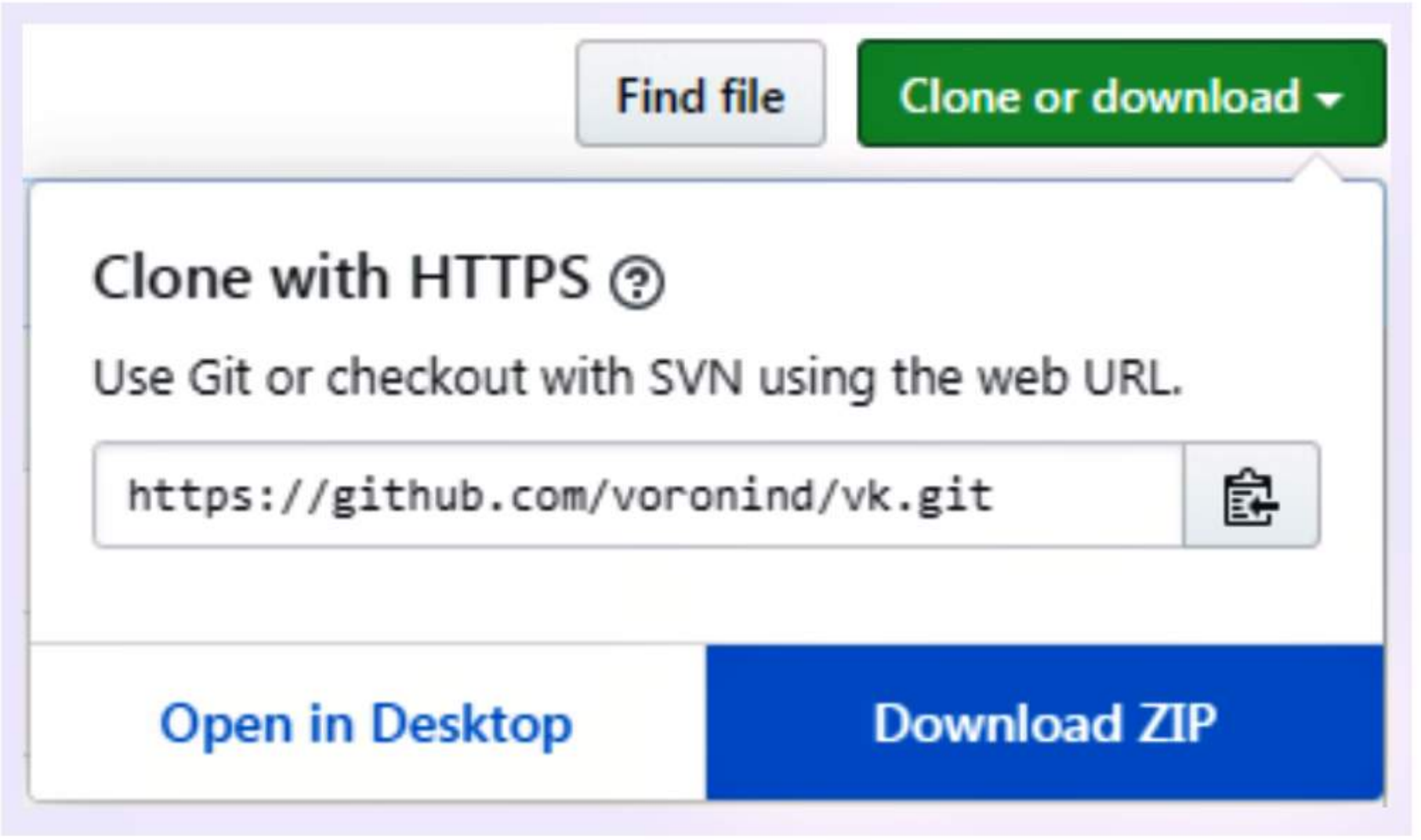
Слева в блоке `languages` выбираем язык Python. Открывается список репозиториев, соответствующих критериям вашего поиска:



Далее, открываете каждый, ознакамливаетесь с функционалом. Практически в каждом публичном репозитории есть раздел `Installation`, в котором прописана инструкция по установке модуля, например, вот так:



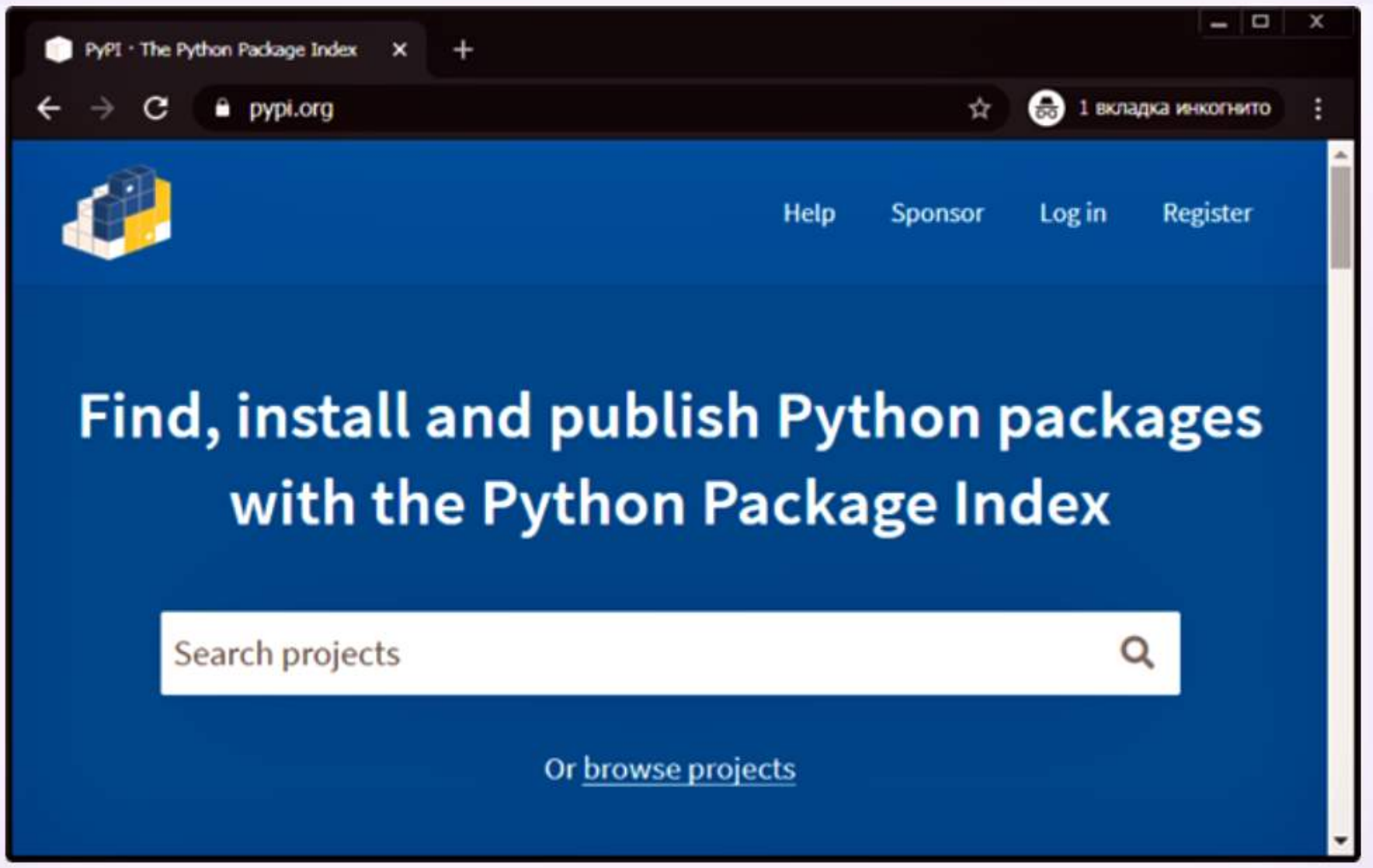
Как видите, здесь указана команда, для установки модуля через менеджер пакетов `pip`. Некоторые репозитории не предполагают установку модуля через `pip`, а представляют просто набор файлов с исходным кодом. В таком случае, вы можете скачать все исходные коды при помощи кнопки `Clone or download`:



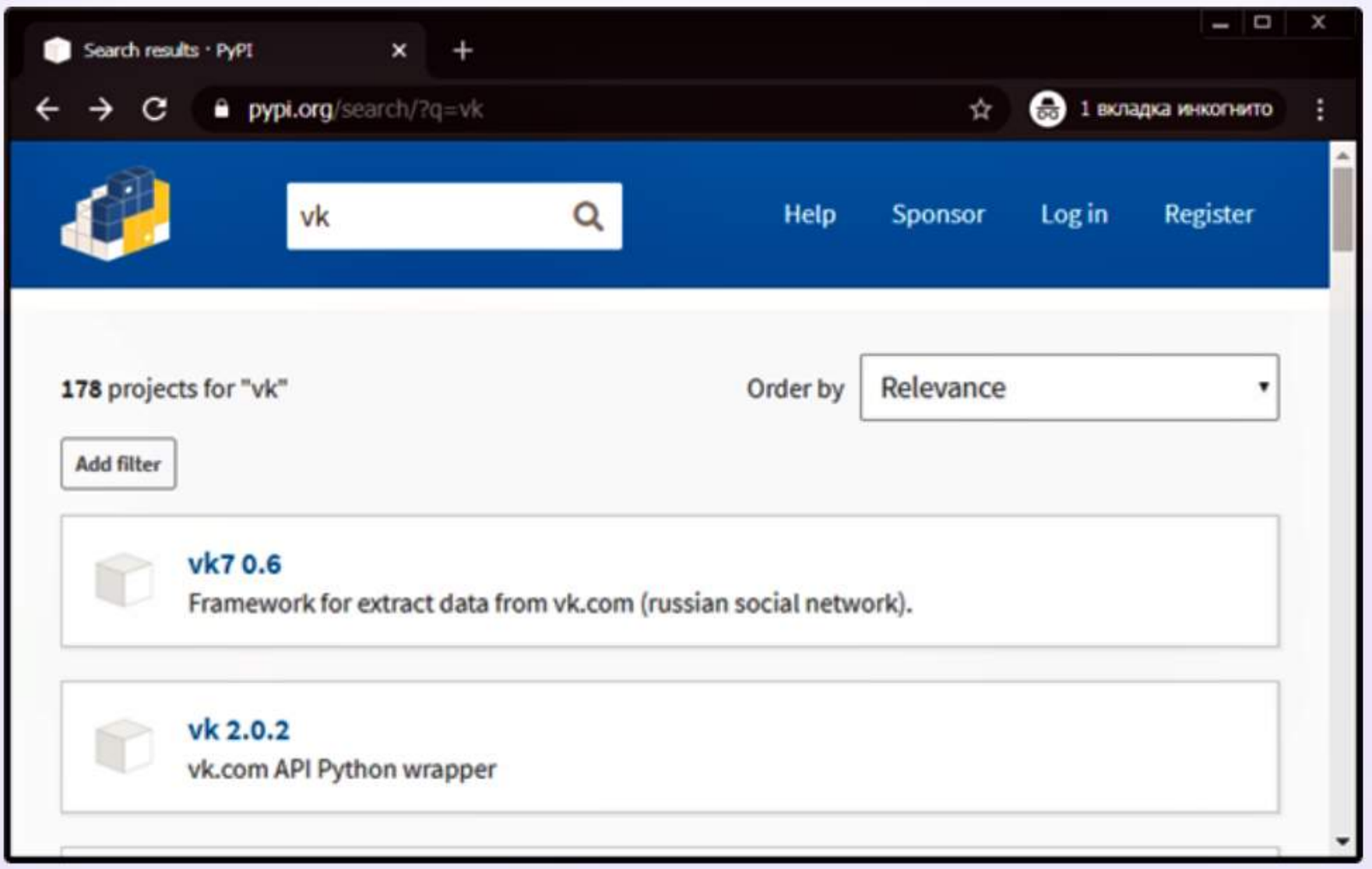
А затем, оформить скачанные файлы в виде собственных модулей. Но если есть возможность установить модуль через `pip` – всегда делайте это.

Python package index: pypi.org

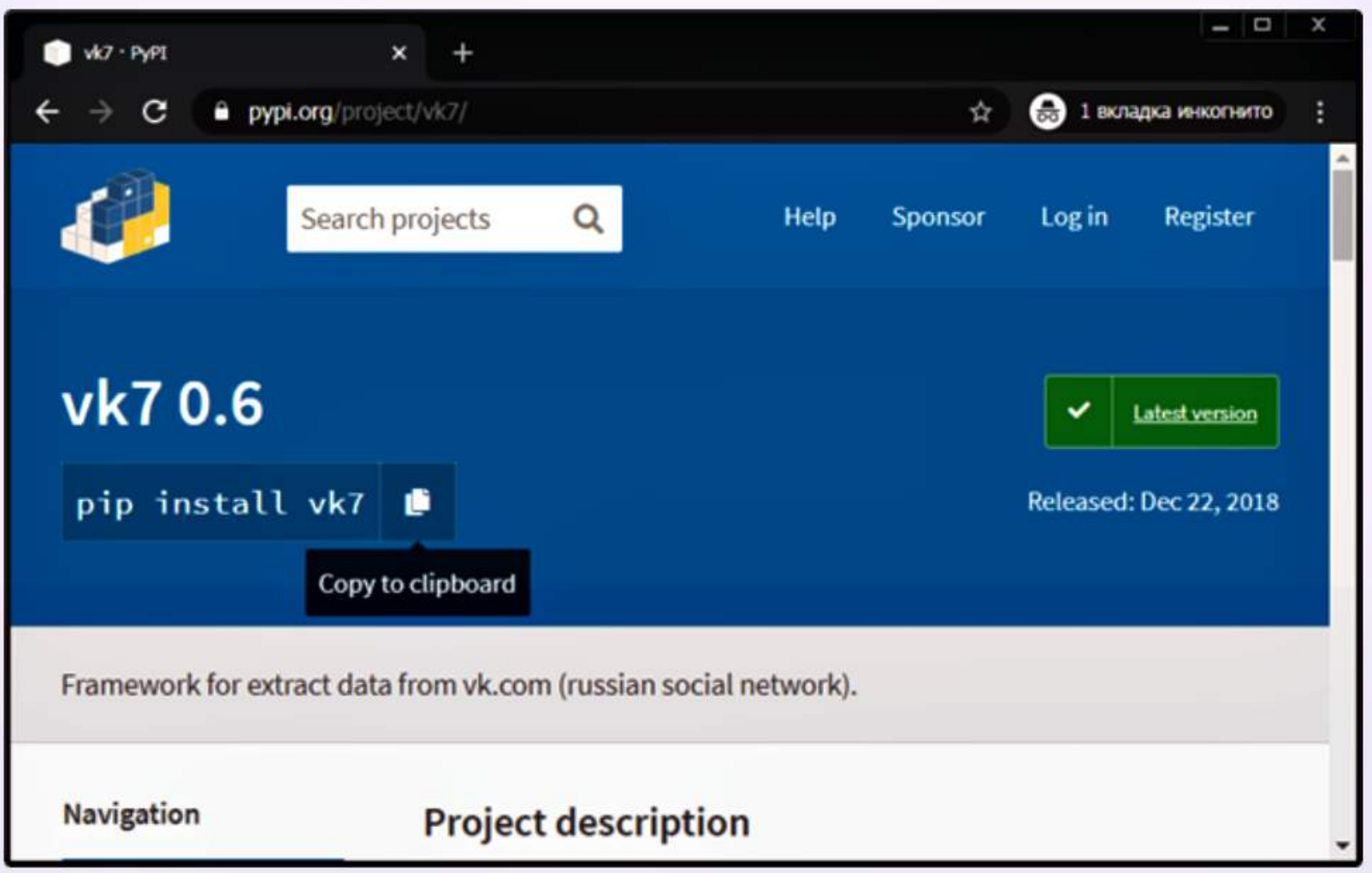
Ещё один репозиторий, в котором можно найти модули для Python – это `pypi.org`:



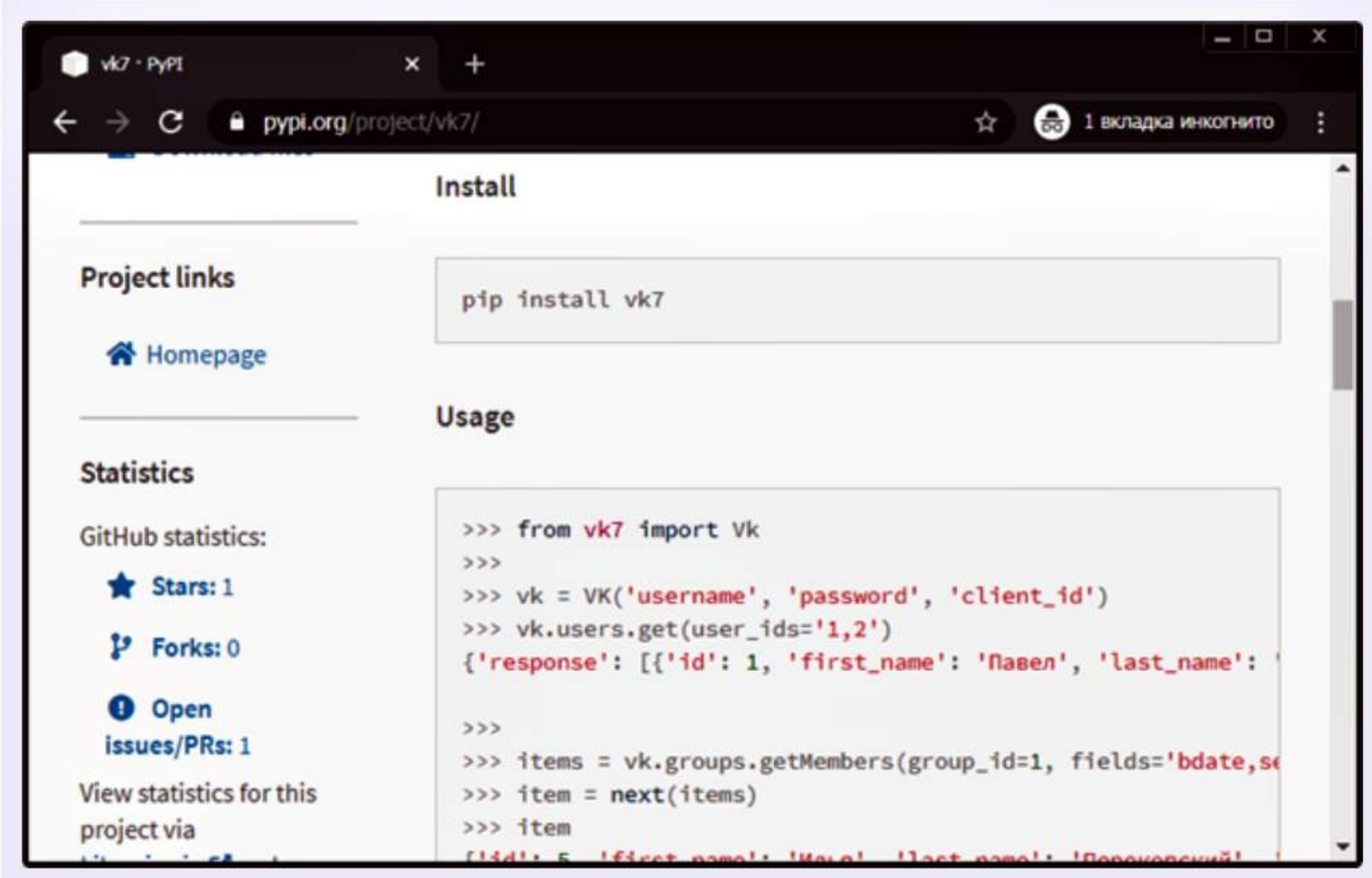
Поиск библиотек в `pypi` осуществляется идентичным образом. В поле для поиска вводим искомые ключевые слова, например, vk:



Видим список библиотек. Просматриваем каждую библиотеку и выбираем подходящую. На странице с библиотекой вы увидите удобную форму, из которой можно скопировать команду для установки:



Также на странице как правило указаны инструкции по установке и работе с библиотекой, а также ссылки на документацию:



Ошибки выполнения программы

Содержание урока

Понятие исключений

Если ваша программа написана правильно с синтаксической точки зрения, то есть, вы нигде не пропустили запятую, аргумент, отступ или не допустили ошибку в одном из ключевых слов или функций, ваша программа всё равно может работать с ошибкой, если при разработке вы написали некорректный алгоритм работы программы, в программу были введены некорректные данные или каким-либо образом были изменены внешние обстоятельства (например, файл который вы пытаетесь открыть занят другой программой).

Данные ошибки чаще называют исключениями. Самая распространённая ситуация, порождающая исключения, случается тогда, когда вы пытаетесь передавать число туда, где требуется строка, например, число 15 в функцию print (). Дело в том, что функция print () в качестве аргументов принимает только строчные значения и если в неё в качестве аргумента передаётся число - она порождает исключение (или ошибку, если вам так больше нравится).

Именно поэтому я, стараясь предвосхитить возможную ошибку делаю так:

```
user = {"name": "Mark", "phone": "+79507864598", "age": 25}

for key in user:
    print(key + ": " + str(user[key]))
```

Выводит:

> name: Mark phone: +79507864598 age: 25

Обратите внимание на эту конструкцию: `str(user[key])`.

Я, заранее зная, что в словаре есть число принудительно привожу все значения к строке, чтобы в функцию `print()` не попало число.

Но что делать, если я не знаю, будет ли в качестве значения число, объект или массив? Допустим, данные мне приходят по сети, и порой, они возвращаются с самыми неподходящими для функции `print()` значениями. По сети я получил 100 словарей, которые мне нужно вывести в консоль, но в 7 словаре в поле phone лежит недопустимое значение, которое не принимает функция print, что будет тогда?

Программа прекратит работу. Она выведет в терминал шесть первых словарей, а на седьмом прекратит работу и остальные 93 словаря так и не будут выведены в терминал. Что с этим делать?

Обработка исключений

Как я уже говорил, функция print порождает исключения, если тип аргумента, передаваемого в неё не является строкой. Но в Python, как и в других языках программирования есть конструкция для обработки исключений.

Конструкция `try..except`. Если мы знаем, что функция порождает исключения (выбрасывает ошибку) при определённых обстоятельствах, мы можем поместить её в конструкцию `try..except`, в таком случае выброшенная ошибка не приведёт к прекращению работы программы, и более того, позволит узнать в чём заключается ошибка и как её обработать.

Давайте сперва запустим наш код, без приведения числа к строке и посмотрим, что из этого выйдет:

```
user = {"name": "Mark", "phone": "+79507864598", "age": 25}

for key in user:
    print(key + ": " + user[key])

print("Конец программы")
```

Выводит:

> name: Mark phone: +79507864598 TypeError: can only concatenate str (not "int") to str

Возникло исключение типа `TypeError`, значение поля age не было выведено и программа сразу завершила работу, не дойдя до инструкции: `print("Конец программы")`

Давайте сделаем вид, что мы не знаем, что в словаре могут попадаться числа и попробуем избежать ошибки другим путём, с помощью конструкции `try..except`:

```
user = {"name": "Mark", "phone": "+79507864598", "age": 25}
```

```
for key in user:
    try:
        print(key + ": " + user[key])
    except TypeError as exception:
        print(key + ": ", exception)


print("Конец программы")
```

Выводит:

> name: Mark phone: +79507864598 age: can only concatenate str (not "int") to str Конец программы

Теперь никакой ошибки не появилось, более того в терминал вывело сообщение о том, где находится проблема и программа не прервала свою деятельность.

Как это работает?

Код, который потенциально может порождать исключение мы помещаем в блок `try`, это гарантирует нам, что если произойдёт ошибка в коде, то программа не завершится с ошибкой. Ошибка либо просто будет проигнорирована, либо обработана так или иначе в блоке `except`.

В блок except помещается логика обработки ошибки, то есть те действия, которые необходимо произвести в случае если возникает какая-то конкретная ошибка, например, сохранить сообщение об ошибке в лог или отправить запись в базу данных и т.д.

Если вам неизвестно заранее, какой тип ошибки может выбросить функция, можно использовать блок `except` без типа ошибки:

```
user = {"name": "Mark", "phone": "+79507864598", "age": 25}

for key in user:
    try:
        print(key + ": " + user[key])
    except:
        print(key + ": ", "Ошибка!")


print("Конец программы")
```

Выводит:

> name: Mark phone: +79507864598 age: Ошибка! Конец программы

Так можно делать, когда допустимо просто проигнорировать ошибку и продолжить выполнение программы без её обработки.

Тип исключения

Исключения всегда имеют какой-либо тип. Чтобы обработать исключение определённого типа, нужно в блоке except указать его тип, и объявить через ключевое слово as для него переменную-псевдоним, имя псевдонима может быть любым и подчиняется правилам именования переменных:

```
except TypeError as err:
    print(key + ": ", err)
```

В псевдоним помещается объект ошибки, что позволяет нам получить текст ошибки.

Несколько блоков except

Порой, в зависимости от ситуации одна и та же конструкция языка или функция может выбрасывать разные исключения, в таком случае, для их обработки можно объявить несколько блоков `except`:

```
try:
    make_america_great_again()
except ImpeachmentError as ex:
    print(ex)
except HillaryError as ex:
    print(ex)
except ImmigrantsError as ex:
    print(ex)
```

Как видите, одно действие может порождать множество разных ошибок, каждую из которых можно обработать при помощи объявления нескольких блоков `except`.

Блок finally

Блок finally – необязателен, смысл этого блока заключается в том, что он срабатывает в любом случае, даже если в блоке try не было выброшено исключения и не сработал не один блок except. Подобный блок нужен для того, чтобы выполнять некоторую работу, которая должна быть выполнена в любом случае, вне зависимости от того, было выброшено исключение или нет.

Данный блок часто используют для закрытия соединений с разного рода ресурсами. Например, для закрытия соединения с сетью, файлом, базой данных или другой программы:

```
user = {"name": "Mark", "phone": "+79507864598", "age": 25}

for key in user:
    try:
        print(key + ": " + user[key])
    except:
        print(key + ": ", "Ошибка!")
    finally:
        print("Этот блок сработает в любом случае")

print("Конец программы")
```

Выводит:

> name: Mark Этот блок сработает в любом случае phone: +79507864598 Этот блок сработает в любом случае age: Ошибка! Этот блок сработает в любом случае Конец программы

Обработка ошибок в реальной жизни

Хоть приведённый выше пример действительно мог произойти в реальном проекте, давайте рассмотрим процесс обработки ошибок на более реалистичном примере.

Допустим, мы работаем с библиотекой (готовым модулем от стороннего разработчика) для работы с ВК. У этой библиотеки есть функция `auth`, которая принимает логин и пароль, авторизует пользователя в ВК и возвращает уникальный ключ, который необходим для взаимодействия с ВК.

А что будет, если вы передадите функции неправильную пару логин/пароль? Разумеется, авторизация не будет пройдена, и ВК вернёт код `401`. В документации к функции `auth` мы прочитали, что в таком случае будет порождено (или «выброшено») исключение под названием `AuthFailed`.

Зная это мы думаем, ага.. У меня есть файл на 400 аккаунтов, каждый из которых мне нуно авторизовать и записать уникальный ключ в новый файл. Но если какая-то пара логин/пароль из 400 будет невалидна, вся программа прекратит работу.

Значит нужно воспользоваться конструкцией `try..except`. Но как я буду обрабатывать ошибку? Я знаю, что ошибка может возникнуть в том случае, если авторизация будет неудачна, но возможны и другие необрабатываемые ошибки:

```
# Открываем файл с аккаунтами
with open("path/to/file.txt") as f:
    # Разделяем файл на строки, каждую строку добавляем в список
    accounts = [account.rstrip('\n') for account in f]

for account in accounts:
    # Каждый аккаунт представлен в виде строки login:password
    # Разбиваем строку login:password на словарь со значениями
    # login и password и кладём их в переменные
    login, password = account.rstrip(':')
    try:
        # Получаем авторизационный ключ
        token = vk_api.auth(login, password)
        # Сохраняем этот ключ в "успешный" файл
        # login:password:token
        with open("path/to/success_file.txt") as f:
            f.write(login + ":" + password + ":" + token + "\n")
    except AuthFailed as ex:
        # Если авторизация не удалась - сохраняем аккаунт
        # в "неуспешный" файл
        with open("path/to/lose_file.txt") as f:
            f.write(login + ":" + password + ":" + ex + "\n")
    except:
        # Если произошла любая другая ошибка,
        # также сохраняем аккайнт в неуспешный файл
        with open("path/to/lose_file.txt") as f:
            f.write(login + ":" + password + ":" + "Неизвестная ошибка!" + "\n")
```

Что здесь происходит?

Мы открываем файл с аккаунтами, где на каждой новой строке хранится значение по типу «login:password»
Разбиваем файл по строкам и кладём каждую строку в список
Проходим список в цикле, на каждой итерации получаем новую строку с логином и паролем «login:password»
Разбиваем каждую строку по символу разделения «:» в две переменные: login и password
Далее, передаём логин и пароль в функцию для авторизации, но так как мы знаем, что данная функция может породить ошибку AuthFailed, мы помещаем её вызов в блок try.
Далее, мы получаем авторизационный токен и помещаем его в файл с успешно авторизованными аккаунтами.
Создаём блок обработчик except для ошибки AuthFailed. Если при авторизации функция вернула эту ошибку, значит мы точно знаем, что с парой логин/пароль что-то не так, а значит записываем эту пару в файл с неуспешно авторизованными аккаунтами.
Создаём общий блок except который сработает в случае любой другой ошибки и также, в случае его срабатывания сохраняем аккаунт в файл с неуспешно авторизованными аккаунтами.

В данном коде есть моменты, которые мы ещё не проходили, советую не обращать внимание на детали реализации, а лишь на общий смысл работы кода.

Использование конструкции try..except очень сильно обезопасило наш код, теперь помимо того, что наш код не прекратит выполнение из-за некорректных данных, так он ещё и отсортирует успешно проработанные аккаунты от неуспешных.

И впоследствии это даст нам возможность понять, что не так стало с неуспешно авторизованными аккаунтами.

Удобно, не так ли?

Расширенная работа со строками

Содержание урока

Форматирование строк

Порой приходится собирать строку из разных частей Например, у нас есть некоторый статичный кусок текста, в который нужно вставить меняющиеся данные Мы можем реализовать это при помощи конкатенации (объединения строк)

```
ref_id = "2194"
link = "http://partner-cash.nu/ref/"
ref_link = link + ref_id

print(ref_link)
```

Выводит:

> http://partner-cash.nu/ref/2194

Форматирование через псевдонимы

А можем при помощи более удобного механизма, именуемого форматирования при помощи функции `format`:

```
ref_link = "http://partner-cash.nu/ref/{ref_id}".format(ref_id=2194)
print(ref_link)
```

Выводит:

> http://partner-cash.nu/ref/2194

Для того, чтобы функция `format` собирала строку из нужных частей, в самой строке, в фигурных скобках необходимо указать произвольное название элемента для замены (например, `{ref_id}`), а затем, в функцию `format` передать это значение в виде именованного аргумента (например, `ref_id=2194`).

Кажется, что в сравнении с конкатенацией разницы нет, но задумайтесь, а если вы форматируете большую строку, в которую нужно вставить 23 разных замены? При форматировании вы явно сможете указать, присвоить имя элементу в тексте, на место которого будет присвоено это значение. А в самой функции вы сможете явно задать на что будет этот элемент текста заменён.

Форматирование через индексы

Функции format не обязательно передавать именованные параметры, она может делать замену по порядковому номеру, вот так:

```
ref_link = "http://partner-cash.nu/ref/{0}".format(2194)
print(ref_link)
```

Выводит:

> http://partner-cash.nu/ref/2194

Или так:

```
ref_link = "http://partner-cash.nu/{0}/{1}".format("mark", 2194)
print(ref_link)
```

Выводит:

> http://partner-cash.nu/Mark/2194

А ещё можно продублировать одну позицию:

```
ref_link = "http://partner-{0}.nu/{0}/{0}".format("mark")
print(ref_link)
```

Выводит:

> http://partner-mark.nu/mark/mark

Форматирование через подстановку переменных

Вы также можете подставлять переменные прямо внутрь строк, для этого перед строкой необходимо указать символ `f`, а в том месте строки, в которое необходимо поместить значение переменной указываем фигурные скобки и внутри них пишем имя переменной:

```
name = "Mark"
words = f"Hello, my name is {name}"
print(words)
```

Выводит:

> Hello, my name is Mark

Функции для работы со строками

Объединение строк. Функция `join`

Объединяет символы строки или элементы последовательностей (например, словарей) между собой, при помощи заданной последовательности символов:

```
word = "Строка"
glue = " / "
join_word = glue.join(word)
print(join_word)
```

Выводит:

> С / т / р / о / к / а

То есть, он как-бы «склеивает» элементы последовательности между собой, при помощи некоторой заданной строки.

Склейка списка:

```
words = ["Раз", "Два", "Три", "Четыре", "Пять"]
glue = " / "
join_word = glue.join(words)
print(join_word)
```

Выводит:

> Раз / Два / Три / Четыре / Пять

Склеили все элементы списка в строку.

Разделение строк. Функция split

Если функция `join` объединяла строки по символам, то функция `split` разделяет строки по символам, например, у нас есть строка:

```
words = "Lorem ipsum dolor sit amet consectetur adipiscing elit"
```

И мы хотим разделить эту строку на отдельные слова и положить их в список. Мы можем сделать это при помощи функции `split`, разбив строку по символу пробела:

```
words = "Lorem ipsum dolor sit amet consectetur adipiscing elit"
words_list = words.split(" ")
print(words_list)
```

Выводит:

> ['Lorem', 'ipsum', 'dolor', 'sit', 'amet', 'consectetur', 'adipiscing', 'elit']

Также можно указать, на сколько максимум частей нужно разделить заданную строку, если нам нужно, например, только первых 3 элемента, то мы можем указать так:

```
words = "Lorem ipsum dolor sit amet consectetur adipiscing elit"
words_list = words.split(" ", 3)
print(words_list)
```

Выводит:

> ['Lorem', 'ipsum', 'dolor', 'sit amet consectetur adipiscing elit']

Первых три элемента были разбиты по символу пробела, остальные – нет.

Проверяем, начинается ли строка с подстроки. Функция startswith

Возвращается `True`, если строка начинается с заданной подстроки:

```
num = "https://vk.com/id329483294"
is_vk_link = num.startswith("https://vk.com/id")
print(is_vk_link)
```

Выводит:

> True

Проверяем, заканчивается ли строка подстрокой. Функция endswith

Возвращается `True`, если строка заканчивается заданной подстрокой:

```
num = "https://vk.com"
is_link = num.endswith(".com")
print(is_link)
```

Выводит:

> True

Замена подстроки в строке. Функция replace

Позволяет найти в тексте заданное строчное значение и заменить её на любое другое строчное значение:

```
id = "https://vk.com/id13545642"
new_id = id.replace("https://vk.com/id", "")
print(new_id)
```

Выведет:

> 13545642

Вместо пустой строки можно подставить любую строку:

```
id = "https://vk.com/id13545642"
new_id = id.replace("https://vk.com/id", "Стоимость единорога в рублях: ")
print(new_id)
```

Выводит:

> Стоимость единорога в рублях: 13545642

Также, можно ограничить количество замен в строке. Допустим в тексте искомое строчное значение входит в строку несколько раз, а мы хотим заменить только часть вхождений, для этого третьим аргументом можно передать количество замен:

```
story = "- Гуси-Гуси? \n-Га-га-га!"
dystopia = story.replace("Гуси", "Человеко", 1)
print(dystopia)
```

Выводит:

> - Человеко-Гуси? -Га-га-га!

Заменилось только первое вхождение слова «Гуси», так как мы передали количество замен равное единице.

Удаление начальных и конечных пробелов. Функции: strip, lstrip, rstrip

Функция `strip`. Удаляет пробелы в начале и в конце строки:

```
words = " Слово с пробелами ".strip()
result = "\"{words}\"".format(words=words)
print(result)
```

Выводит:

> "Слово с пробелами"

Пробелы в начале и в конце строки были удалены.

Функция `lstrip`. Удаляет пробелы в начале строки:

```
words = " Слово с пробелами ".lstrip()
result = "\"{words}\"".format(words=words)
print(result)
```

Выводит:

> "Слово с пробелами "

Пробел в начале строки был удалён.

Функция `rstrip`. Удаляет пробелы в конце строки:

```
words = " Слово с пробелами ".rstrip()
result = "\"{words}\"".format(words=words)
print(result)
```

Выводит:

> " Слово с пробелами"

Пробел в конце строки был удалён.

Перевод первой буквы строки в верхний регистр. Функция capitalize

```
word = "предложение с большой буквы"
result = word.capitalize()
print(result)
```

Выводит:

> Предложение с большой буквы

Перевод первого символа всех слов строки в верхний регистр. Функция title

Если вы хотите, чтобы каждое слово в вашей строке начиналось с большой буквы – используйте функцию `title`:

```
word = "предложение с большой буквы"
result = word.title()
print(result)
```

Выводит:

> Предложение С Большой Буквы

Является ли строчное значение числом. Функция isnumeric

Узнать находится ли в вашей строке число в строчном представлении:

```
num = "100"
result = num.isnumeric()
print(result)
```

Выводит:

> True

```
num = "100ss"
result = num.isnumeric()
print(result)
```

Выводит:

> False

Файловая система. Модуль работы с OS

Содержание урока

Управление файлами и папками является очень полезной возможностью. Например, вы пишите парсер, который сохраняет фотографии пользователей на вашем диске, и удобно было бы для каждого пользователя сохранять папку, название которой соответствует id этого пользователя, а внутрь неё сохранять фотографии.

Разумеется, создавать папки необходимо на лету, для этого у нас есть модуль OS и ряд функций для работы с файлами и папками. По правде говоря, у данного модуля намного больше функций, но в контексте данного урока мы рассмотрим только четыре:

Функция remove – удалить файл
Функция rename – переименовать файл
Функция rmdir – удалить папку
Функция mkdir – создать папку

Для того, чтобы воспользоваться перечисленными функциями необходимо импортировать модуль OS:

```
import os
```

Функция remove

Для того, чтобы удалить файл при помощи функции remove достаточно передать в неё путь до этого файла:

```
os.remove("deleted_file.txt")
```

Если файл который вы передаёте в функцию не существует – выбрасывается исключение `FileNotFoundError`.

Функция rename

Для того, чтобы переименовать файл при помощи функции `rename`, в неё необходимо передать два аргумента: первый – путь до файла, который вы хотите переименовать, второй – путь до файла с новым именем:

```
os.rename("raname_file.txt", "new_raname_file.txt")
```

Если файл который вы передаёте в функцию не существует – выбрасывается исключение `FileNotFoundError`.

Функция rmdir

Для того, чтобы удалить папку при помощи функции `rmdir`, в неё необходимо передать один аргумент – путь до папки, которую вы хотите удалить:

```
os.rmdir("delete_dir")
```

Если папки которую вы передаёте в функцию не существует – выбрасывается исключение `FileNotFoundError`.

Функция mkdir

Для того, чтобы создать папку при помощи функции `mkdir`, в неё необходимо передать один аргумент – путь до папки, которую вы хотите создать:

```
os.mkdir("new_dir")
```

Если вы указывает путь до папки, которая находится внутри несуществующей на данный момент папки – выбрасывается исключение `FileNotFoundError`.

Работа с текстовыми файлами

Содержание урока

Открытие и закрытие файла. Конструкция with

Работа с файлами – это достаточно распространённая деятельность, особенно если ты используешь Python для работы в сфере манимейкинга. Всегда нужно читать, перезаписывать, сохранять какие-то файлы. И вообще активно работать с файловой системой ОС.

С файлами производят два функциональных действия: их читают и в них записывают. Для того, чтобы производить над ними эти действия их нужно открыть, а по окончанию работы с файлом – закрыть.

Открытие файла. Функция open

Открыть файл и получить его дескриптор позволяет функция open:

```
open(путь_к_файлу, режим_открытия)
```

Путь к файлу – строковое значение описывающее путь к заданному файлу, относительный или абсолютный.

Режим открытия – определяет для какой цели открывается файл (прочитать, записать, дописать и т.д.). Режимов открытия файла в Python четыре:

`r (read)`. Открытие для чтения. Если файл не существует, то выбрасывается исключение `FileNotFoundError`.
`w (write)`. Открытие для записи. Если файл не существует, то он создается. Если подобный файл существует, он создаётся заново и затирает все старые данные.
`a (append)`. Открытие для дозаписи. Если файл не существует, то он создается. Если заданный файл существует, то данные дописываются в конец файла.
`b (binary)`. Открытие для работы с бинарными данными. Используется совместно с режимами - w или r.

Пример:

```
file = open("path/to/file.txt", "a")
```

Мы открыли файл для дозаписи. В переменной file находится дескриптор файла, с помощью которого мы можем управлять файлом.

Кодировка файла при открытии

Кодировка как известно – важна, при несоответствии кодировки наши данные ломаются и при записи/чтении мы можем получить не совсем то, что ожидаем. Поэтому при открытии файла я советую сразу установить кодировку, делается это при помощи передачи параметра `encoding` в функцию `open`:

```
with open("путь_к_файлу.txt", "r", encoding="utf-8") as f:
```

Я советую ставить `utf-8`, потому что это одна из самых универсальных кодировок на данный момент.

Закрытие файла. Функция close

После того как мы открыли файл, провели над ним какие-то действия нам обязательно нужно его закрыть, поскольку открытый файл — это ресурс, который занимает вычислительные мощности ПК.

Для этого используется функция `close`:

```
file = open("path/to/file.txt", "a")
# Действия над файлом
file.close()
```

Автоматическое закрытие. Конструкция with

Во время работы с файлом нередко возникают ошибки, порождающие исключения. Как мы знаем, если выбрасывается исключение, программа завершается на том же месте, где это исключение «всплыло».

Соответственно, при выбрасывании исключения файл не будет закрываться, т.к. интерпретатор остановит исполнение скрипта и не дойдёт до инструкции закрытия соединения с файлом.

Мы можем использовать, например, конструкцию `try..except.finally`:

```
try:
    file = open("путь_к_файлу.txt", "w", encoding="utf-8")
    try:
        file.write("Мой новый текст")
    except Exception as error:
        print(error)
    finally:
        print("Закрываем файл")
        file.close()
except FileNotFoundError as error:
     print(error)
```

Проблема решена. Но она типична и для её разрешения в Python есть более эстетичное решение, а именно конструкция `with`:

```
with open("путь_к_файлу", "режим") as переменная_для_файла:
    # инструкции по работе с фалом
```

Суть этой конструкции в том, что при любых условиях, соединение с файлом будет закрыто. Она выполняет всё тот же код, что и конструкция `try..except..finally`, только более лаконично. Давайте перепишем прошлый пример, с конструкцией `with`:

```
with open("путь_к_файлу.txt", "w", encoding="utf-8") as f:
    f.write("Мой новый текст")
```

Гораздо удобнее, верно? Разумеется, данную конструкцию следует использовать вместе с конструкцией `try..except..finally`, так как выбрасываемые в процессе работы ошибки в любом случае необходимо обрабатывать.

В дальнейшем, я буду работать с файлами именно в конструкции `with`.

Чтение текстового файла

Для того, чтобы прочесть текстовой файл, его нужно открыть для чтения:

```
with open("путь_к_файлу.txt", "r", encoding="utf-8") as f:
```

Итак, файл открыт, теперь его можно прочитать. Прочитать файл можно тремя способами:

По одной строке (метод readline)
Прочитать целиком и вернуть в виде одной строки (метод read)
Прочитать все строки в список (метод readlines)

Чтение файла по строке, метод readline

Я создал файлик, в котором разместил цифры от 0 до 4 по одной на каждую строку. Чтобы их считать я могу воспользоваться методом `readline`:

```
with open("путь_к_файлу.txt", "r", encoding="utf-8") as f:
    print(f.readline())
    print(f.readline())
    print(f.readline())
    print(f.readline())
    print(f.readline())
```

Выводит:

> 0 1 2 3 4

Для того, чтобы построчно прочитать весь файл в цикле можно использовать следующую конструкцию:

```
with open("путь_к_файлу.txt", "r", encoding="utf-8") as f:
    for item in f:
        print(item)
```

Выведет:

> 0 1 2 3 4

То есть, при переборе дескриптора файла (переменная `f`) в цикле for, на каждой итерации автоматически запускается функция `readline` и возвращает содержимое строки одну за другой, как значения элемента массива (в переменную `item`).

Чтение всего файла в строку. Функция read

Функция read читает весь файл и возвращает в виде строки:

```
with open("путь_к_файлу.txt", "r", encoding="utf-8") as f:
    data = f.read()
    print(type(data))
    print(data)
```

Выводит:

> 0 1 2 3 4

Построчное чтение файла в список. Функция readlines

Мы сразу можем прочитать все строки файла в список, где каждая строка будет отдельным элементом списка, делается это при помощи функции `readlines`:

```
with open("путь_к_файлу.txt", "r", encoding="utf-8") as f:
    list = f.readlines()
    print(type(list))
    print(list)
```

Выводит:

> ['0\n', '1\n', '2\n', '3\n', '4']

Запись в текстовой файл

Информацию в текстовой файл можно:

Записывать
Дозаписывать

Метод `write`, который реализует процесс записи в файл принимает на вход строку, эта строка и записывается в файл.

Для записи необходимо открыть файл с модификатором `w`. В таком случае, если файл уже существует, его содержимое будет полностью перезаписано:

```
with open("путь_к_файлу.txt", "w", encoding="utf-8") as f:
    f.write("Новая строка")
```

Содержимое файла:

> Новая строка

Для дозаписи необходимо открыть файл с модификатором a. В таком случае, если файл уже существует, то информация будет дописана в конец файла. Отмечу, не на новой строке, а в конце последней:

```
with open("путь_к_файлу.txt", "a", encoding="utf-8") as f:
    f.write("Новая строка")
```

Содержимое файла:

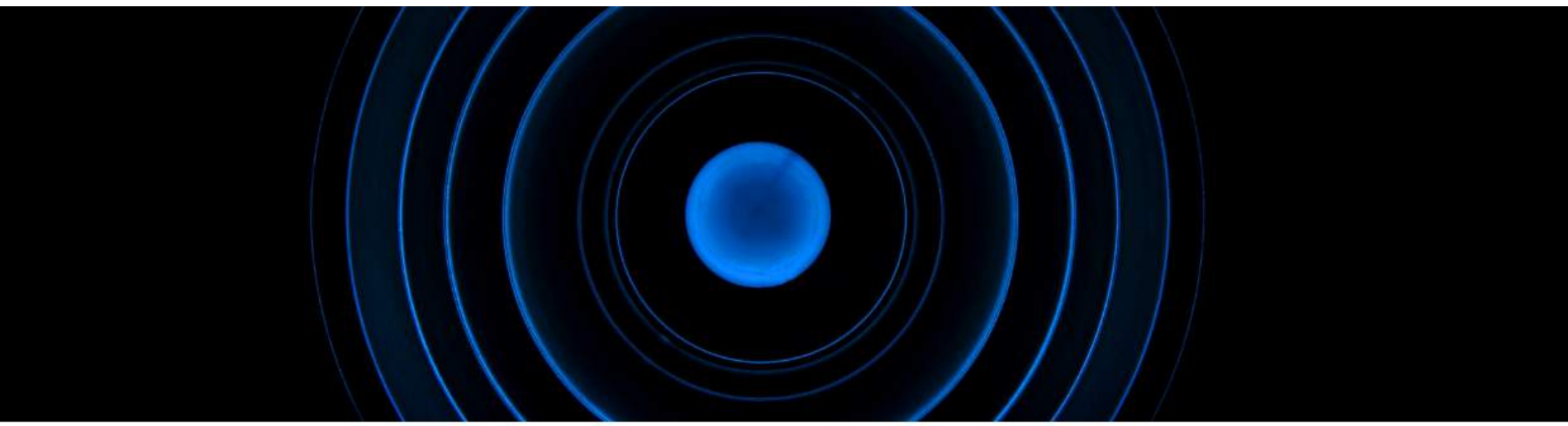
> Новая строкаЕщё одна новая строка

Если вы хотите, чтобы дозапись производилась не в конце последней строки, а на новую строку, то нужно добавить управляющую последовательность переноса строки:

```
with open("путь_к_файлу.txt", "a", encoding="utf-8") as f:
    f.write("\nЗначение с новой строки")
```

Содержимое файла:

> Новая строкаЕщё одна новая строка Значение с новой строки



Работа с CSV-файлами

Содержание урока

Ну и куда же без таблиц. Таблицы, в общем, имеют большое распространение в сфере бизнеса, маркетинга и манимейкинга. Достаточно удобно размещать данные в таблицах, а также анализировать эти данные, поэтому будет полезным разобраться как с ними работать в Python.

Вообще, CSV-формат, это тот же текстовой файл, с которым можно работать как с текстовым файлом, но на наше счастье в Python есть встроенная библиотека по работе с `CSV`, которая значительно упрощает работу с CSV-файлами, она так и называется `csv`.

Для того чтобы с ней работать, нужно её сначала подключить. В самом верху файла, в котором мы будем писать код нужно указать инструкцию импорта этого модуля:

```
import csv
```

Далее, нам нужно открыть наш CSV-файл. Делается это практически также, как и с обычным файлом, за исключением одной особенности, при открытии файла мы также передаём именованный параметр: `newline=""`

Это необходимо для того, чтобы избежать проблем с совместимостью между разными ОС:

```
with open("table.csv", "w", encoding="utf-8", newline="") as csv_file:
```

Запись в CSV-файл

Запись производится при помощи двух функций:

`writerow` – записывает одну строку, в функцию передаётся список со значениями
`writerows` – записывает сразу несколько строк, в функцию передаётся список со вложенными списками

Перед тем, как начать запись, нам нужно запустить специальную функцию `writer`, которая подготовит наш файл, для записи в него CSV-данных:

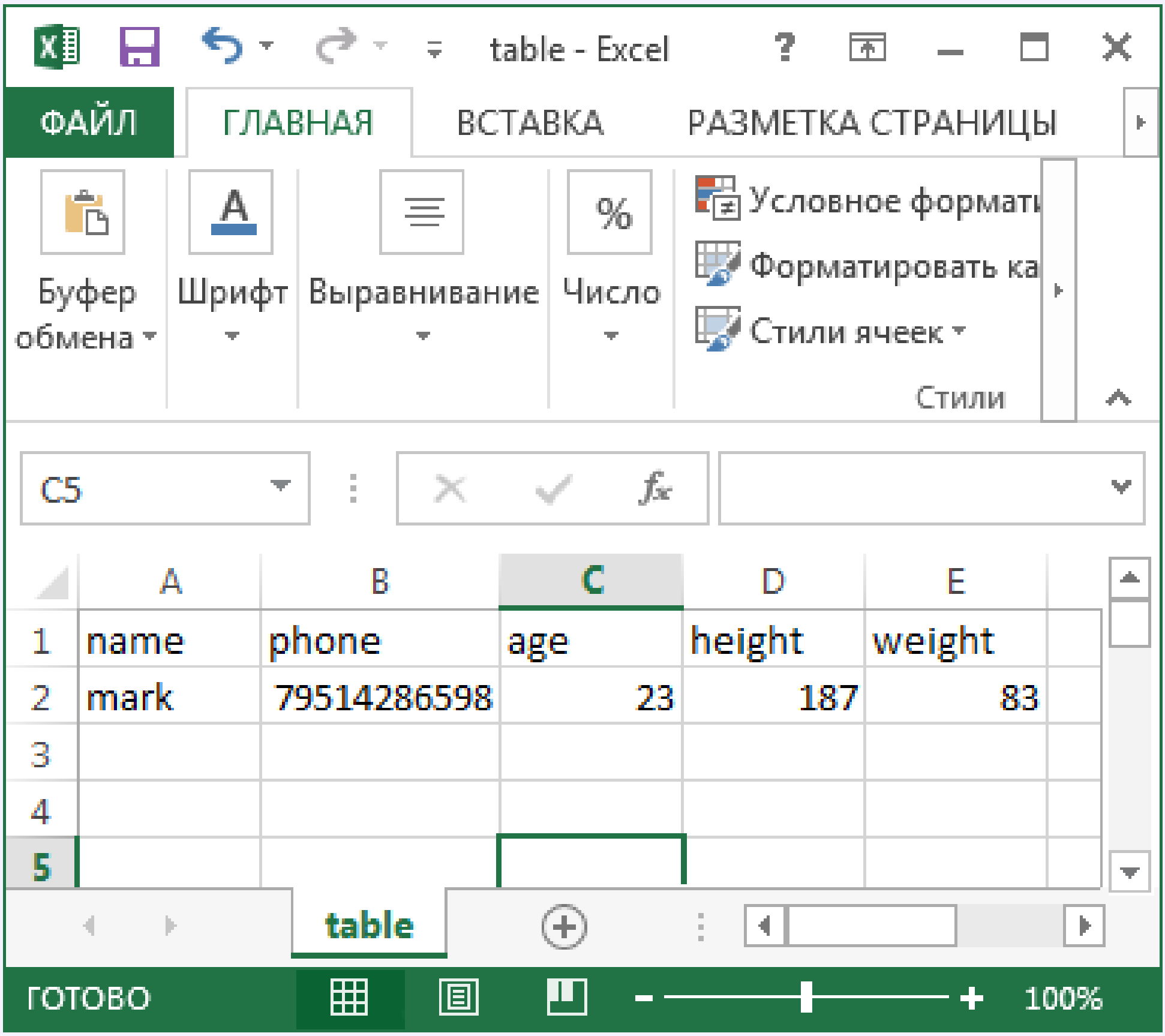
```
writer = csv.writer(csv_file, delimiter=';')
```

В функцию мы передаём первым параметром дескриптор нашего файла, а вторым параметром указываем, каким разделителем будут отделяться столбцы. По умолчанию, разделителем является символ «,», но если мы откроем такой файл в excel то он не увидит таблицы, потому что в excel по умолчанию разделитель это «;», поэтому, если вы хотите чтобы созданные вами CSV файлы открывались в excel, то принудительно указывайте разделитель «;».

Запись через `writerow` добавляет строку в конец таблицы, каждое отдельное значение списка как отдельный столбец:

```
import csv
with open("table.csv", "w", encoding="utf-8", newline="") as csv_file:
    writer = csv.writer(csv_file, delimiter=';')
    writer.writerow(["name", "phone", "age", 'height', "weight"])
    writer.writerow(["mark", "+79514286598", "23", '187', "83"])
```

Откроем получившийся файл:

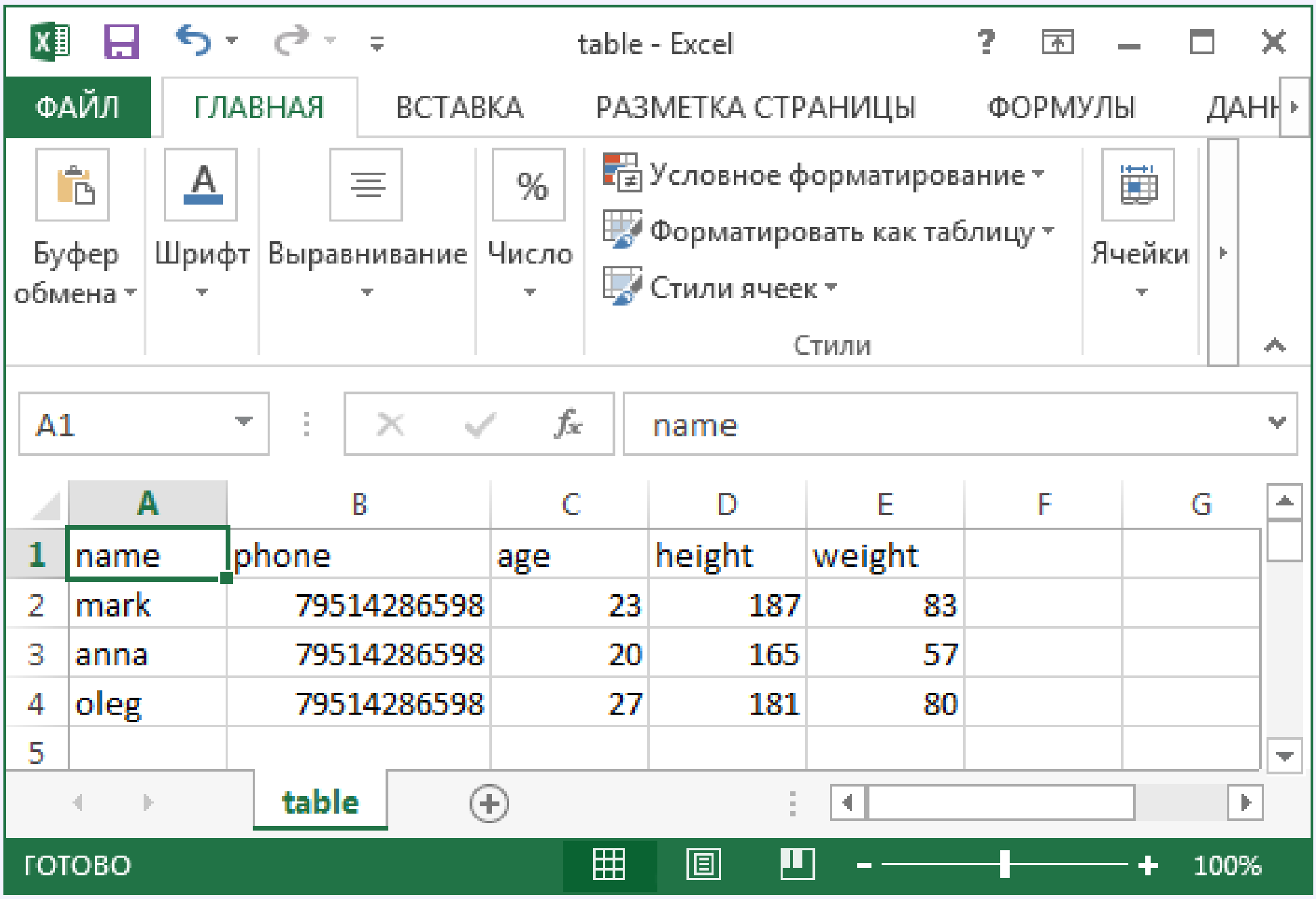


Каждый список полностью вписывается в одну строку, каждое значение в новый столбец.

Запись через `writerows` добавляет сразу несколько строк в конец таблицы, в неё нужно передать список со вложенными списками:

```
import csv

rows = [
    ["name", "phone", "age", 'height', "weight"],
    ["mark", "+79514286598", "23", '187', "83"],
    ["anna", "+79514286598", "20", '165', "57"],
    ["oleg", "+79514286598", "27", '181', "80"],
]

with open("table.csv", "w", encoding="utf-8", newline="") as csv_file:
    writer = csv.writer(csv_file, delimiter=';')
    writer.writerows(rows)
```

Откроем получившийся файл:



Запись через DictWriter

Запись через `DictWriter` позволяет записывать данные в cs-файл из словаря. При этом, процесс записи будет отличаться, т.к. он будет записывать всё содержимое не по строкам, а по столбцам, где столбцом будет являться ключ от значения.

Перед тем, как начать запись, нам нужно запустить специальную функцию `writer`, которая подготовит наш файл, для записи в него CSV-данных:

```
writer = csv.DictWriter(csv_file, fieldnames=columns, delimiter=';')
```

В функцию мы передаём первым параметром дескриптор нашего файла, вторым параметром указываем, какие имена будут у столбцов нашего файла, третьим параметром указываем каким разделителем будут отделяться столбцы.

Для записи через `DictWriter` есть также 2 функции:

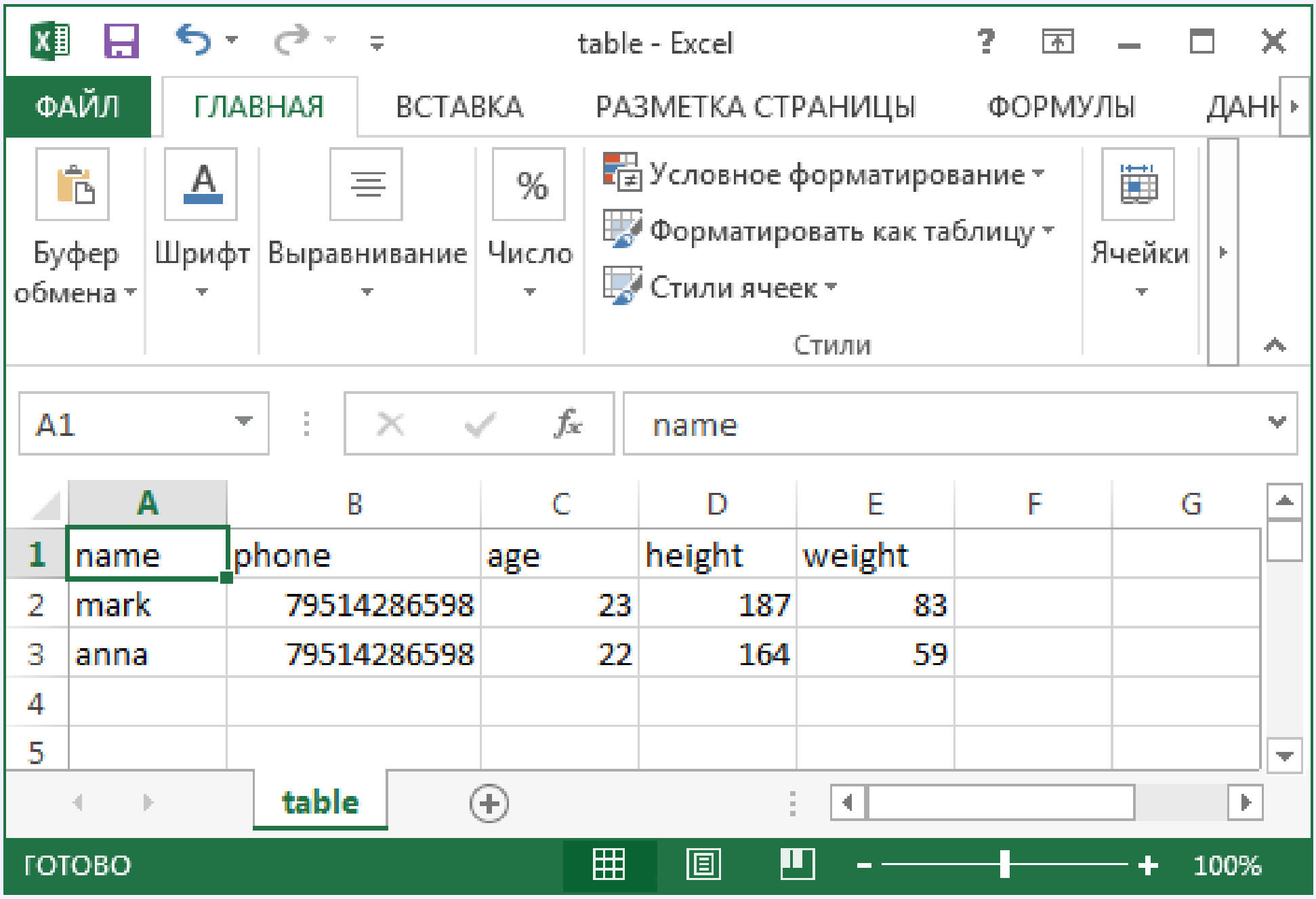
`writerow` – записывает одну строку, в функцию передаётся словарь со значениями
`writerows` – записывает сразу несколько строк, в функцию передаётся список со вложенными словарями

Функция `writerow`:

```
import csv

with open("table.csv", "w", encoding="utf-8", newline="") as csv_file:
    # Определили список заголовков столбцов
    columns = ["name", "phone", "age", 'height', "weight"]
    # Подготовили дескриптор фала для записи CSV
    writer = csv.DictWriter(csv_file, fieldnames=columns, delimiter=';')
```

```
    writer.writeheader()

    # Записываем словарь с данными, где ключ равен заголовку столбца
    writer.writerow({"name": "mark", "phone": "+79514286598", "age": "23", "height": "187", "weight": "83"})
    writer.writerow({"name": "anna", "phone": "+79514286598", "age": "22", "height": "164", "weight": "59"})
```

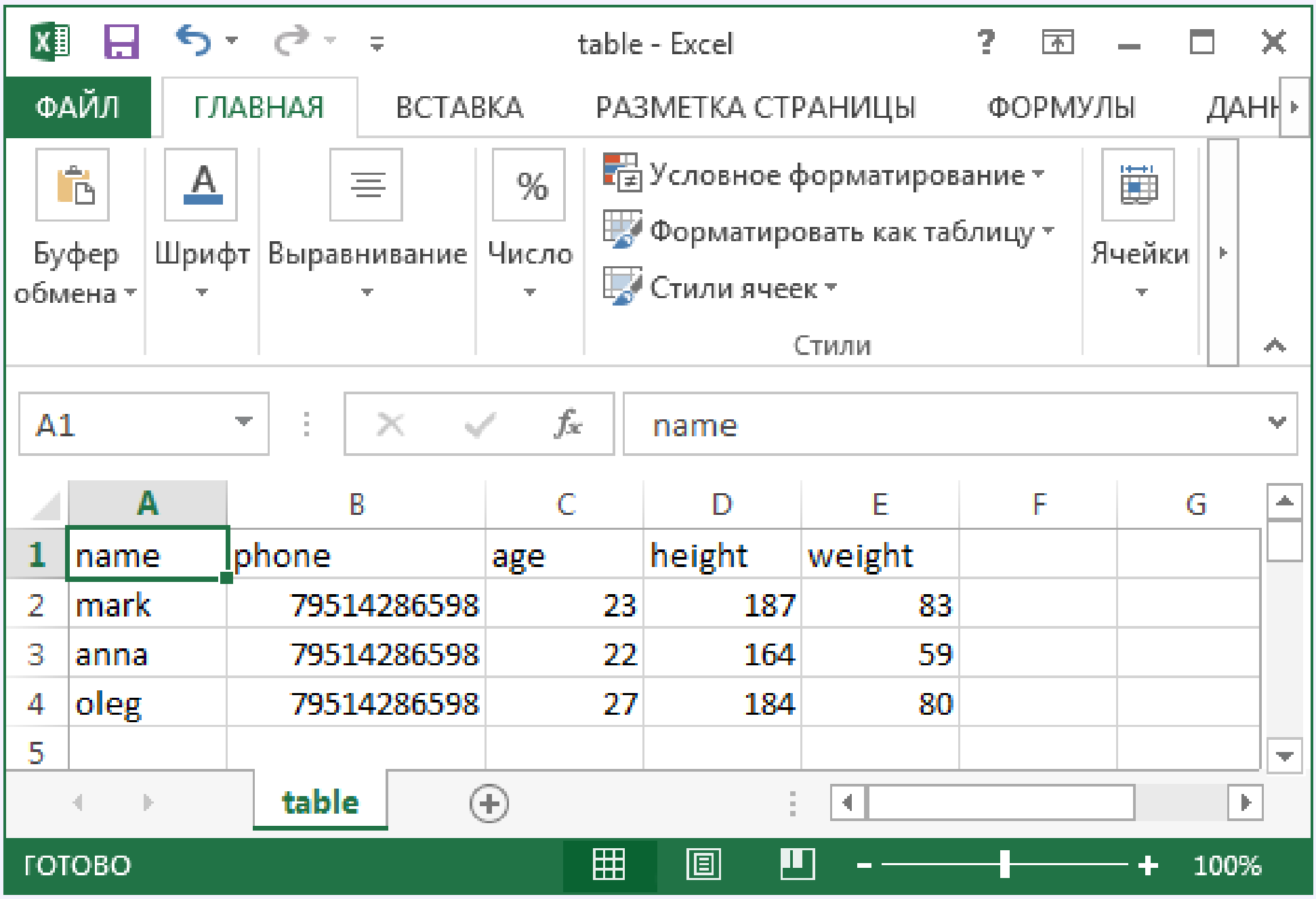
Открываем файл и смотрим, что получилось:



Функция `writerows`:

```
import csv

rows = [
    {"name": "mark", "phone": "+79514286598", "age": "23", "height": "187", "weight": "83"},
    {"name": "anna", "phone": "+79514286598", "age": "22", "height": "164", "weight": "59"},
    {"name": "oleg", "phone": "+79514286598", "age": "27", "height": "184", "weight": "80"},
]

with open("table.csv", "w", encoding="utf-8", newline="") as csv_file:
    # Определили список заголовков столбцов
    columns = ["name", "phone", "age", 'height', "weight"]
    # Подготовили дескриптор фала для записи CSV
    writer = csv.DictWriter(csv_file, fieldnames=columns, delimiter=';')
    writer.writeheader()

    # Записываем список словарей с данным, где ключ равен заголовку столбца
    writer.writerows(rows)
```

Открываем файл и смотрим, что получилось:



Чтение CSV-файлов

Чтение файла производится через функцию `reader`:

```
reader = csv.reader(csv_file, delimiter=';')
```

Если вы записывали csv-файл с разделителем «;», то и при чтении нужно указать этот разделитель.

В переменную reader возвращается список (скорее объект, но он ведёт себя как список) с данными таблицы, каждый элемент – вложенный список со всеми данными строки.

Давайте обойдём этот список через цикл for и посмотрим, что у него внутри:

```
import csv


with open("table.csv", "r", encoding="utf-8", newline="") as csv_file:
    reader = csv.reader(csv_file, delimiter=';')
    for line in reader:
        print(line)
```

Выводит:

> ['name', 'phone', 'age', 'height', 'weight'] ['mark', '+79514286598', '23', '187', '83'] ['anna', '+79514286598', '22', '164', '59'] ['oleg', '+79514286598', '27', '184', '80']

Каждый элемент является вложенным списком, который мы также можем обойти при помощи вложенного цикла:

```
with open("table.csv", "r", encoding="utf-8", newline="") as csv_file:
    reader = csv.reader(csv_file, delimiter=';')
    for line in reader:
        for item in line:
            print(item)
        print("\n")
```

Выводит:

> name phone age height weight mark +79514286598 23 187 83 anna +79514286598 22 164 59 oleg +79514286598 27 184 80

Работа с бинарными файлами

Содержание урока

Понятие бинарных файлов

Бинарные файлы хранят набор байт, а не привычный нам текст. К бинарным файлам относятся изображения, аудио, видео и многие другие файлы. Зачем нам нужны бинарные файлы? Одно слово – `pickle`, это модуль который умеет преобразовывать структуры данных (например, списки, словари или объекты) в файлы содержащие бинарный код, а затем также легко их считывать и преобразовывать обратно в структуры данных.

Так зачем нам это нужно-то? Объясню на примере рассылок ВК. Для авторизации и отправки запросов на сайте ВК нам нужны `cookies` заданного аккаунта, полученные после авторизации. И большинство библиотек для работы с сетью (для отправки запросов на сайты) принимают cookies в виде словаря.

Но не будем же мы проходить авторизацию и загружать в скрипт `cookies` каждый раз, когда он выполняет какие-то действия на сайте ВК. Нет, разумнее один раз сохранить наши cookies в виде словаря, а затем каждый раз брать их и отправлять на сервер.

Но как это сделать, ведь в файл сохраняется лишь текст, а нам нужно где-то сохранить полноценную структуру данных. Именно в этом нам и помогает `pickle`, он упаковывает сложные структуры данных в бинарный файл, а затем полностью восстанавливает их из файла в первозданном виде.

Модуль `pickle` необходимо импортировать:

```
import pickle
```

У модуля `pickle` есть два метода:

`dump(структура_данных)`: записывает структуру данных в файл в бинарном формате
`load(файл)`: читает данные из файла в структуру данных

Для того, чтобы сохранить или прочитать бинарный файл при помощи pickle сперва необходимо открыть этот файл стандартным методом:

```
import pickle

with open("path/to/file.pkl", "wb") as f:
```

Обращаю внимание, что открытие бинарного файла требует передачи другого режима открытия. Если вы открываете файл для записи, то режим нужно указать как `wb`, если для чтения, то `rb`.

Упаковка структуры данных в бинарный файл

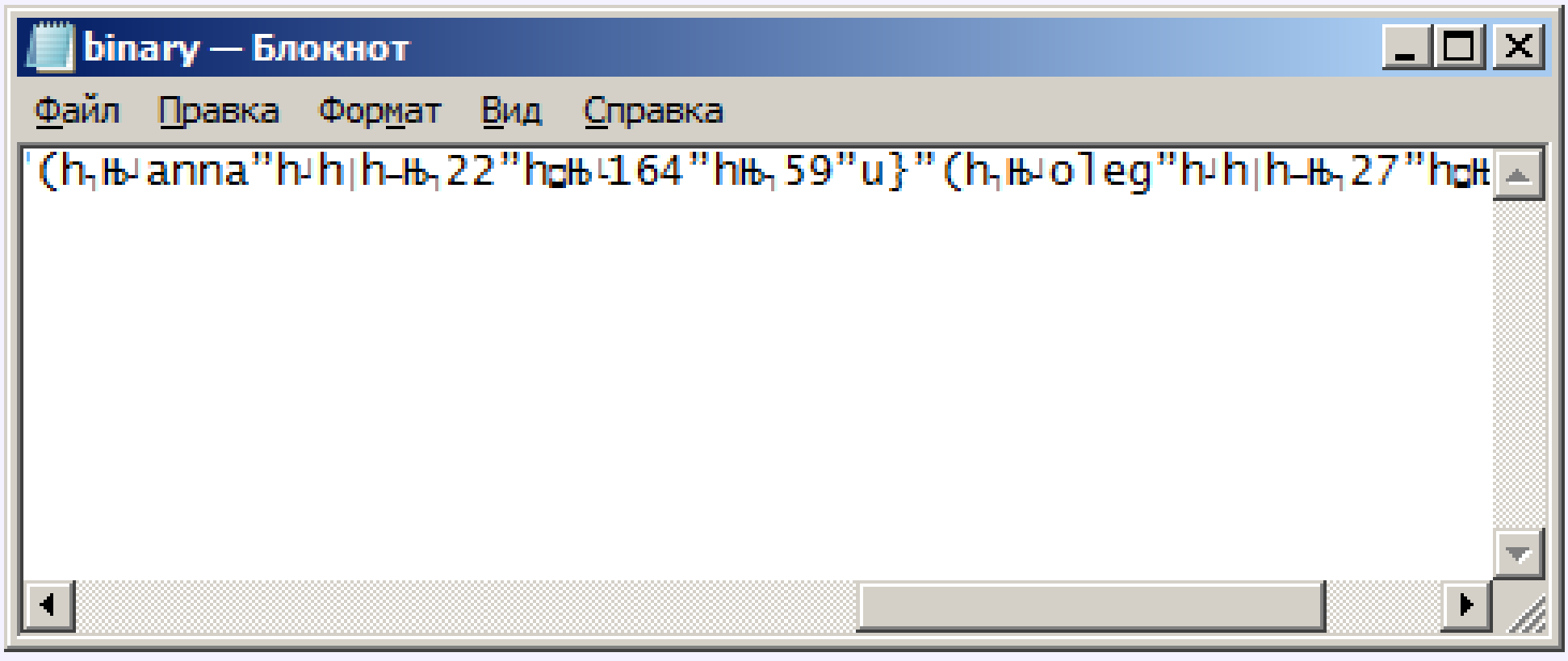
Возьмём структуру данных из прошлого урока:

```
data = [
    {"name": "mark", "phone": "+79514286598", "age": "23", "height": "187", "weight": "83"},
    {"name": "anna", "phone": "+79514286598", "age": "22", "height": "164", "weight": "59"},
    {"name": "oleg", "phone": "+79514286598", "age": "27", "height": "184", "weight": "80"},
]
```

И запишем её в бинарный файл:

```
import pickle

data = [
    {"name": "mark", "phone": "+79514286598", "age": "23", "height": "187", "weight": "83"},
    {"name": "anna", "phone": "+79514286598", "age": "22", "height": "164", "weight": "59"},
    {"name": "oleg", "phone": "+79514286598", "age": "27", "height": "184", "weight": "80"},
]

with open("binary.pkl", "wb") as f:
    pickle.dump(data, f)
```

Откроем созданный нами файл:



Нашу структуру данных упаковали в бинарный файл.

Распаковка структуры данных из бинарного файла

Распаковка из бинарного файла производится через функцию `load`. Сперва мы открываем файл в режиме бинарного чтения (`rb`), затем передаём дескриптор этого файла функции `load` и получаем распакованную структуру данных:

```
with open("binary.pkl", "rb") as f:
    data = pickle.load(f)
    print(type(data))
    print(data)
```

Выводит:

> [{'name': 'mark', 'phone': '+79514286598', 'age': '23', 'height': '187', 'weight': '83'}, {'name': 'anna', 'phone': '+79514286598', 'age': '22', 'height': '164', 'weight': '59'}, {'name': 'oleg', 'phone': '+79514286598', 'age': '27', 'height': '184', 'weight': '80'}]

Мы упаковали список словарей в файл, а затем его же распаковали из файла. И можем пользоваться им также как и раньше.

Случайности в Python. Модуль random

Содержание урока

Очень полезно будет рассмотреть модуль `Random`. Как можно догадаться из названия, это модуль про генерацию случайных значений. Однако, лишь генерацией случайных чисел он не ограничивается, а добавляет ещё ряд достаточно удобных функций, которые можно успешно применять в повседневной деятельности.

У модуля `Random` огромное количество функций всех цветов и расцветок, однако я хочу поделиться самыми распространёнными и удобными.

Случайное число из диапазона. Функция randint

Возвращает случайное число, в промежутке «от-до», принимает два аргумента для задания диапазона, минимальное и максимальное значение:

```
import random
print(random.randint(5, 50))
```

Выводит:

> 34

Выводится случайное число в диапазоне от 5 до 50.

Перемешивание списка. Функция shuffle

В списке все элементы расположены последовательно. Функция shuffle «перемешивает» все значения, переставляя их на другие позиции:

```
import random

first = [1, 2, 3, 4, 5, 6, 7, 8, 9]
random.shuffle(first)
print(first)
```

Выводит:

> [3, 2, 5, 9, 4, 6, 1, 8, 7]

Очень удобная функция, когда нам нужно получить несколько случайных, неповторяющихся элемента списка. Например, мы хотим написать модуль для сайта, который выводит на страницу с записью 5 случайных, не повторяющихся записей из базы данных.

Для этого нам достаточно получить, скажем, двадцать последних записей из БД, затем перемешать их и при помощью обрезки списка (слайсов) извлечь из списка 5 записей:

```
import random

articles = [
    "1 статья",
    "2 статья",
    "3 статья",
    "4 статья",
    "5 статья",
    "6 статья",
    "7 статья",
    "8 статья",
    "9 статья",
    "10 статья",
    "11 статья",
    "12 статья",
    "13 статья",
    "14 статья",
]
random.shuffle(articles)
print(articles[:5:])
```

Выводит:

> ['8 статья', '9 статья', '1 статья', '2 статья', '5 статья']

При повторном запуске:

> ['3 статья', '9 статья', '4 статья', '13 статья', '14 статья']

Случайный элемент списка. Функция choice

Данная функция позволяет вернуть случайный элемент списка:

```
import random

nums = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print(random.choice(nums))
```

Выводит:

> 3

При повторном запуске:

> 1

Случайное число с плавающей точкой от 0.0 до 1.0. Функция random()

Если вам нужно случайное число с плавающей точкой, вы можете воспользоваться функцией `rand`. Она возвращает случайное значение в диапазоне от 0.0 до 1.0.

```
import random
print(random.random())
```

Выводит:

> 0.9475910087761283

Если вам нужно случайное число с плавающей в некотором диапазоне, от 0.0 до n.n. Достаточно просто умножить результат на это число, например:

```
import random
# Диапазон от 0 до 20
print(random.random()*20)
```

Выводит:

> 5.564611877316854

При повторном запуске:

> 17.8048107733193

Расширенные математические операции

Содержание урока

Проведение сложных вычислений в нашей сфере пригождается редко, однако будет не лишним знать несколько самых распространённых математических функций.

Со списком всех функций модуля math, можно ознакомиться по ссылке:
https://docs.python.org/3/library/math.html

Округление числа, функции floor и ceil

Если вам необходимо округлить число с плавающей точкой до целого числа в меньшую сторону, воспользуйтесь функцией floor:

```
print(math.floor(10.70))
```

Выведет:

> 10

Если вам необходимо округлить число с плавающей точкой до целого числа в большую сторону сторону, воспользуйтесь функцией `ceil`:

```
import math

print(math.ceil(10.30))
```

Выведет:

> 11

Извлечение квадратного корня. Функция sqrt

Извлекает квадратный корень из числа:

```
import math

print(math.sqrt(25))
```

Выведет:

> 5.0

Возведение в степень. Функция pow

Если вы хотите возвести число x в степень y, вам поможет функция pow, где первым аргументом передаётся число, которое нужно возмести в степень, а вторым аргументом сама степень:

```
import math

print(math.pow(5, 2))
```

Понятие БД. СУБД MySQL

Содержание урока

Понятие СУБД

Как вы, наверное, знаете, практически все данные в интернете хранятся в базах данных (далее, БД). Что такое база данных простыми словами? БД – это совокупность данных, организованная по определённому принципу и находящаяся в памяти компьютера.

Чтобы пользоваться базой данных, необходима специальная система. Система управления базами данных, или СУБД. СУБД – это совокупность программных и языковых инструментов, направленных на создание БД, управление БД (получить данные, обновить данные, вставить данные и т.д.).

Работа с БД MySQL построена на языке запросов SQL, она наилучшим образом подходит для малых и средних приложений. Таким образом, для хранения и манипуляций данными в БД, мы используем программные инструменты MySQL и язык запросов SQL.

Например, отправка такого запроса:

```
SELECT * FROM article WHERE id = '55'
```

Выведет из таблицы БД с именем `article` все записи, с `id` равным `55`.

Таблицы MySQL

Данные внутри БД MySQL хранятся в таблицах, по форме подобных таблицам `excel`.
Предположим, что мы создаём блог. Внутри блога есть: статьи, комментарии, пользователи.

Всё это – отдельные сущности, которые при этом могут взаимодействовать друг с другом.

Так вот, каждая такая сущность — это отдельная таблица. Т.е. в БД для блога, можно поместить таблицу со статьями, таблицу с комментариями, таблицу с пользователями.

Например, таблица со статьями в MySQL выглядит примерно так:

id	title	content	author
1	Заголовок статьи 1	Текст статьи 1	Автор 1
2	Заголовок статьи 2	Текст статьи 2	Автор 2
3	Заголовок статьи 3	Текст статьи 3	Автор 3
4	Заголовок статьи 4	Текст статьи 4	Автор 4
5	Заголовок статьи 5	Текст статьи 5	Автор 5
6	Заголовок статьи 6	Текст статьи 6	Автор 6
7	Заголовок статьи 7	Текст статьи 7	Автор 7
8	Заголовок статьи 8	Текст статьи 8	Автор 8
9	Заголовок статьи 9	Текст статьи 9	Автор 9
10	Заголовок статьи 10	Текст статьи 10	Автор 10

Это в общем смысле таблица `excel`, но в MySQL таблицы имеют точно такой же вид. Как и обычная таблица, таблица в MySQL состоит из записей:

3	Заголовок статьи 3	Текст статьи 3	Автор 3

И столбцов:

2

3

4

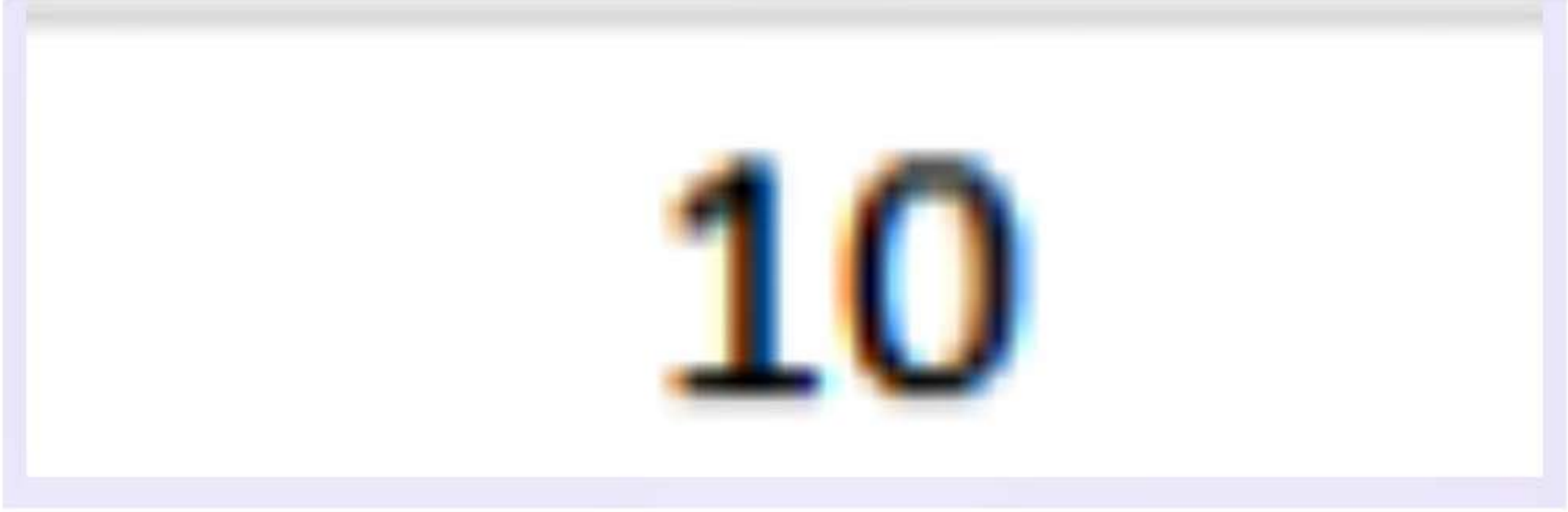
5

6

7

8

9



Каждый столбец имеет своё имя, назначение и тип данных. Например, в таблице нашей БД содержатся следующие столбцы: id, title, content, author, date.

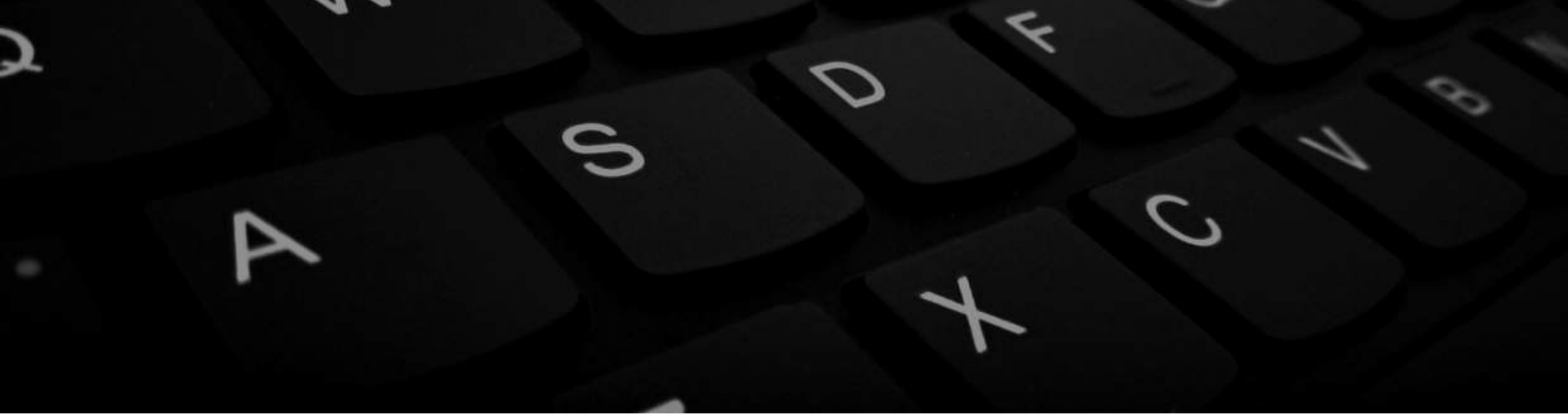
В столбце id хранятся уникальные для каждой отдельной записи идентификаторы, числового типа
В столбце title хранятся заголовки статей, строкового типа
В столбце content хранятся сами статьи
В столбце author хранятся авторы
В столбце date хранятся даты

Каждая запись (строка) состоит из ячеек, содержание которых определяется столбцом, в котором находится эта ячейка.

В нашем случае, запись состоит из 5 ячеек:

id	title	content	author	date
4	Updated Title	Updated Content	555	17 485 484

Ячейка 1 – определяется столбцом id, ячейка 2 – определяется столбцом title, ячейка 3 определяется столбцом content, ячейка 4 определяется столбцом author, ячейка 5 определяется столбцом date.



Устанавливаем БД и осуществляем подключение

Содержание урока

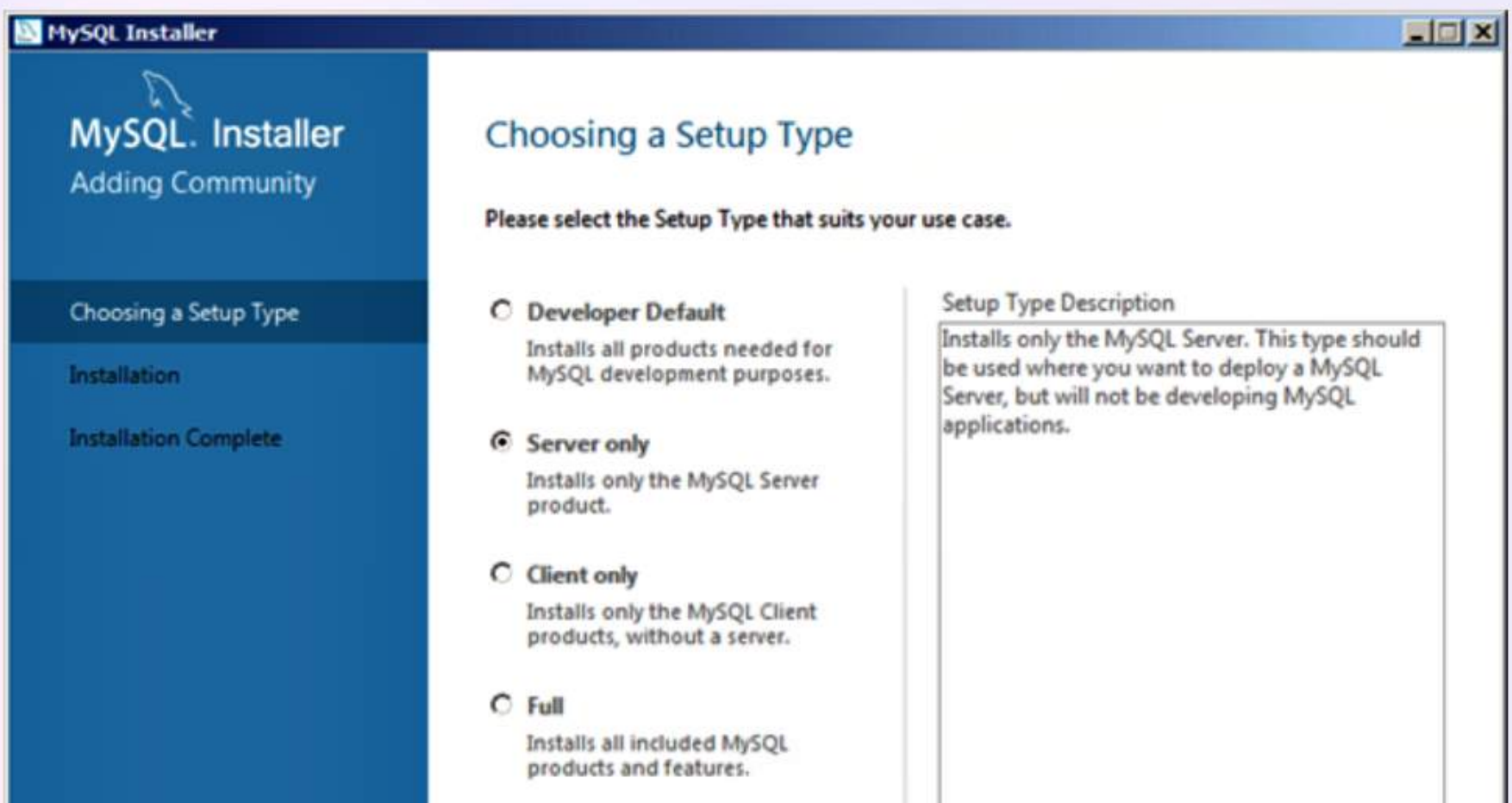
Зачем нам нужна БД и как она выглядит разобрались. Осталось самое простое, установить сервер MySQL, подключить библиотеку для работы Python с MySQL и установить настройки соединения. Приступим.

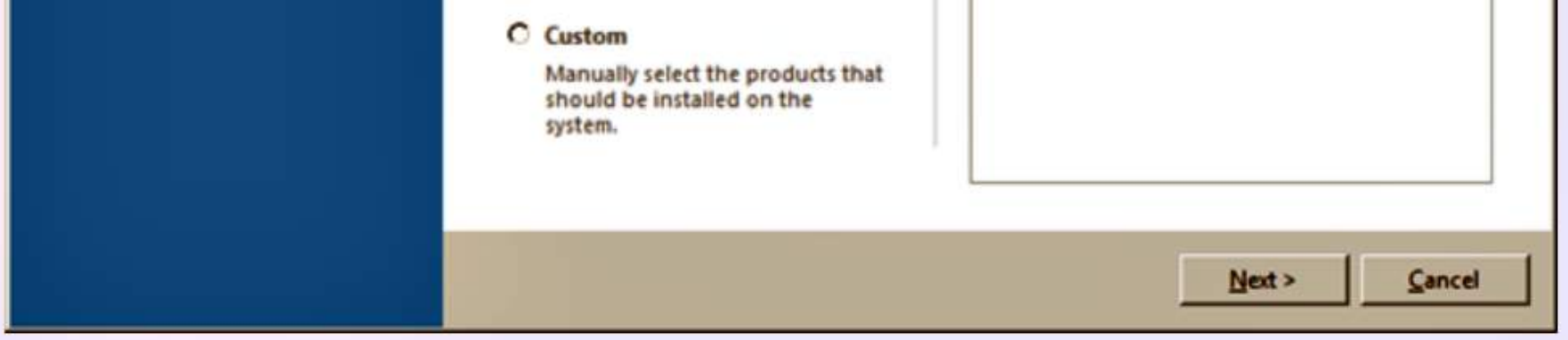
Устанавливаем сервер

Сперва необходимо скачать MySQL-сервер: https://dev.mysql.com/downloads/installer/

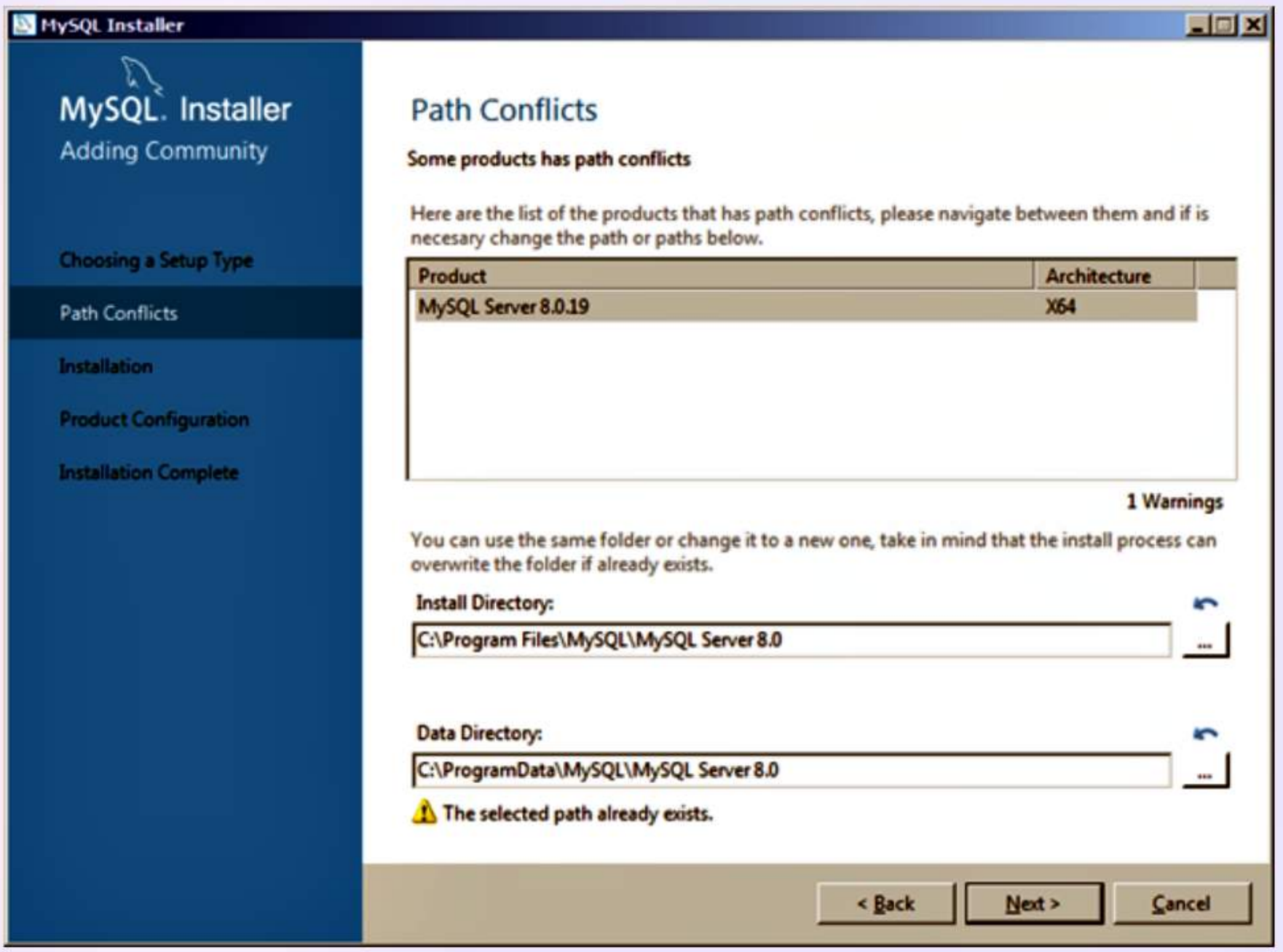
Выбирайте тот вариант, который больше весит.

Запускаем инсталлятор и выбираем пункт «Server only»:

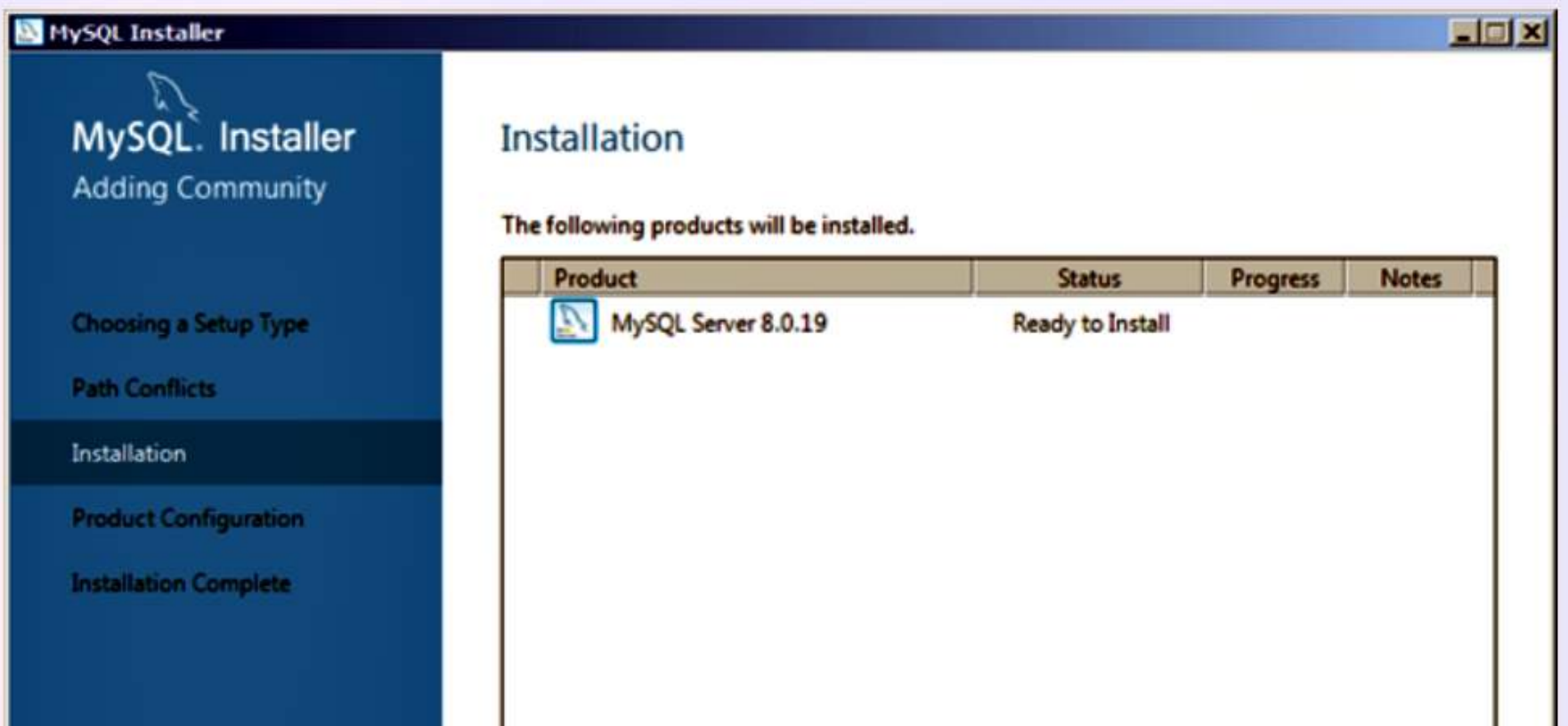


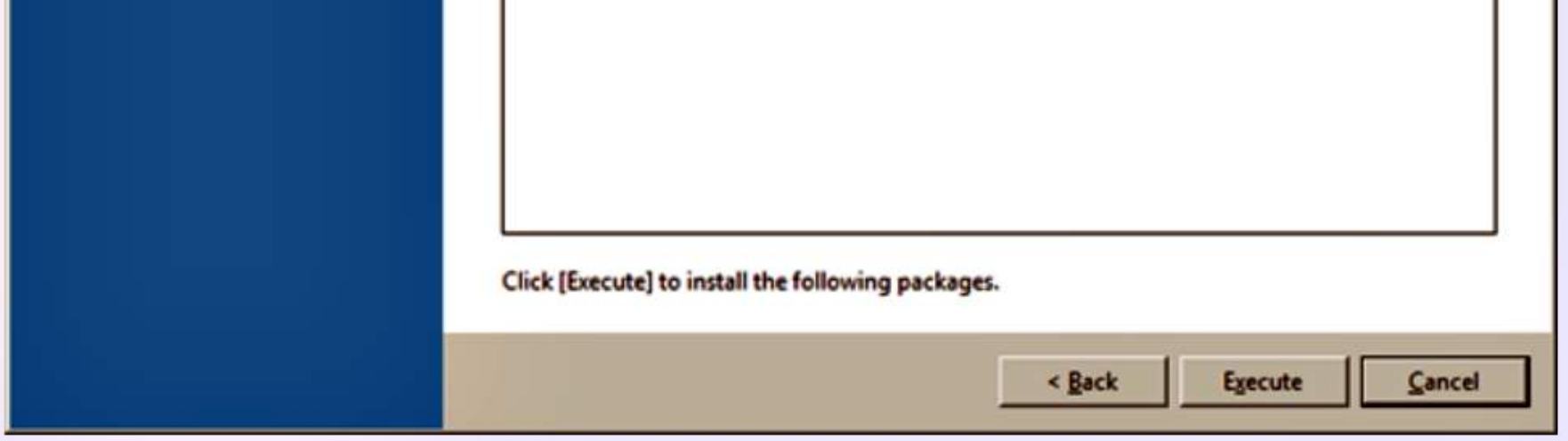


На следующем экране оставляем всё как было:

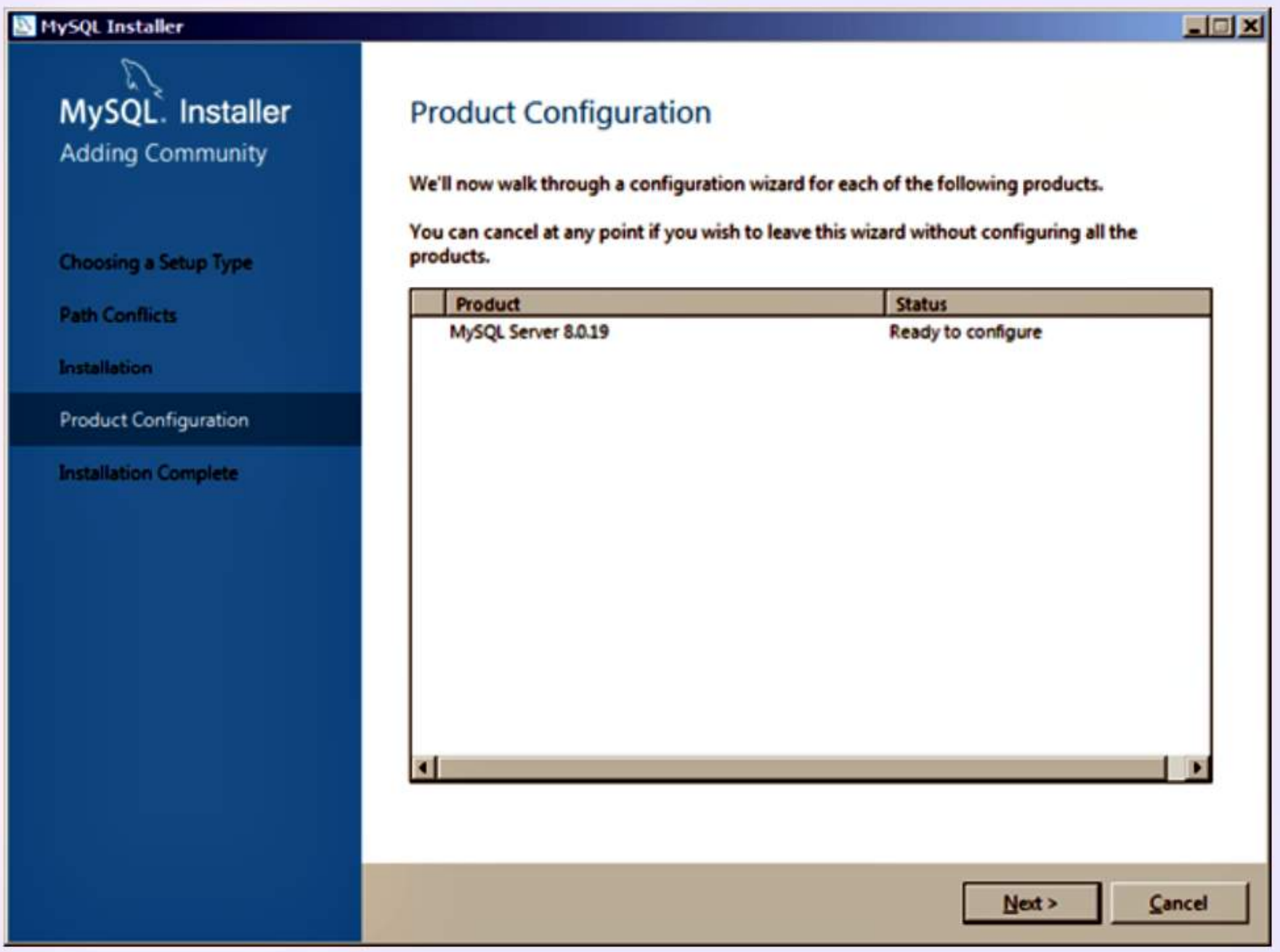


Далее, также оставляем всё как есть:

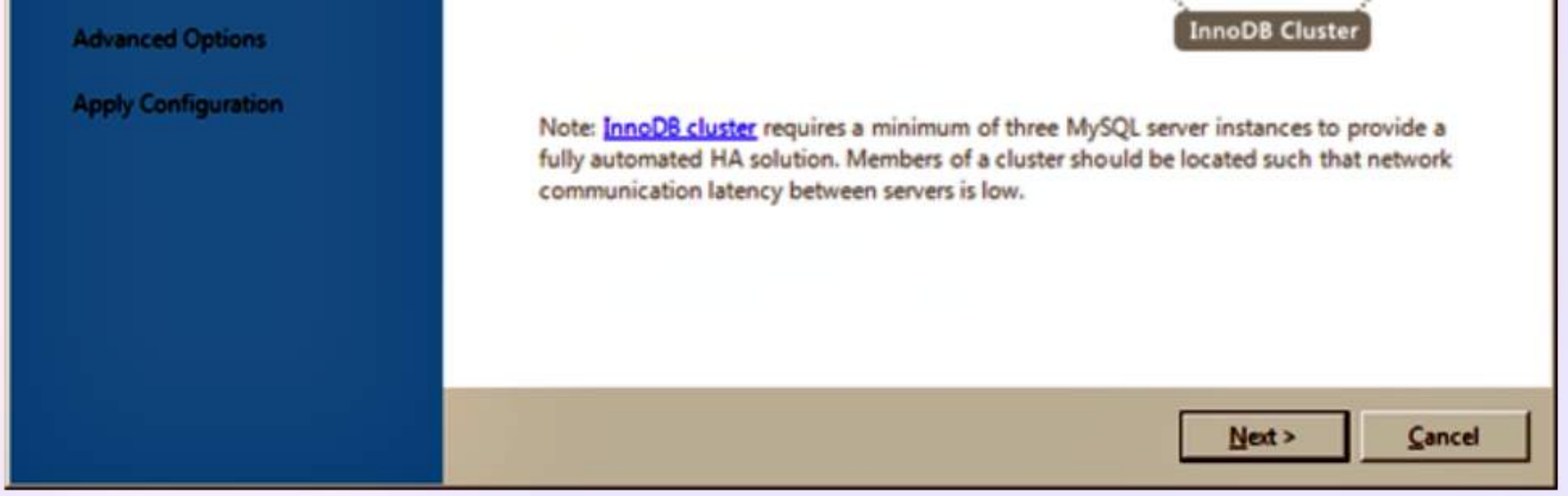




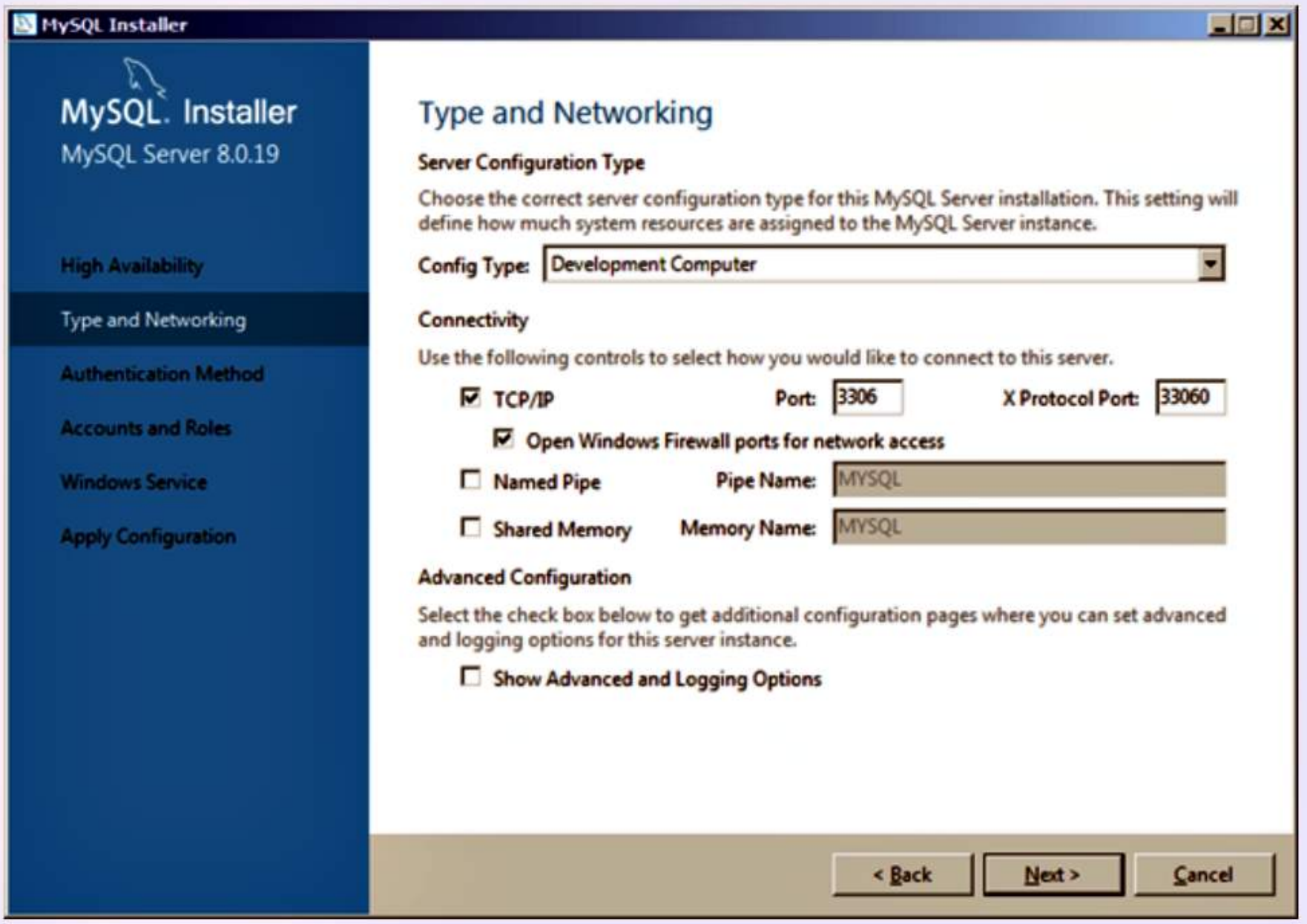
Нажимаем на кнопку «Execute». Затем, «Next»:



Оставляем всё как есть:

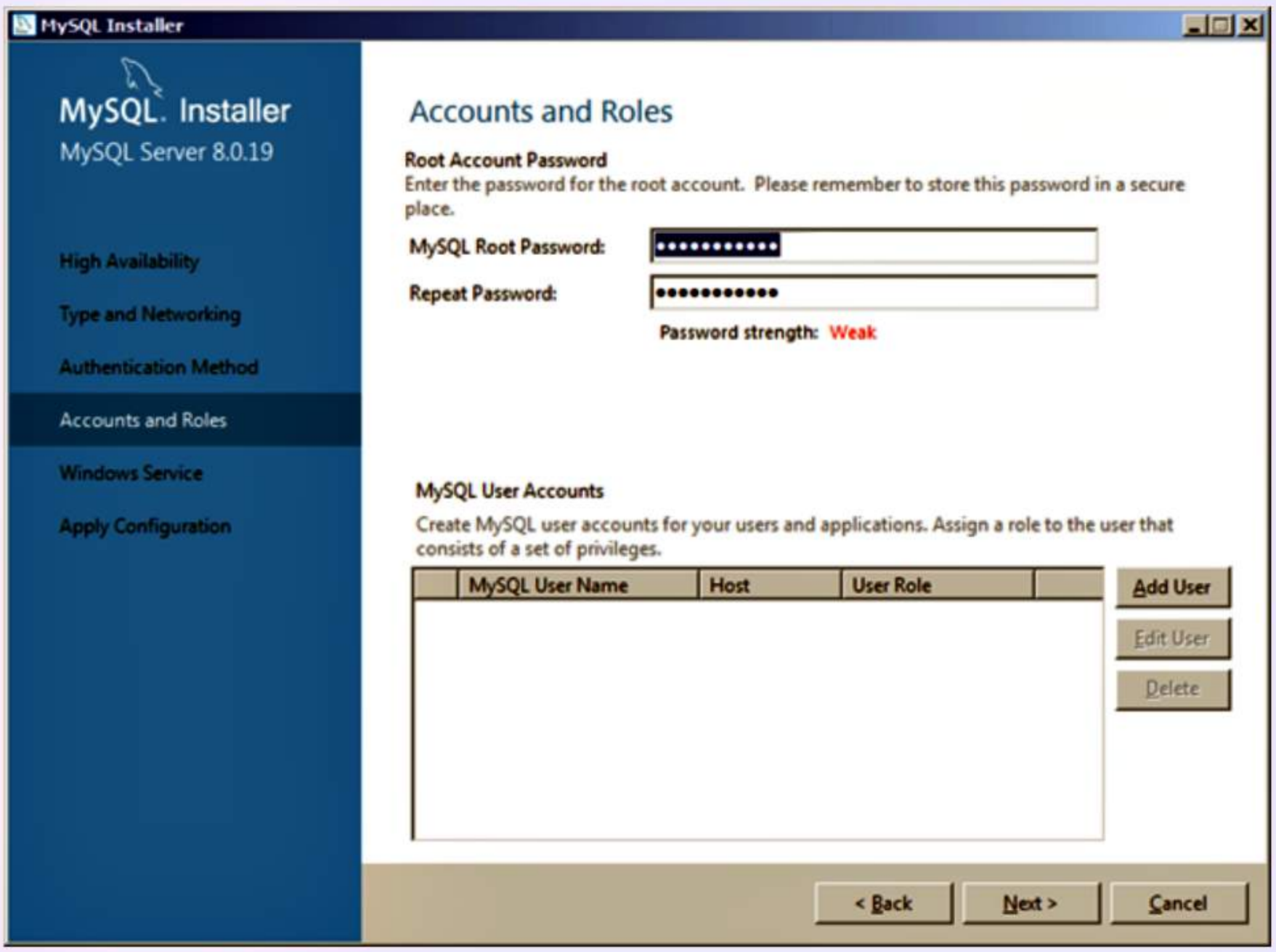


Оставляем всё как есть:

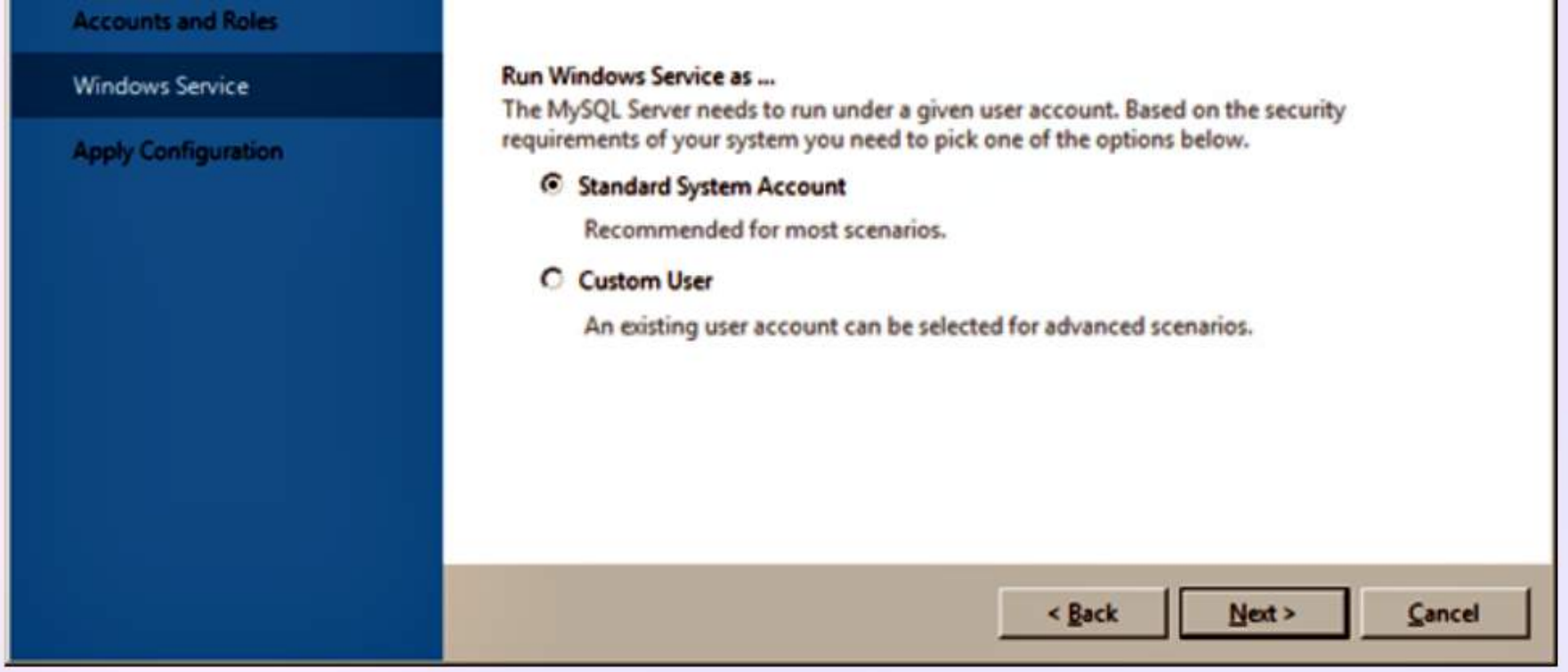


Также, ничего не меняем:

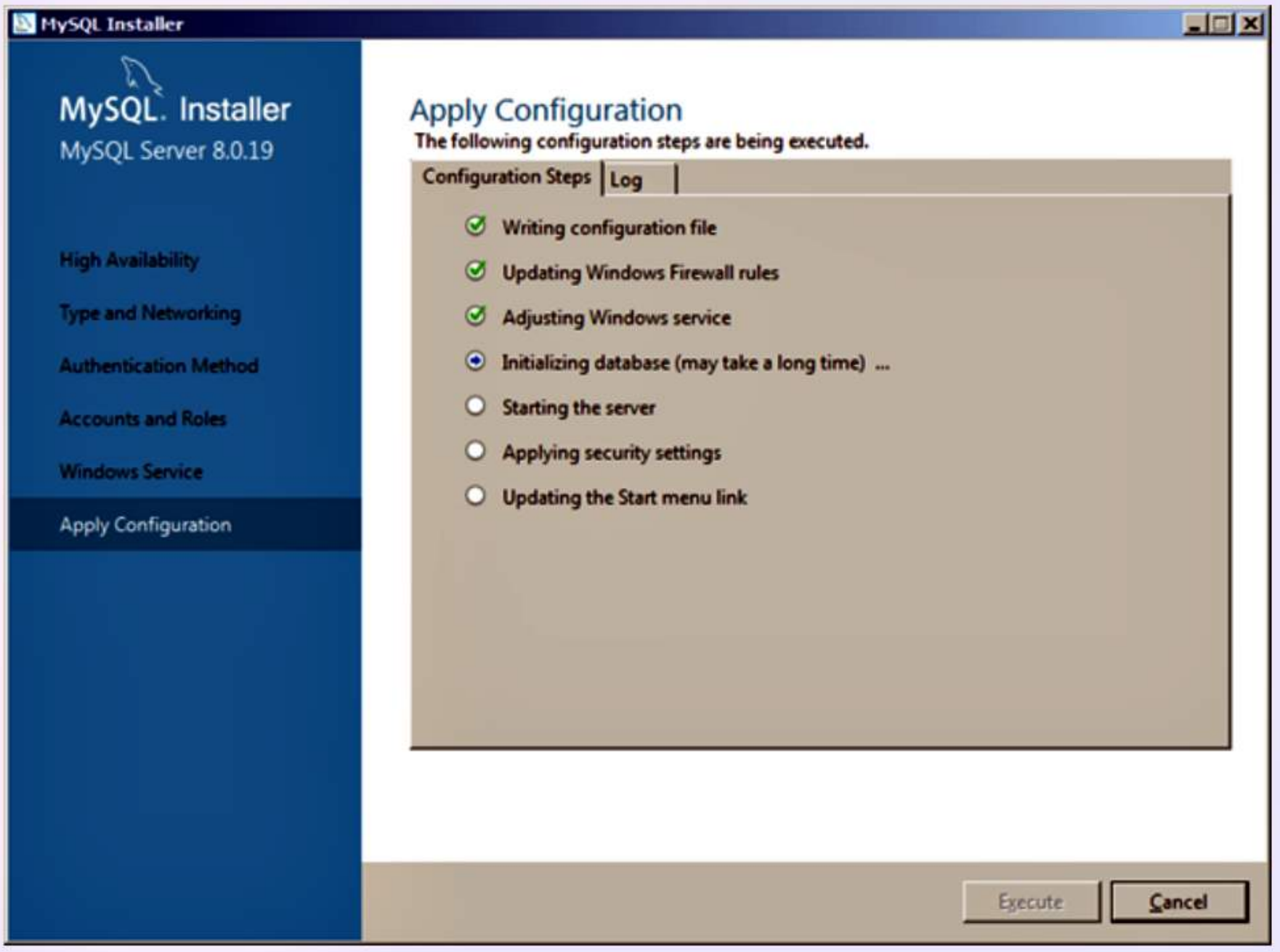
Устанавливаем пароль:



Ничего не меняем:



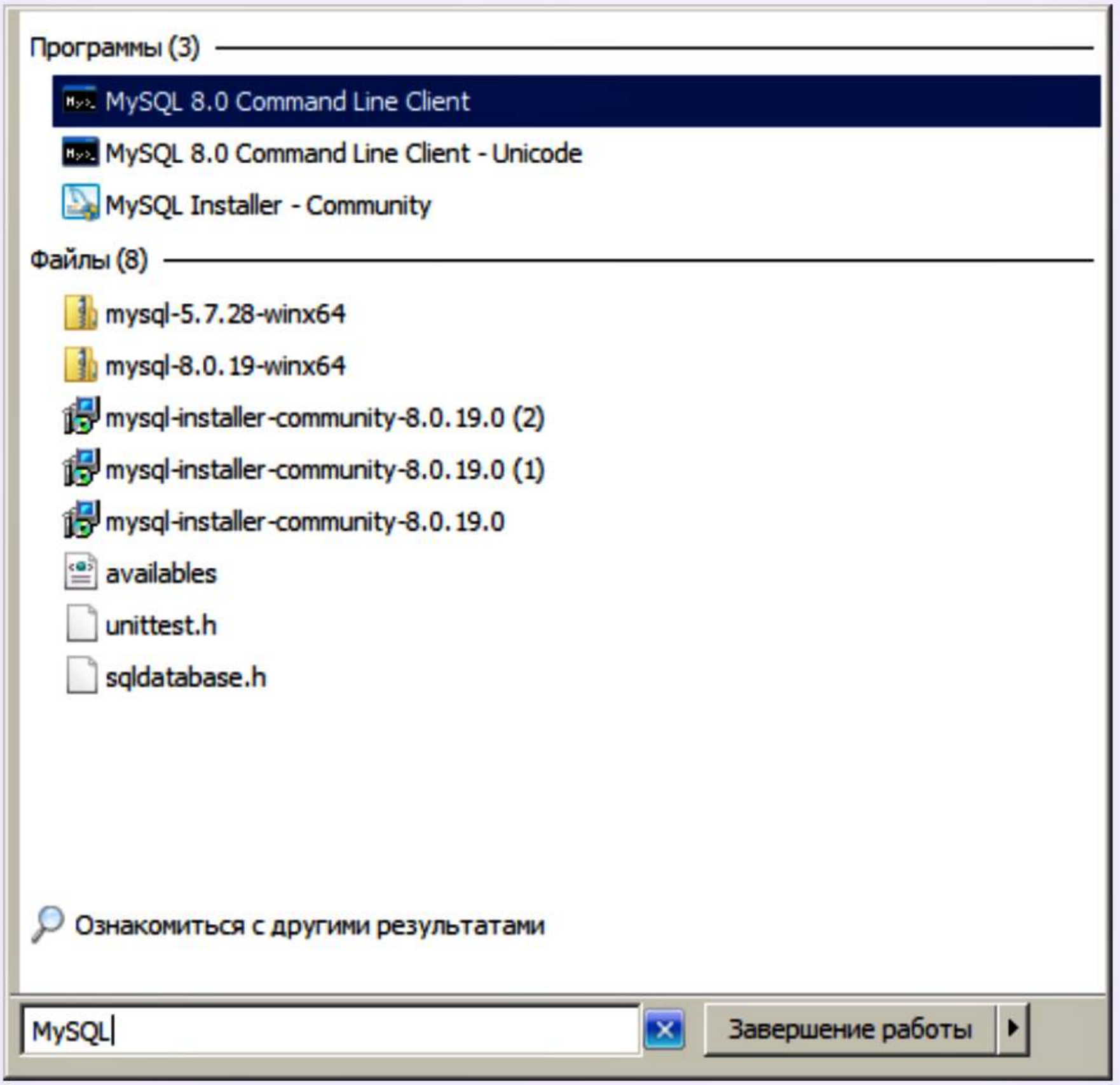
Нажимаем «Execute»:



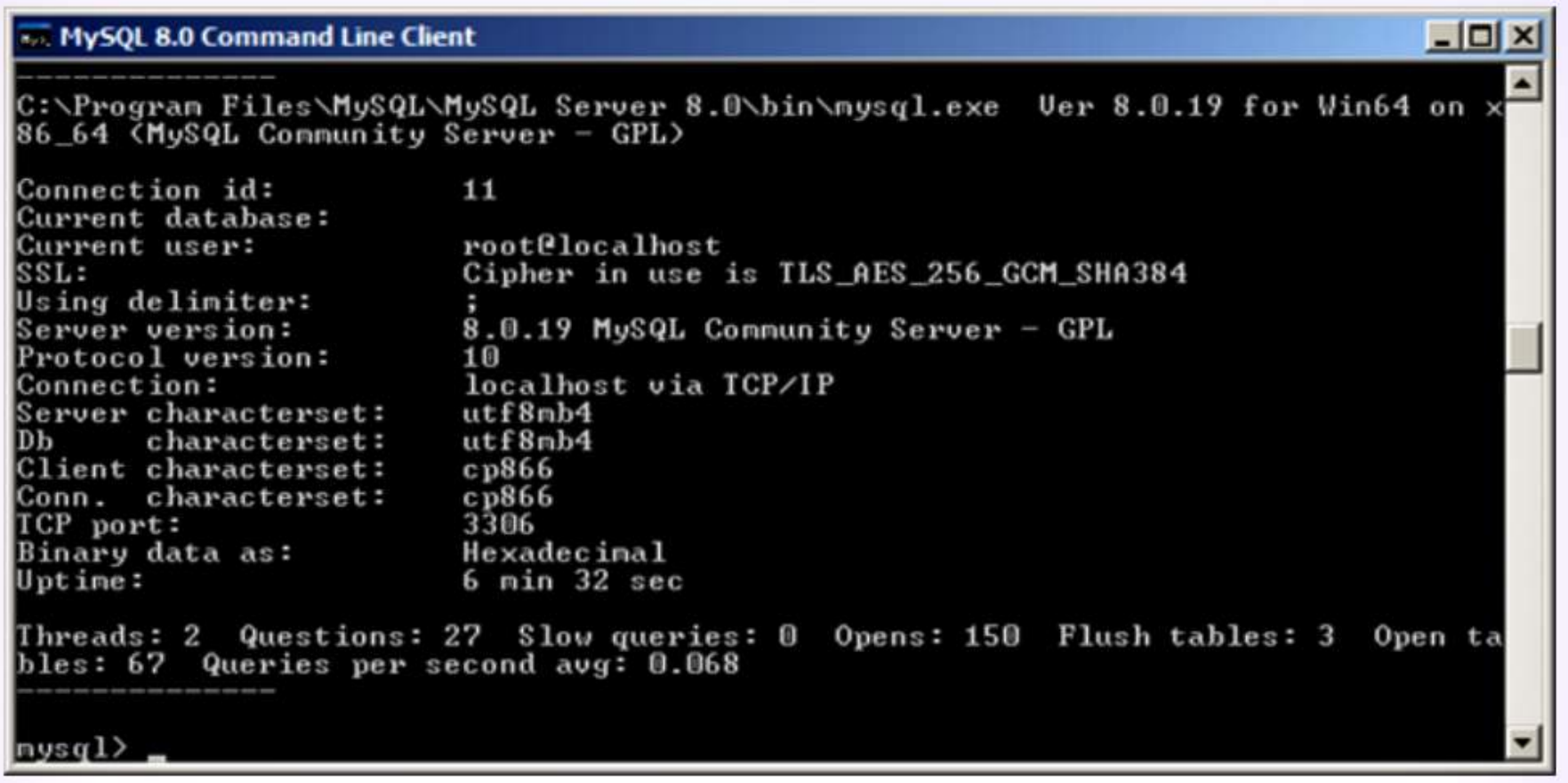
Нажимаем «Finish» несколько раз, пока программа установки не закроется.

Устанавливаем соединение с БД

Теперь в списке программ найдём командную строку MySQL:



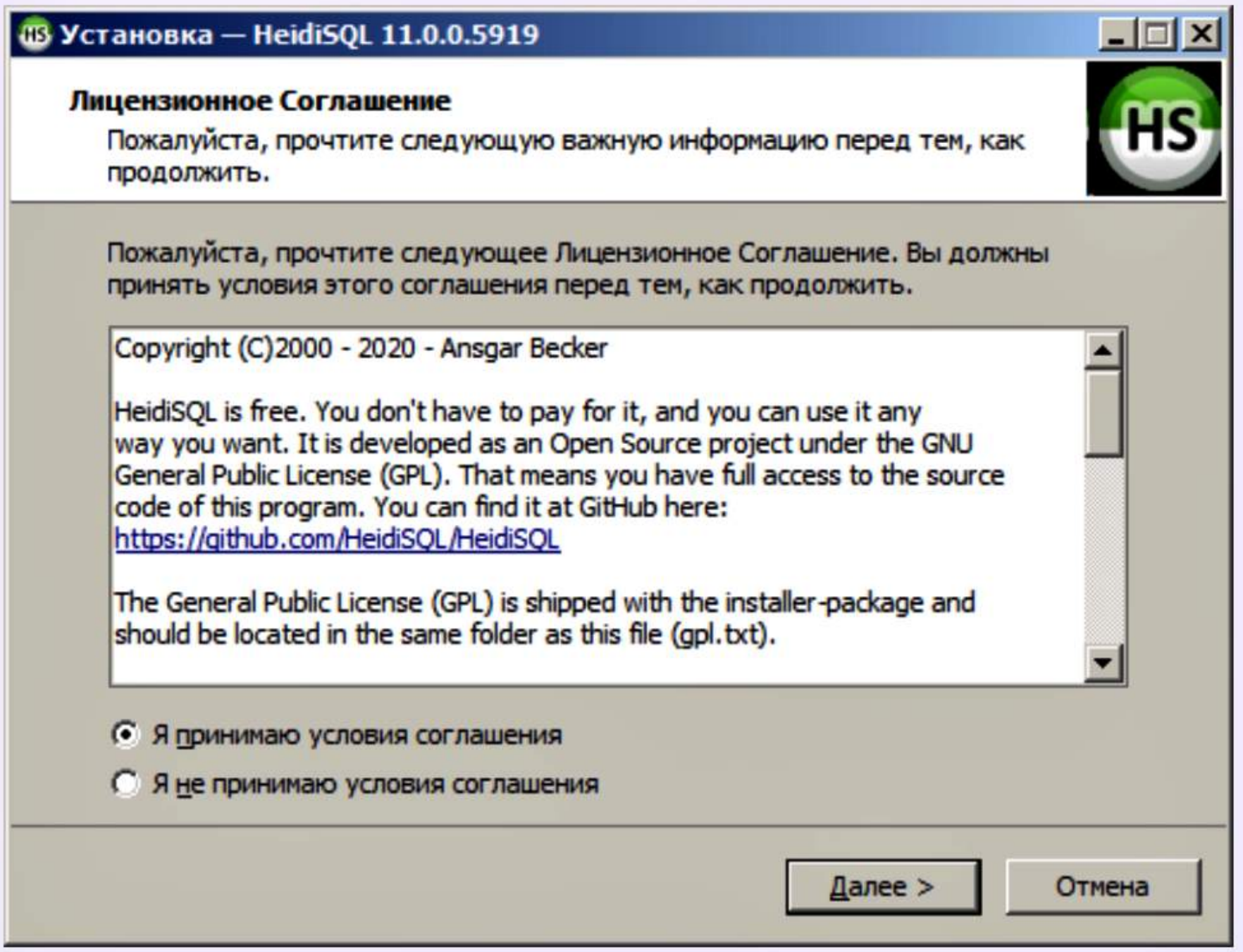
Открываем консоль, вводим пароль и нажимаем Enter, после этого вводим в консоль команду: «\s»:



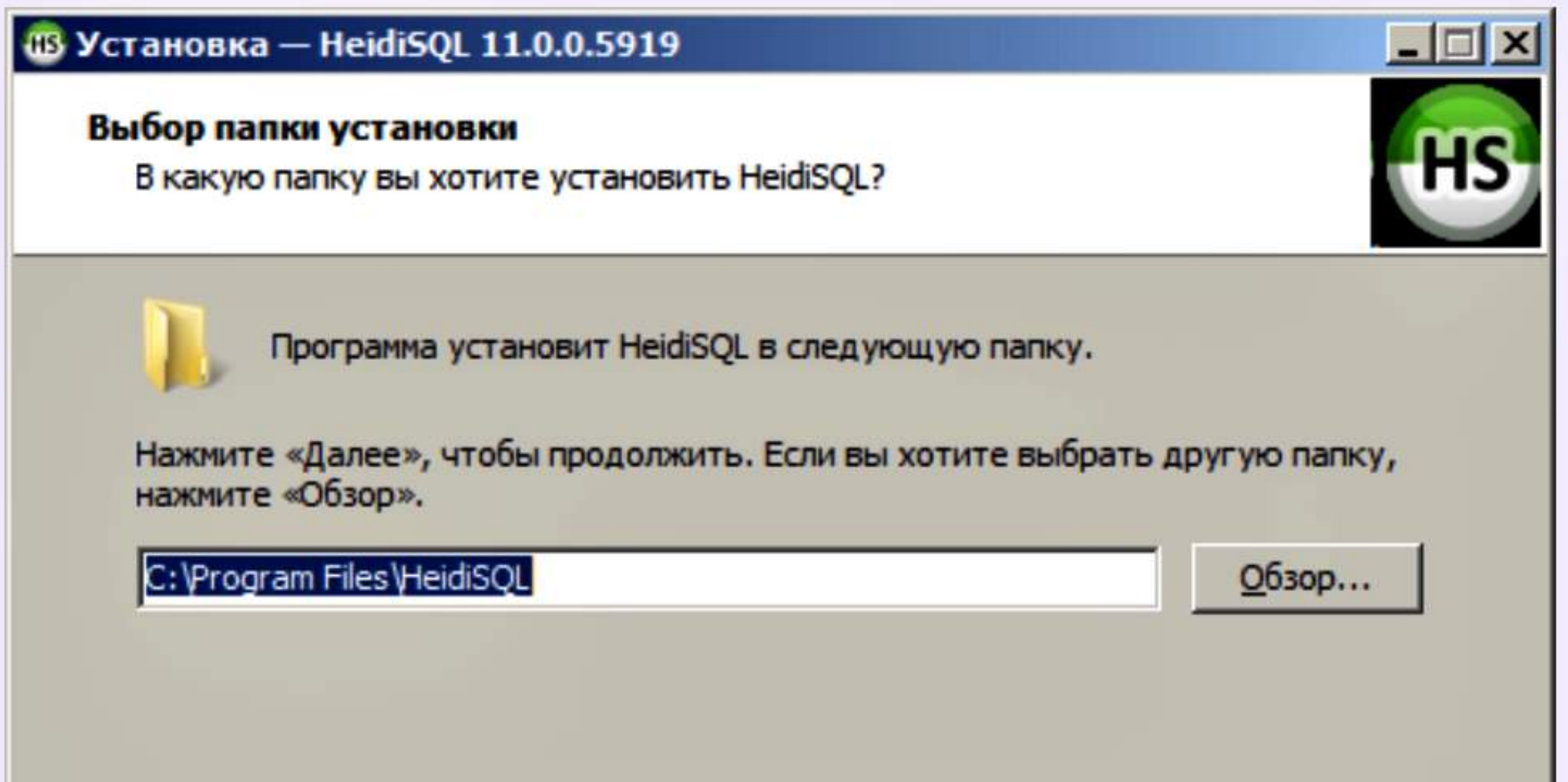
Запишите куда-нибудь значение строки «TCP port», оно вам пригодится.

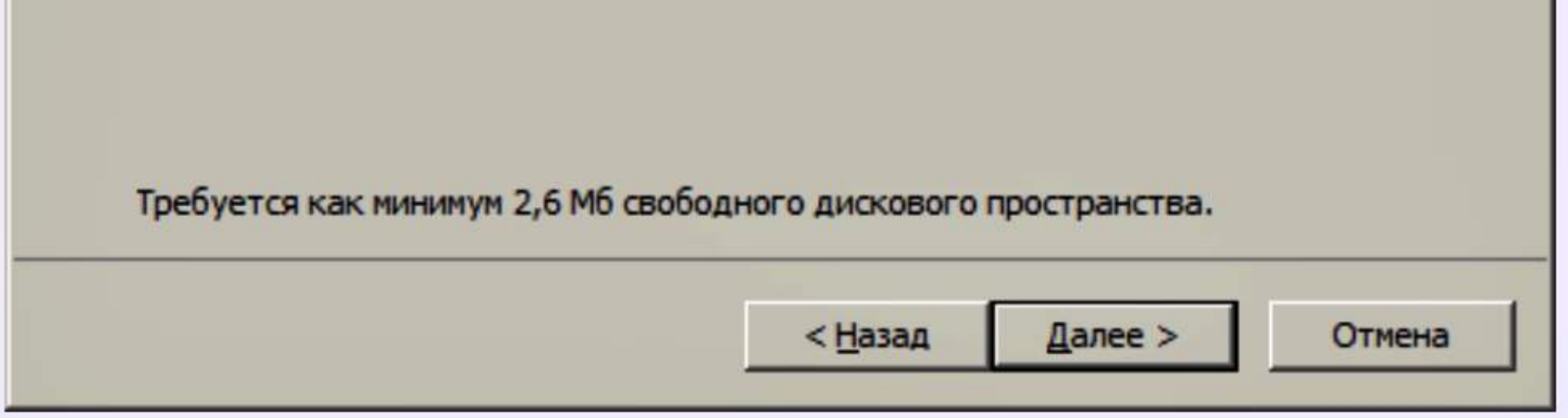
Сервер MySQL был установлен и теперь мы можем сохранять данные в MySQL, однако для более удобного визуального управления, нам пригодится MySQL менеджер. Установим его по ссылке: https://www.heidisql.com/download.php?download=installer

Принимаем соглашение:

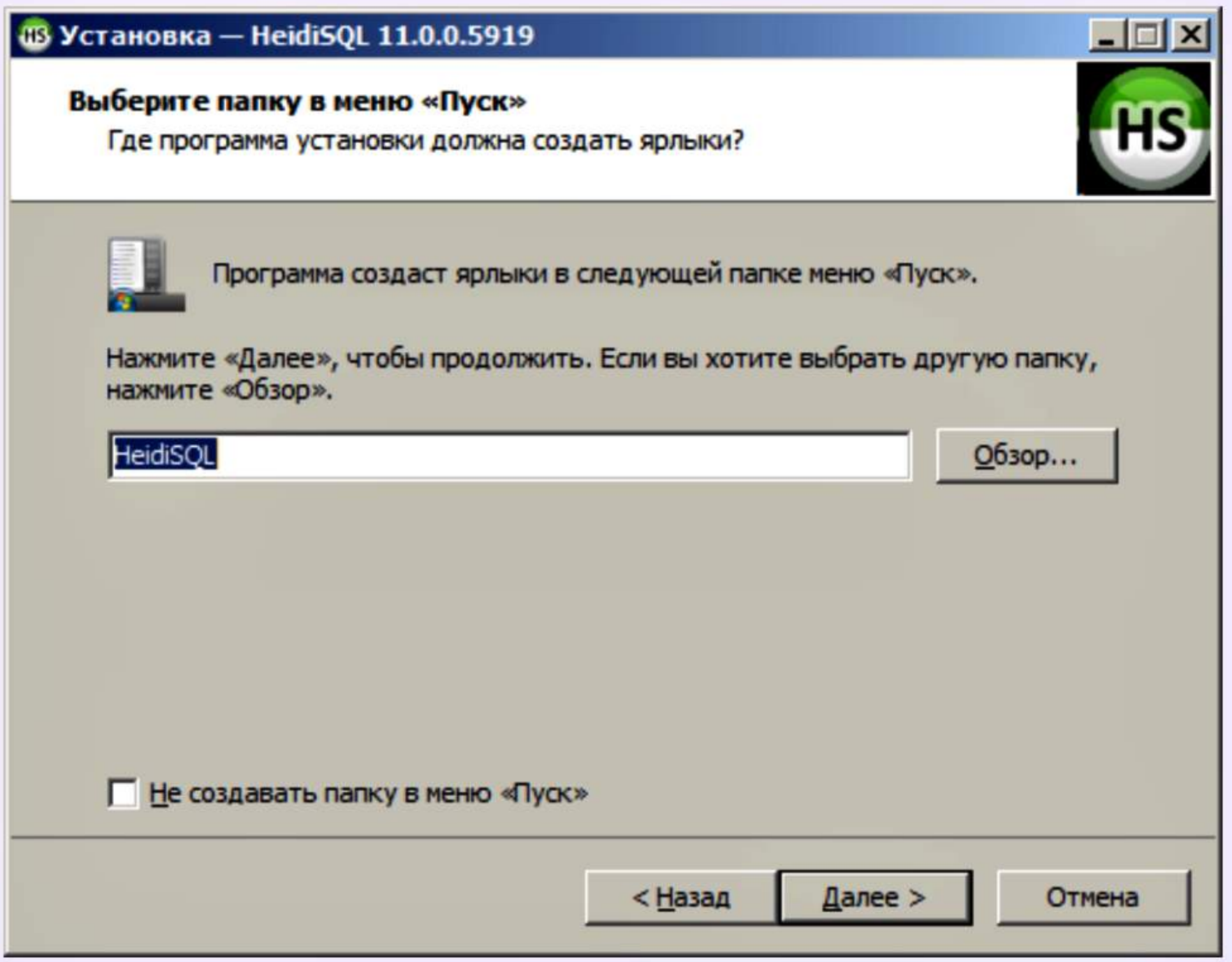


Указываем папку для инсталляции:

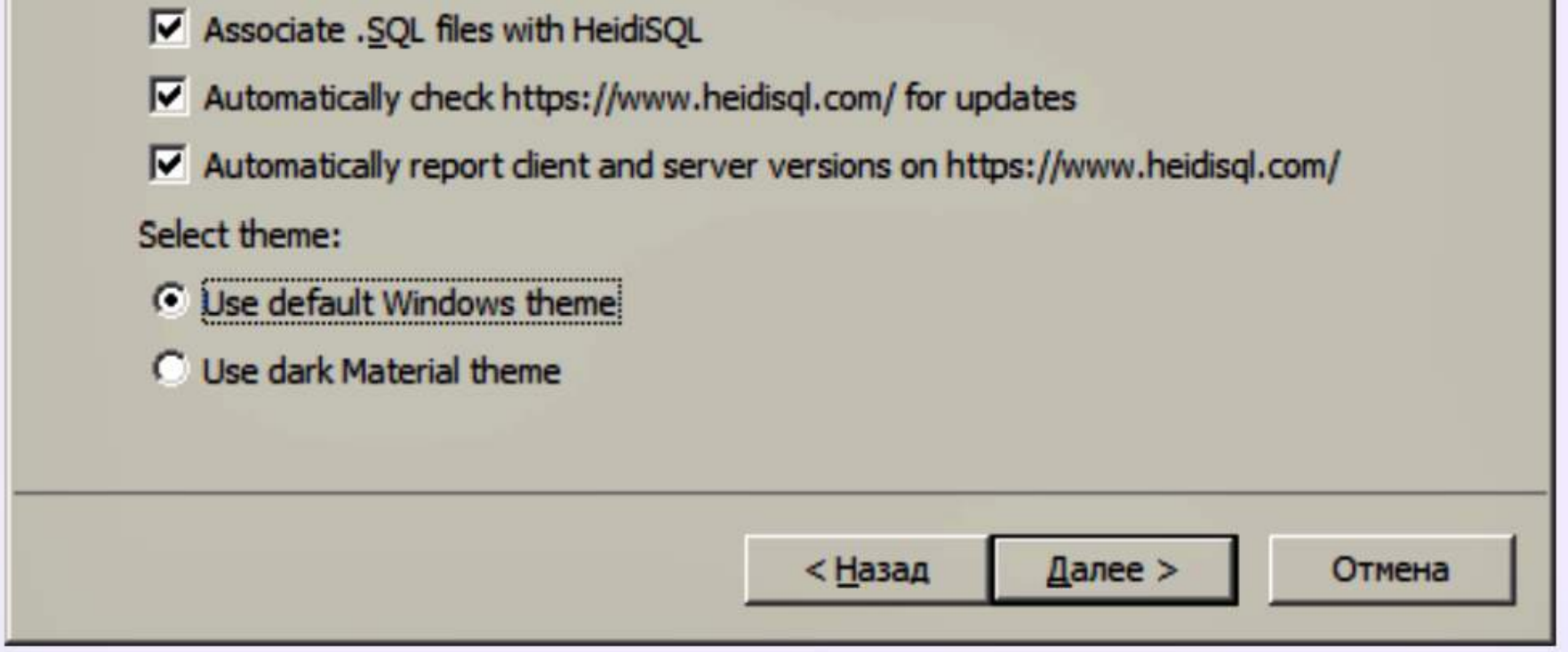




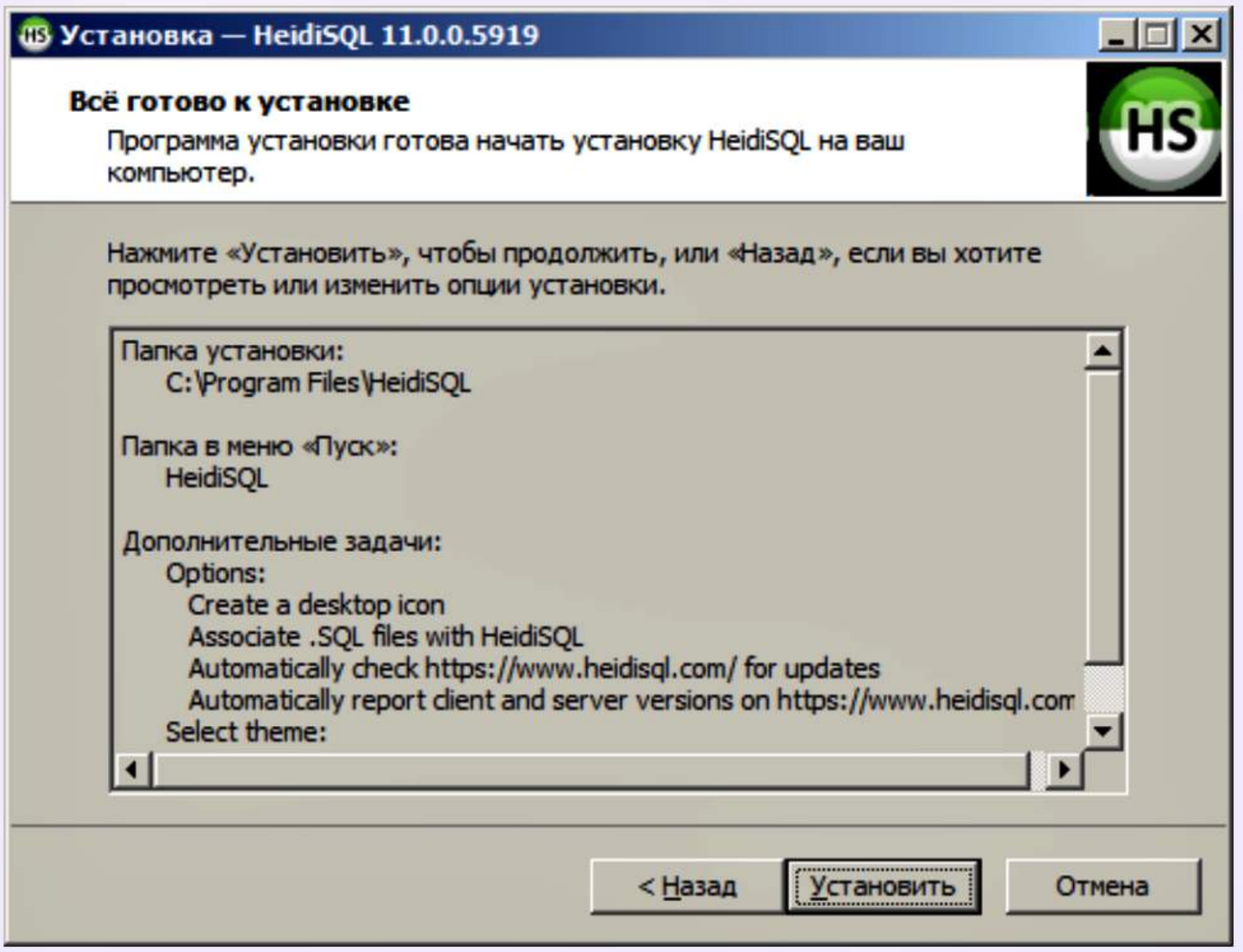
Оставляем как есть:



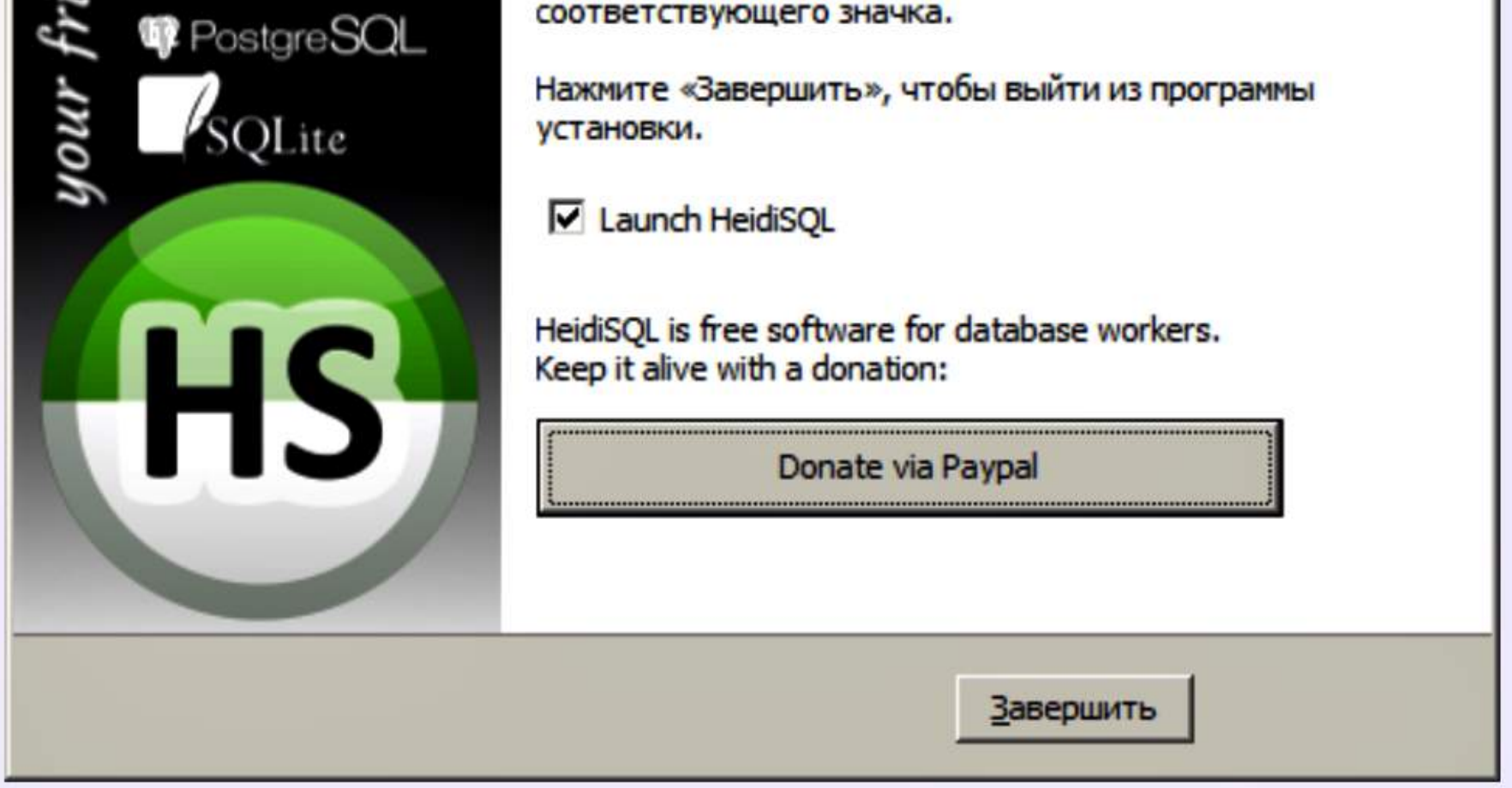
Оставляем как есть:



Нажимаем «Установить»:

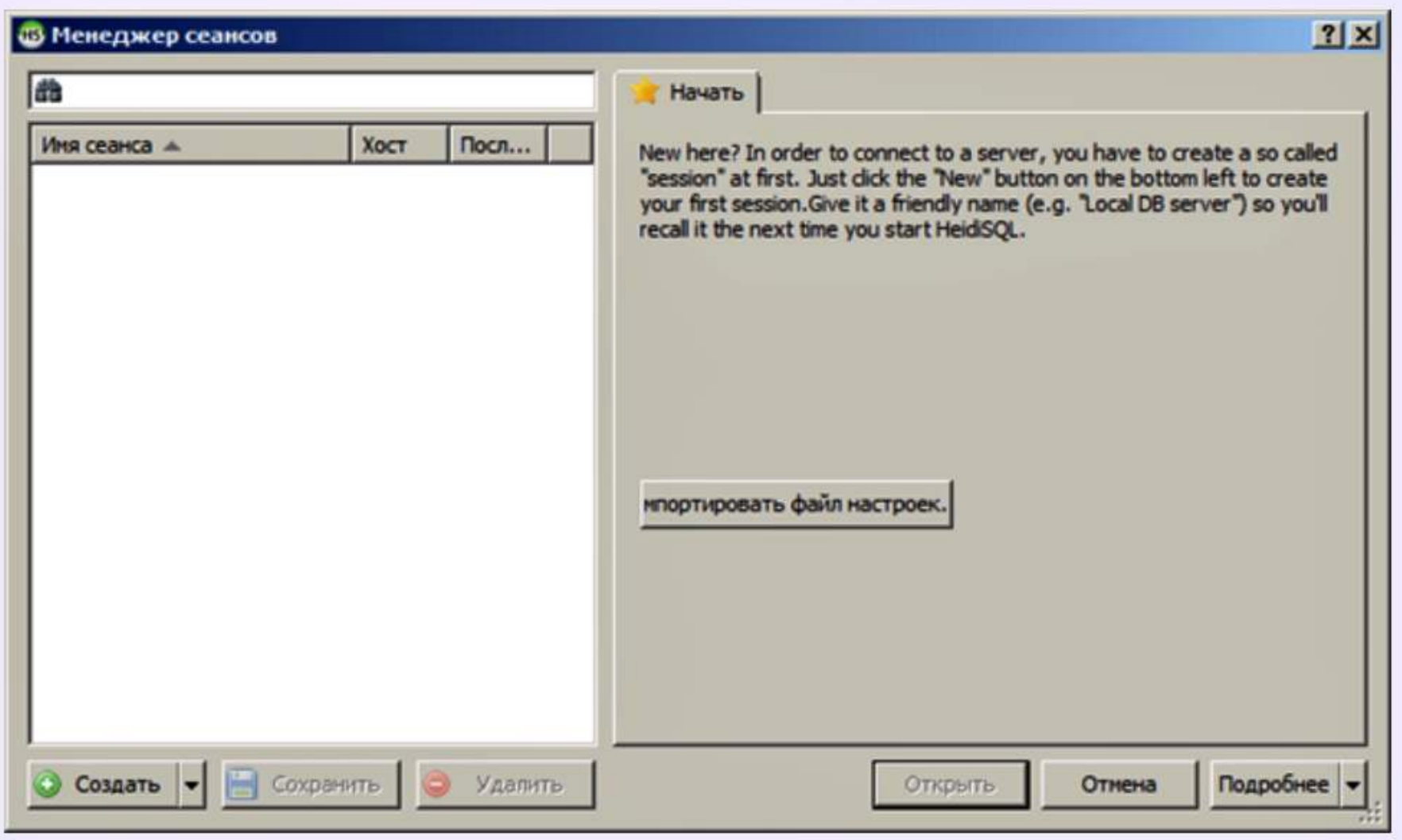


Нажимаем «Завершить»:

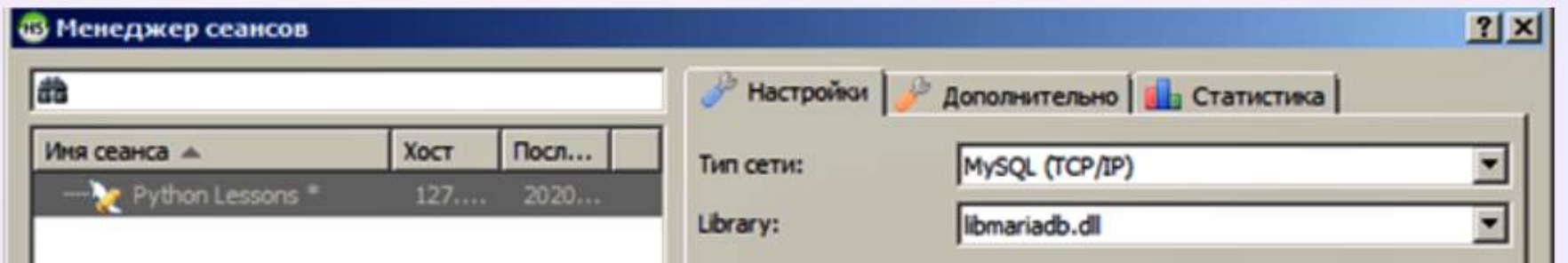


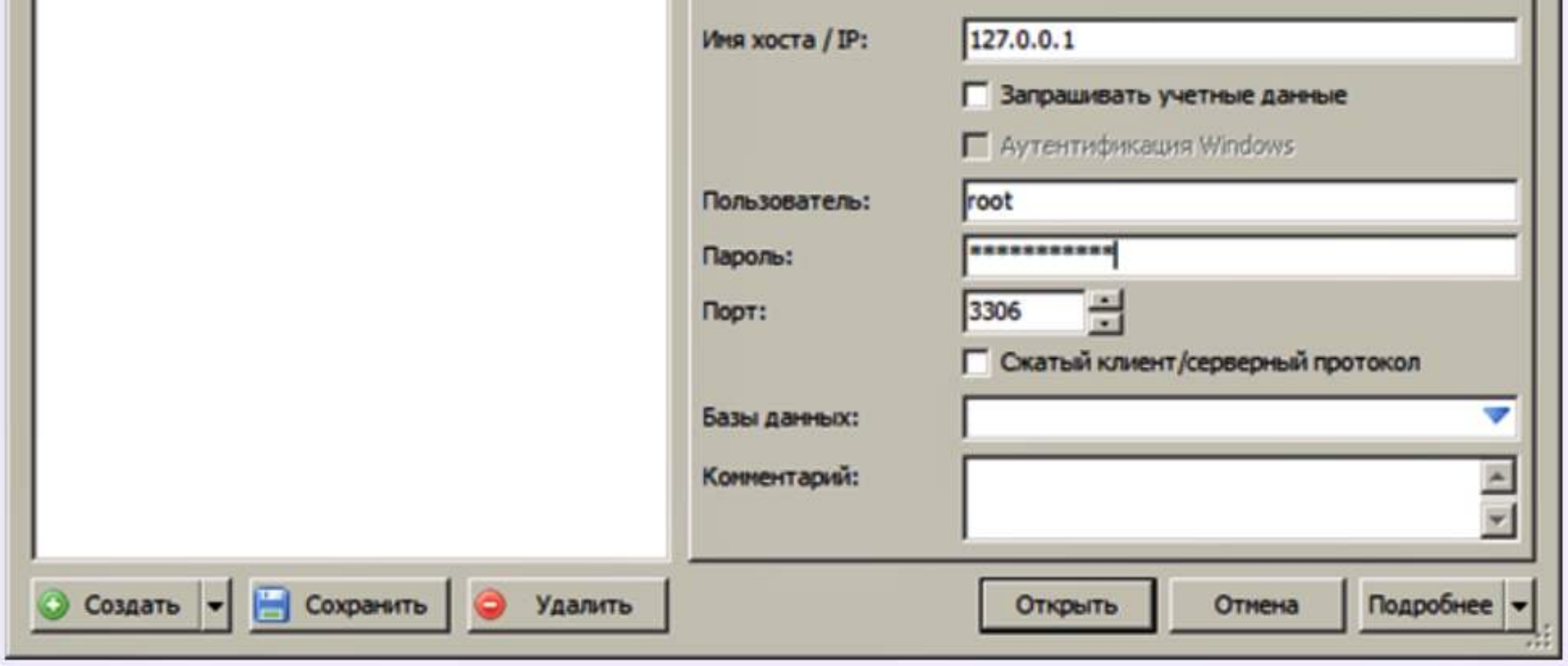
После завершения открылось окно менеджера MySQL. Давайте установим соединение с нашим сервером.

В левом нижнем углу, нажмём на кнопку «Создать»:

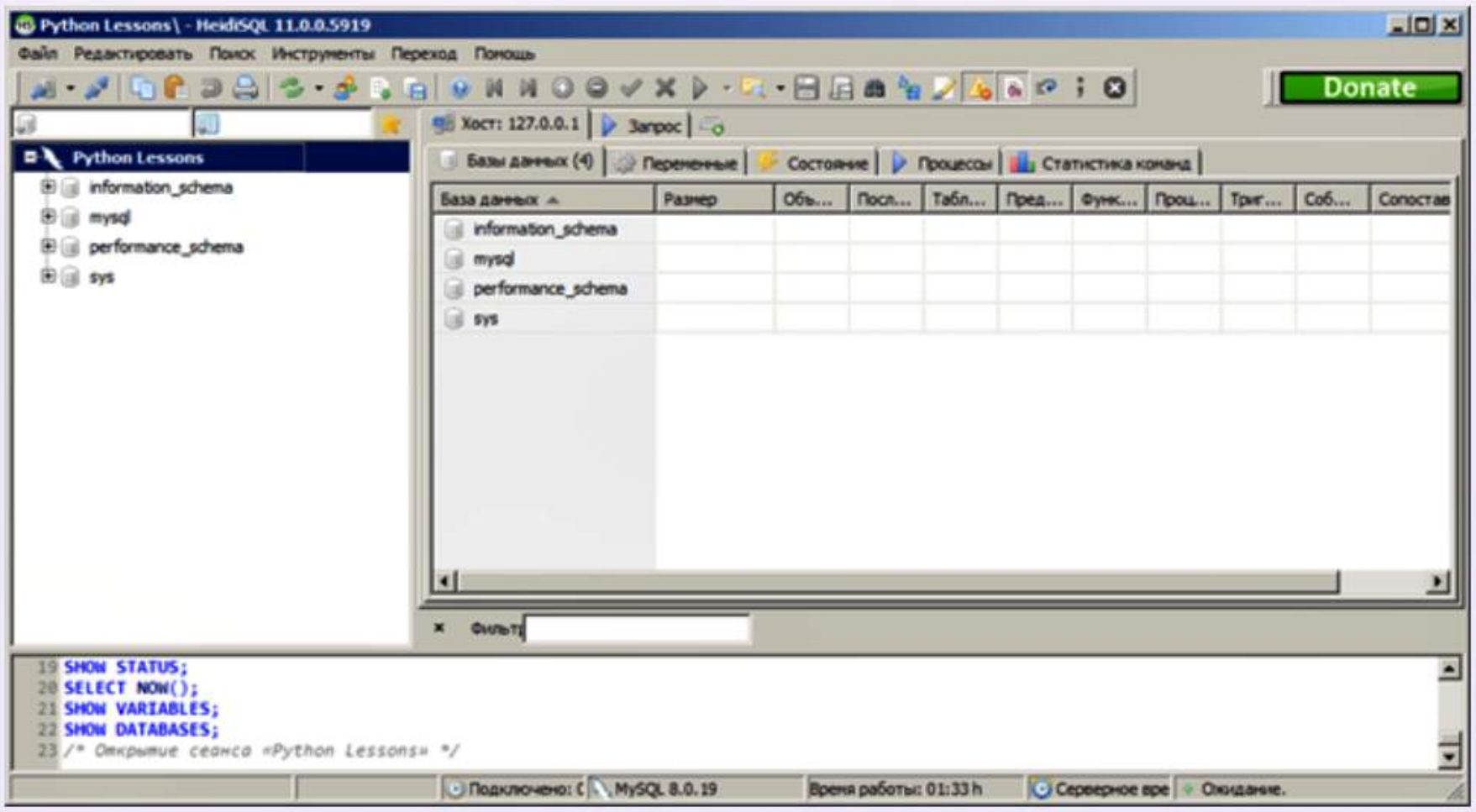


В открытом окне можно установить «Имя сеанса», а также ввести пароль от сервера, который вы ввели во время установки:

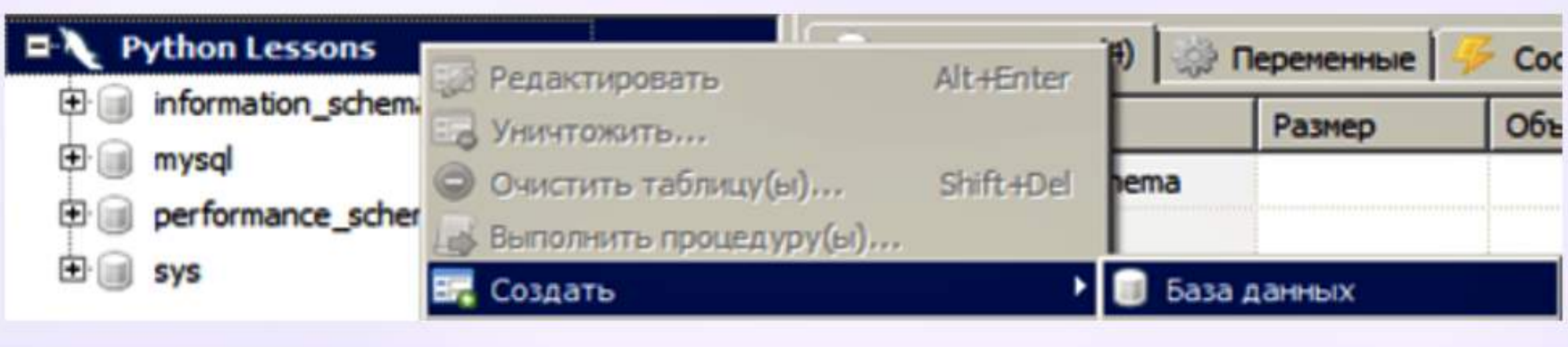




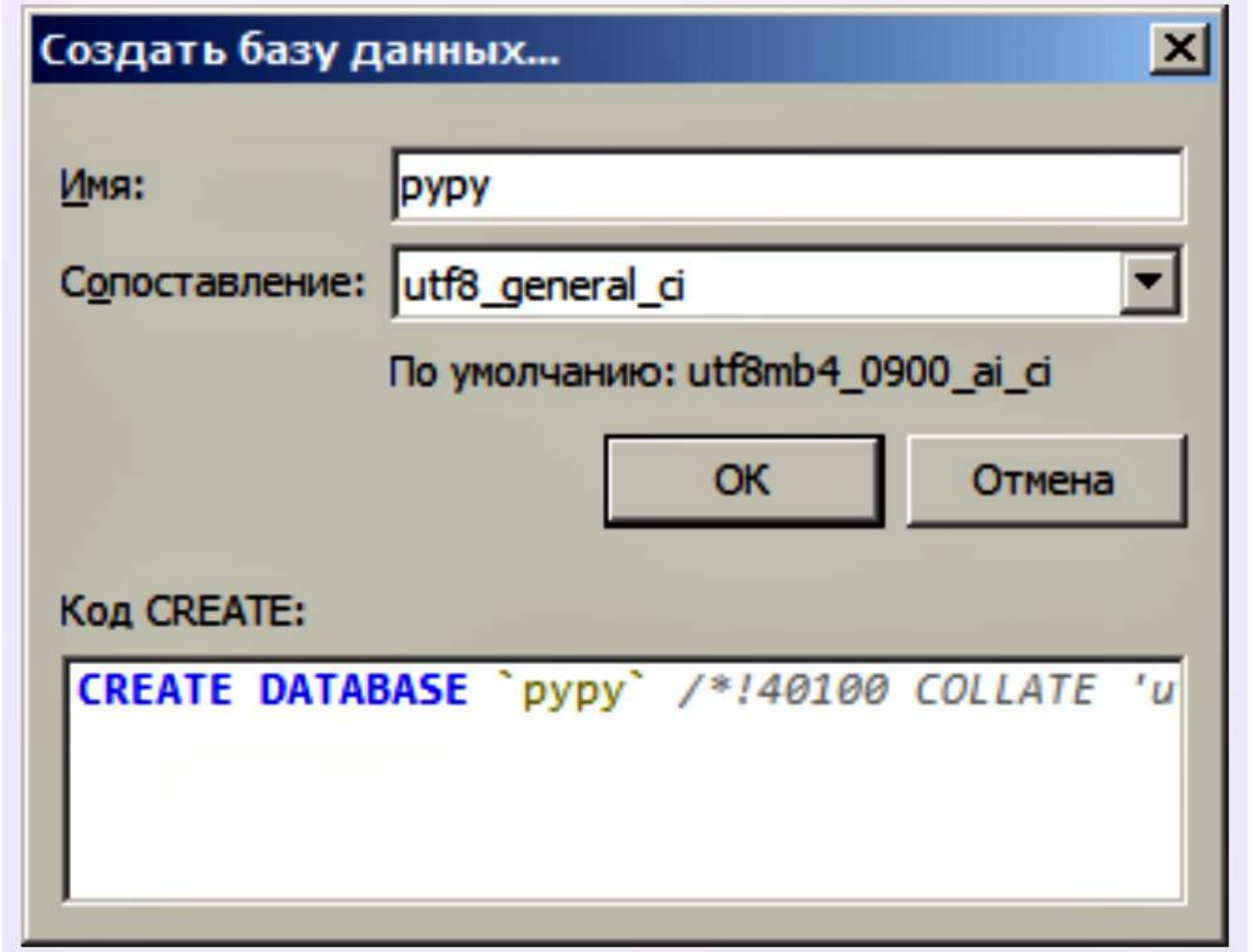
Если сервер MySQL вы устанавливали по моей инструкции, то здесь ничего менять не нужно. Нажимаем кнопку «Открыть»:



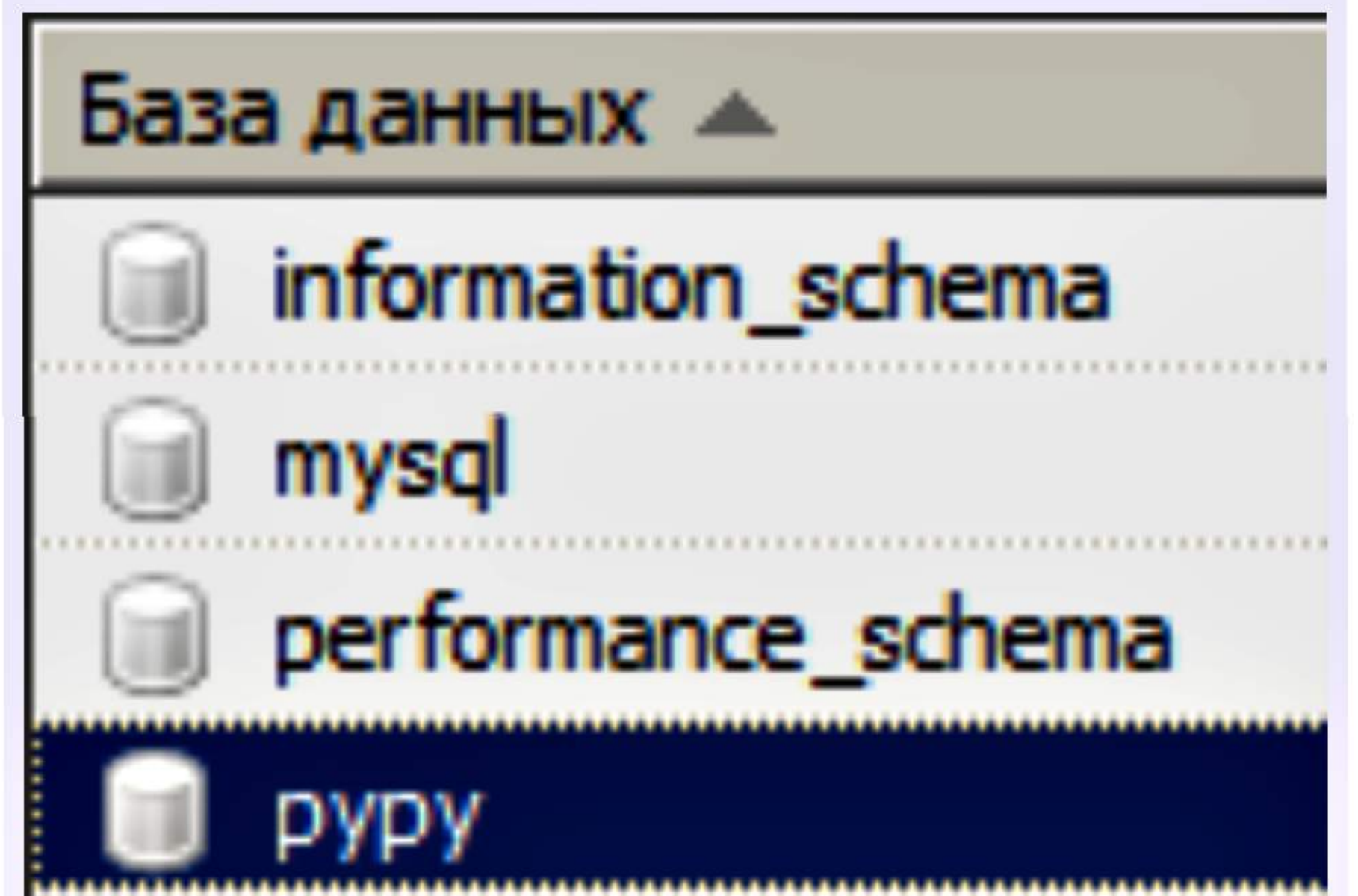
Открылся интерфейс управления сервером MySQL. Здесь вы можете создать новые базы данных и таблицы для них. Давайте сперва создадим новую базу данных. Нажимаем ПКМ на название нашего соединения, затем «Создать», затем «База данных»:



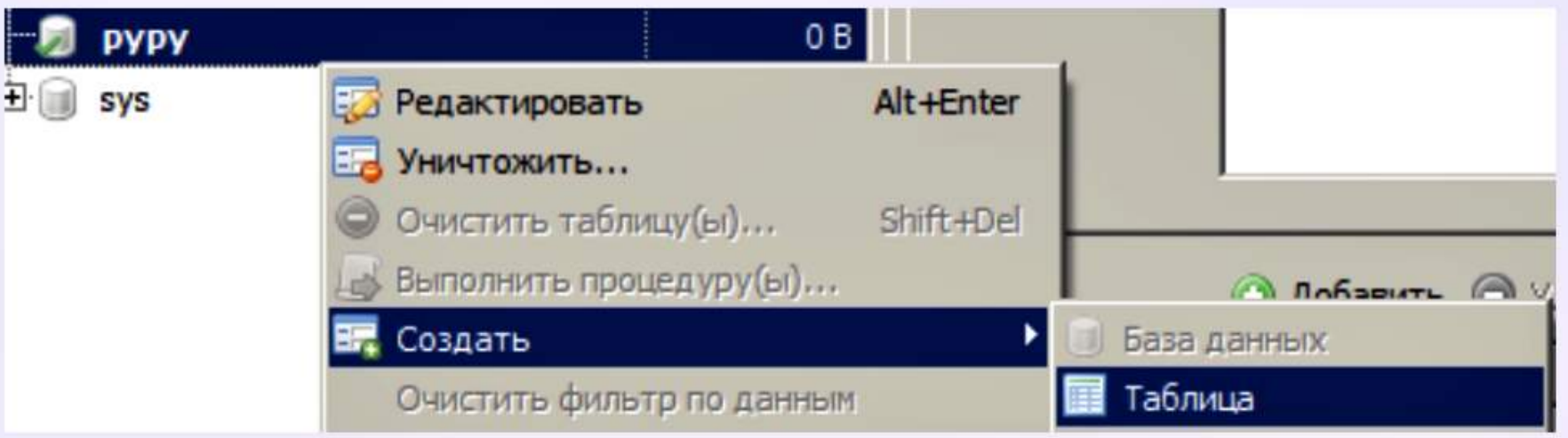
В открывшемся окне вводим название БД и указываем кодировку, в которой будут сохраняться данные:



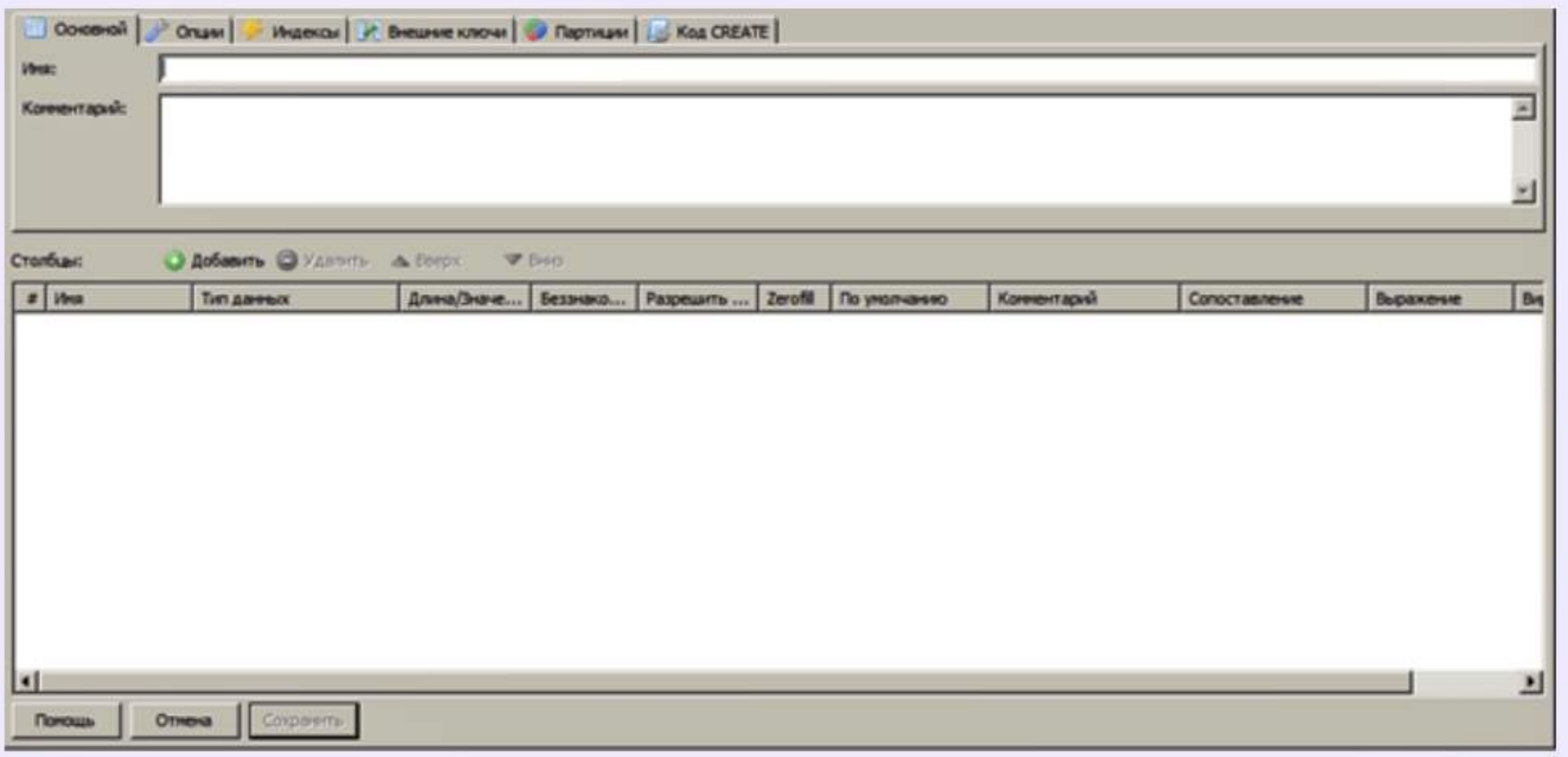
Указывайте `utf8_general_ci`. Нажимаем кнопку «ОК»:



БД была создана. Теперь создадим таблицу для нашей БД. В ней будут храниться статьи. Создадим таблицу. Нажмите ПКМ по названию нашей БД (руру), затем, «Создать», затем «Таблица»:

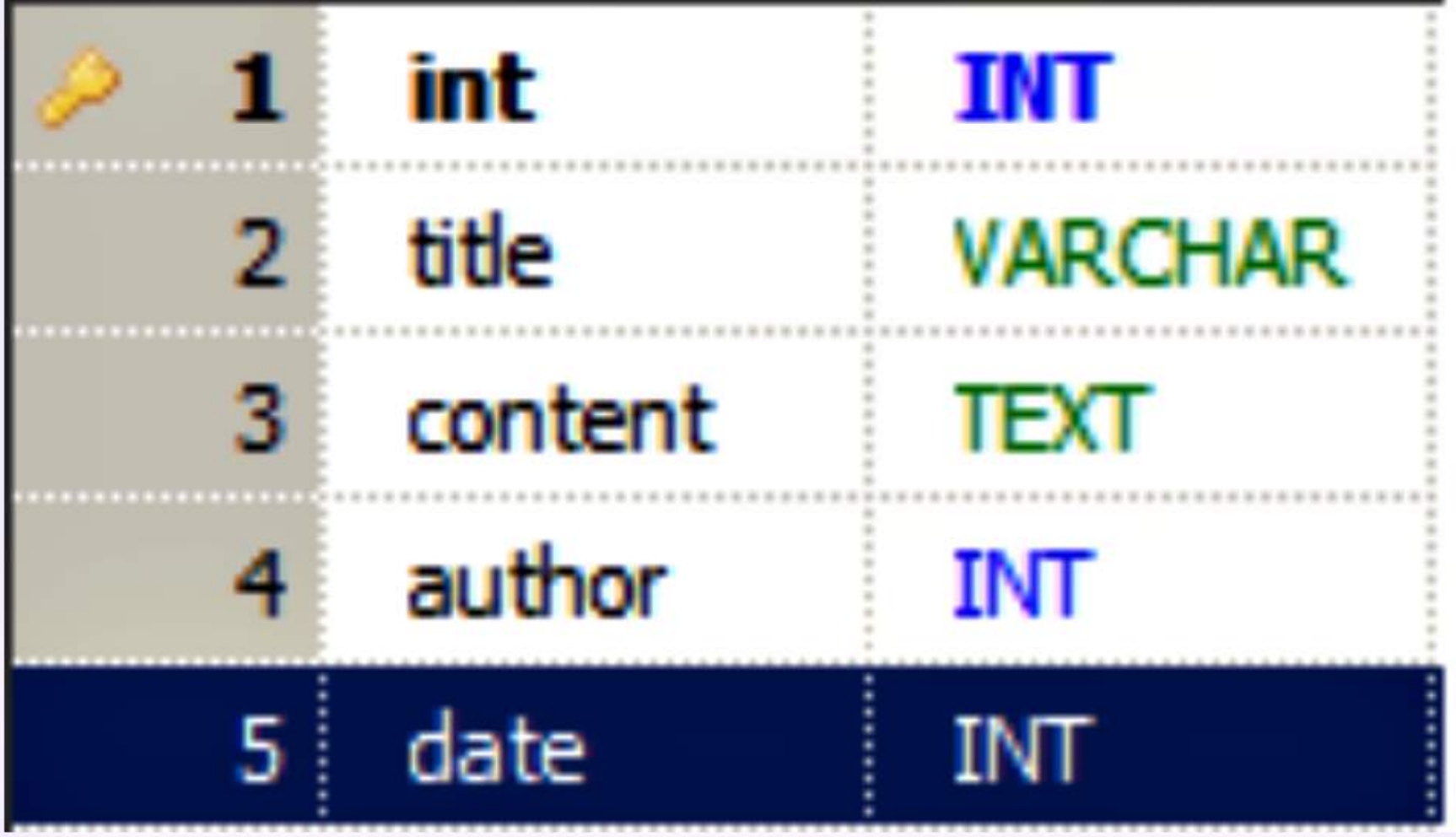


Откроется окно создания таблицы:



В поле «Имя» введём имя нашей БД, например, «Articles». Теперь нам нужно добавить столбцы. Для добавления столбца нужно нажать на кнопку «Добавить»:

Вспомним прошлый урок. Каждая отдельная строка на странице настройки БД – это отдельный столбец, например:



1	int	INT
2	title	VARCHAR
3	content	TEXT
4	author	INT
5	date	INT

Столбцы у таблицы следующие:

id (INT) – уникальный ключ записи, это значение будет уникально для каждой записи, по нему мы сможем обращаться к каждой конкретной записи
title (VARCHAR) – заголовок статьи
content (TEXT) – содержимое статьи
author (INT) – автор статьи
date (INT) – дата создания статьи

Для каждого столбца вы можете увидеть некоторые настройки:

Имя
Тип данных
Значения
Длина
Беззнаковое
Разрешить NULL
Zerofill
По умолчанию
Сопоставление
Индекс

Графа «Имя»

В этом поле, указывается имя значения записи. Например, каждая запись имеет 5 полей, чтобы обратиться к полю конкретной записи, этому полю нужно присвоить имя.

Графа «Тип данных»

Указывает тип данных, к которому относится этот столбец (строка, число, дата и т.д.). Давайте рассмотрим основные типы, которые нам пригодятся:

VARCHAR: Строка длинной до 20к+ символов (если устанавливаете кодировку utf-8). Обычно VARCHAR используют для хранения коротких строк, например, заголовков или имён пользователей.

TEXT: Строка длинной 65535 символов. Как правило используется для хранение большого объёма данных, например, статей.

INT: Числовой тип в диапазоне от -2 147 483 648 до 2 147 483 647. Если указываете атрибут UNSIGNED (Беззнаковое), то в диапазоне от 0 до 4 294 967 295.

Графа «Длина/Значения»

Указывает максимально допустимую длину значения. В зависимости от типа принимает разные условия, например, если тип VARCHAR, то количество символов.

Графа «Беззнаковое»

Указывает, что данное число положительное, в таком случае увеличивается максимальная длинна, т.к. не нужно резервировать память числа на диапазон отрицательных значений.

Графа «Разрешить null»

Это поле, которое указывает, может ли наше поле иметь значение null.

Графа «Zerofill»

Вам это навряд ли пригодится. Это поле, которое дополняет незанятые разряды числа нулями.

Графа «По умолчанию»

Позволяет указать значение, которое будет подставляться автоматически при создании записи.

Графа «Сопоставление»

Указывает кодировку текста, хранимого в данном столбце. Если столбец числовой, то нет необходимости указывать значение этой графы. Если строковый, то выбираете наиболее распространённую и корректную кодировку: `utf8_general_ci`

Индекс

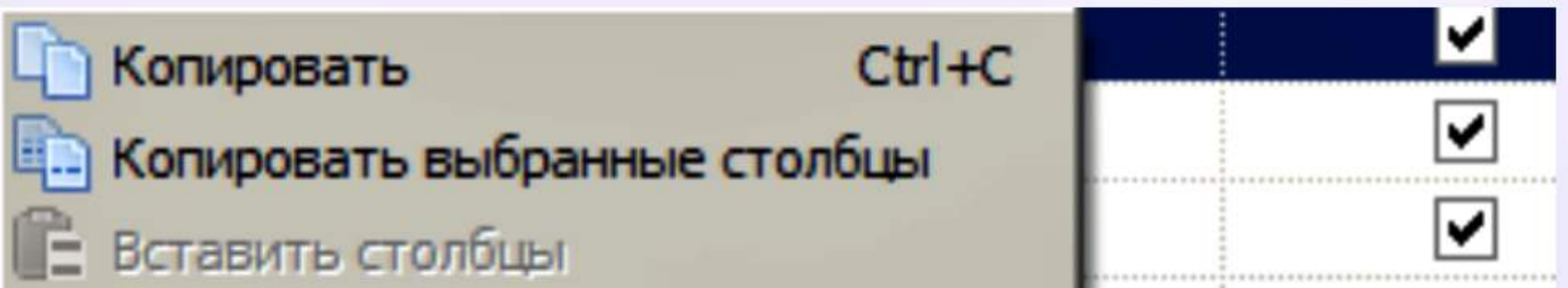
Индекс определяет то, как будет вести себя значение заданного столбца. Например, значение PRIMARY указывает, что значение данного поля должно быть уникальным, то есть, если у поля есть индекс PRIMARY, то внутри одной таблицы, не может быть двух одинаковых значений этого поля.

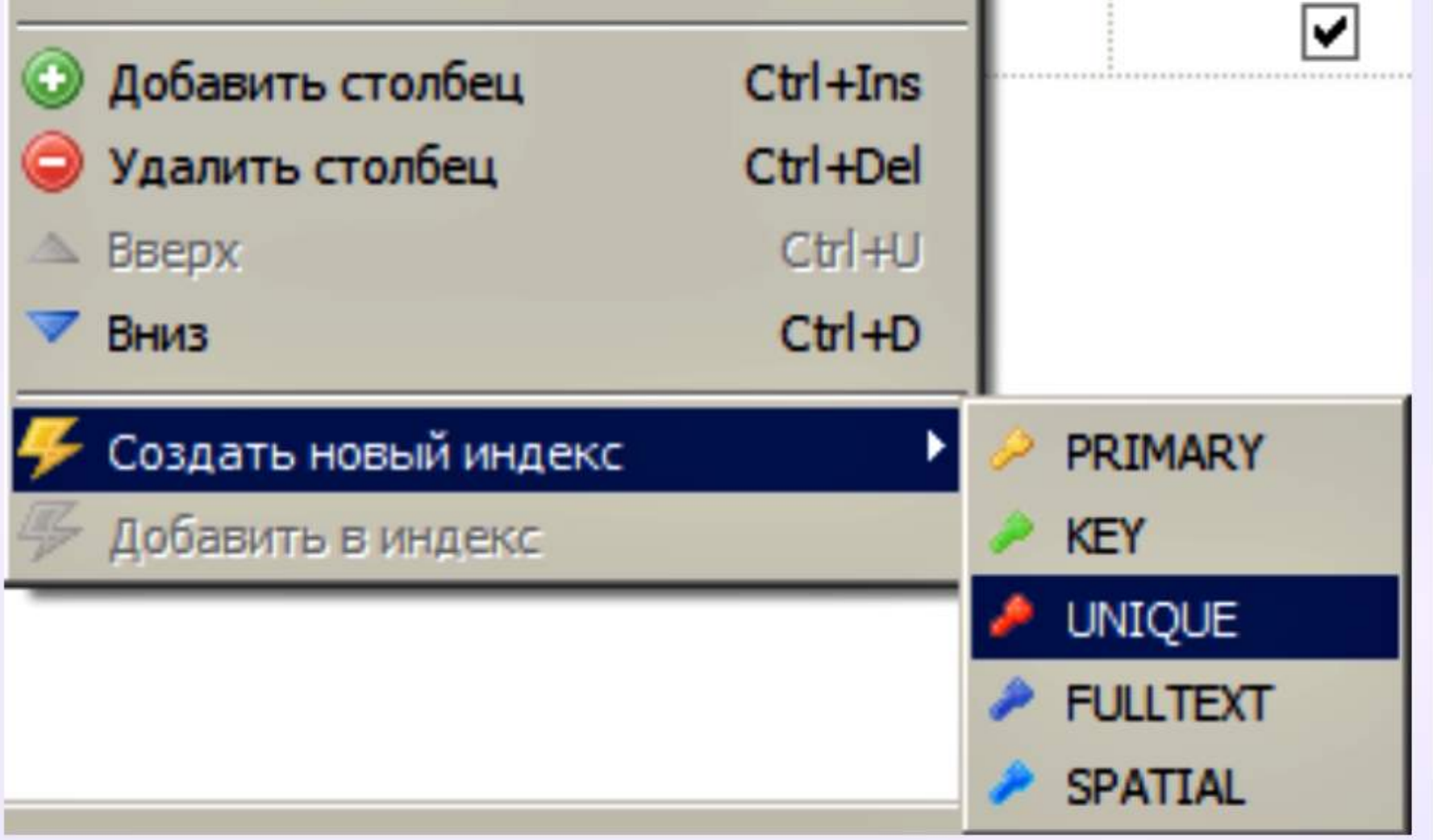
Т.е. если, например, для поля id мы устанавливаем индекс PRIMARY, то другого такого id в этой таблице уже быть не может, допустим, если у записи ячейка id равна 741, то второй записи в данной таблице с id равному 741 быть не может.

Если одному из полей (например, id) вы присвоили индекс PRIMARY, то другому полю (например, title) индекс PRIMARY вы присвоить не можете.

Помимо PRIMARY у графы Index есть также значение UNIQUE оно похоже на PRIMARY, за одним исключением, данное значение можно присвоить любому количеству полей. Например, и полю id и полю title.

Чтобы выбрать индекс, нажимаем по столбцу ПКМ, затем «Создать новый индекс»:





Когда нам нужно использовать индекс? Когда нужно задать столбцу уникальную индексацию. Например, когда нужно установить, что значение данного столбца должно быть строго уникальным. Как мы сделаем это со столбцом id.

Столбец id

Индекс. Открываем настройки индекса. Так как столбец id – это уникальный ключ, по которому можно получить данные определённой записи, присваиваем этой графе значение PRIMARY.

Как выглядят заполненные поля столбца id:

#	Имя	Тип данных	Длина/Значения	Беззнаковое	Разрешить NULL	Zerofill	По умолчанию
1	id	INT		☑	☐	☐	AUTO_INCREM...

Столбец title

Как выглядят заполненные поля столбца `title`:

#	Имя	Тип данных	Длина/Значения	Беззнаковое	Разрешить NULL	Zerofill	По умолчанию	Комментарий	Сопоставление
1	id	INT		☑	☐	☐	AUTO_INCREM...		
2	title	VARCHAR	500	☐	☐	☐	Нет значения п...		utf8_general_ci

Столбец content

Так как статьи могут быть достаточно объёмными, не ставим ограничение по размеру. Как выглядят заполненные поля столбца `content`:

#	Имя	Тип дан...	Длина/...	Беззнаковое	Разрешить NULL	Zerofill	По умолчанию	Сопоставление
1	id	INT		☑	☐	☐	AUTO_INCREM...	
2	title	VARCHAR	500	☐	☐	☐	Нет значения п...	utf8_general_ci
3	content	TEXT		☐	☐	☐	Нет значения п...	utf8_general_ci

Столбец author

В столбце автор будет храниться его id, соответственно в формате INT.

#	Имя	Тип дан...	Длина/...	Беззнаковое	Разрешить NULL	Zerofill	По умолчанию	Сопоставление
1	int	INT		☑	☐	☐	AUTO_INCREM...	
2	title	VARCHAR	500	☐	☐	☐	Нет значения п...	utf8_general_ci
3	content	TEXT		☐	☐	☐	Нет значения п...	utf8_general_ci
4	author	INT		☑	☐	☐	0	
5	date	INT		☑	☐	☐	Нет значения п...	

Столбец date

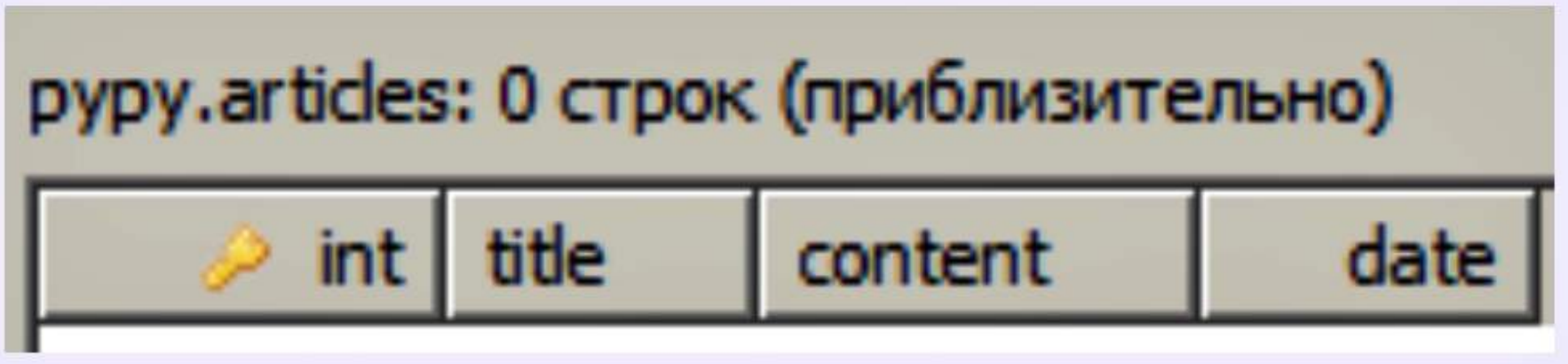
Дата будет храниться в формате unix, то есть как количество секунд, прошедших с 1 января 1970 года, поэтому проще хранить их в переменной типа INT.

#	Имя	Тип дан...	Длина/...	Беззнаковое	Разрешить NULL	Zerofill	По умолчанию	Сопоставление
1	id	INT		☑	☐	☐	AUTO_INCREM...	
2	title	VARCHAR	500	☐	☐	☐	Нет значения п...	utf8_general_ci
3	content	TEXT		☐	☐	☐	Нет значения п...	utf8_general_ci
4	date	INT		☑	☐	☐	Нет значения п...	

Все столбцы созданы, осталось только сохранить новую таблицу. Нажимаем кнопку «Сохранить». Теперь в списке таблиц, мы можем увидеть созданную нами таблицу «Articles»:

Чтобы посмотреть её содержимое, нужно кликнуть ЛКМ по названию таблицы, в открывшемся окне кликнуть по вкладке «Данные»:

Здесь мы увидим список всех записей таблицы:



Но так как таблица пока пустая, данных в ней нет.

В следующем уроке мы разберёмся как работать с таблицей, как добавлять, считывать, редактировать и удалять записи.

Основные операции с БД. CRUD

Содержание урока

Устанавливаем PyMySQL

База данных создана. Таблица тоже. Теперь мы можем спокойно с ней работать. А именно, использовать основные (и по желанию не только) операции для работы с БД. Основные операции многим известны под распространённой аббревиатурой CRUD: `Create` (создать запись), `Read` (прочитать запись), `Update` (обновить запись), `Delete` (удалить запись).

Но сперва, нам необходимо установить библиотеку по работе с `MySQL` в Python, под названием `pymysql`.

Документация:

https://pymysql.readthedocs.io/en/latest/
https://github.com/PyMySQL/PyMySQL

Запустим наш проект в PyCharm, откроем терминал и введём следующую команду:

```
pip install PyMySQL
```

Отлично, теперь у нас есть модуль, при помощи которого мы можем подключиться к MySQL и управлять ей. Но сперва, давайте напишем свой простой модуль для подключения к БД, чтобы удобно пользоваться им впоследствии. В корневой директории нашего проекта, создадим пакет под названием `db`, внутри него создадим модуль `connection.py`:

Внутри модуля подключим библиотеку PyMySQL:

```
import pymysql.cursors
from pymysql.cursors import DictCursor
```

Теперь создадим функцию make, которая будет создавать соединение с БД и возвращать модуль, при помощи которого можно управлять нашей БД:

```
def make(host="127.0.0.1", user="root", password="my_password", db="pypy", charset="utf8", cursorclass=DictCursor):
    connection = pymysql.connect(host=host, user=user, password=password, db=db, charset=charset, cursorclass=cursorclas
    return connection
```

Что тут происходит? В данную функцию передаются данные, необходимые для подключения к БД. Для подключения к БД необходимы следующие аргументы:

host (адрес) БД, по умолчанию это `127.0.0.1`
user (логин пользователь) БД, по умолчанию это root
password (пароль пользователя) БД, его вы создавали сами, во время установки сервера MySQL
db (имя БД), базу данных с некоторым именем вы создали в прошлом уроке
charset (кодировка данных) БД, по умолчанию ставьте `utf-8`, самая достойная кодировка
cursorclass, определяет форму в которой данные будут возвращаться из БД. По умолчанию – список. Но я импортировал DictCursor и передал его в качестве аргумента по умолчанию.

Для каждого аргумента я прописал значения по умолчанию, чтобы можно было каждый раз не прописывать данные для подключения к основной БД.

Далее, я просто передал эти аргументы в функцию pymysql.connect эта функция вернула в переменную connect объект соединения с БД. Далее, я при помощи инструкции return возвращаю этот объект соединения из функции в любой вызывающий эту функцию код.

Зачем это нужно? Это гораздо удобнее, чем авторизоваться в разных участках кода. Теперь, в любом месте приложения где нам нужно поработать с БД, мы можем сделать так:

```
from db import connection
db = connection.make()
```

Всё, теперь в переменной `db` лежит объект для работы с БД.

Создаём модуль для работы с таблицей статей

Давайте рассмотрим этот процесс на примере и реализуем все основные операции для нашей таблицы articles.
Создадим новый пакет (папку) `articles`, в ней модуль `article-model.py`, обратите внимание, `article-model.py`:

Обычно, модули которые содержат функции по работе с базой данных называют моделями (model), это удобно, потому что теперь нам становится ясно с первого взгляда, чем занимается данный модуль.

Работа с таблицей. CRUD

Общий принцип работы с БД

Взаимодействие с БД производится при помощи SQL запросов, аля:

```
INSERT INTO articles (title, content, author, date) VALUES ('Заголовок статьи 1', 'Содержание статьи 1', '64', '42949672
```

Запросов в MySQL существует большое количество и каждый тип запросов выполняет определённые задачи. Например, запрос указанный мной сверху буквально говорит БД: вставь в таблицу articles запись, со значениями `'Заголовок статьи 1'`, `'Содержание статьи 1'`, `'64'`, `'4294967295'` в столбцы `title`, `content`, `author`, `date`.

По такому принципу и происходит работа с БД.

Общий принцип работы с pymysql

1. Открываем соединение с БД
2. Получаем объект для управления БД
3. Пишем SQL запрос, не передавая в него данные
4. Передаём данные в SQL запрос (подготовленные запросы)
5. Отправляем запрос на сервер MySQL
6. Сохраняем изменения
7. Закрываем соединение с БД

Create. Создать новую запись

Для записи в БД используют оператор INSERT. Синтаксис записи запроса на запись данных в таблицу выглядит следующим образом:

```
INSERT INTO имя_таблицы (список_столбцов) VALUES (значения_столбцов)
```

Пример для нашей таблицы:

```
INSERT INTO articles (title, content, author, date) VALUES ('Заголовок статьи 1', 'Содержание статьи 1', '64', '4294967
```

Объявим в файле `article.py` функцию `create`:

```
from db import connection


def create(title, content, author, date):
    # Открытие соединения с БД
    db = connection.make()
    try:
        # Объект взаимодействия с БД
        cursor = db.cursor()
        # Сам SQL-запрос
        sql = "Insert into articles (title, content, author, date) values (%s, %s, %s, %s)"
        # Выполняем запрос
        cursor.execute(sql, (title, content, author, date))
        # Сохраняем изменения в БД
        db.commit()
        # Возвращаем результат успешного выполнения
        return True
    except Exception as error:
        # Возвращаем содержание ошибки
        return error
    finally:
        # Закрытие соединения с БД
        db.close()
```

Что происходит в данном коде:

1. Создаём соединение с БД:

```
db = connection.make()
```

Отлично, теперь мы можем отправлять запросы к БД. Любой запрос к БД необходимо осуществлять в конструкции try..except..finally.

2. Вызываем модуль, при помощи которого будем взаимодействовать с БД:

```
# Объект взаимодействия с БД
cursor = db.cursor()
```

3. Формируем SQL запрос, который будет отправлен в БД:

```
sql = "Insert into articles (title, content, author, date) values (%s, %s, %s, %s)"
```

Формируем стандартную запись insert запроса для БД, с одной оговоркой, вместо значений, которые мы должны направить в БД, мы вставляем символы `%s` , это нужно для того, чтобы обезопасить запрос и не напрямую подставлять данные из переменных в строку, а пропустить их через специальный механизм, который заменит эти символы на данные, и очистит эти данные от возможных опасных символов.

3. Подготавливаем и выполняем наш запрос:

```
cursor.execute(sql, (title, content, author, date))
```

То, о чём шла речь в прошлом шаге. Функция execute не только отправляет запрос к БД, но ещё и формирует SQL-запрос в безопасном для БД виде. Первым аргументом, мы передаём сам запрос, с символами для замены `%s` , а вторым кортеж с данными, которые заменят символы «%s» в том же порядке.

4. Сохраняем изменения в БД:

```
db.commit()
```

5. Возвращаем результат в случае успешного выполнения:

```
return True
```

В противном случае, исключение будет выброшено раньше, чем дойдёт до инструкции `return`.

6. Обрабатываем ошибку. Если Ошибка произошла, возвращаем сообщение об ошибке:

```
except Exception as error:
    return error
```

7. Закрываем соединение в блоке `finally`:

```
finally:
    db.close()
```

Запустим модель:

```
from articles import article_model

print(article_model.create("Title", "Hello, world!", 5, 3423542543))
```

Выводит:

> True

В базе данных новая запись:

id	title	content	author	date
4	Updated Title	Updated Content	555	17 485 484

Ещё раз полный листинг:

```
def create(title, content, author, date):
    # Открытие соединения с БД
    db = connection.make()
    try:
        # Объект взаимодействия с БД
        cursor = db.cursor()
        # Сам SQL-запрос
        sql = "Insert into articles (title, content, author, date) values (%s, %s, %s, %s)"
        # Выполняем запрос
        cursor.execute(sql, (title, content, author, date))
        # Сохраняем изменения в БД
        db.commit()
        # Возвращаем результат успешного выполнения
        return True
    except Exception as error:
        # Возвращаем содержание ошибки
        return error
    finally:
        # Закрытие соединения с БД
        db.close()
```

Метод read_one. Получить одну конкретную запись

Для получения данных из таблицы используют оператор `SELECT`. Запрос `SELECT` пишется по следующему шаблону:

```
SELECT (список_столбцов) FROM (имя_таблицы)
```

Пример для нашей таблицы:

```
SELECT * FROM articles
```

Символ «`*`» означает буквально «вернуть все столбцы».

Оператор WHERE

Оператор указывающий условие запроса, например, при помощи него можно указать, что нам нужно вывести запись с конкретным `id`.

Пример для нашей таблицы:

```
SELECT * FROM articles where id=5
```

Дословно, выбрать все записи из таблицы `articles`, где `id` равно `5`.

Таким образом, БД вернёт нам все записи у которых `id = 5`, но так как столбец `id` у нас содержит только уникальные значения, будет возвращён только один элемент (если элемент с таким `id` существует).

Объявим функцию `read_one`. Данная функция принимает один аргумент, `id` статьи, которую необходимо вернуть:

```
def read_one(article_id):
    # Открытие соединения с БД
    db = connection.make()
    try:
        # Объект взаимодействия с БД
        cursor = db.cursor()
        # Сам SQL-запрос
        sql = "SELECT * FROM articles where id=%s"
        # Выполняем запрос
        cursor.execute(sql, (article_id))
        # Возвращаем словарь с заданной записью
        return cursor.fetchone()
    except Exception as error:
        # Возвращаем содержание ошибки
        return error
    finally:
        # Закрытие соединения с БД
        db.close()
```

Логика работы функции идентична прошлой, с двумя оговорками. Во-первых, мы не сохраняем изменения (потому что мы ничего не изменяли). Во-вторых, мы возвращаем уже не логическое значение, а запись, которая нам была возвращена из БД, при помощи функции `fetchone`. Данную функцию используют, когда нужно вернуть одно значение из БД.

Выполним код:

```
print(article_model.read_one(4))
```

Выводит:

> {'id': 4, 'title': 'Title', 'content': 'Hello, world!', 'author': 5, 'date': 3423542543}

Метод read_many. Возвращаем несколько записей

Объявим функцию `read_many`. Данная функция принимает один аргумент, количество статей которое необходимо вернуть из БД:

```
def read_many(how_many):
    # Открытие соединения с БД
    db = connection.make()
    try:
        # Объект взаимодействия с БД
        cursor = db.cursor()
        # Сам SQL-запрос
        sql = "SELECT * FROM articles"
        # Выполняем запрос
        cursor.execute(sql)
        # Возвращаем словарь с заданным количеством записей
        return cursor.fetchmany(how_many)
    except Exception as error:
        # Возвращаем содержание ошибки
        return error
    finally:
        # Закрытие соединения с БД
        db.close()
```

Логика выполнения не изменилась. Но теперь мы не передаём никаких данных, мы просто читаем записи из БД и возвращаем их через функцию `fetchmany`, данная функция возвращает заданное количество записей из БД.

Выполним данный метод:

```
print(article_model.read_many(2))
```

Выводит:

> [{'id': 4, 'title': 'Title', 'content': 'Hello, world!', 'author': 5, 'date': 3423542543}, {'id': 5, 'title': 'Title 2', 'content': 'Hello world 2', 'author': 45, 'date': 1515454668}]

Метод read_all. Возвращаем все записи из БД

Объявим метод `read_all`, который не принимает никаких аргументов:

```
def read_all():
    # Открытие соединения с БД
    db = connection.make()
    try:
        # Объект взаимодействия с БД
        cursor = db.cursor()
        # Сам SQL-запрос
        sql = "SELECT * FROM articles"
        # Выполняем запрос
        cursor.execute(sql)
        # Возвращаем словарь со всеми записями
        return cursor.fetchall()
    except Exception as error:
        # Возвращаем содержание ошибки
        return error
    finally:
        # Закрытие соединения с БД
        db.close()
```

Принцип остался тем же, за исключением того, что записи возвращаются через функцию `fetchall`. Данная функция не принимает никаких аргументов, она просто возвращает все записи, вернувшиеся в запросе.

Метод update. Обновляем запись в БД

Обновление данных происходит при помощи команды `UDATE`. Она производится следующим образом:

```
UPDATE название_таблицы SET столбец1 = значение1, столбец2 = значение2 WHERE условие_обновления
```

Пример для таблицы articles:

```
update articles set title='Новый заголовок статьи 1', content='Новый содержание статьи 1', author='64', date='429496729
```

Объявим метод update. Данный метод принимает те же, аргументы что и метод create, но в дополнение передаётся и id записи, чтобы мы могли указать какую конкретно запись должны обновить:

```
def update(article_id, title, content, author, date):
    # Открытие соединения с БД
    db = connection.make()
    try:
        # Объект взаимодействия с БД
        cursor = db.cursor()
        # Сам SQL-запрос
        sql = "update articles set title=%s, content=%s, author=%s, date=%s where id=%s"
        # Выполняем запрос
        cursor.execute(sql, (title, content, author, date, article_id))
        # Сохраняем изменения
        db.commit()
        # Возвращаем результат
        return True
    except Exception as error:
        # Возвращаем содержание ошибки
        return error
    finally:
        # Закрытие соединения с БД
        db.close()
```

Так как мы изменяем содержимое БД, нам нужно сохранить изменения при помощи уже известного нам метода: `db.commit()`

Запустим код:

```
print(article_model.update(4, "Updated Title", "Updated Content", 555, 17485484))
```

Вывод:

> True

Посмотрим на изменения в БД:

id	title	content	author	date
4	Updated Title	Updated Content	555	17 485 484

Метод delete. Удаление записи

Предположим, у вас есть запись, которую необходимо удалить. Это делается при помощи оператора `DELETE`, по следующему шаблону:

```
DELETE FROM имя_таблицы WHERE условие_удаления
```

Пример для таблицы `articles`:

```
DELETE FROM articles WHERE id=5
```

Создадим метод `delete`, который принимает один аргумент – `id` записи которую необходимо удалить:

```
def delete(article_id):
    # Открытие соединения с БД
    db = connection.make()
    try:
        # Объект взаимодействия с БД
        cursor = db.cursor()
        # Сам SQL-запрос
        sql = "delete from articles where id = %s"
        # Выполняем запрос
        cursor.execute(sql, (article_id))
        # Сохраняем изменения
        db.commit()
        # Возвращаем результат
        return True
    except Exception as error:
        # Возвращаем содержание ошибки
        return error
    finally:
        # Закрытие соединения с БД
        db.close()
```

После удаления записи нужно также сохранить изменения в БД при помощи метода commit.

Понятие вёрстки

Содержание урока

Прежде чем говорить о вёрстке, давайте поговорим о базовом для вёрстке понятии – веб странице. В каноническом смысле, веб страница это эокумент имеющий свой собственный адрес, расположенный в сети интернет, состоящий из гипертекста. Веб-страница содержит в себе текст, изображения, медиафайлы и т.д.

Просмотр веб-страницы осуществляется через веб-браузер.

Если говорить более понятным языком, то веб-страница, это документ, содержащий в себе HTML-разметку, внутри которой содержится текст и медиафайлы.

Принцип вёрстки: взаимодействие HTML и CSS

Возможно, не все знают, но веб-страница - это не картинка, которую вы видите в браузере. Веб страница, это код, который при попадании в браузер, этот самый браузер обрисовывает на основе кода страницы.

Код страницы состоит из:

1. HTML-разметки
2. CSS-стилей

Давайте посмотрим, как примерно выглядит код страницы:

HTML-разметка

```
<блок имя="верхнее_меню">ссылка ссылка ссылка ссылка
<блок имя="блок_с_текстом"текст текст текст текст текст
```

CSS-стили

```
.верхнее_меню {
    ширина: 500 пикселей;
    высота: 150 пикселей;
    цвет_фона: чёрный;
    цвет_текста: белый;
}

.блок_с_текстом {
    ширина: 500 пикселей;
    высота: автоматическая;
    цвет_фона: белый;
    цвет_текста:
    чёрный;
}
```

Попадая в браузер, документ как бы говорит ему: "распечатай два блока". Первому нарисуй ширину 500, а высоту в 150, сделай его чёрным, а текст внутри него - белым. Второй пусть будет по ширине 500 пикселей, по длине пусть выравнивается по содержимому, цвет фона пусть будет белым, а цвет текста чёрным.

В итоге браузер отрисует примерно такую страницу:

Вёрстка представляет собой совокупность разметки и стилей, определяющих внешний вид этой разметки.

Разметка определяет содержание, блоки, из которых состоит страница и их общее поведение. А вёрстка определяет форму, внешний вид этих блоков.

В рамках данного курса, мы лишь в общих чертах коснёмся темы вёрстки, чтобы иметь общее понятие о том, из чего строится страница и как её парсить.

Основы HTML

Содержание урока

Понятие веб-сервера

Итак, мы уже говорили что вёрстка состоит из HTML-разметки и CSS-стилей. В свою очередь, HTML-разметка состоит из HTML-тегов и текста.

HTML-теги

Любая веб-страница состоит из элементов, именно теги позволяют создавать эти элементы.

Например, элементу "текст", соответствует тег p. Элементу "ссылка" соответствет тег a. Элементу "список" соответствует элемент ul. Каждому структурному элементу в разметке соответствет какой-то тег.

Теги бывают парными и непарными.

К парнымтегам относятся теги, имеющие пару из открывающего и закрывающего тега, например:

```
<p>Здесь могла бы быть ваша реклама</p>
```

Здесь, - открывающий тег, а - закрывающий тег.

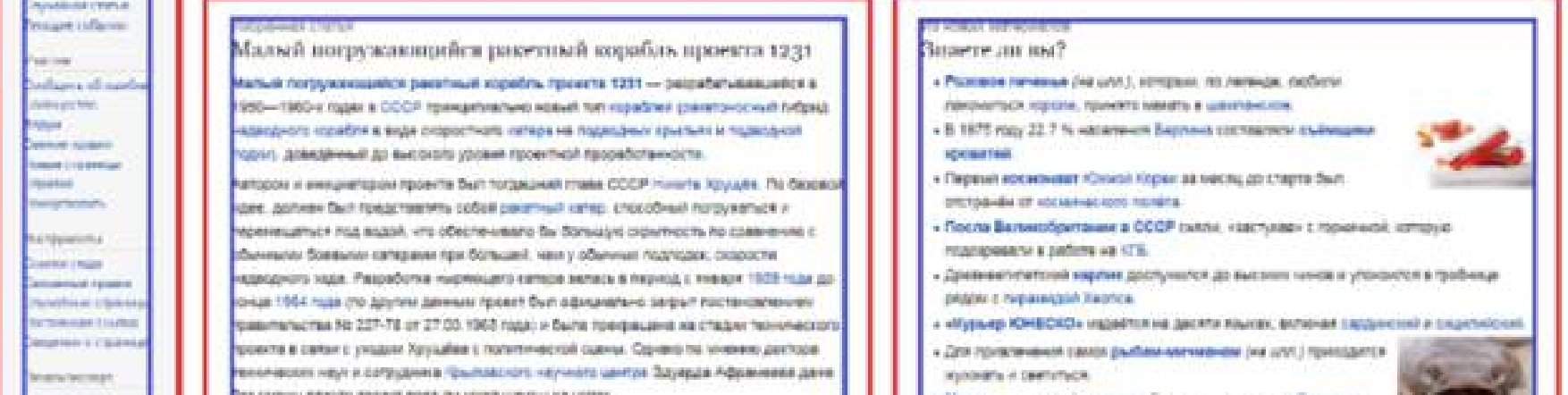
Семантика HTML такова, что в парном теге, закрывающий тег выглядит как открывающий, только со слешем / перед именем тега:

К непарным относятся некоторые теги, не имеющие закрывающего тега, например, тег
. Этот тег выполняет перенос строки.

Теги можно вкладывать друг в друга, например:

```
<p>Здесь могла бы быть <br>ваша реклама</p>
```

В данном случае, браузер отрисует текстовый блок, и с помощью тега
 словосочетание "ваша реклама" будет перенесено на новую строку:



Из чего состоит страница. Как правило страница построена из, из блоков и их содержимого:

Красной рамкой обведены блоки, синим - содержимое блоков. Таки образом, вёрстка страницы предполагает разметку основных блоков и встраивание в эти блоки содержимого.

В упрощённом виде, разметка основных блоков могла бы выглядеть так:

```
<aside>
 <img src="/logo.png" alt="" class="logo">
 <ul>
 <li>Меню</li>
 <li>Меню</li>
```

```
 <li>Меню</li>
 </ul>
</aside>
<div class="content">
 <div class="hero">
 <h1>Добро пожаловать в Википедию</h1>
 <a href="#">Создать статью</a>
 </div>
 <div class="faw_article">
 <h2>Избранная статья</h2>
 <p>Малый погружающийся ракетный корабль проекта 1231 — разрабатывавшийся в 1950—1960-х</p>
 </div>
 <div class="new_article">
 <h2>А знаете ли вы?</h2>
 <ul>
 <li>Розовое печенье (на илл.), которым, по легенде, любили лакомиться короли, принято макать в шампанское.</li>
 <li>В 1875 году 22,7 % населения Берлина составляли съёмщики кроватей.</li>
 </ul>
 </div>
</div>
```

Обратите внимание на данный код, здесь теги aside и div определяют блоки, т.е. эти блоки строят скелет страницы. А теги: `h1`, `h2`, `p`, `ul`, `li`, `img` - определяют содержимое и зависят от тегов, определяющих блоки, так как находятся внутри них.

Ознакомиться с основными тегами и их описанием можно здесь: https://html5book.ru/html-html5

Таким образом, при вёрстке html-страниц, сперва мы строим каркас (сетку) страницы, а затем этот каркас заполняем содержимым.

А теперь, после того как мы разобрались с принципом вёрстки, рассмотим из чего должен HTML-документ.

В первую очередь, при создании HTML-документа, вы должны создать файл, с расширением `*.html`, например: `index.html`

Внутри каждого HTML-файла должен быть указан тип документа, в нашем случае, в соответствии с последними стандартами декларации HTML5, необходимо указать следующий тип:

```
<!DOCTYPE html>
```

Далее, все html теги необходимо записывать внутри тега <html>, таким образом:

```
<!DOCTYPE html>
<html></html>
```

Внутри тега html находится всё остальное содержимое страницы. Далее, внутри парного тега html, указывается тег <head>. Этот тег содержит техническую информацию о странице, в частности: заголовок сраницы, кодировку, описание, подключение стилей страницы и т.д.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Заголовок документа</title>
        <link rel="stylesheet" href="/style.css">
    </head>
</html>
```

Тег <meta> описывает содержимое страницы, в частности для поисковых систем, позволяет указать ключевые слова, описание страницы, кодировку и т.д.

Подробнее о этом незаменимом теге: https://html5book.ru/osnovy-html/#part5

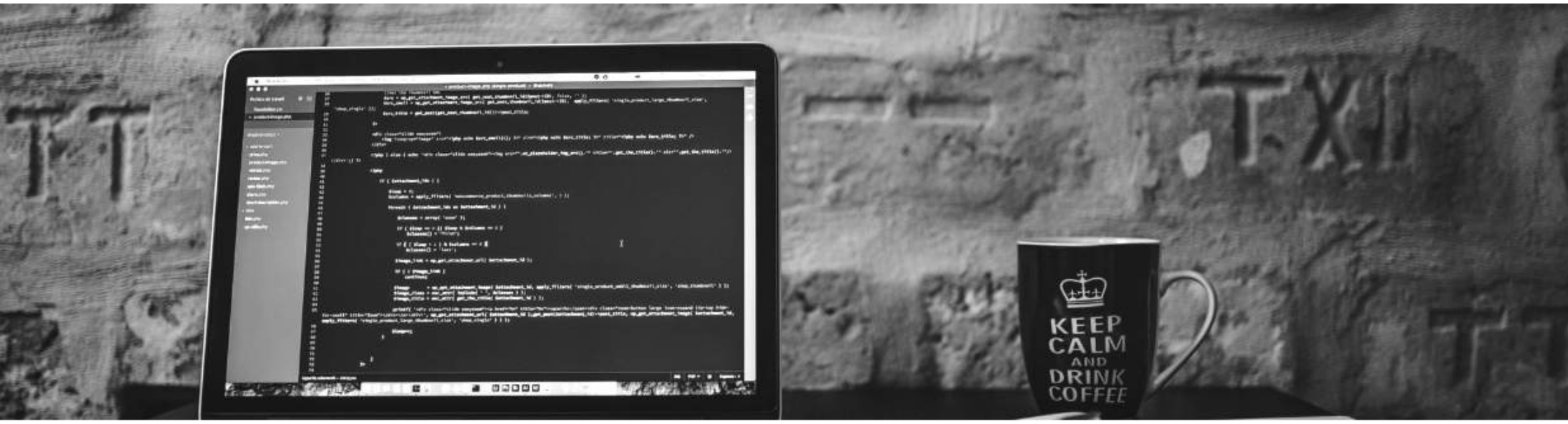
Тег <title> определяется браузером и поисковыми системами как тег заголовка, таким образом любой текст длиной от 0 до 60 символов, размещённый внутри тега `title` определяется браузером и поисковыми системами как заголовок страницы.

Тег <link> устанавливает ссылку на внешний ресурс, таким образом связывая страницу и внешний ресурс. Он используется для подключения к странице шрифтов, файлов стилей и т.д. В отличии от тега <a> тег <link> не создаёт ссылок и может указываться только внутри тега `head`.

После тега <head> идёт тег <body>. В данном теге и размещается всё остальное содержимое документа, если объединить все примеры что я привёл в данном уроке, то получится валидная HTML-разметка:

```
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Заголовок документа</title>
 <link rel="stylesheet" href="/style.css">
 </head>
     <body>
         <aside>
             <img src="/logo.png" alt="" class="logo">
                 <ul>
                     <li>Меню</li>
                     <li>Меню</li>
                     <li>Меню</li>
                 </ul>
         </aside>
     <div class="content">
         <div class="hero">
             <h1>Добро пожаловать в Википедию</h1>
             <a href="#">Создать статью</a>
         </div>
         <div class="faw_article">
             <h2>Избранная статья</h2>
             <p>Малый погружающийся ракетный корабль проекта 1231 — разрабатывавшийся в 1950—1960-
             </p>
         </div>
         <div class="new_article">
             <h2>А знаете ли вы?</h2>
             <ul>
                 <li>Розовое печенье (на илл.), которым, по легенде, любили лакомиться короли, принято макать в шампанское.</li>
                 <li>В 1875 году 22,7 % населения Берлина составляли съёмщики кроватей.</li>
             </ul>
         </div>
        </div>
     </body>
 </html>
```

Итак, мы разобрались с тем, из чего состоит валидный HTML-документ, поняли, из каких этапов состоит вёрстка страниц, осталось разобраться с CSS-стилями.



Основы CSS

Содержание урока

Понятие CSS-стилей

CSS-стили используются для описания внешнего вида HTML-тегов. CSS-стили определяют внешний вид HTML-элементов при помощи свойств.

Если определить краткую модель, по изменению внешнего вида тега при помощи CSS-стилей, то она выглядит так:

1. Элементу или группе элементов присваивается метка (селектор)
2. В CSS файле прописывается инструкция для этой метки (селектора)

Таким образоми, в CSS-файле указывается, что в отношении определённого селектора необходимо присвоить конкретные визуальные свойства.

Например, у нас есть тег <p> в котором расположен некоторый текст. Мы хотим, чтобы фон у тега <p> был серого цвета, цвет текста в теге <p> был синим, а размер текста в теге <p> был 16 пикселей, то мы можем сделать так:

```
//index.html
<!DOCTYPE html>
<html>
    <head>
         <meta charset="UTF-8">
         <title>Заголовок документа</title>
         <link rel="stylesheet" href="/style.css">
    </head>
    <body>
        <p>Здесь могла бы быть ваша реклама</p>
    </body>
</html>
//style.css
p {
    background: gray;
    color: black;
    font-size: 16px;
}
```

Здесь мы видим, что файл со стилями был подключён к документу через тег `link`, затем, в самом файле `style.css` мы указали условия для тега `p`, которые буквально определили внешний вид для этого тега.

CSS-селекторы

В основном в качестве селектора выступет:

1. HTML-тег
2. Класс элемента
3. Идентификатор

HTML-тег как селектор. Для присваивания стилей к конкретному тегу в CSS-стилях используется его имя:

```
p {
   color: black;
}

a {
   font-size: 15px;
}
```

Но нужно учесть, что когда в качестве селектора выступает непосредственно тег, применение стилей к тегу приводит к тому, что стиль присваивается ко ВСЕМ тегам на странице.

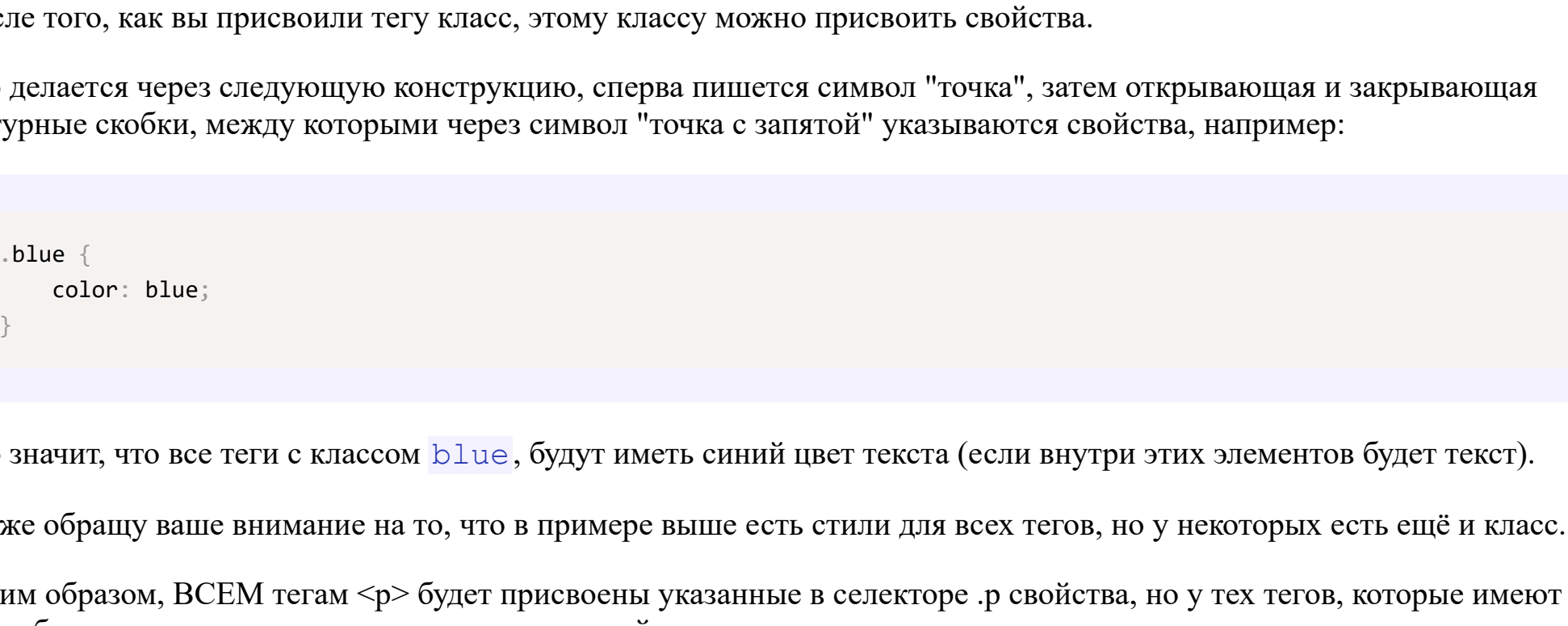
Например, в примере выше, ВСЕ теги a будут иметь размер шрифта `15px`.

Класс элемента — это идентификатор, который можно присвоить любому элементу или нескольким элементам, например:

```
//index.html
<!DOCTYPE html>
<html>
     <head>
         <meta charset="UTF-8">
         <title>Заголовок документа</title>
         <link rel="stylesheet" href="/style.css">
     </head>
    <body>
         <p>Здесь могла бы быть ваша реклама</p>
         <p class="green">Здесь могла бы быть ваша ЗЕЛЁНАЯ реклама</p>
```

```
        <p class="blue">Здесь могла бы быть ваша СИНЯЯ реклама</p>
        <p class="green">Здесь могла бы быть ваша ЗЕЛЁНАЯ реклама</p>
    </body>
 </html>
//style.css
p {
    background: yellow;
    color: black;
    font-size: 16px;
}
.green {
    color: green;
}
.blue {
    color: blue;
}
```

Посмотрим на результат:



Как можно заметить, чтобы присвоить тому или иному тегу класс, необходимо прописать у открывающего тега атрибут class, поставить знак равно и в кавычках указать название на кириллице, например: , .

После того, как вы присвоили тегу класс, этому классу можно присвоить свойства.

Это делается через следующую конструкцию, сперва пишется символ "точка", затем открывающая и закрывающая фигурные скобки, между которыми через символ "точка с запятой" указываются свойства, например:

```
.blue {
    color: blue;
}
```

Это значит, что все теги с классом `blue`, будут иметь синий цвет текста (если внутри этих элементов будет текст).

Также обращу ваше внимание на то, что в примере выше есть стили для всех тегов, но у некоторых есть ещё и класс.

Таким образом, ВСЕМ тегам будет присвоены указанные в селекторе .p свойства, но у тех тегов, которые имеют класс будут перезаписываться установленные свойства.

То есть, в случае со строкой:

```
<p class="green">Здесь могла бы быть ваша ЗЕЛЁНАЯ реклама</p>
```

Фоновый цвет этого блока с текстом будет жёлтый, а размер текста будет составлять 16px у всех элементов, даже у тех, которыми присвоились стили. Но цвет текста будет зелёный, несмотря на то, что в стиле для тега <p>, был указан чёрный цвет.

Это связано с тем, что общие стили переопределяются более частными.

Грубо говоря, стили "смешиваются", то есть, если одному и тому же элементу присваиваются стили через разные селекторы, например через тег и через класс, то стили обоих селекторов будут присвоены этому элементу. Одному элементу можно присвоить сколько угодно классов, для этого в атрибуте class прописывайте название классов через пробел: <p class="one_class two_class three_class">

Идентификатор элемента. Идентификатор элемента ведёт себя также как и класс, с 2 принципиальными отличиями:

1. Идентификатор должен быть уникальным на страницы, то есть присваиваться только для 1 элемента. В отличии от класса, который можно присвоить к любому количеству элементов.
2. Идентификатор более специфичен, поэтому, если к одному и тому же элементу присвоен и класс и идентификатор и в стилях обоих есть одно и то же свойство, то сработает свойство идентификатора, например:

```
<p class="green" id="blue">Здесь могла бы быть ваша СИНЯЯ реклама</p>

//style.css
#blue {
   color: blue;
}
.green {
   color: green ;
}
```

В результате, текст будет синим, а не зелёным.

Как вы можете заметить, при присваивании идентификатора элементу, необходимо указать сперва атрибут `id`, затем знак `=` и в кавычках указать имя идентификатора, например: <p id="blue">

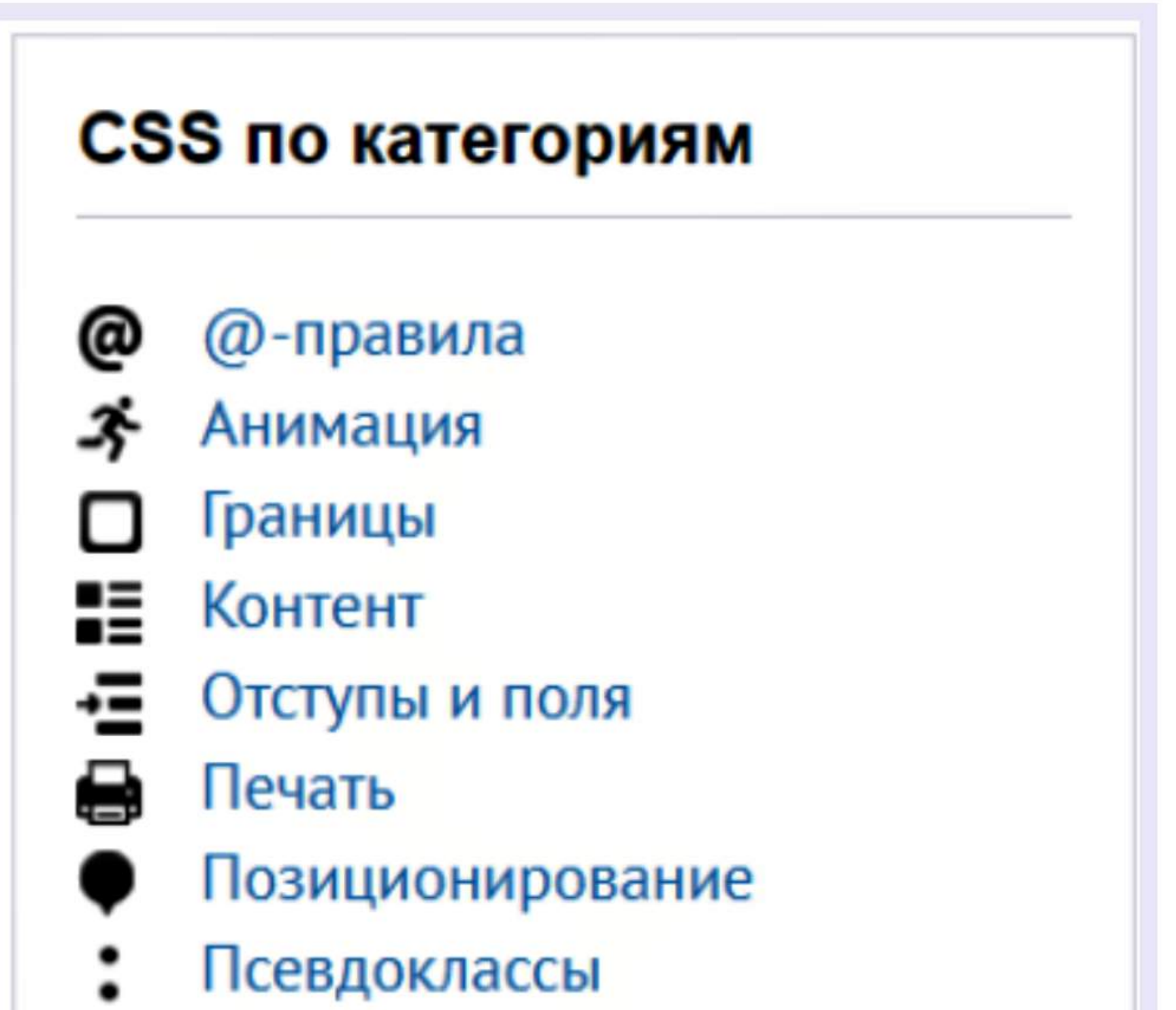
CSS-свойства

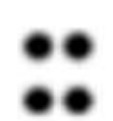
Нам осталось поговорить непосредственно про свойства. В сущности, CSS состоит из инструкций, каждая инструкция это совокупность двух составляющих:

1. Селектор, указывающий на тег/класс/идентификатор и т.д. например: .green, p, #blue
2. Свойства, которые определяют внешний вид элемента, например: color: blue, background: yellow, color: black

CSS-свойств существует великое множество, приводить их не имеет смысла, поэтому для того, чтобы ознакомиться с необходимыми вам свойствами советую обратитья: https://webref.ru/css

В особенности посмотрите на раздел:

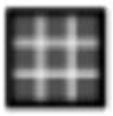




Псевдоэлементы

Размеры

Списки



Таблицы

Текст и шрифт

Флексы

Форматирование

Формы

Цвет и фон

Сейчас вам важно уяснить принцип их действия. Поэтому ещё раз повторим, указав определённый селектор и присваивая к нему свойства, эти свойства определят внешний вид всех элементов, с этим селектором.

Например, если на странице существует 20 тегов <p> с текстом внутри, к 10 тегам <p> вы присвоили класс `.green` и указали в css-свойствах зелёный цвет, для этого селектора, то все 10 тегов <p>, а точнее содержимое тегов (текст) будут иметь зелёный цвет.

Несмотря на то, что свойств великое множество, не все свойства работают с определёнными тегами, например, тегу <p> нельзя установить фиксированную ширину и высоту, а тегу - запросто.

Поэтому при вёрстке следует это учитывать.

Управление браузером. Selenium

Содержание урока

При работе со сторонними сайтами, а именно при написании автоматизации для них можно работать двумя способами.

Первый – отправлять http-запросы без браузера (об этом мы поговорим в следующем уроке), в таком случае – сайт отправляет вам чистый текст (в основном html-код, изображения, JSON), этот текст не интерпретируется браузером, и вам самостоятельно нужно его разбирать и извлекать из него данные.

Плюс первого варианта в том, что данные собираются и выполняются гораздо быстрее, так как работа происходит напрямую с сайтом и не используется посредник в виде браузера.

Минус первого варианта в том, что для того чтобы отправить или получить данные без браузера, приходится повозиться с составлением запроса, т.к. бразуер на основе исходного кода страницы легко может его составить, а вот в ручную извлекать параметры запроса и определять логику их формирования – сложно.

Второй – отправлять http-запросы через браузер, в этом случае сайт также возвращает вам некоторый текст, но теперь этот текст интерпретируется браузером и выводит на экран интерфейс сайта.

Плюс второго варианта в том, что отправка запроса на сервер через браузер реализуется гораздо проще, ведь, как мы помним из прошлого пункта – браузер формирует запросы за вас.

Минус второго варианта в том, что получение и отправка данных на сайт происходит гораздо медленнее.

В данном блоке уроков, мы разберём принцип автоматизации работы с веб-сайтами через браузер на основе `Selenium WebDriver`.

Selenium WebDriver

`Selenium WebDriver` – набор драйверов для управления различными браузерами, а также набор библиотек для различных языков программирования, при помощи которых происходит управление браузерами.

При помощи Selenium мы можем запустить реальный браузер, написать определённый сценарий (например, зайти на сайт, ввести логин и пароль, нажать на кнопку «Войти») и управлять им так, будто действия производятся реальным человеком. Selenium поддерживает работу с такими браузерами как: `Chrome`, `Opera`, `Firefox`, `InternetExplorer`, `Safari`, `Andriod`, `iOS`.

Selenium позволяет не просто управлять браузером, но также и делать его относительно уникальным. Мы можем установить для каждого запущенного браузера свой прокси-сервер, свой юзерагент, локаль, куки, реферер и многое другое.

И самое главное, Selenium позволяет нам запускать автоматизацию в фоновом режиме, то есть, наш бот будет работать с сайтом через браузер, но никаких окон браузера при этом вы не увидите, ваш бот будет работать как-бы в трее.s

Selenium это нечто вроде браузерного проекта в Zennoposter.

Установка Selenium на ПК

Содержание урока

Устанавливаем Selenium

Для того, чтобы управлять браузером через специальную библиотеку, его необходимо сперва скачать. Точнее, необходимо скачать специальный веб-драйвер, соответствующий тому браузеру, которым необходимо управлять.

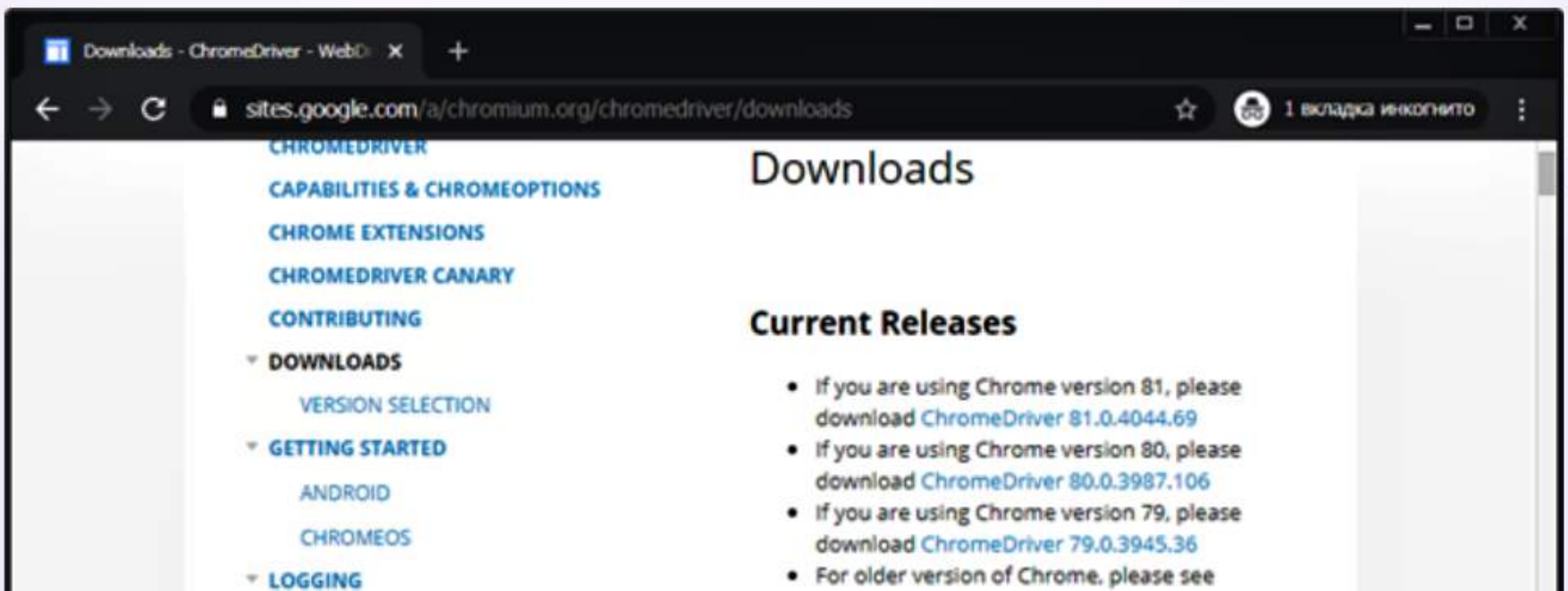
Скачать необходимый веб-драйвер можно по ссылкам:

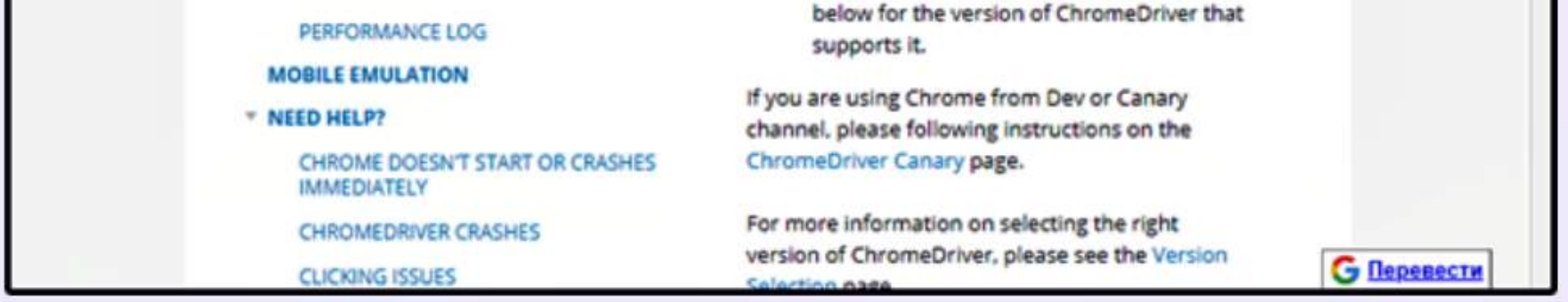
Chrome: https://sites.google.com/a/chromium.org/chromedriver/downloads
Edge: https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/
Firefox: https://github.com/mozilla/geckodriver/releases
Safari: https://webkit.org/blog/6900/webdriver-support-in-safari-10/

Установку (скорее, просто скачивание нужного драйвера) покажу на примере браузера Chrome, остальные драйвера ставятся по тому же принципу.

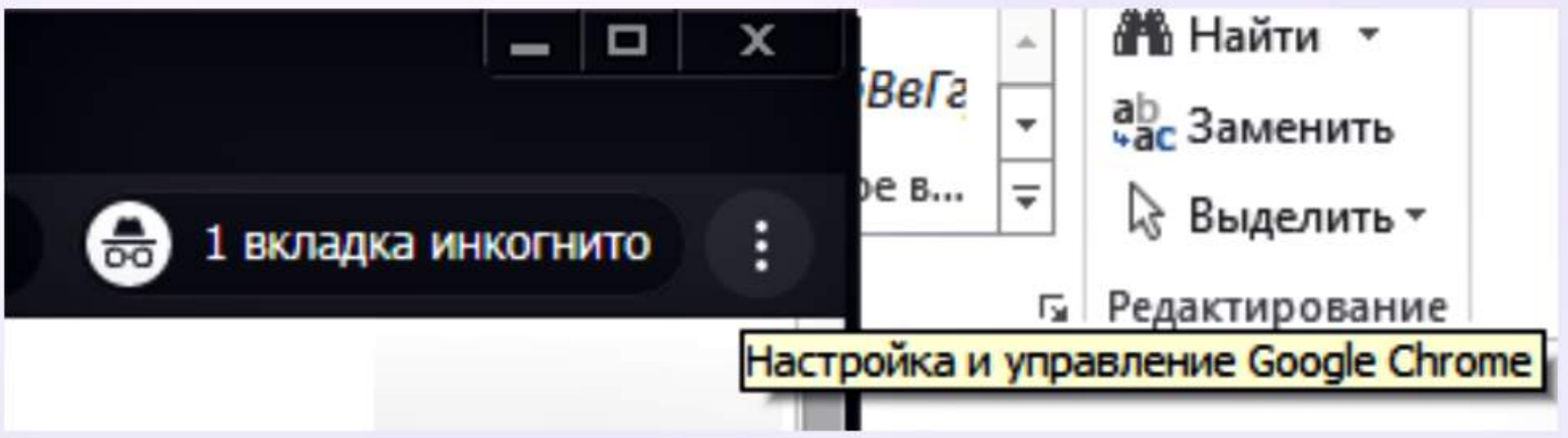
Переходим на сайт для скачивания драйвера:
https://sites.google.com/a/chromium.org/chromedriver/downloads

Видим разные версии драйвера:

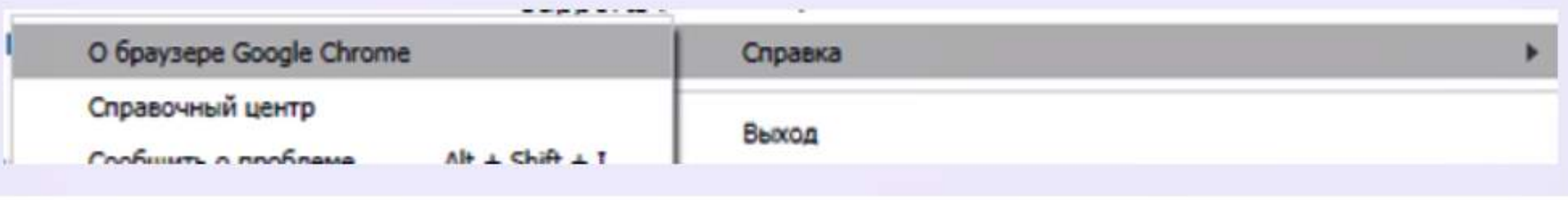




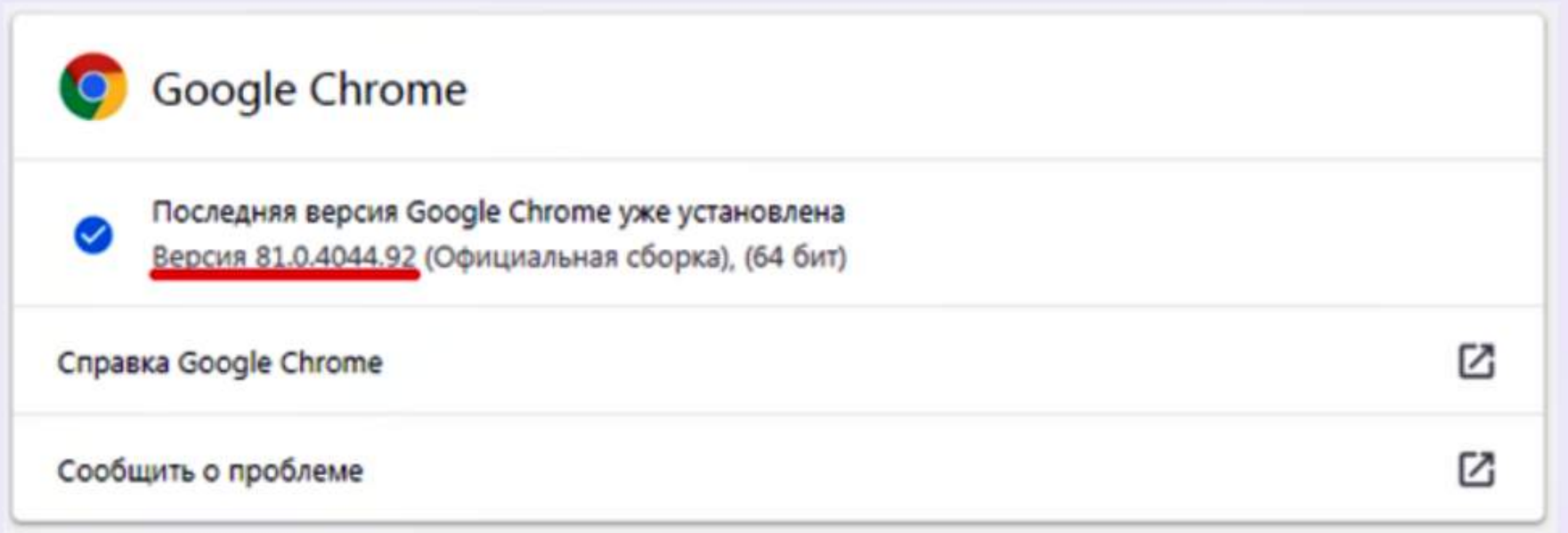
Какую версию следует выбрать? Для этого нажимаете кнопку «Настройка и управление Google Chrome»:



Раздел «Справка» -> «О браузере»:



Откроется окно, где будет указана версия вашего текущего браузера Chrome:



Эту версию драйвера и скачивайте:

Нажимаем на ссылку, открывается новая вкладка:

Name	Last modified	Size	ETag
Parent Directory		-	
chromedriver_linux64.zip	2020-03-17 16:16:51	4.73MB	11bc281b27db997b5045b376866b8ed5
chromedriver_mac64.zip	2020-03-17 16:16:53	6.69MB	2d27f0b1b4cdc9e7f2e535a88223efbb
chromedriver_win32.zip	2020-03-17 16:16:54	4.19MB	e6006040f914e704f591a6abdb3833ef
notes.txt	2020-03-17 16:25:31	0.00MB	adf2a9dacb0ae755b7328973b31271d2

Здесь кликаем по той версии драйвера, которая соответствует вашей ОС. В моём случае – это `windows`, соответственно, выбираю `win32`. Сохранится архив, с файлом драйвера:

Имя	Размер	Сжат	Тип
..			Папка с файлами
chromedriver.exe	8 124 416	4 393 341	Приложение

Этот файл мы будем использовать в следующем уроке.

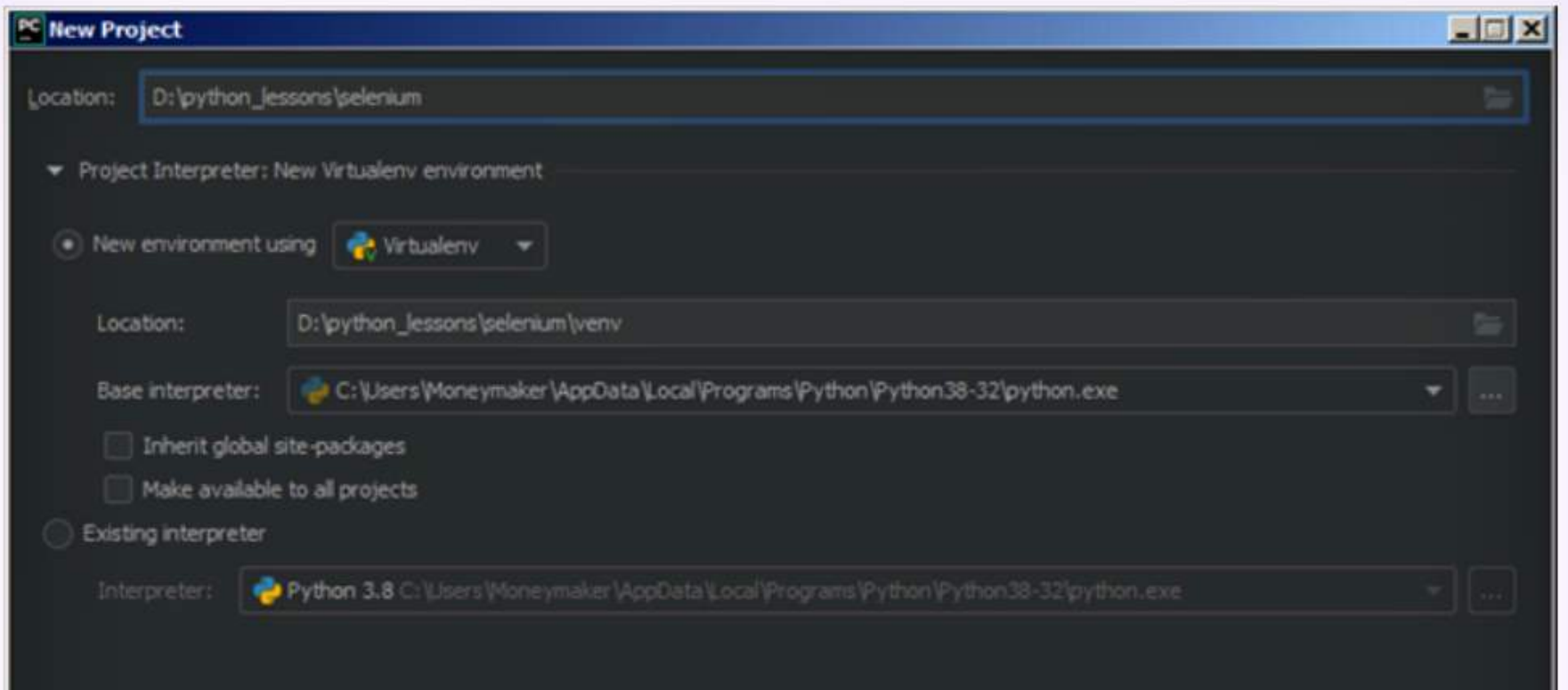
Подключение Selenium к Python проекту

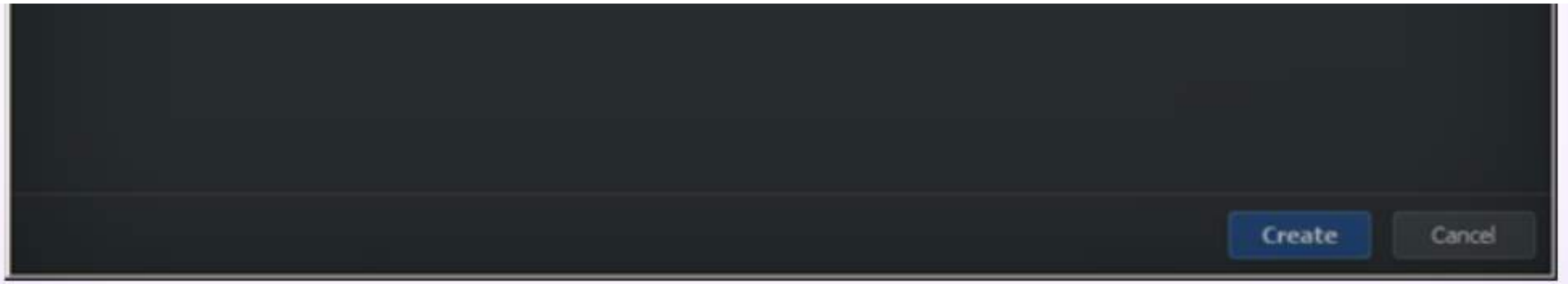
Содержание урока

Подготавливаем проект

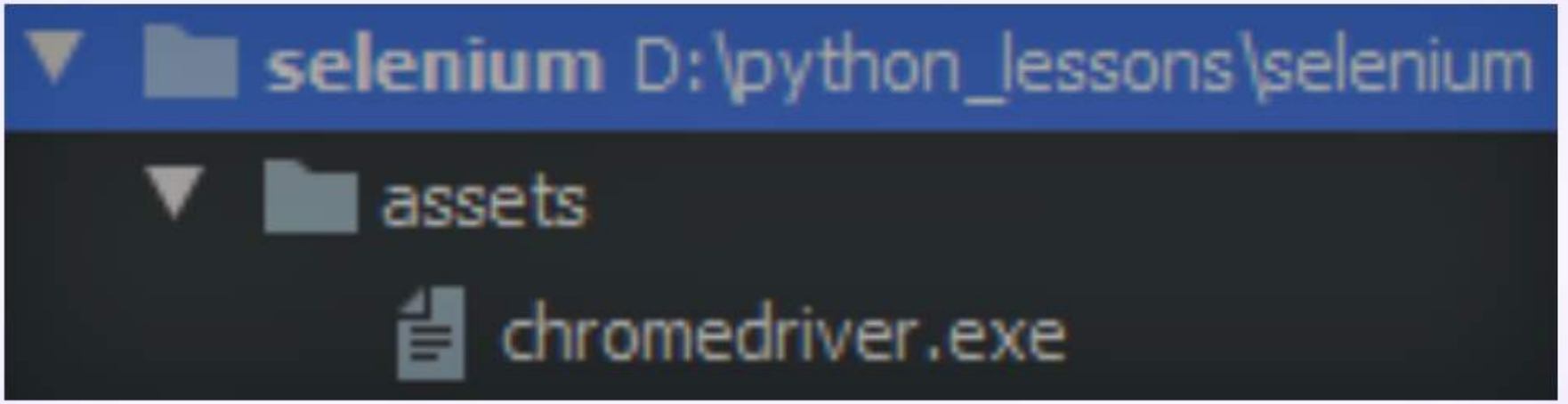
Для того, чтобы мы могли работать с Selenium через Python нам необходимо установить библиотеку. Но сперва, давайте создадим новую директорию и новый проект для неё, чтобы вся практика по Selenium была в отдельной папке, например:

Далее, создаём проект в PyCharm:





После создания проекта в корневой директории проекта создаём папку «assets» (в ней будут помещаться внешние ресурсы) в эту папку поместим файл со скачанным в прошлом уроке веб-драйвером:



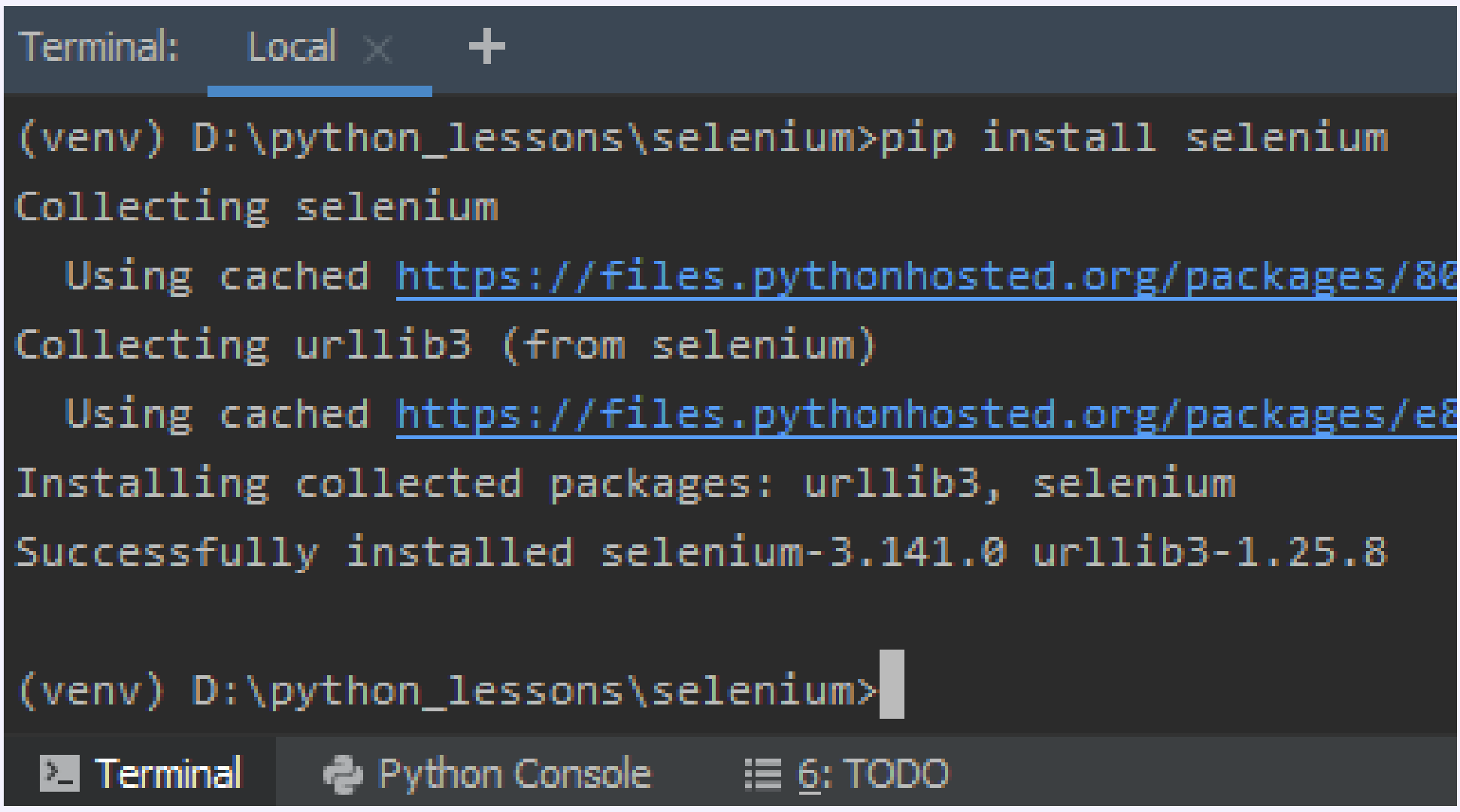
Устанавливаем библиотеку для управления Selenium

Отлично, мы поместили веб-драйвер в наш проект, теперь установим библиотеку selenium. Документация по библиотеке selenium для Python: https://selenium-python.readthedocs.io/

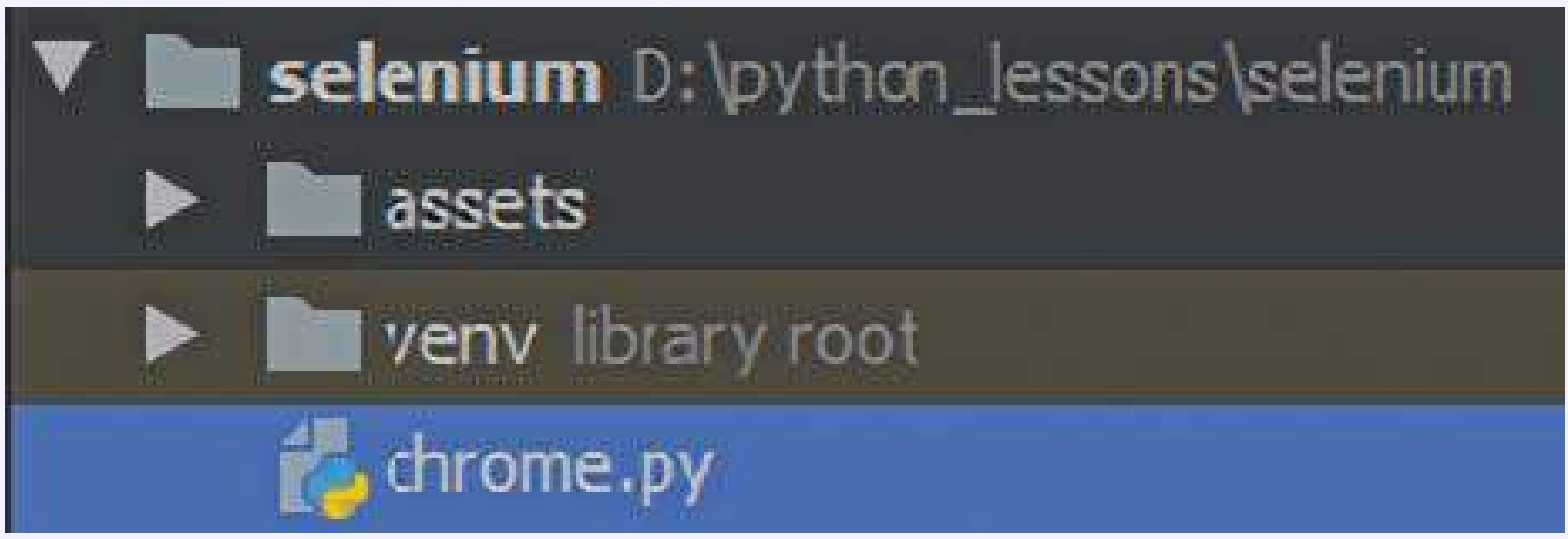
Для этого в терминале введём команду:

```
pip install selenium
```

Нажимаем «enter»:



Установка прошла успешно, теперь мы можем использовать `selenium` в нашем проекте. Создадим в корневой папке нашего проекта файл `chrome.py`:



Откроем файл и импортируем в него установленный модуль `selenium`, а точнее, модуль по управлению браузером `chrome`:

```
from selenium.webdriver import Chrome
```

Теперь нужно запустить наш браузер и получить к нему доступ. Для этого необходимо создать экземпляр модуля `webdriver`. Так как мы используем и импортируем Chrome, создаём объект веб-драйвера Chrome:

```
driver = Chrome(executable_path="assets/chromedriver.exe")
```

В качестве аргумента передаём путь до нашего файла с драйвером. Таким образом, мы как-бы указываем библиотеке, чем именно она будет управлять.

После того, как мы создали объект веб-драйвера и выполнили над ним некоторые действия, нам обязательно необходимо его закрыть, при помощи метода `close()`:

```
from selenium.webdriver import Chrome

# Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe")

# Некоторая инструкция
# Некоторая инструкция
# Некоторая инструкция
# Некоторая инструкция

# Закрываем браузер
driver.close()
```

В противном случае, запущенный браузер (или 100 браузеров, если вы захотите запустить 100 штук) может остаться незакрытым и сильно нагружать оперативную память ПК.

Управление страницей при помощи Selenium

Содержание урока

При создании объекта управления браузером, мы получаем ряд методов, которые и реализуют механизмы управления. Давайте познакомимся с основными методами подробнее. Полная документация по методам Selenium в Python: https://selenium-python.readthedocs.io/

Перейти на страницу. Метод get

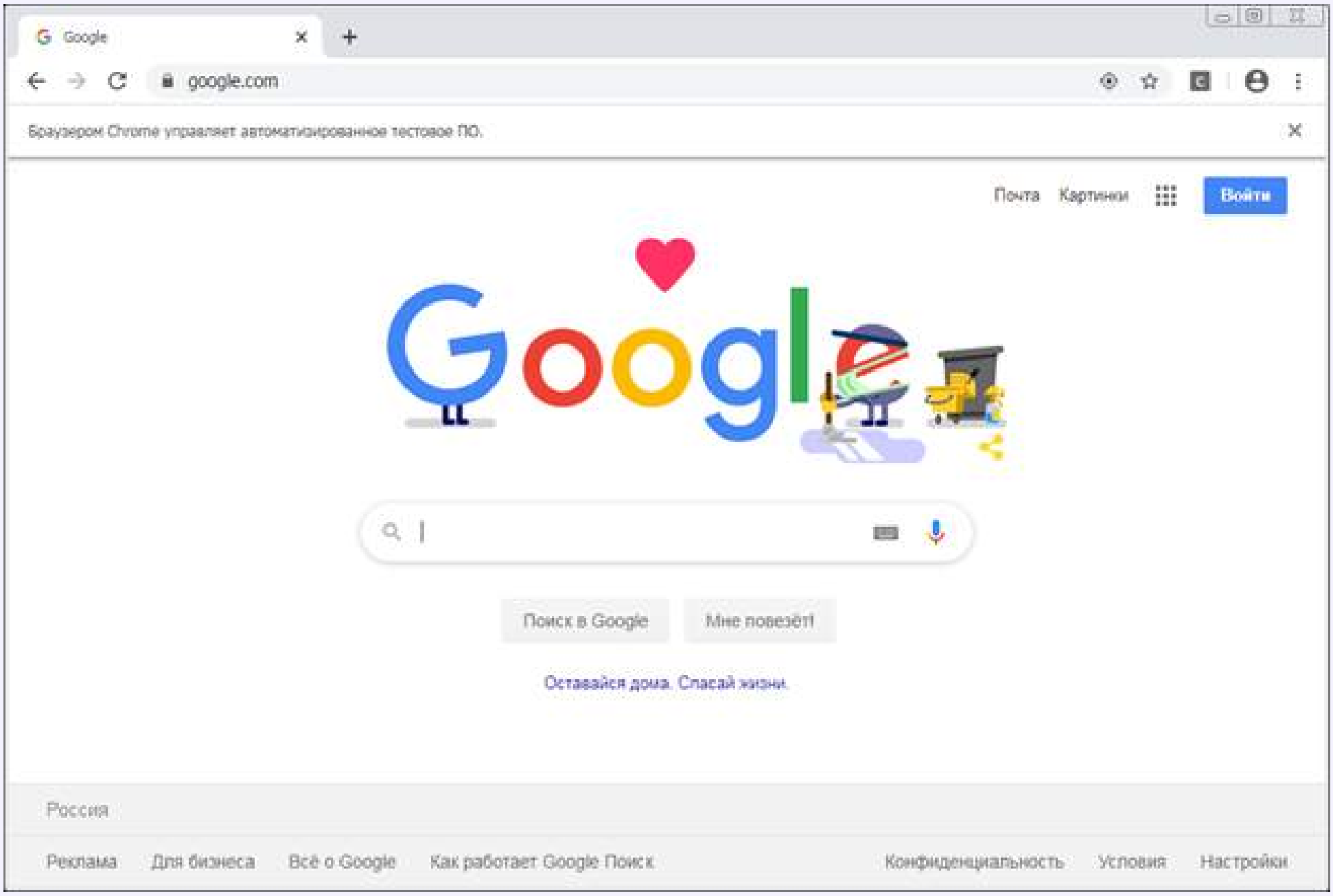
Давайте попробуем перейти на главную страницу ПС Google.

Для этого воспользуемся функцией get. Функция get принимает в качестве параметра ссылку на страницу, на которую необходимо перейти. При вызове этого метода, веб-драйвер будет ждать полной загрузки страницы. Однако, если на странице какой-то контент подгружается динамически, через AJAX – драйвер не будет ждать загрузки. О том, как решить эту проблему поговорим позже.

Напишем следующий скрипт:

```
from selenium.webdriver import Chrome
import time


# Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe")
# Открываем ссылку, при помощи метода get
driver.get("https://www.google.com/")
# Ожидаем 5 секунд
time.sleep(5)

# Закрываем браузер
driver.close()
```

Запустив этот код, мы увидим, как откроется наш браузер, а затем, он перейдёт по ссылке, «подумает» 5 секунд, а затем закроется:



Поиск элемента на странице. Группа методов find_element

Пожалуй, второй по популярности метод, а точнее, группа методов – это группа методов `find`. Её используют для взаимодействия со страницей, а точнее, для поиска элемента, к которому впоследствии применяются некоторые действия, например, произвести клик по элементу.

Отмечу, эта группа методов ищет только один элемент. Если найдено несколько элементов, вернётся только первый.

Группа методов `find` содержит много методов, я расскажу только о самых основных и доступных для новичков:

Метод find_element_by_name

Ищет элемент, с заданным атрибутом name. Например, чтобы найти элемент: <input type="text" name="login" />

Нужно прописать следующий код:

```
elem = driver.find_element_by_name("login")
```

Метод find_element_by_id

Ищет элемент, с заданным атрибутом id. Например, чтобы найти элемент: <input type="text" id="password" />

Нужно прописать следующий код:

```
elem = driver.find_element_by_id("password")
```

Метод find_element_by_class_name

Ищет элемент, по имени класса. Например, чтобы найти элемент: <input type="text" class="form_login" />

Нужно прописать следующий код:

```
elem = driver.find_element_by_class_name("login")
```

Метод find_element_by_css_selector

Ищет элемент по CSS-селектору. Это один из самых мощных и удобных методов для поиска элементов, так как он позволяет искать элементы очень точно.

Давайте для примера рассмотрим вот такой html-код:

```
<div class="links">
    <a href="#" class="gold">Ссылка</a>
    <a href="#">Ссылка</a>
    <a href="#">Ссылка</a>
</div>
<div class="another_blocks">
    <a href="#" class="silver">Ссылка</a>
    <div class="gold">HI</div>
    <a href="#" class="gold">Ссылка</a>
</div>
```

Нам нужна, например, ссылка с классом `gold`, которая находится в блоке с классом `another_blocks`:

```
<div class="links">
    <a href="#" class="gold">Ссылка</a>
    <a href="#">Ссылка</a>
    <a href="#">Ссылка</a>
</div>
<div class="another_blocks">
    <a href="#" class="silver">Ссылка</a>
    <div class="gold">HI</div>
    <a href="#" class="gold">Ссылка</a> <!-- Вот эта ссылка-->
</div>
```

Мы бы могли воспользоваться методом `find_element_by_class_name` и найти элемент с классом `gold`, но тогда был бы найден самый первый элемент в коде, а именно ссылка в блоке с классом `links`:

```
<div class="links">
    <a href="#" class="gold">Ссылка</a>    <!-- Вот эта ссылка-->
    <a href="#">Ссылка</a>
    <a href="#">Ссылка</a>
</div>
```

```
elem = driver.find_element_by_css_selector("div.another_blocks a.gold")
```

Что это значит? Дословно мы передали этому методы следующее, внутри блока `div` с классом `another_blocks` найти ссылку (a) с классом `gold`.

Принцип построения CSS-селектора по имени класса: <элемент>.<имя_класса>, например: div.another_blocks

Найди блок div с классом `another_blocks`.

Принцип построения CSS-селектора по имени класса: <элемент>#<id_класса>, например: div#login

Найти блок `div` с `id` равному `login`.

Вложенные селекторы

Если нужно найти элемент, который лежит внутри другого элемента, ты мы используем вложенные селекторы. Когда мы пишем так <элемент1>.<имя_класса1> <элемент2>.<имя_класса2>, например: `div.another_blocks a.gold`:

```
div class="another_blocks">
    <a href="#" class="gold">Ссылка</a> <!-- Вот эта ссылка-->
</div>
```

То это значит, что нужно найти ссылку, с классом `gold`, внутри блока `div` с классом `another_blocks`. Каждый элемент справа, ищется внутри предыдущего элемента.

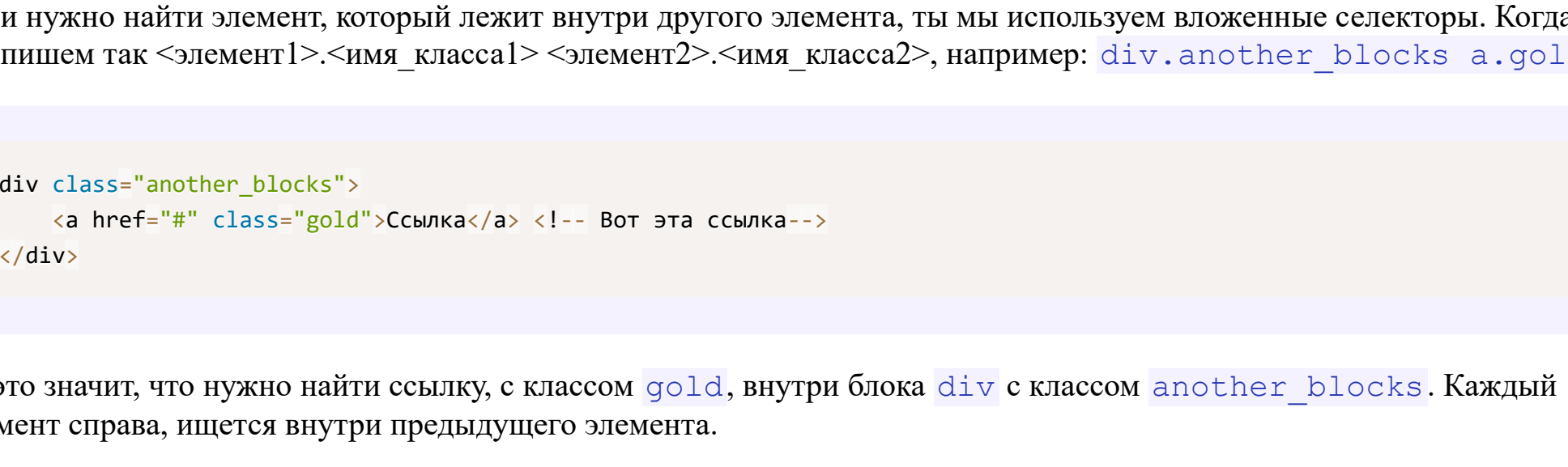
Ещё пример: <элемент1> <элемент2> <элемент3>

Такая последовательность селекторов звучит как: найти элемент 3, при этом, элемент три лежит внутри элемента 2, а элемент 2 лежит внутри элемента 1.

Метод find_element_by_css_selector. Пример на реальном сайте

Давайте зайдём на сайт google.com и получим ссылку на изображение логотипа.

1. Откройте сайт https://www.google.com/
2. Нажмите ПКМ по логотипу, далее нажмите «Просмотреть код»:



Блок с кодом логотипа выделен, но у самого логотипа (тега img) нет идентификатора, например id, или класса. Чтобы получить к нему доступ, нужно обращаться к тем блокам, внутри которых находится сам логотип.

Например, у нас есть блок див с id «lga»:

Внутри этого блока лежит наш логотип. Так `id` всегда уникально, значит мы можем указать точный и однозначный путь к нашему логотипу через этот блок, вот так:

```
elem = driver.find_element_by_css_selector("div#lga a img")
```

Что здесь происходит. Мы указываем функции: найди блок div с id равному lga, внутри этого блока найди ссылку (a), внутри которой найди изображение (img):

Если вы ищете элемент не по имени его класса, а по id то вместо символа «.», ставится символ «#». Давайте выведем ссылку на изображение логотипа на экран:

```
from selenium.webdriver import Chrome

# Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe")
# Переходим на сайт google.com
driver.get("https://www.google.com/")
# Ищем изображение с логотипом на странице
elem = driver.find_element_by_css_selector("div#lga a img")
# Получаем ссылку на логотип
link = elem.get_attribute("src")
# Выводим ссылку на экран
print(link)
# Закрываем браузер
driver.close()
```

Выводит:

> https://www.google.com/logos/doodles/2020/thank-you-custodial-and-sanitation-workers-6753651837108756-law.gif

Откроем ссылку:

Поиск элементов на странице. Группа методов find_elements

Данная группа методов идентична предыдущей и работает таким же образом, как и рассмотренные выше, за тем исключением, что они возвращают не один элемент, а все найденные элементы на странице (в виде списка, который можно, разумеется, перебрать в цикле):

find_elements_by_name
find_elements_by_class_name
find_elements_by_css_selector

Даже названия функций отличаются только тем, что вместо `element` (элемент), вводится `elements` (элементы), аргументы у данных функций идентичны.

Клик по элементу. Метод click

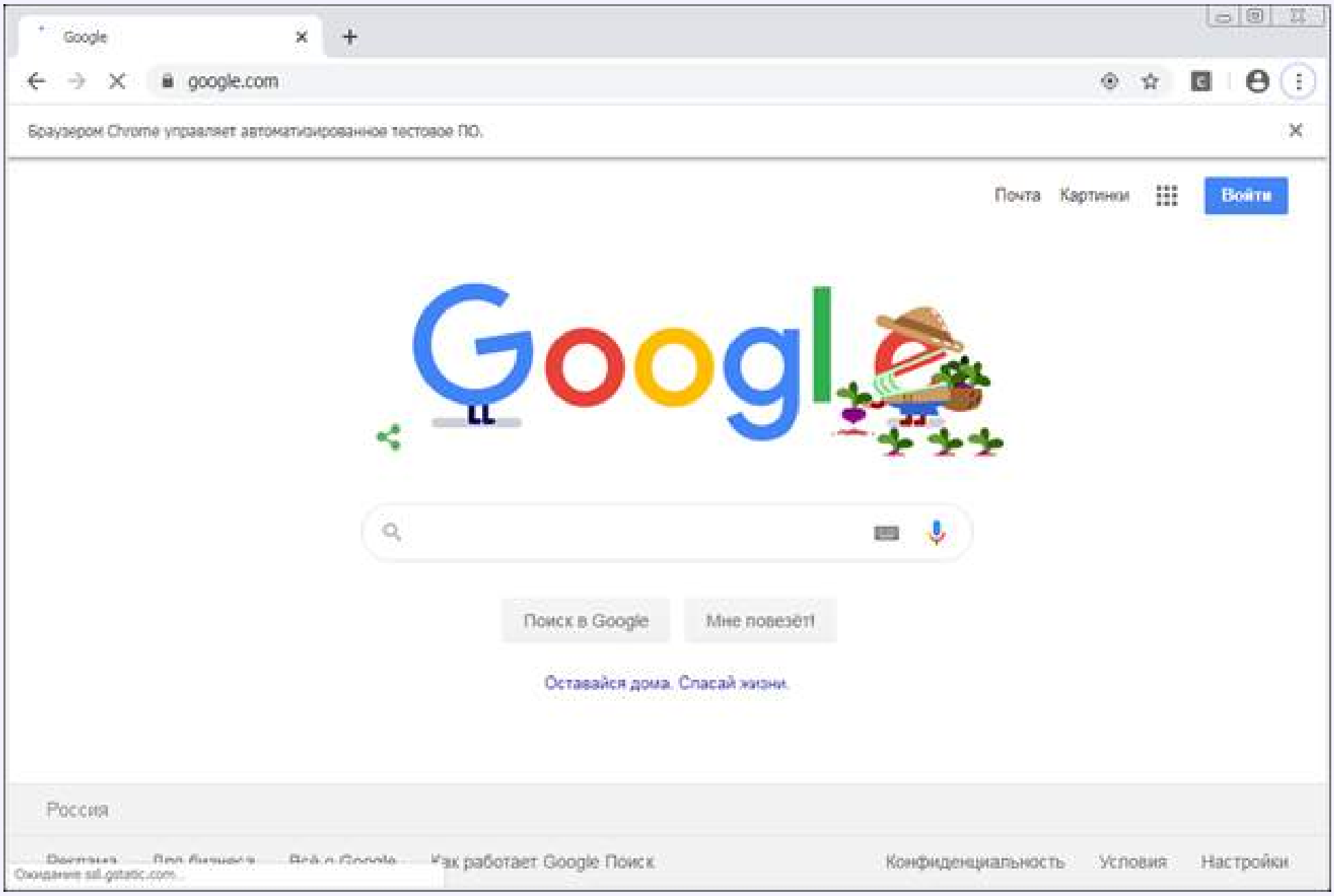
Для того, чтобы кликнуть на тот или иной элемент необходимо этот элемент найти, а затем вызвать у него метод `click`:

```
from selenium.webdriver import Chrome

# Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe")
# Переходим на сайт google.com
driver.get("https://www.google.com/")
# Ищем изображение с логотипом на странице
elem = driver.find_element_by_css_selector("div#lga a img")
# Производим действие клик по элементу
elem.click()
# Закрываем браузер
driver.close()
```

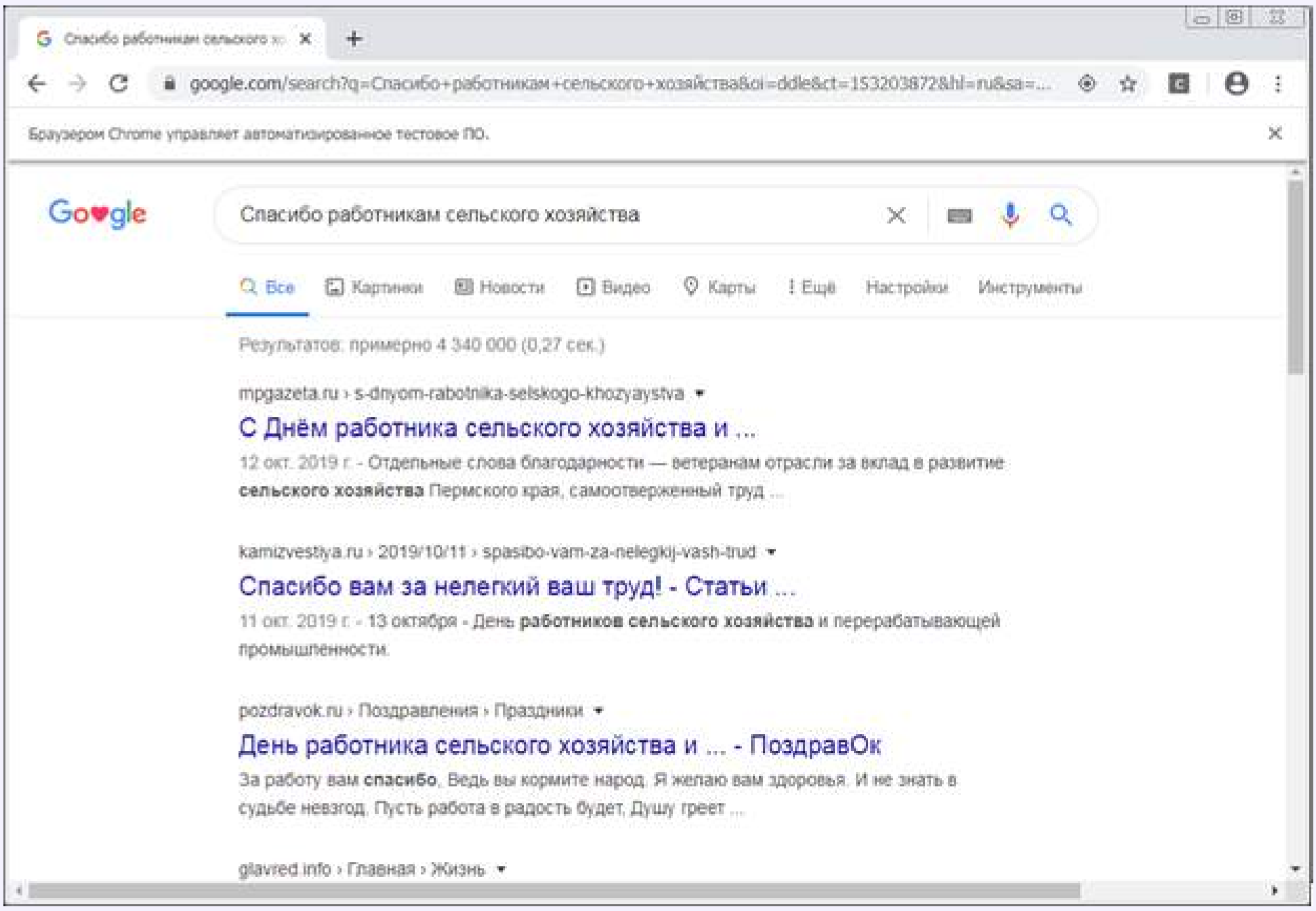
Что тут происходит:

1. Нашли элемент с логотипом
2. Совершили над этим элементом действие «Клик»

На экране увидим следующее. Первая страница:



После действия «Клик»:



Произвести действие «Клик» можно над любым элементом.

Получить значение атрибута. Метод get_attribute

Мы уже сталкивались с этим методом в данном уроке, в разделе `find_element_by_css_selector`, данный метод получает значение атрибута у любого html-элемента, например, атрибут `href` у ссылки, или атрибут `src` у изображения, для получения ссылки на исходное изображение, или атрибут `value`, для получения значения введённого в форму текста или любой другой атрибут, например, `id`, `class` и т.д.

Давайте разберёмся, что такое атрибут html-элемента. Атрибут элемента – это дополнительная декларация возможностей тега. Это дополнительные значения, которые вписывают в тег, чтобы изменить его форму и поведение.

Вот пример тега img (изображение): <img src="/images/title.gif" width="438" height="118">

Сам тег выглядит так: <img>

А его атрибуты:

src – ссылка на отображаемое изображение
width – ширина изображения
height – высота

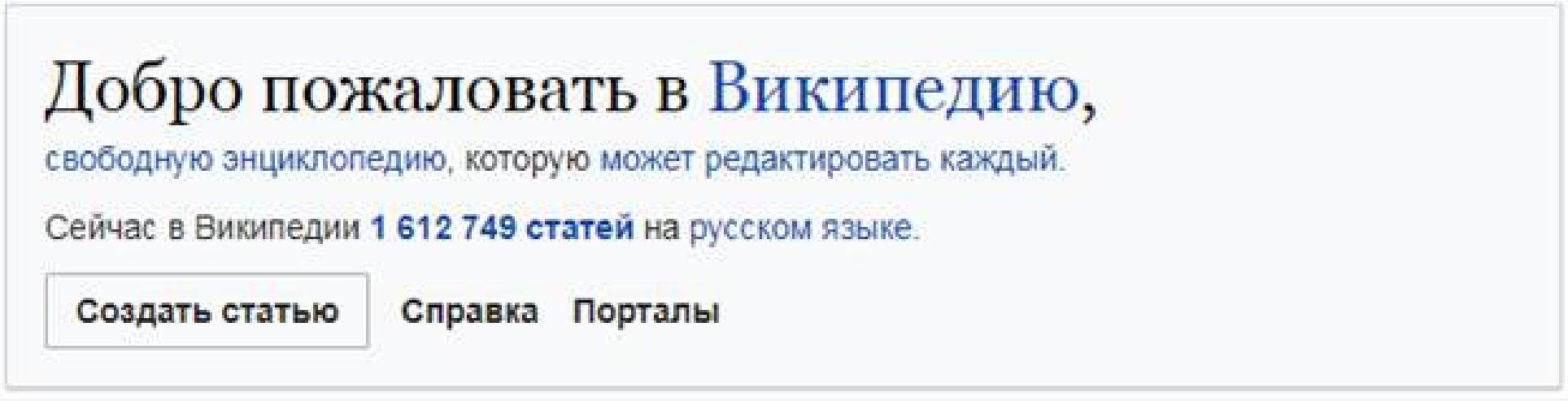
Таким образом, все вложенные внутри тега параметры вида имя_атрибута="значения" являются атрибутами.

А метод `get_attribute` позволяет получить значение атрибута у элемента.

Для того, чтобы получить значение атрибута у конкретного тега, необходимо сперва найти этот тег на странице, а затем вызвать у него метод `get_attribute`, в качестве параметра передав ему тот атрибут, который мы хотим получить:

```
# Ищем элемент со ссылкой для создания статьи
elem = driver.find_element_by_css_selector("div.main-top li.main-top-createArticle a")
# Получаем ссылку на создание статьи из элемента
link = elem.get_attribute("href")
```

Давайте для примера зайдём на википедию и попробуем получить ссылку на страницу создания новой статьи. Перейдём на сайт википедии вручную и найдём сперва саму ссылку:



Как видите, ссылка в виде кнопки «Создать статью», нажимаем «Просмотреть код» у самой кнопки и видим:

Видим, что наша ссылка имеет пусть и не уникальные атрибуты, но по иерархии вложенности её можно найти. Ссылка находится внутри тега li с классом main-top-createArticle, который в свою очередь находится внутри тега div с классом main-top.

CSS-селектор для поиска будет таким: `div.main-top li.main-top-createArticle a`

Найди тег div с классом `main-top`, затем внутри этого тега найди тег `li` с классом `main-top-createArticle`, внутри которого найди тег a (искомая сылка).

Напишем код:

```
from selenium.webdriver import Chrome

# Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe")
# Переходим на сайт ru.wikipedia.org
driver.get("https://ru.wikipedia.org/")
# Ищем элемент со ссылкой для создания статьи
elem = driver.find_element_by_css_selector("div.main-top li.main-top-createArticle a")
# Получаем ссылку на создание статьи из элемента
link = elem.get_attribute("href")
# Выводим ссылку на экран
print(link)
# Закрываем браузер
driver.close()
```

Выводит:

> https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%BA%D0%B8%D0%BF%D0%B5%D0%B4%D0%B8%D1%8F:%D0%9C%D0%B0%D1%81%D1%82%D0%B5%D1%80_%D1%81%D1%82%D0%B0%D1%82%D0%B5%D0%B9

То, что мы и искали.

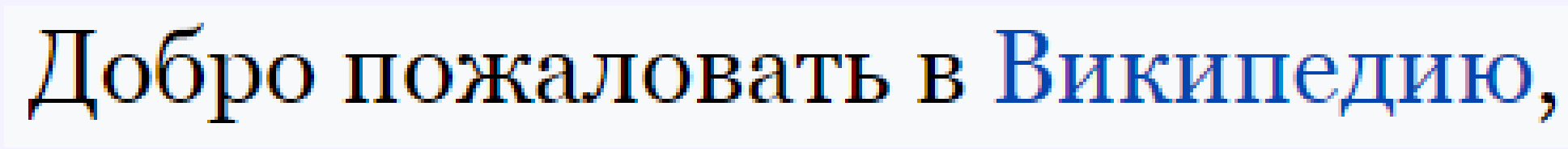
Получить содержимое тега: get_attribute("innerHTML")

Отдельно стоит заметить, что если вам необходимо получить содержимое тега, например текст из тега:

```
<p>
    Текст, который вы хотите получить
</p>
```

То следует воспользоваться такой конструкцией: `get_attribute("innerHTML")`.

У элемента, содержимое которого вы хотите извлечь, вызываете функцию get_attribute с параметром innerHTML. Давайте попробуем с википедии спарсить вот этот текст:



Он находится в этом теге:

Парсим:

```
from selenium.webdriver import Chrome

# Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe")
# Переходим на сайт ru.wikipedia.org/
driver.get("https://ru.wikipedia.org/")
welcome = driver.find_element_by_css_selector(".main-top-header .mw-headline")
print(welcome.get_attribute("innerHTML"))
# Выводим Cookie на экран
driver.close()
```

> Добро пожаловать в Википедию,

Если вы хотите спарсить только текст, то нужно запустить функцию `get_attribute` со значением `innerText`:

```
from selenium.webdriver import Chrome
from selenium.webdriver.chrome.options import Options

# Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe")
# Переходим на сайт ru.wikipedia.org/
driver.get("https://ru.wikipedia.org/")
welcome = driver.find_element_by_css_selector(".main-top-header .mw-headline")
print(welcome.get_attribute("innerText"))
# Выводим Cookie на экран
driver.close()
```

Выводит:

> Добро пожаловать в Википедию,

Ввод текста в форму. Метод send_keys

Чтобы ввести текст в некоторую форму, например, в форму поиска можно воспользоваться методом send_keys. Для этого необходимо:

1. Найти тег input в который вы хотите ввести текст
2. Вызвать у элемента метод click, чтобы сделать форму активной
3. Вызвать у этого элемента метод send_keys, в который в качестве аргумента передаётся текст, который нужно ввести

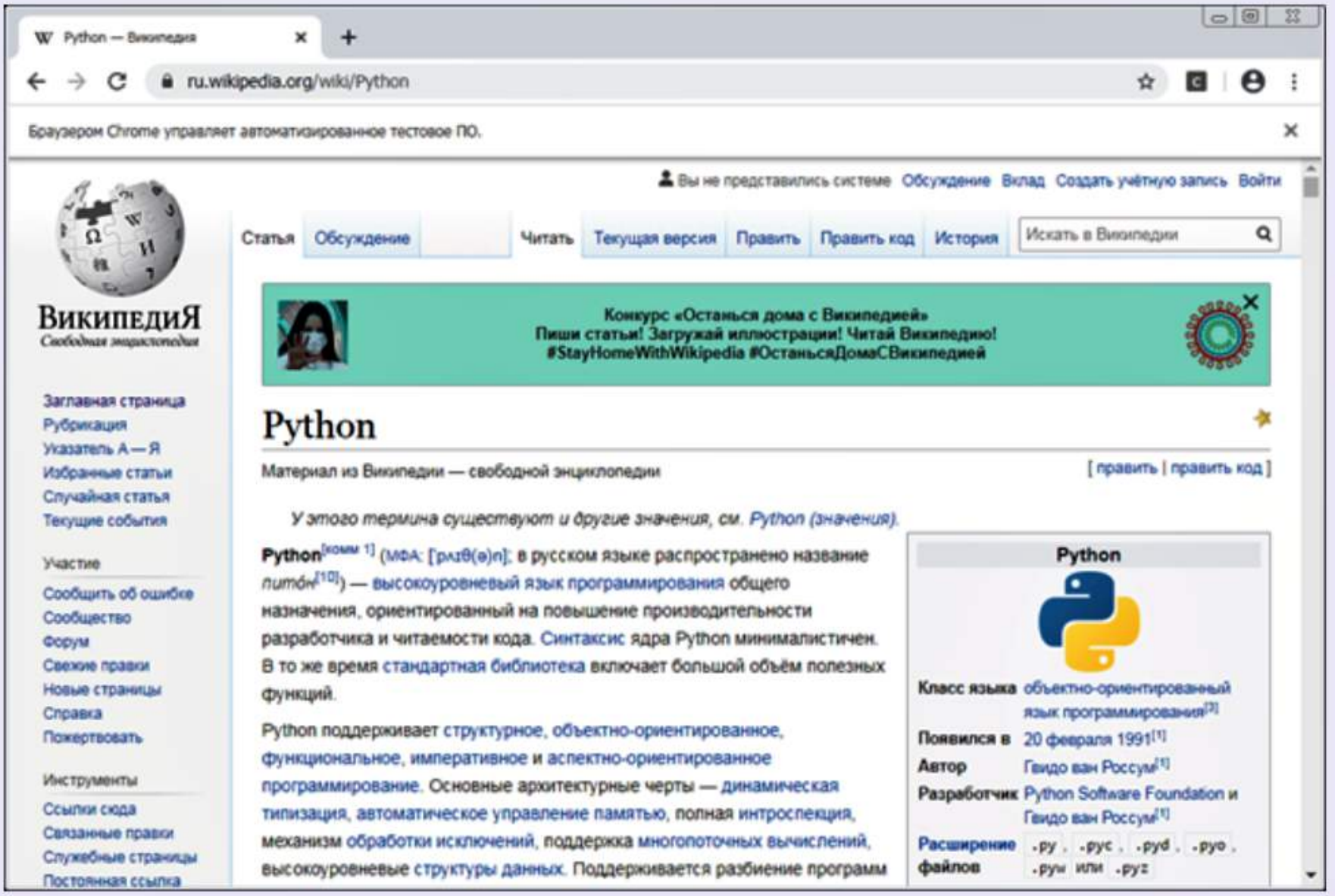
Пример:

```
# Ищем элемент input для ввода поисковго запроса
elem = driver.find_element_by_tag_name("input#searchInput")
# Совершаем действие click по элементу input
elem.click()
# Вводим текст в элемент input
elem.send_keys("Python")
```

Давайте попробуем найти для примера статью о языке программирования Python на сайте `ru.wikipedia.org`:

```
from selenium.webdriver import Chrome

# Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe")
# Переходим на сайт ru.wikipedia.org
driver.get("https://ru.wikipedia.org/")
# Ищем элемент input для ввода поисковго запроса
elem = driver.find_element_by_tag_name("input#searchInput")
# Совершаем действие click по элементу input
elem.click()
# Вводим текст в элемент input
elem.send_keys("Python")
# Ищем элемент input для поиска
submit = driver.find_element_by_css_selector("input#searchButton")
# Совершаем действие click по элементу
submit.click()
# Закрываем браузер
driver.close()
```

В итоге браузер откроет вот такую страницу:



Это перечень основных действий. С остальными можно познакомиться в официальной документации.

Расширенные настройки Selenium

Содержание урока

Selenium обладает большим количеством настроек и позволяет подробно сконфигурировать ваш браузер. Рассмотрим только самые основные настройки Selenium.

Напомню, подробная документация по Selenium: https://selenium-python.readthedocs.io/

Установка Proxy

Чтобы работать с одним и тем же сайтом, через множество разных аккаунтов нередко требуется устанавливать прокси-сервера, чтобы сайт не мог идентифицировать разные аккаунты по одному и тому же ip-адресу.

Для начало, нам нужно подключить к проекту специальный модуль:

```
from selenium.webdriver import DesiredCapabilities
```

Теперь нам нужен прокси-сервер. Если у вас уже есть свой – используйте его. Можете взять бесплатный в интернете, например, здесь: https://hidemy.name/ru/proxy-list/

Создаём словарь с настройкой наших прокси-серверов:

```
# Proxy
proxy = {
    "httpProxy": "163.172.146.119:8811",  # http
    "ftpProxy": "163.172.146.119:8811",  # ftp
    "sslProxy": "163.172.146.119:8811",  # https
    "noProxy": [],
    "proxyType": "MANUAL"
}
```

Как видите, мы можем здесь указать прокси для `http`, `ftp`, `https` соединений. Теперь, обращаемся к модулю `DesiredCapabilities` и передаём в него настройки:

```
DesiredCapabilities.CHROME['proxy'] = proxy
```

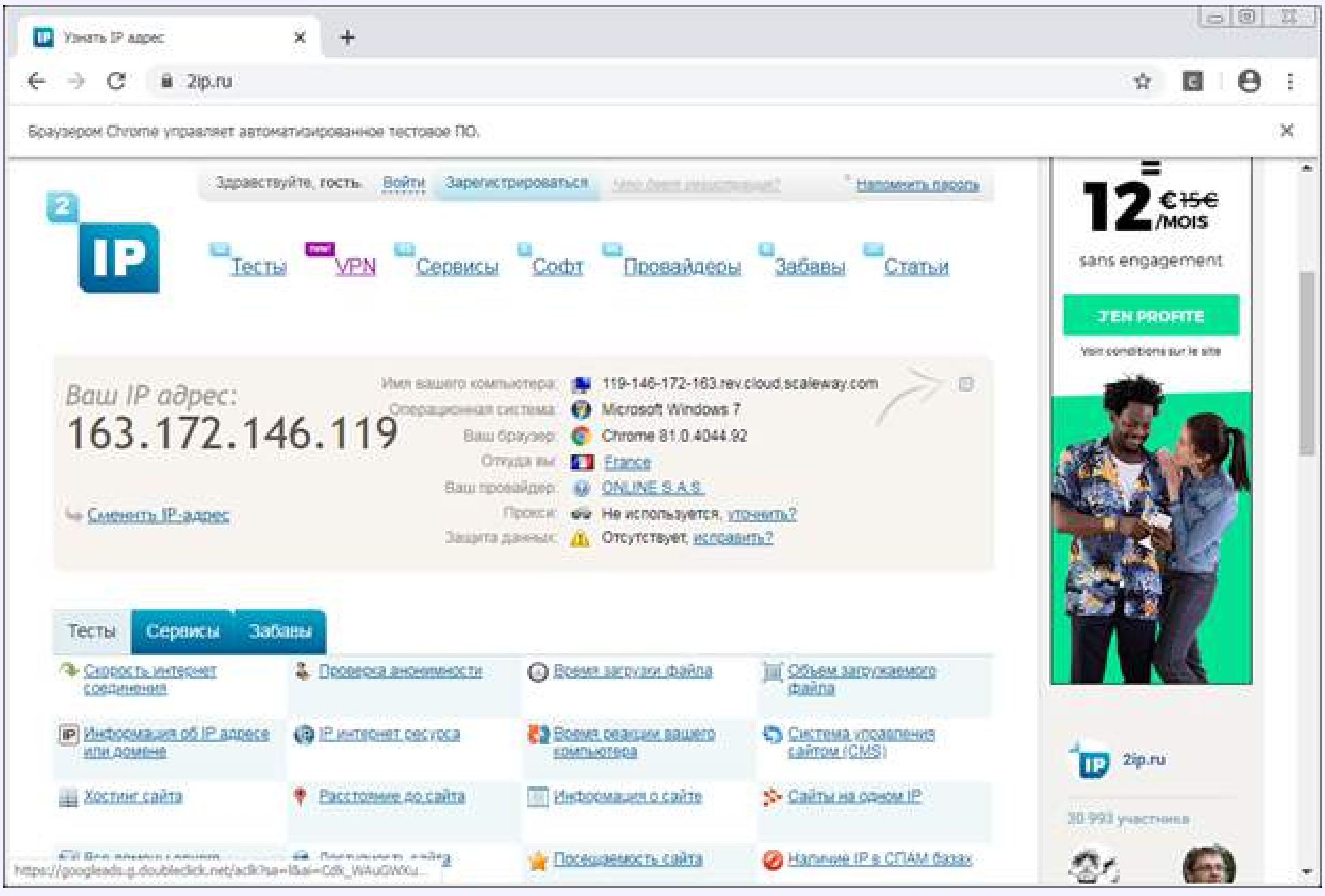
Здесь мы указываем, что модулю по работе с браузером CHROME мы устанавливаем настройку proxy, и присваиваем наш словарь с записанными туда прокси-серверами.

Давайте сперва попробуем зайти через Selenium на сайт: https://2ip.ru/

И посмотрим, подключился наш прокси-сервер или нет:

```
import time
from selenium.webdriver import Chrome
from selenium.webdriver import DesiredCapabilities

# Proxy
proxy = {
    "httpProxy": "163.172.146.119:8811",  # http
    "ftpProxy": "163.172.146.119:8811",  # ftp
    "sslProxy": "163.172.146.119:8811",  # https
    "noProxy": [],
    "proxyType": "MANUAL"
}

DesiredCapabilities.CHROME['proxy'] = proxy

# # Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe")
# # Переходим на сайт 2ip.ru
driver.get("https://2ip.ru/")
time.sleep(15)
# # Закрываем браузер
driver.close()
```

Запускаем скрипт. Открывается страница:



Как видите, мой российский `ip` сменился на французский.

Установка Proxy с авторизацией

Из коробки Selenium «не умеет» работать с прокси-серверами, требующими авторизации. Но тем не менее, достаточно удобное решение существует. Установить Proxy с авторизацией можно, если загрузить в заданный запускаемый объект браузера специальное расширение.

Для этого существует следующая библиотека: https://pypi.org/project/SeleniumAuthProxy/

Данная библиотека, создаёт в заданной вами папке расширение Chrome с записанным в неё прокси с авторизацией, а при запуске самого драйвера, это расширение в свою очередь устанавливает прокси в браузер.

Установим расширение в наш проект: `pip install SeleniumAuthProxy`

Теперь импортируем его в проект:

```
from SeleniumAuthProxy import SeleniumAuthProxy
```

Далее, нам необходимо создать папку, в которую будут записываться расширения с нашими прокси-серверами, например, proxy:

Давайте попробуем установить прокси в наш браузер. Импортируем все библиотеки в наш проект:

```
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from SeleniumAuthProxyimport SeleniumAuthProxy
```

Библиотека которую мы установили просто создаёт расширение для хрома, в которое записывает данные вашего прокси, затем это расширение устанавливается в браузер который вы запускаете и оно устанавливает заданные прокси с авторизацией.

Реализуется это при помощи метода SeleniumAuthProxy.get_proxy_object, например:

```
SeleniumAuthProxy.get_proxy_object(ип_адрес_прокси, порт, логин, пароль, папка_в_которую_будет_сохранено_расширение)
SeleniumAuthProxy.get_proxy_object("211.170.52.120", "9923", "DKTuee", "9rttL3", "proxy")
```

Перед тем, как работать с этим методом, обязательно в корневой директории создадть папку, в которую будут сохраняться наши расширения с прокси.

Вызовем этот метод и получим путь до созданного нами расширения:

```
# Создаём в директории прокси и получаем путь до него
proxy = SeleniumAuthProxy.get_proxy_object("213.166.72.220", "9923", "DKкuee", "9rttL3", "proxy")
```

Далее, нам нужно загрузить это расширение в наш объект хрома, для этого вызываем объект настроек и добавляем в него расширение:

```
# Создаём объект настроек
chrome_options = Options()
# Добавляем в текущий объект хрома расширение
chrome_options.add_extension(proxy)
```

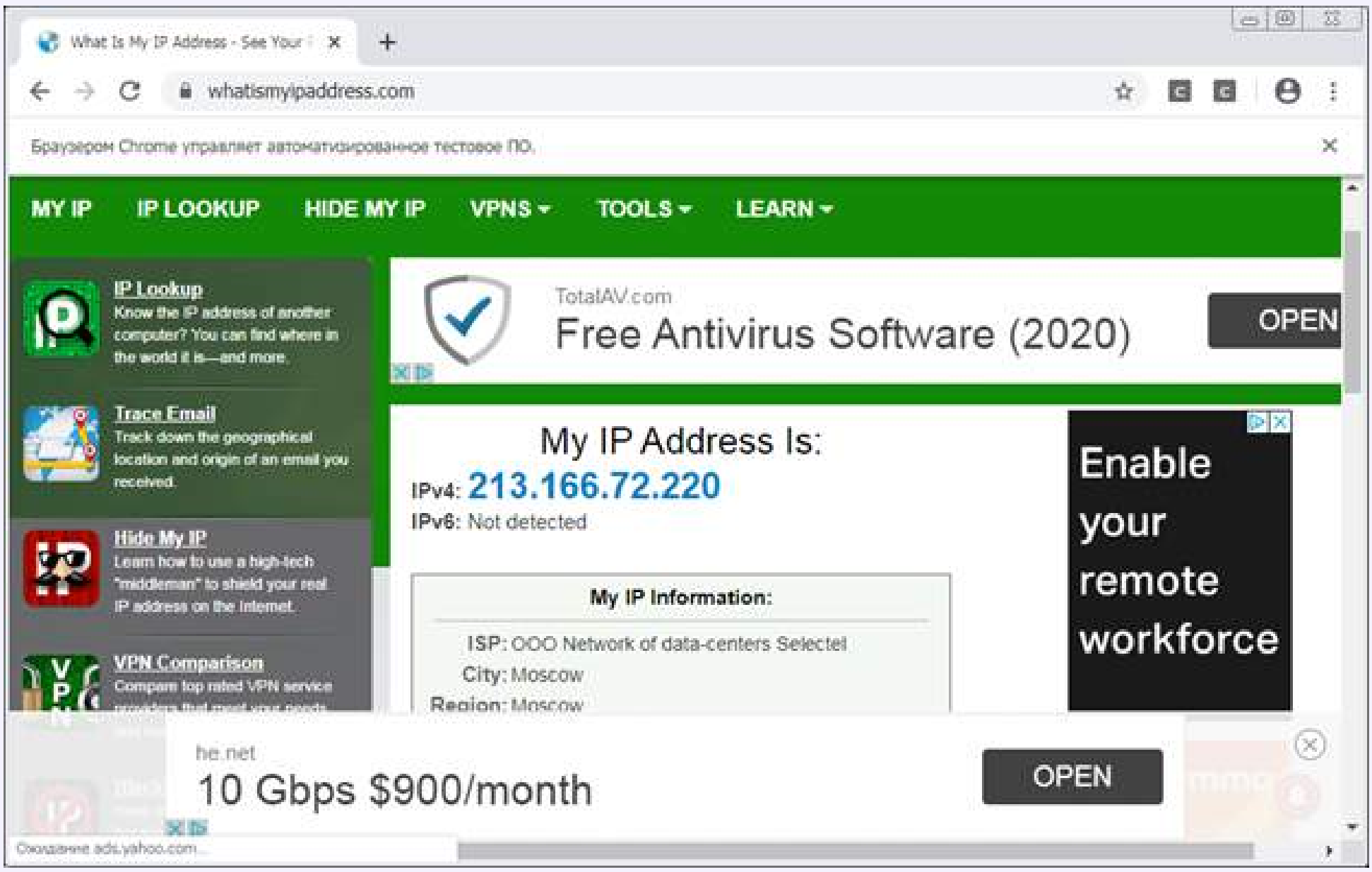
Передаём объект с настройками в наш браузер и запускаем его:

```
# Создаём новый объект хрома с установленным расширением
driver = webdriver.Chrome(executable_path='assets/chromedriver.exe', options=chrome_options)
```

Переходим на сайт, определяющий ip-адрес:

```
# Переходим на страницу и видим что наш прокси успешно установился
driver.get("https://whatismyipaddress.com/")
```

Видим, что заданный адрес был успешно установлен:



После установки обязательно необходимо удалить заданное расширение, чтобы не было лишнего мусора в вашем проекте, для этого есть метод:

```
SeleniumSetterAuthProxy.delete_proxy_object(путь_до_расширения)
```

Удалить расширение необходимо в любом случае, поэтому необходимо запускать всю конструкцию в `try..finally`:

```
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from SeleniumAuthProxy import SeleniumAuthProxy


try:
    # Создаём в директории прокси и получаем путь до него
    proxy = SeleniumAuthProxy.get_proxy_object("213.166.72.220", "9923", "DKkuee", "9rttL3", "proxy")
    # Создаём объект настроек
    chrome_options = Options()
    # Добавляем в текущий объект хрома расширение
    chrome_options.add_extension(proxy)
    # Создаём новый объект хрома с установленным расширением
```

```
    driver = webdriver.Chrome(executable_path='assets/chromedriver.exe', options=chrome_options)
    # Переходим на страницу и видим что наш прокси успешно установился
    driver.get("https://whatismyipaddress.com/")
finally:
    # В обязательном порядке удаляем расширение с прокси из нашей директории
    SeleniumAuthProxy.delete_proxy_object(proxy)
    # Закрываем браузер
    driver.close()
```

Установка User-Agent

Также будет не лишним, если ваши аккаунты будут иметь уникальные юзерагенты, для того, чтобы установить юзерагент, необходимо импортировать модуль настройки вашего браузера, например, для `chrome`:

```
from selenium.webdriver.chrome.options import Options
```

Создаём строку с юзерагентом:

```
# Указываем юзерагент
user_agent = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:63.0) Gecko/20100101 Firefox/63.0"
```

Создаём объект с настройками браузера:

```
# Создаём объект настроек браузера
opts = Options()
```

Устанавливаем юзерагент:

```
# Устанавливаем юзер-агент
opts.add_argument("user-agent=" + user_agent)
```

Передаём объект с опциями, при создании драйвера:

```
# Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe", options=opts)
```

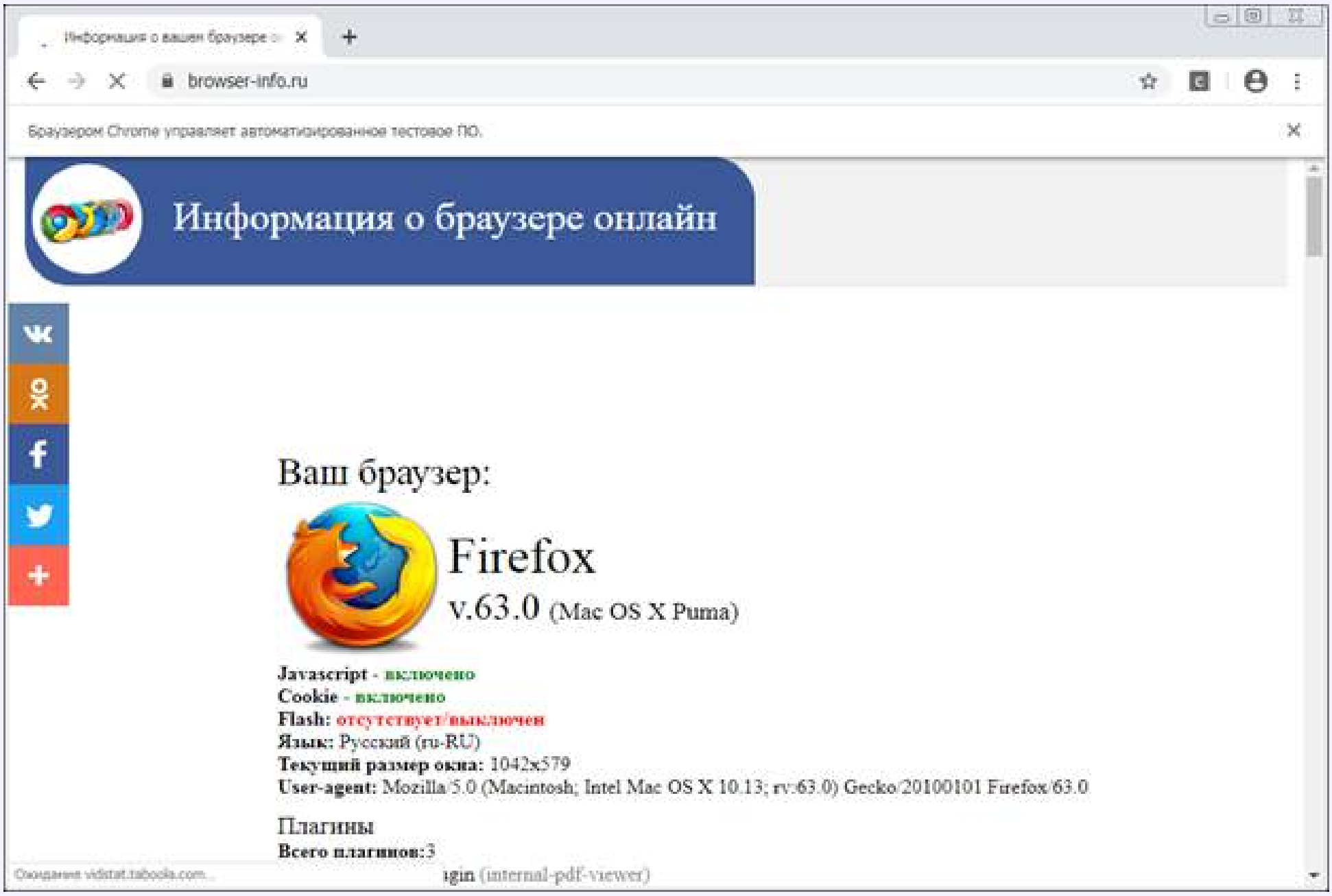
Переходим на сайт: https://browser-info.ru/

Он покажет какой юзерагент был установлен в нашем браузере:

```
import time
from selenium.webdriver import Chrome
from selenium.webdriver.chrome.options import Options

# Указываем юзерагент
user_agent = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:63.0) Gecko/20100101 Firefox/63.0"

# Создаём объект настроек браузера
```

```
opts = Options()
# Устанавливаем юзер-агент
opts.add_argument("user-agent=" + user_agent)

# Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe", options=opts)

# Переходим на сайт browser-info.ru
driver.get("https://browser-info.ru/")
time.sleep(15)
# Закрываем браузер
driver.close()
```

Запустим скрипт. На сайте отобразится:



Как видите, был установлен заданный юзерагент.

Работа с Cookies

Получить cookies с сайта

Например, вы авторизовались на сайте и хотите получить куки, при помощи которых впоследствии вы можете пользоваться сайтом без повторной авторизации.

Для того, чтобы получить куки с сайта, достаточно воспользоваться методом `get_cookies()` у объекта нашего браузера. Давайте перейдём на сайт википедии и посмотрим, какие куки она нам установит:

```
from selenium.webdriver import Chrome

# Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe")

# Переходим на сайт browser-info.ru
```

```
driver.get("https://ru.wikipedia.org/")
print(driver.get_cookies())

# # Закрываем браузер
driver.close()
```

Запустим скрипт. Вот что будет выведено в консоль:

> [{'domain': '.wikipedia.org', 'expiry': 1589674501.454278, 'httpOnly': True, 'name': 'WMF-Last-Access-Global', 'path': '/', 'secure': True, 'value': '15-Apr-2020'}, {'domain': 'ru.wikipedia.org', 'expiry': 1589674501.454001, 'httpOnly': True, 'name': 'WMF-Last-Access', 'path': '/', 'secure': True, 'value': '15-Apr-2020'}]

Вывело список словарей с куками, которые были установлены в наш браузер, в последствии мы можем ими пользоваться (если это нужно).

Установить Cookies на сайт

Предположим, что у вас уже есть куки от некоторого сайта, и чтобы не производить авторизацию повторно, вы хотите передать ему куки для авторизации. Для этого у Selenium есть метод `add_cookie`.

Данный метод принимает в качестве аргумента словарь:

```
cookie = {
    "name": "python",  # Имя куки
    "domain": "ru.wikipedia.org",  # Домен, для которого кука устанавливается
    "value": "python"  # Значение куки
}
```

А добавляется в Selenium таким образом:

```
driver.add_cookie(cookie)
```

Эта функция устанавливает один экземпляр `cookies`. Чтобы установить несколько, можно вызвать данную функцию несколько раз:

```
driver.add_cookie(cookie_1)
driver.add_cookie(cookie_2)
driver.add_cookie(cookie_3)
driver.add_cookie(cookie_4)
```

Или в цикле:

```
for cookie in cookies:
    driver.add_cookie(cookie)
```

Давайте установим наш экземпляр cookie на сайт Wikipedia, а затем, проверим, был он установлен или нет:

```
from selenium.webdriver import Chrome

# Создаём объект для управления драйвером
```

```
driver = Chrome(executable_path="assets/chromedriver.exe")

# Переходим на сайт browser-info.ru
driver.get("https://ru.wikipedia.org/")

cookie = {
    "name": "Korben",  # Имя куки
    "domain": "ru.wikipedia.org",  # Домен, для которого кука устанавливается
    "value": "Привет от Корбена!"  # Значение куки
}

# Добавляем экземпляр Cookie
driver.add_cookie(cookie)

# Выводим Cookie на экран
print(driver.get_cookies())

# # Закрываем браузер
driver.close()
```

Запустим скрипт. В консоль выводит:

[{'domain': '.ru.wikipedia.org', 'httpOnly': False, 'name': 'Korben', 'path': '/', 'secure': True, 'value': 'Привет от Корбена!'}, {'domain': '.wikipedia.org', 'expiry': 1589717731.080771, 'httpOnly': True, 'name': 'WMF-Last-Access-Global', 'path': '/', 'secure': True, 'value': '15-Apr-2020'}, {'domain': 'ru.wikipedia.org', 'expiry': 1589717731.080157, 'httpOnly': True, 'name': 'WMF-Last-Access', 'path': '/', 'secure': True, 'value': '15-Apr-2020'}]

Наш объект Cookie был успешно установлен.

Selenium без головы

Когда пишешь скрипт для автоматизации сайтов, нередко приходится запускать много экземпляров Selenium, например, в многопоточном режиме запускаешь 20 браузеров и каждый открывается на вашем экране, как обычный браузер.

Не самая удобная ситуация. Хотелось бы, чтобы selenium работал где-нибудь в трее и никому не мешал. И Selenium в Python поддерживает такую возможность. Для этого необходимо воспользоваться модулем Options, с которым мы уже работали до этого.

Создаём объект объект options:

```
# Создаём объект Options
options = Options()
```

У этого объекта есть переменная `headless`, именно она отвечает за то, чтобы браузер не открывался на экране, а работал в трее. Этой переменной нужно установить значение `True`:

```
# Устанавливаем настройку скрытия экрана в значение True
options.headless = True
```

При создании объекта браузера не забываем передавать объект options в качестве аргумента:

```
# Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe", options=options)
```

Браузер был сконфигурирован на работу в трее. Давайте попробуем запустить код из прошлого примера, где мы получали куки от сайта википедия и выводили их в терминал:

```
from selenium.webdriver import Chrome
from selenium.webdriver.chrome.options import Options

# Создаём объект Options
options = Options()
# Устанавливаем настройку скрытия экрана в значение True
options.headless = True

# Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe", options=options)
# Переходим на сайт ru.wikipedia.org/
driver.get("https://ru.wikipedia.org/")
# Выводим Cookie на экран
print(driver.get_cookies())
driver.close()
```

Запустим скрипт. На экран выводится:

> [{'domain': '.wikipedia.org', 'expiry': 1589674501.454278, 'httpOnly': True, 'name': 'WMF-Last-Access-Global', 'path': '/', 'secure': True, 'value': '15-Apr-2020'}, {'domain': 'ru.wikipedia.org', 'expiry': 1589674501.454001, 'httpOnly': True, 'name': 'WMF-Last-Access', 'path': '/', 'secure': True, 'value': '15-Apr-2020'}]

Самого окна браузера мы не увидели, но тем не менее посещение сайта было произведено и куки были получены.

Безопасный запуск Selenium

После того, как все действия с браузером были выполнены, его обязательно нужно закрыть, для этого мы используем метод `close()`.

Но как вы знаете, метод `close()` может не сработать, если перед ним возникнет ошибка, поэтому всегда запускайте код связанный с Selenium в конструкции `try..except..finally`:

```
from selenium.webdriver import Chrome
from selenium.webdriver.chrome.options import Options

# Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe")

try:
    # Переходим на сайт ru.wikipedia.org/
    driver.get("https://ru.wikipedia.org/")
    welcome = driver.find_element_by_css_selector(".main-top-header .mw-headline")
    print(welcome.get_attribute("innerText"))
    # Выводим Cookie на экран
finally:
    driver.close()
```

Разумеется, при желании можно использовать блок except, чтобы обработать исключение, давайте, например, введём несуществующий классу тега:

```
driver.find_element_by_css_selector(".main-top-header .mw-headlinefttrybhtyj6yj")
```

И обработаем ошибку:

```python
from selenium.webdriver import Chrome

# Создаём объект для управления драйвером
driver = Chrome(executable_path="assets/chromedriver.exe")

try:
    # Переходим на сайт ru.wikipedia.org/
    driver.get("https://ru.wikipedia.org/")
    welcome = driver.find_element_by_css_selector(".main-top-header .mw-headlinefttrybhtyj6yj")
    print(welcome.get_attribute("innerText"))
    # Выводим Cookie на экран
except Exception as error:
    print(error)
finally:
    driver.close()
```

Выводит:

> Message: no such element: Unable to locate element: {"method":"css selector","selector":".main-top-header .mw-headlinefttrybhtyj6yj"}(Session info: chrome=81.0.4044.113)

Ошибка из-за неверного селектора всплывёт, но программа не завершится с шибкой, при этом, драйвер будет закрыт, а в блоке для обработке ошибок мы просто выведем содержимое ошибки в консоль (или можем, например где-то её сохранить).

Практика. Авторизуемся в ВК

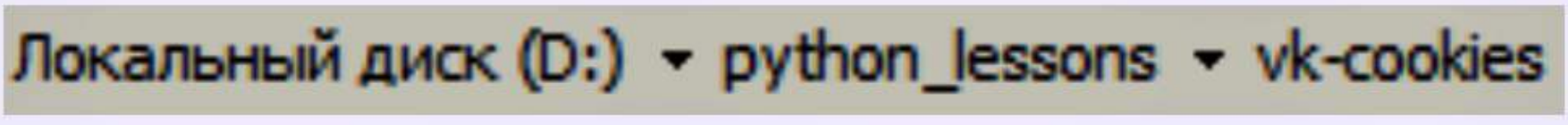
Содержание урока

Прницип скрипта

Напишем модуль для авторизации в ВК через Selenium. Модуль выполнит три простых действия:

1. Перейдёт на сайт
2. Введёт логин и пароль
3. Сохранит cookies, полученные после авторизации

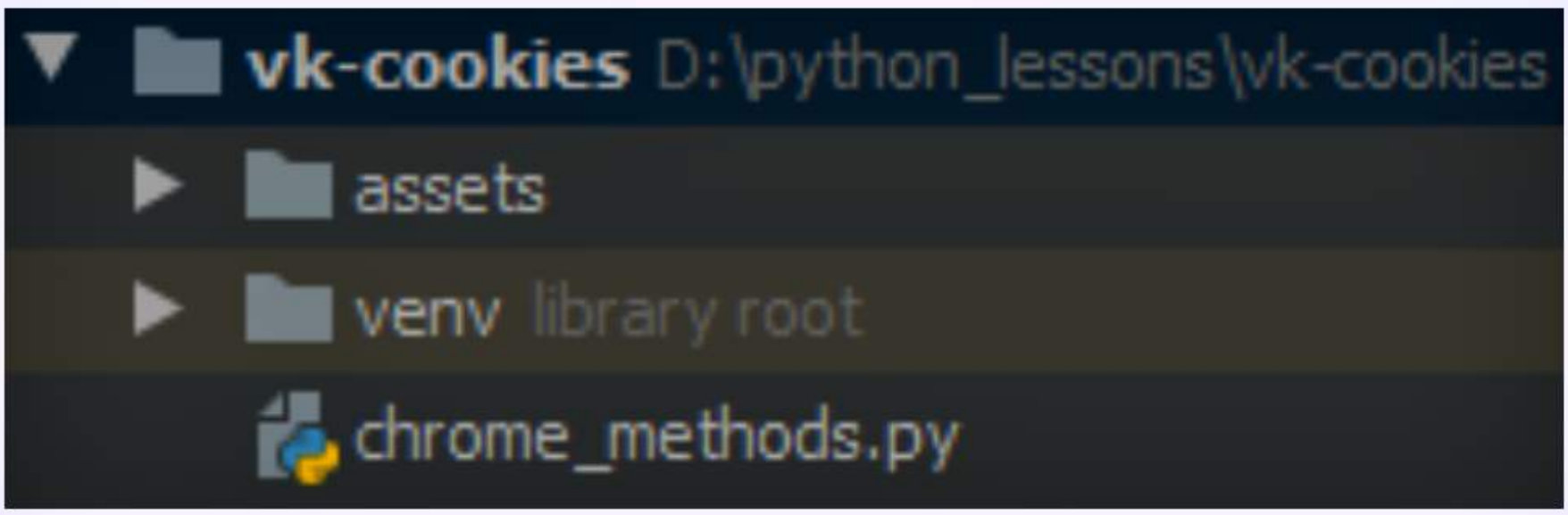
Приступим. Сперва, создайте новую папку для нашего модуля авторизации, допустим, vk-cookies:



Не забудьте установить Selenium в новый проект по инструкции из урока номер два, данного раздела.

Создание и настройка браузера вещь достаточно типичная, поэтому давайте создадим модуль, в который вынесем функционал для работы с браузером.

Создадим новый модуль и назовём его `chrome_methods.py`:



В данный модуль включим модули по работе с Selenium:

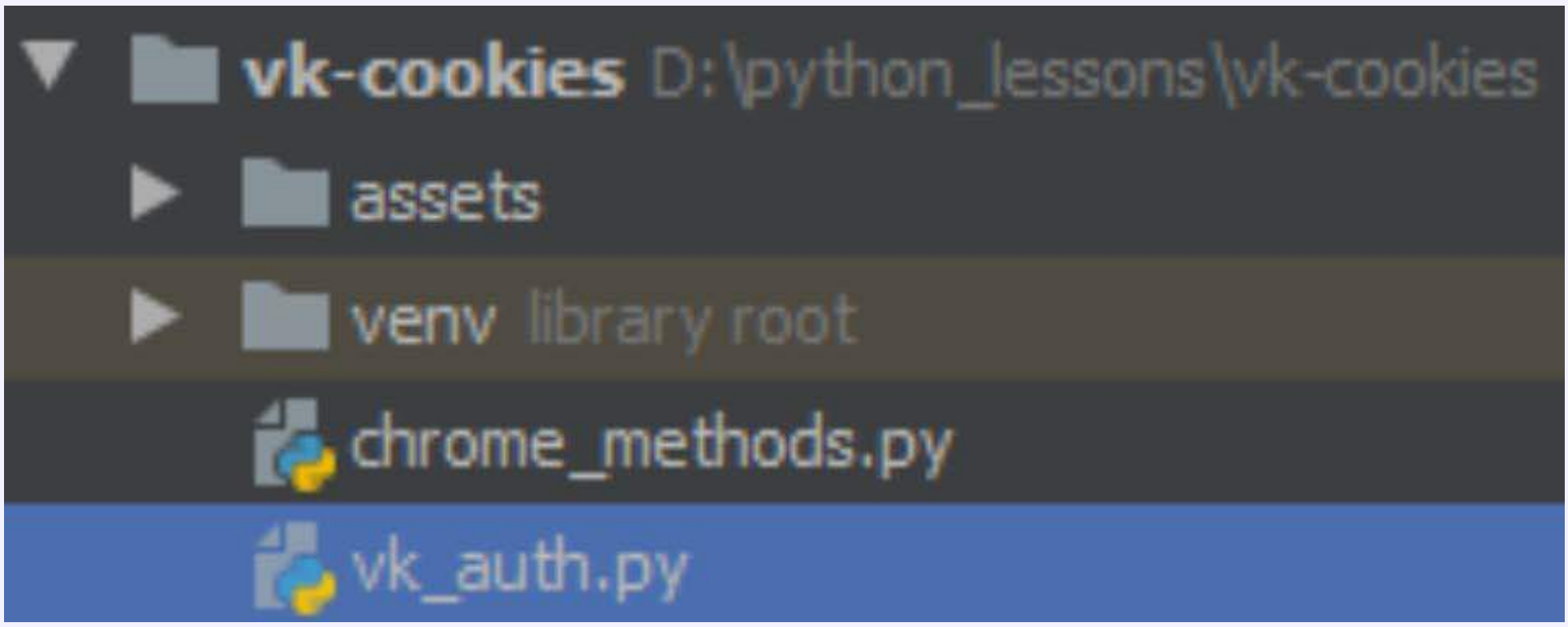
```
from selenium.webdriver import Chrome
from selenium.webdriver.chrome.options import Options
from selenium.webdriver import DesiredCapabilities
```

Далее, создадим функцию, которая как мы разбирали в прошлом уроке, создаёт объект браузера:

```
from selenium.webdriver import Chrome
from selenium.webdriver.chrome.options import Options
from selenium.webdriver import DesiredCapabilities


def init_chrome(path="assets/chromedriver.exe", proxy=None, user_agent=None):

    # Устанавливаем прокси
    proxy = {
        "httpProxy": proxy,  # http
        "ftpProxy": proxy,  # ftp
        "sslProxy": proxy,  # https
        "noProxy": [],
        "proxyType": "MANUAL"
    }

    # Если мы передали прокси в проект, то установим их в браузер
    if proxy is not None:
        DesiredCapabilities.CHROME['proxy'] = proxy

    # Создаём объект настроек браузера
    opts = Options()

    # Если мы передали юзер-агент в проект, то установим их в брвузер
    if user_agent is not None:
        opts.add_argument("user-agent=" + user_agent)

    # Создаём объект для управления драйвером
    return Chrome(executable_path=path, options=opts)
```

Здесь всё просто, функция устанавливает настройки (в том случае, если мы их передали), создаёт браузер и возвращает его объект.

Хорошо, создадим ещё один модуль, в этом модуле будет записан скрипт для автоматизации. Назовём его `vk_auth.py`:



Так как мы будет сохранять куки в виде словаря, нам нужен удобный инструмент, для «консервирования» данных, например, pickle (о нём мы говорили в уроке по файлам).

Импортируем его, а также модуль для работы с браузером:

```
import pickle
import chrome_methods
```

Создадим объект браузера:

```
# Создаём объект браузера
driver = chrome_methods.init_chrome()
```

Далее, определяем логин и пароль:

```
# Определяем данные для авторизации
login = "79126948825"
password = "yWMHsY2Jsw3B1wsf"
```

После этого, напишем сам скрипт для авторизации, а именно, переходим на сайт:

```
# Переходим на сайт vk.com
driver.get("https://vk.com/")
```

Вводим логин:

```
# Ищем инпут для ввода логина
vk_login_input = driver.find_element_by_css_selector("#index_email")
# Кликаем по форме для ввода логина
vk_login_input.click()
# Ввводим логин в форму
vk_login_input.send_keys(login)
```

Вводим пароль:

```
# Ищем форму для ввода пароля
vk_password_input = driver.find_element_by_css_selector("#index_pass")
# Кликаем по форме для ввода парроля
vk_password_input.click()
# Вводим пароль в форму
vk_password_input.send_keys(password)
```

Нажимаем кнопку «Войти»:

```
# Ищем кнопку для авторизации
vk_auth_button = driver.find_element_by_css_selector("#index_login_button")
```

```
# Кликаем по форме для ввода пароля
vk_auth_button.click()
```

Получаем куки из браузера:

```
# Получаем куки из браузера
cookies = driver.get_cookies()
```

Формируем название для имени файла, в который мы сохраним куки:

```
# Формируем название для сохранения файла с куками из логина и пароля вформате: login+password.cookie
file_name = f"{login}+{password}.cookie"
```

Сохраняем куки в файл:

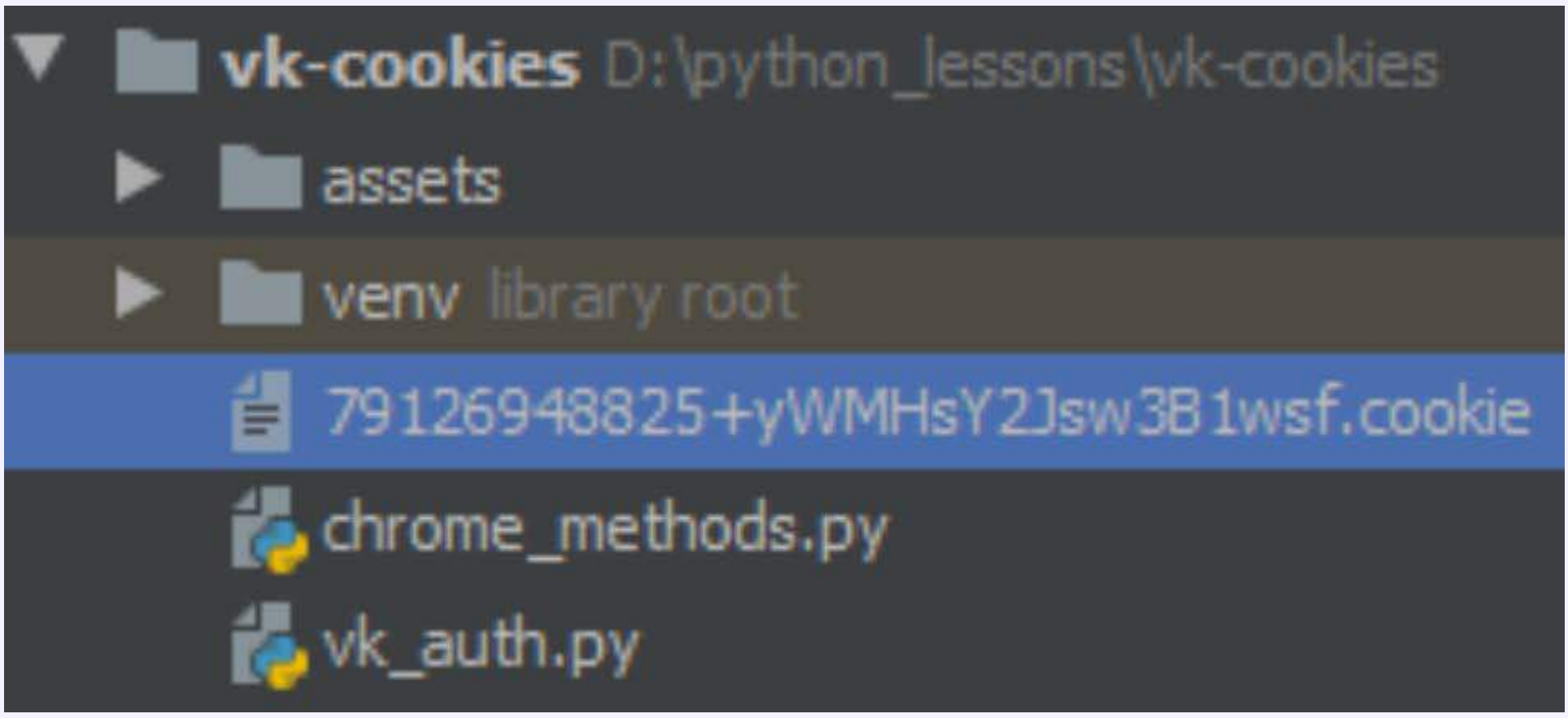
```
# Сохраняем куки в файл
with open(file_name, 'wb') as f:
    pickle.dump(cookies, f)
```

Не забываем, обернуть весь код в конструкцию `try`:

```
import pickle
import chrome_methods

# Создаём объект браузера
driver = chrome_methods.init_chrome()

# Определяем данные для авторизации
login = "79126948825"
password = "yWMHsY2Jsw3B1wsf"

try:
    # Переходим на сайт vk.com
    driver.get("https://vk.com/")

    # Ищем инпут для ввода логина
    vk_login_input = driver.find_element_by_css_selector("#index_email")
    # Кликаем по форме для ввода логина
    vk_login_input.click()
    # Ввводим логин в форму
    vk_login_input.send_keys(login)

    # Ищем форму для ввода пароля
    vk_password_input = driver.find_element_by_css_selector("#index_pass")
    # Кликаем по форме для ввода парроля
    vk_password_input.click()
    # Вводим пароль в форму
    vk_password_input.send_keys(password)

    # Ищем кнопку для авторизации
    vk_auth_button = driver.find_element_by_css_selector("#index_login_button")
    # Кликаем по форме для ввода пароля
    vk_auth_button.click()

    # Получаем куки из браузера
```

```
    cookies = driver.get_cookies()

    # Формируем название для сохранения файла с куками из логина и пароля вформате: login+password.cookie
    file_name = f"{login}+{password}.cookie"

    # Сохраняем куки в файл
    with open(file_name, 'wb') as f:
        pickle.dump(cookies, f)

finally:
    # Закрываем браузер
    driver.close()
```

Запустим скрипт, и после того как он отработает, в нашей директории появится файлик, с именем вида: логин+пароль.cookie:



Теперь эти куки можно открыть и при помощи модуля pickle конвертировать в словарь, а далее загрузить в браузер или использовать при работе с сайтом через запросы.

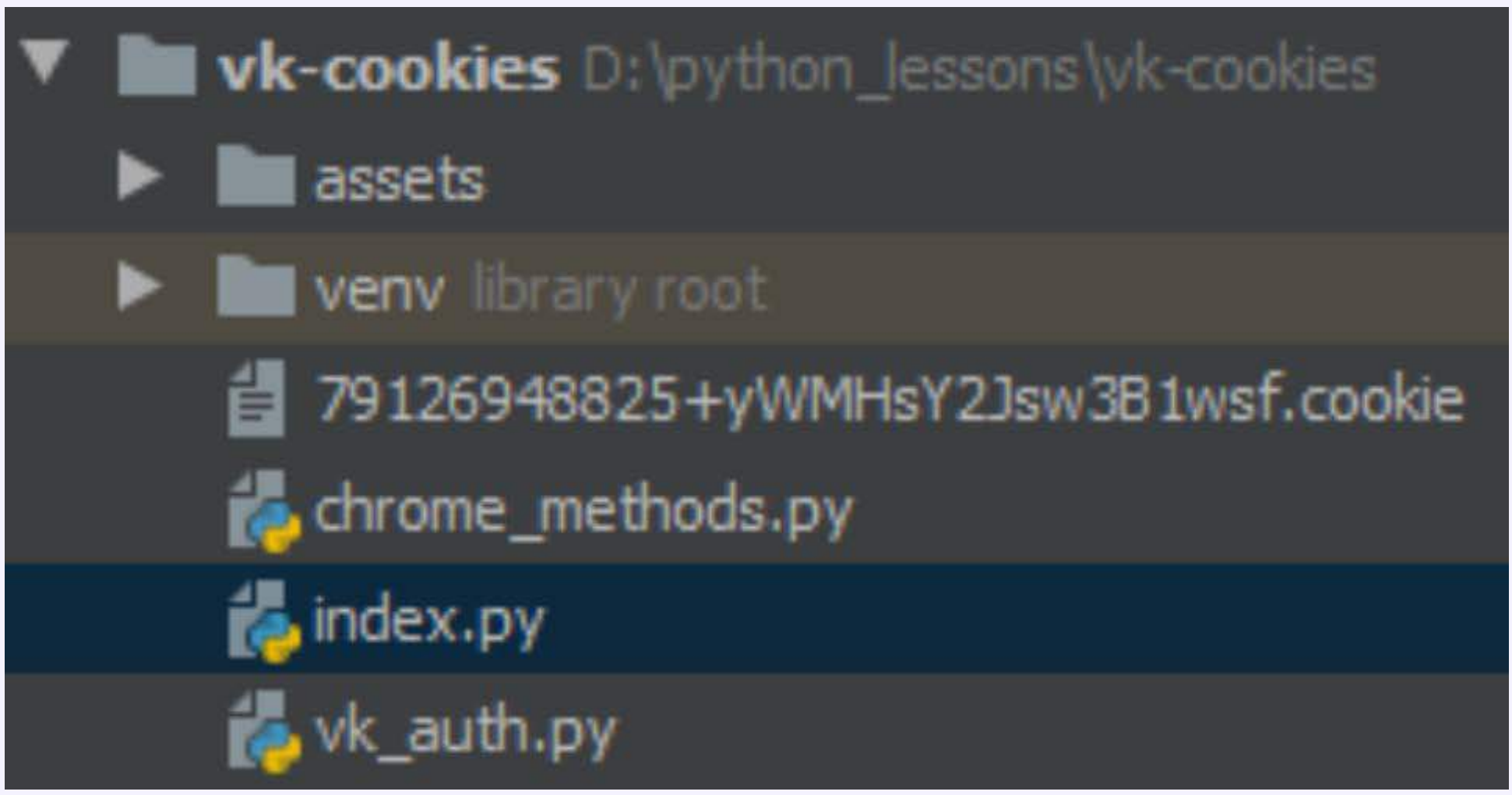
Давайте сделаем модуль чуть более удобным, и завернём его в функцию, в которую можно будет передать логин и пароль для авторизации, а затем многоразово вызывать этот модуль там, где вам это необходимо:

```
import pickle
import chrome_methods


def login(login, password):

    # Создаём объект браузера
    driver = chrome_methods.init_chrome()

    try:
        # Переходим на сайт vk.com
        driver.get("https://vk.com/")

        # Ищем инпут для ввода логина
        vk_login_input = driver.find_element_by_css_selector("#index_email")
        # Кликаем по форме для ввода логина
        vk_login_input.click()
        # Ввводим логин в форму
        vk_login_input.send_keys(login)

        # Ищем форму для ввода пароля
```

```
        vk_password_input = driver.find_element_by_css_selector("#index_pass")
        # Кликаем по форме для ввода парроля
        vk_password_input.click()
        # Вводим пароль в форму
        vk_password_input.send_keys(password)

        # Ищем кнопку для авторизации
        vk_auth_button = driver.find_element_by_css_selector("#index_login_button")
        # Кликаем по форме для ввода пароля
        vk_auth_button.click()

        # Получаем куки из браузера
        cookies = driver.get_cookies()

        # Формируем название для сохранения файла с куками из логина и пароля вформате: login+password.cookie
        file_name = f"{login}+{password}.cookie"

        # Сохраняем куки в файл
        with open(file_name, 'wb') as f:
            pickle.dump(cookies, f)

    finally:
        # Закрываем браузер
        driver.close()
```

Пишем контроллер

Теперь давайте создадим новый файл, в котором для примера будем вызывать нашу функцию, назовём его index:



В файл index импортируем наш модуль vk_auth:

```
import vk_auth
```

Далее, создадим список с данными для авторизации нескольких аккаунтов:

```
# Создаём список словарей с логнинами и паролями для авторизации
auth_base = [
    {"login": "7985632358", "password": "sfecr433t3rgd"},
    {"login": "79081256357", "password": "fcgtrgvb"},
    {"login": "79501475639", "password": "dcxgvgtv"},
    {"login": "79135997493", "password": "bjunki"},
]
```

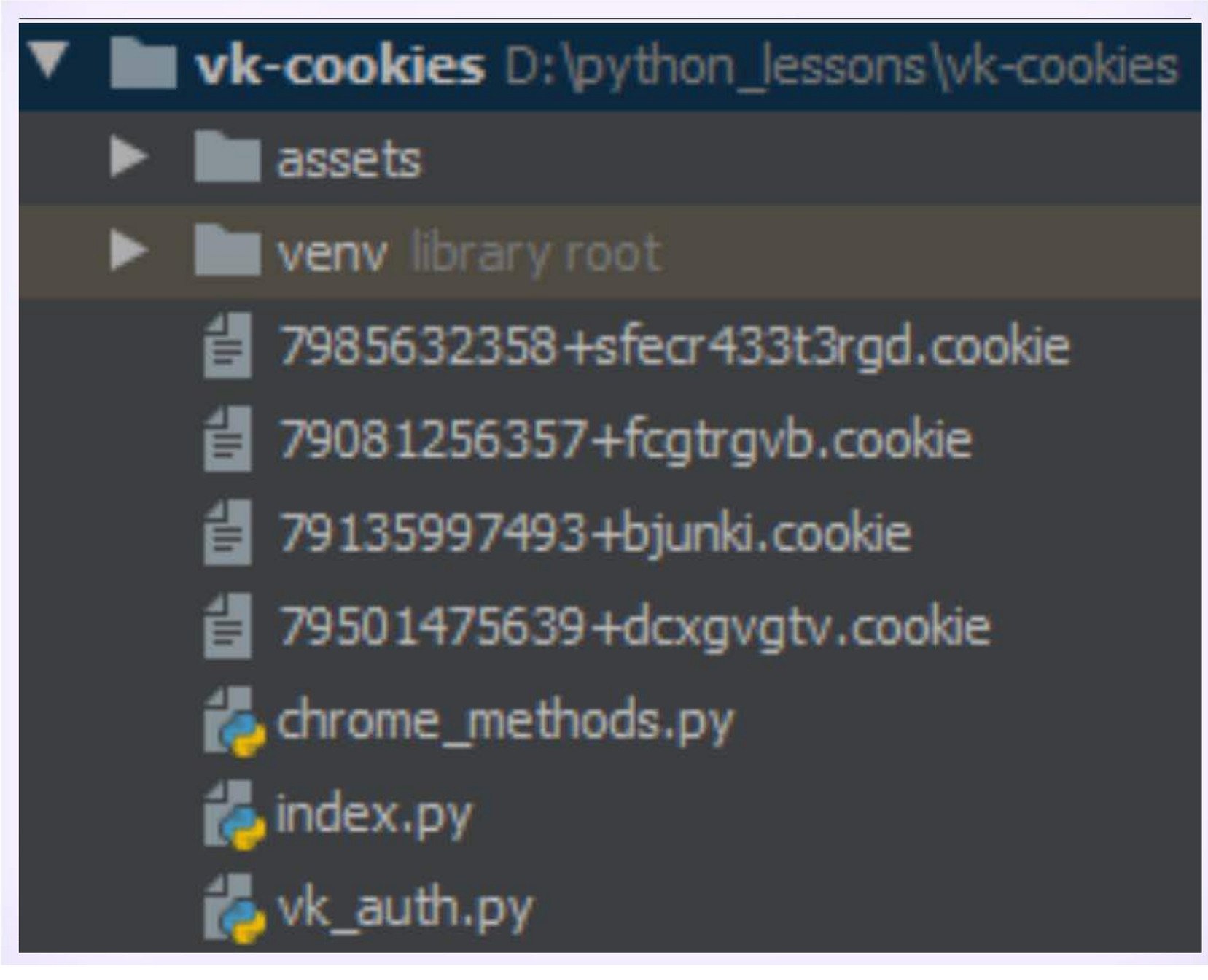
Теперь в цикле запустим процесс авторизации для каждого аккаунта:

```
# Запускаем в цикле процесс
# авторизации для каждого аккаунта
for auth_item in auth_base:
    vk_auth.login(auth_item["login"], auth_item["password"])
```

Запустим этот код:

```
import vk_auth

# Создаём список словарей с логнинами и паролями для авторизации
auth_base = [
    {"login": "7985632358", "password": "sfecr433t3rgd"},
    {"login": "79081256357", "password": "fcgtrgvb"},
    {"login": "79501475639", "password": "dcxgvgtv"},
    {"login": "79135997493", "password": "bjunki"},
]

# Запускаем в цикле процесс
# авторизации для каждого аккаунта
for auth_item in auth_base:
    vk_auth.login(auth_item["login"], auth_item["password"])
```

В итоге было создано 4 файла, с куками для каждого аккаунта:



Отлично, теперь мы освоили работу с Selenium на практике.

Учимся сканировать запросы

Содержание урока

Понятие HTTP

Многие пользователи знают, как работает сеть интернет, однако, думаю стоит разобрать этот момент подробнее. По большей части, интернет работает через протокол http/s.

HTTP – это в сущности клиент-серверный протокол передачи данных, который позволяет отправлять запросы от клиента (например от браузера) и принимать их на сервере, и на оборот, отправлять запросы с сервера на клиент.

Проще говоря, HTTP протокол – это некоторый свод правил, по которому происходит «общение» между клиентом (например, браузером) и сервером.

Общение по HTTP протоколу, производится посредством HTTP-запросов. Пользователь отправляет HTTP-запрос на сервер (например, запрос на авторизацию), серевер принимает запрос, обрабатывает входящие от клиента данные (например, логин и пароль) и возвращает клиенту (браузеру) ответ.

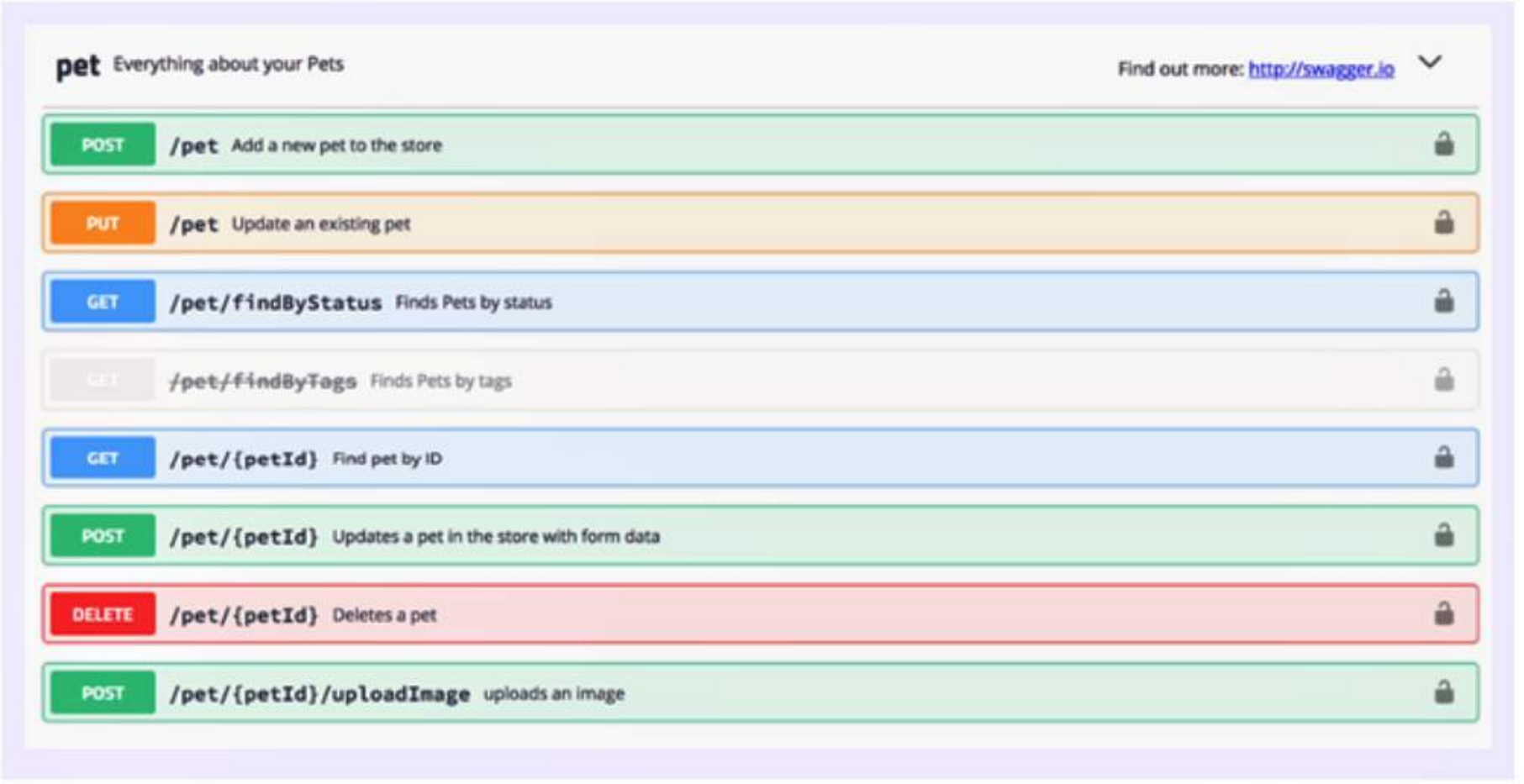
Например, HTML-страницу с данными, которые могут быть доступны только авторизованному пользователю. В свою очередь, клиент эти данные как-то обрабатывает, например, в случае с браузером выводит HTML-код в виде интерфейса, или парсит этот HTML извлекая из него нужные данные для дальнейших профитных схем :)

HTTP-запросы

HTTP-запросы бывают разными, в зависимости от метода, который отправляет запрос. Полный список методов HTTP: https://developer.mozilla.org/ru/docs/Web/HTTP/Methods

Что такое методы запроса? Условно, это некоторый маркер, который отправляется вместе с запросом и позволяет серверу понять, что нужно сделать в ответ на этот запрос. Создать новую запись, удалить её и т.д.

На картинки методы обозначены в цветных прямоугольниках:



На самом деле, в современном вебе как правило используют 2 типа запросов (и оба вы скорее всего слышали):

1. GET
2. POST

Метод GET по спецификации должен только возвращать ресурс (например, веб-страницу). Когда вы переходите по ссылке или вводите адрес сайта в браузере (vk.com) – это GET запрос, и он должен (но не обязан) возвращать HTML код главной страницы VK.

Метод POST по спецификации используется для создания каких-либо ресурсов на сервере. Например, метод POST используют для отправки (а соответственно и создания) сообщения в ВК. Когда вы пишите сообщение на форуме или в соц. сети, нажимая на кнопку «Отправить» скорее всего ваше сообщение уходит на сервер именно через метод POST.

Но очень часто, метод POST используют и для обновления, и для удаления ресурсов. Например, удалить или обновить статью на сайте.

При помощи какого метода необходимо отправлять запрос определяется сервером, точнее кодом, который написан на сервере. Поэтому для того, чтобы узнать, какой тип запросов нужен для того, чтобы добавить, например, новый комментарий на некотором сайте, необходимо перейти на этот сайт через браузер, оставить комментарий и просканировать, на какой адрес, с какими данными и типом был отправлен запрос.

Подведём итог под вышесказанным. Взаимодействие с любым веб-сайтом, это отправка http-запросов на сервер. Даже когда вы работаете с сайтом через браузер, вы всё равно отправляете те же самые http-запросы на сервер.

Чтобы отправлять запросы без браузера, достаточно узнать, на какой адрес и с какими данными отправляется запрос через браузер, а потом эмулировать это же действие через http-клиент.

Чистые запросы без браузера

В прошлой главе мы рассматривали браузерную автоматизацию и у вас может возникнуть вопрос. А зачем нам париться с чистыми запросами, когда мы можем просто автоматизировать браузер. Проблема браузера в том, что он является своего рода медиапроигрывателем.

При работе с браузером, открывая страницу вам приходится ждать, пока браузер отрендерит HTML и проинтерпретирует JS-код. Вам придётся:

1. Зайти на страницу, дождаться её загрузки
2. Найти кнопку авторизации и нажать на неё
3. Заполнить всплывающую форму
4. Отправить запрос через форму

Каждый этот пункт требует от 5 до 25 секунд на выполнение. В то время как отправка POST-запроса затратит 1.5-2 секунды. Да, POST запрос нужно сперва составить, иногда это гораздо сложнее чем написать просто браузерную автоматизацию, но автоматизация на запросах, без браузера в разы быстрее.

Так что лучше использовать? Лучше использовать и то, и то. Например, регеры и скрипты для получения cookies я пишу обычно в виде браузерной автоматизации, т.к. самая сильная защита от автоматическмх запросов реализована как правило на страницах регистрации и автоматизации.

Поэтому для дальнейшей работы аккаунтов гораздо безопаснее, если они зарегистрированы через браузер.

Сканируем запросы

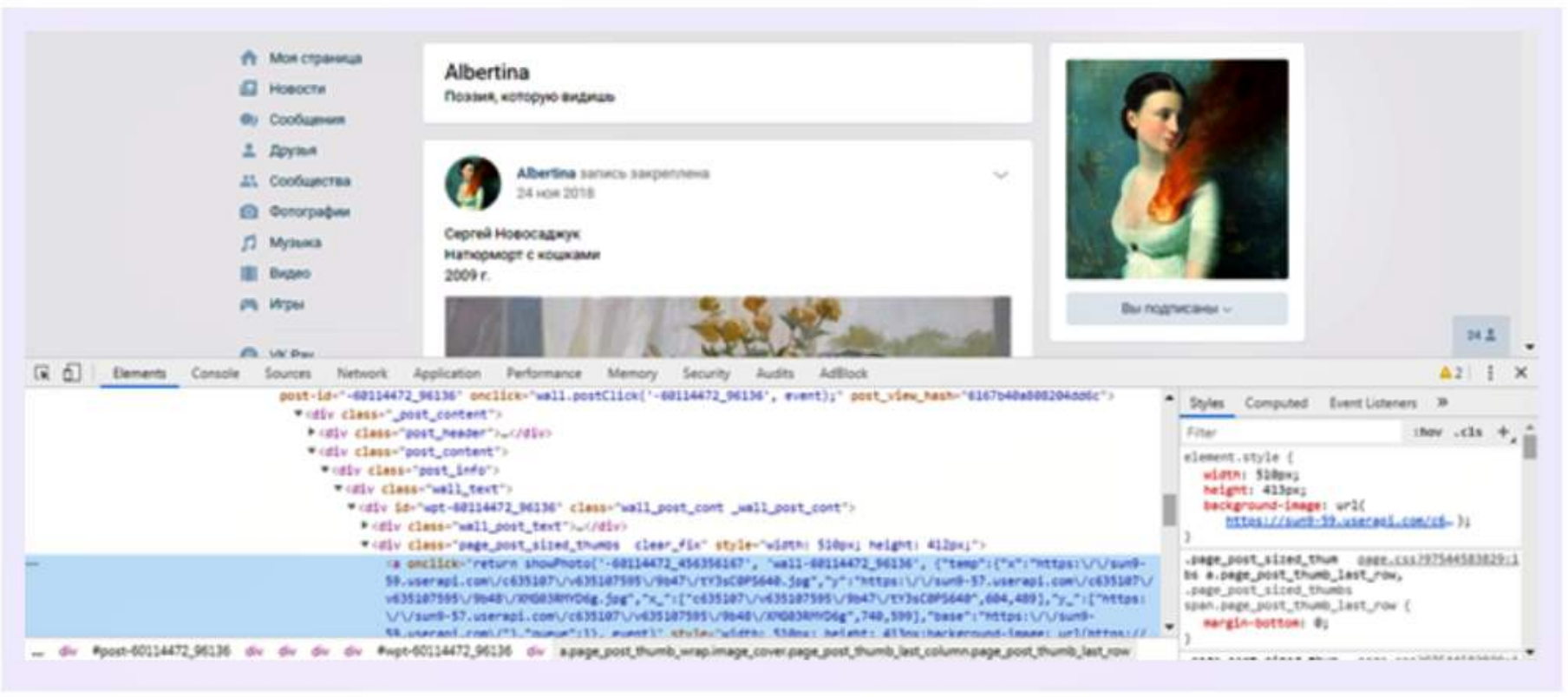
Процесс сканирования запросов сводится к тому, что мы при помощи встроенных инструментов браузера выполняем определённые целевые действия (например, ставим лайк к посту) и смотрим:

1. На какой адрес был оправлен запрос
2. Какой метод был у запроса
3. Какие данные необходимы для запроса

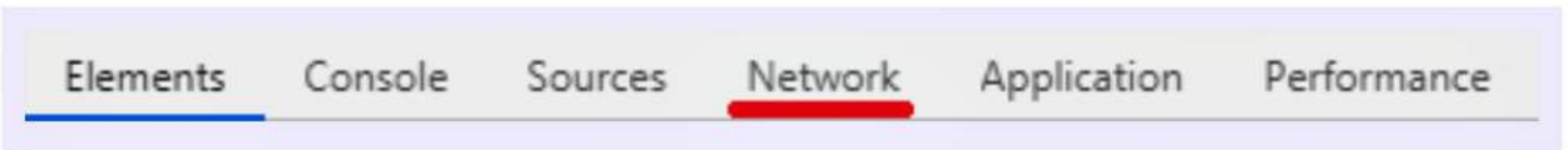
Давайте попробуем просканировать запрос, который отправляется, когда мы ставим лайк к какому-нибудь посту в ВК. Сканировать запросы мы будем при помощи инструментов разработчика Google Chrome.

Перейдём в браузере Google Chrome в какое-нибудь сообщество, например: https://vk.com/al_bertina

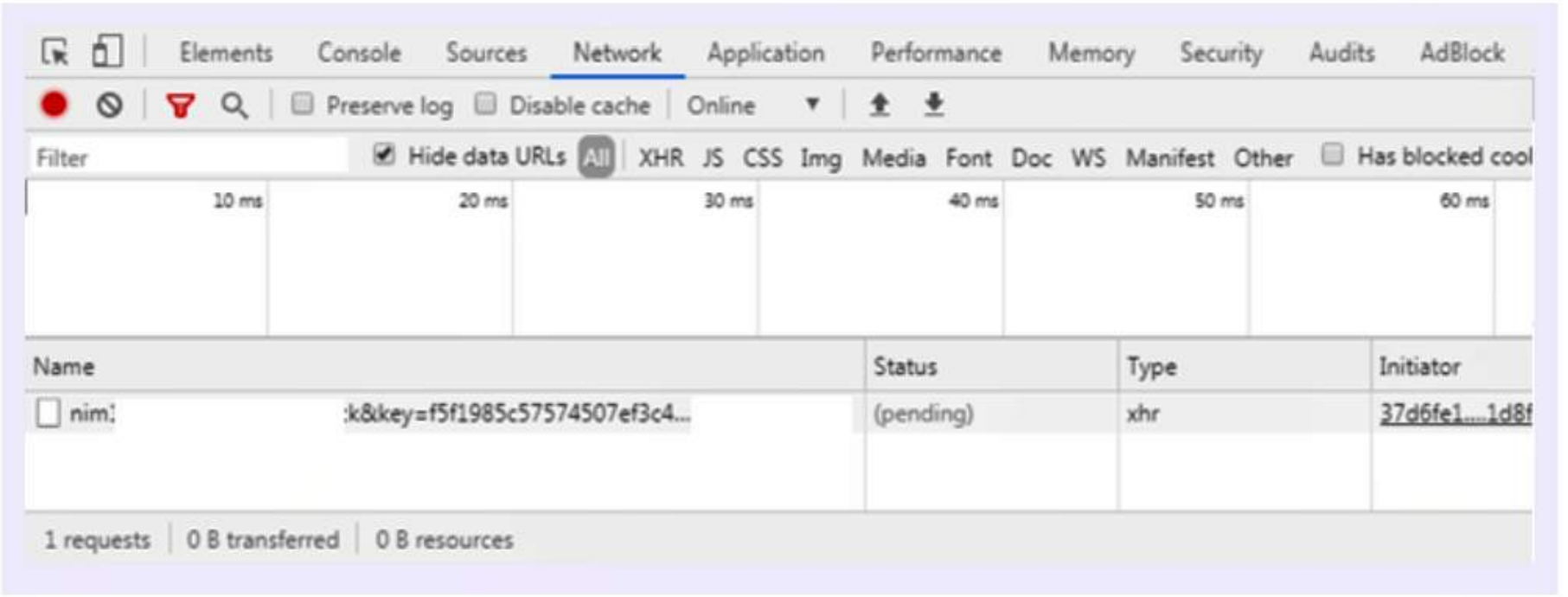
Теперь в любой области веб-страницы нажимаем ПКМ, далее, «Просмотреть код», откроется такое окно:



В этом окошке вы можете увидеть вкладку Network:



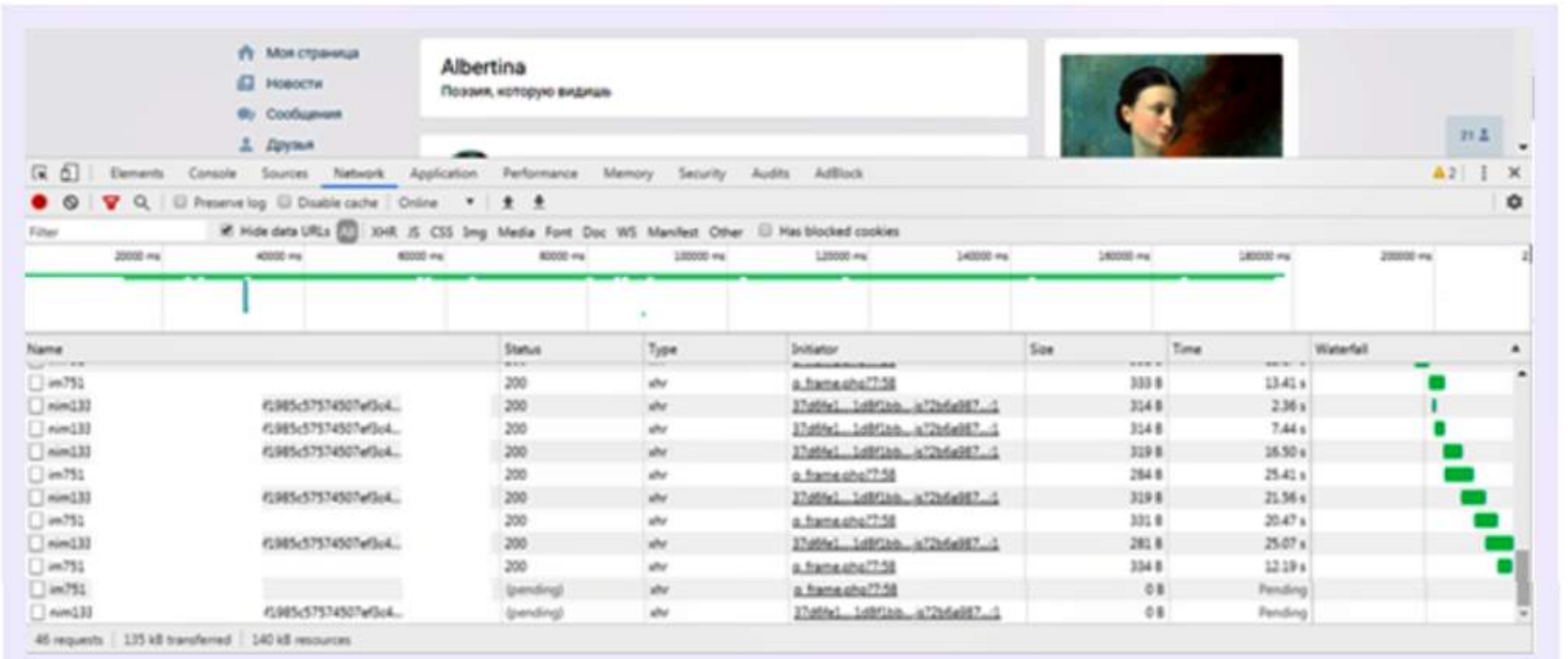
Кликаем по ней, открывается такой интерфейс:



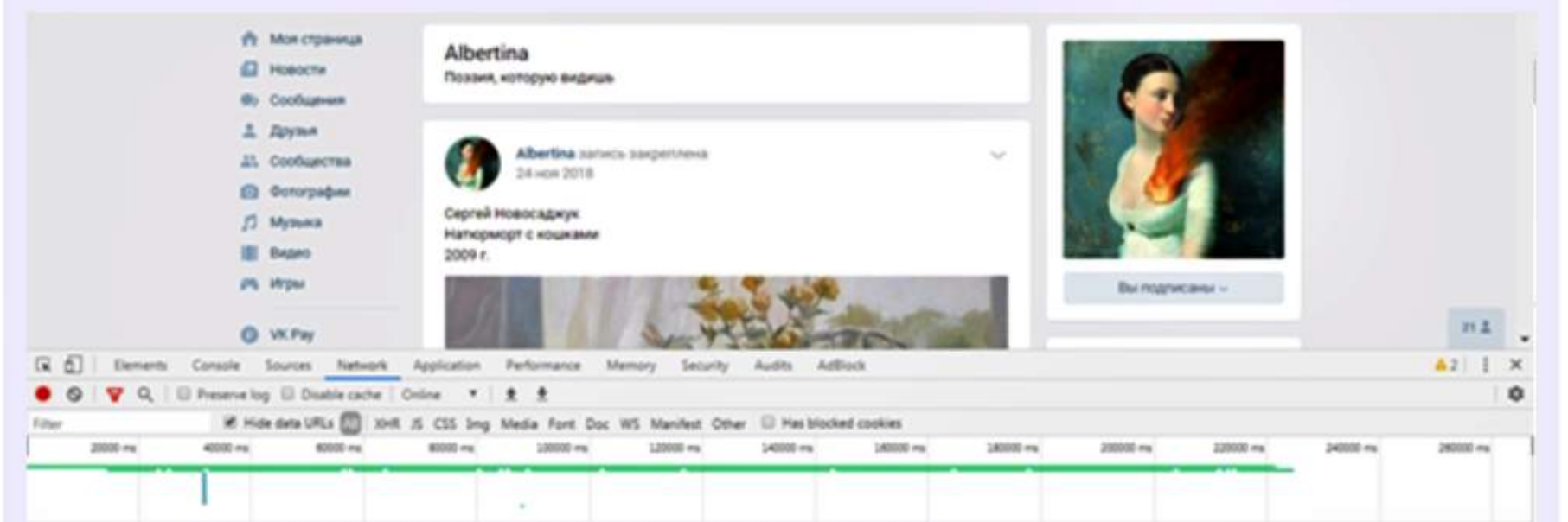
Перед сканированием, обязательно убедитесь, что кружок находящийся вверху панели по левой стороне горел красным цветом. Если он горит серым, нужно кликнуть по нему тем самым вы начнёте запись сетевых запросов.

Теперь разберёмся с самой панелью. Если вы наведёте на вершнюю грань панели мышью, её можно делать выше или ниже.

Поднять:

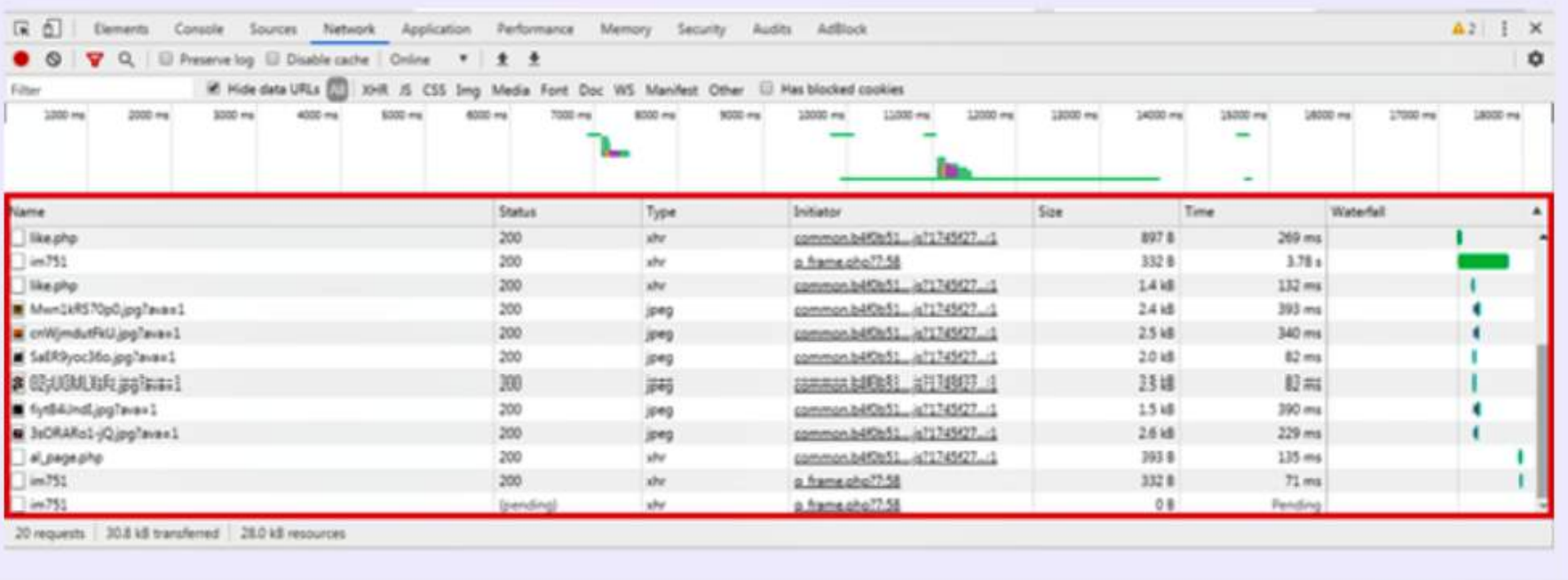


Или опустить:



Это удобно. Когда нам нужно что-то сделать со страницей, мы сворачиваем панель. Когда нам нужно работать с панелью, мы напротив её растягиваем.

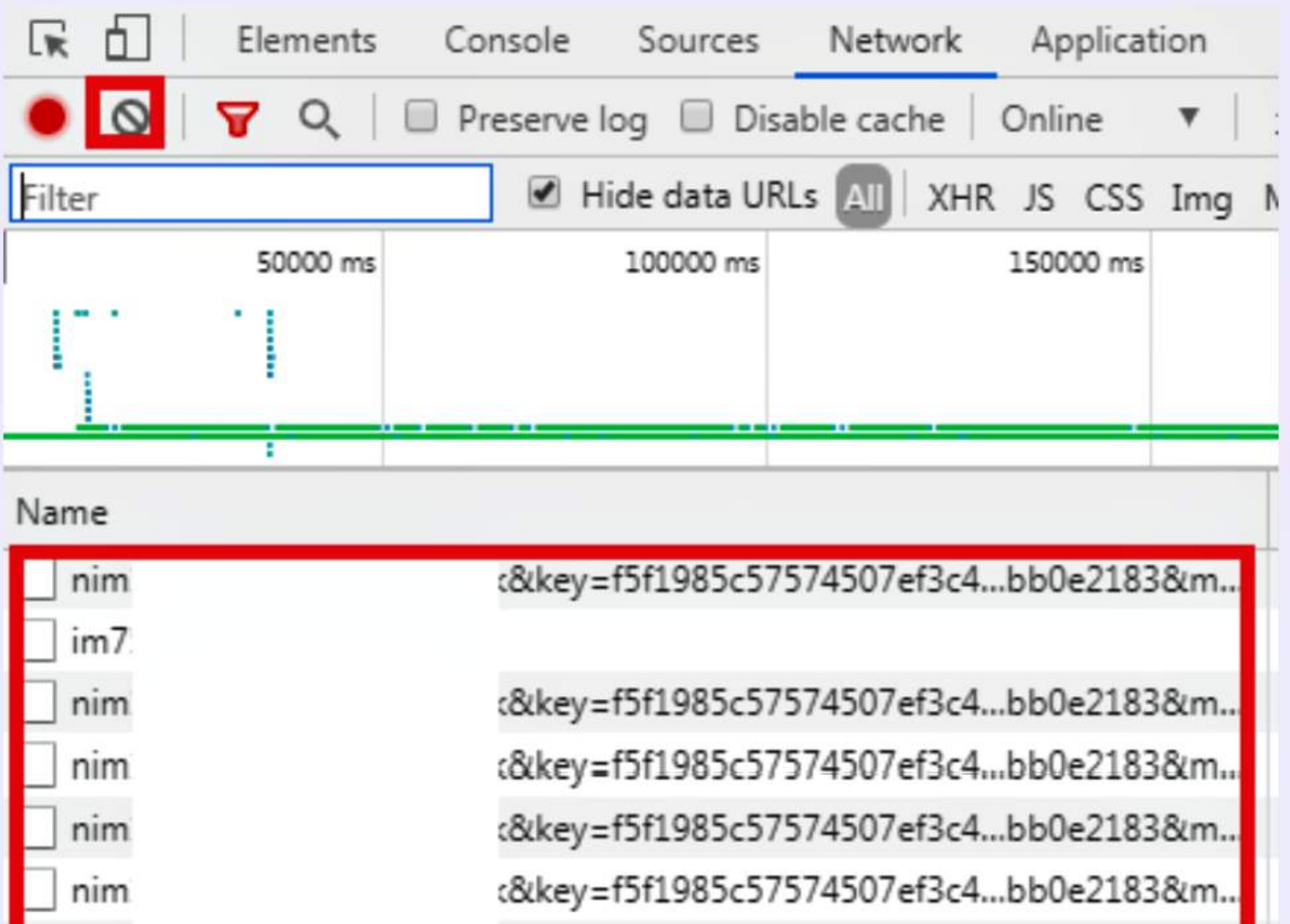
Красной рамкой обозначено окно:

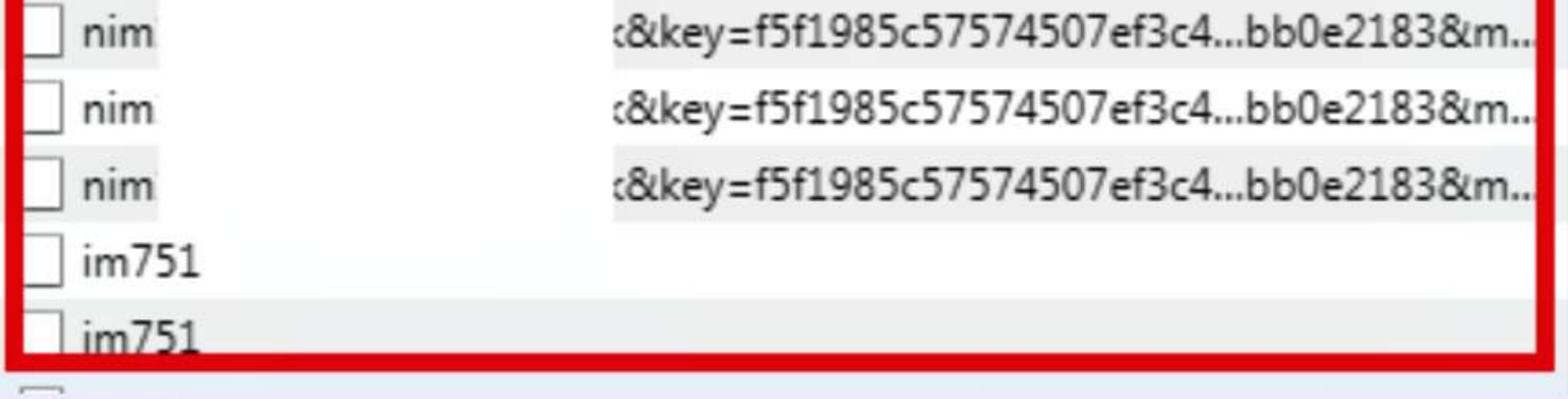


Внутри этого списком будут отображаться все запросы, отправленные браузером на сервер. Именно здесь мы и будем выискивать нужные нам.

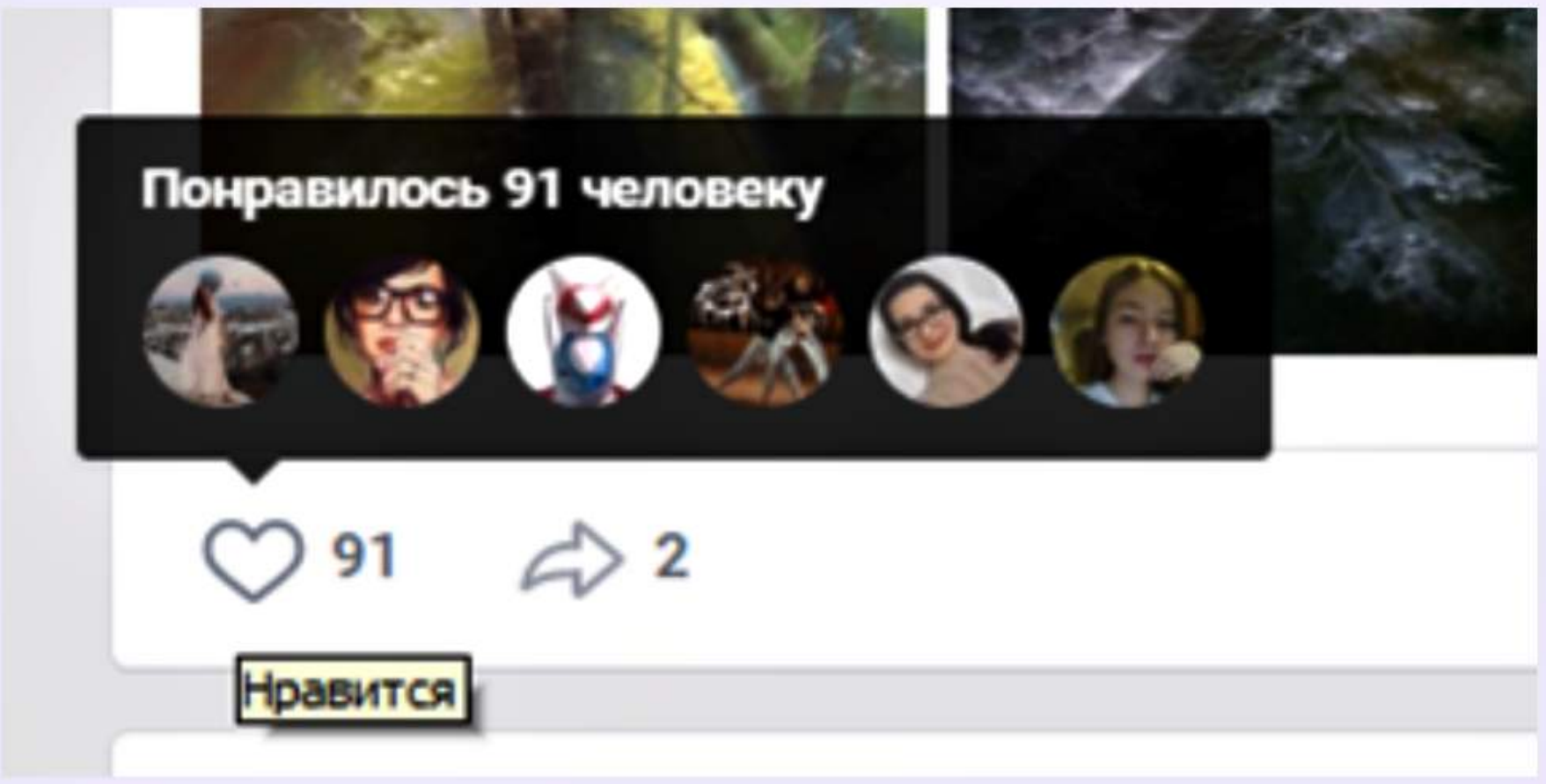
Итак, мы перешли на страницу сообщества. Теперь нам нужно выполнить целевое действие (сделать лайк записи), но перед этим давайте очистим список запросов, чтобы нам ничего не мешало.

Нажмите на кнопку, выделенную на скриншоте и список запросов очистится:

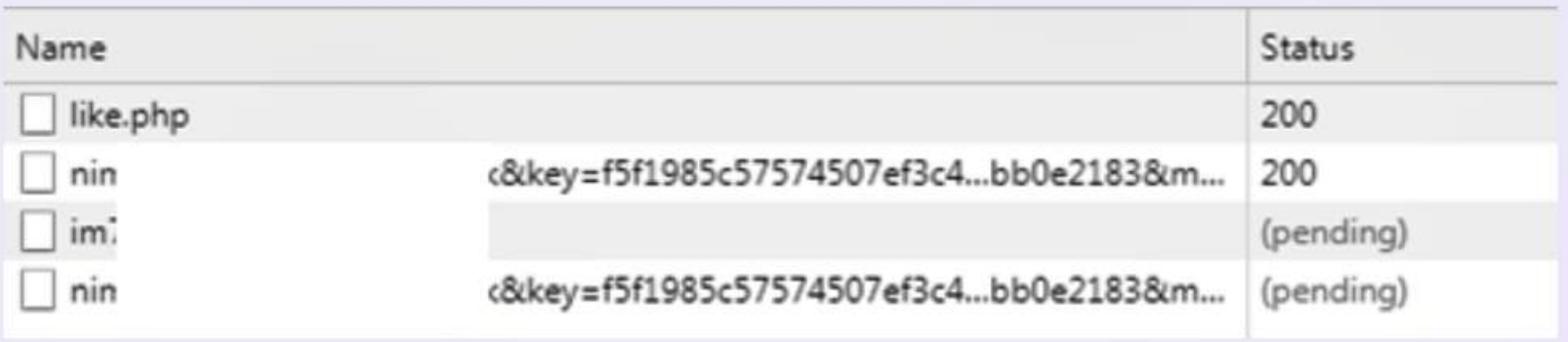




Выполним целевое действие, нажмём на кнопку «Мне нравится» у какой-нибудь записи:

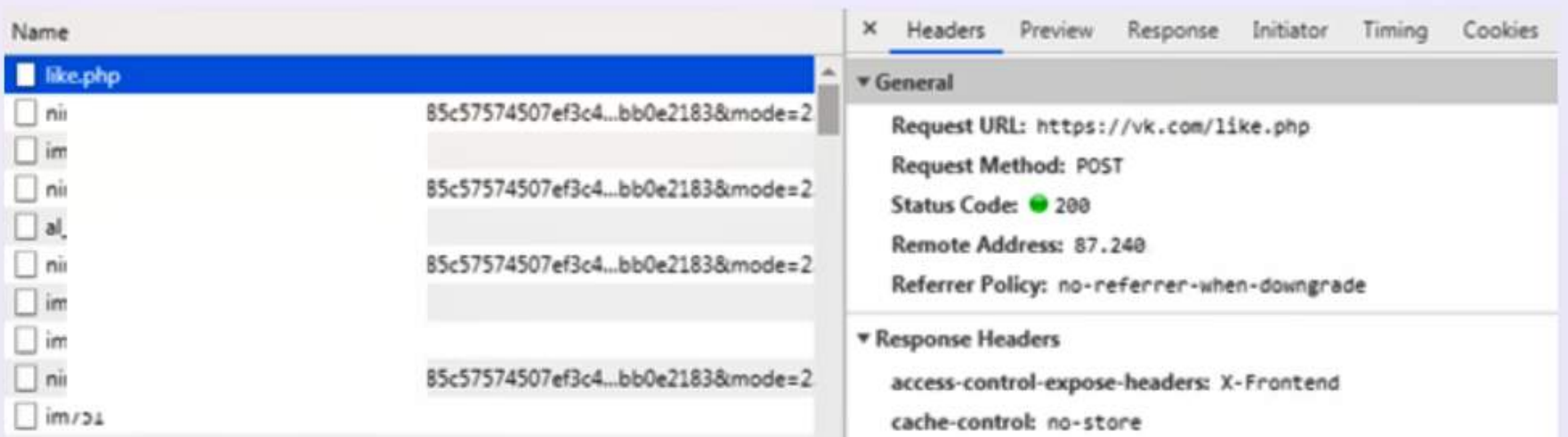


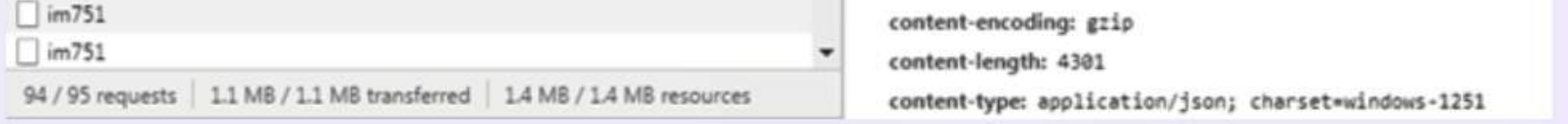
Нажав на кнопку мне нравится, в списке запросов мы видим:



Путём несложных логических умозаключений несложно догадаться, что запрос, при помощи которого мы поставили лайк имеет имя `like.php`

Кликнем по этому запросу, откроется вкладка с подробностями:





На какой адрес был отправлен запрос

Здесь вся информация по запросу, на какой адрес отправлен запрос отмечается в графе Request URL.

Какой метод использовался при отправке запросов

HTTP-метод в графе Request Method.

Какие данные необходимы для запроса

Как правило, данные отправляются при помощи заголовков запроса и данных форм (обычных или скрытых форм, по типу формы регистрации).

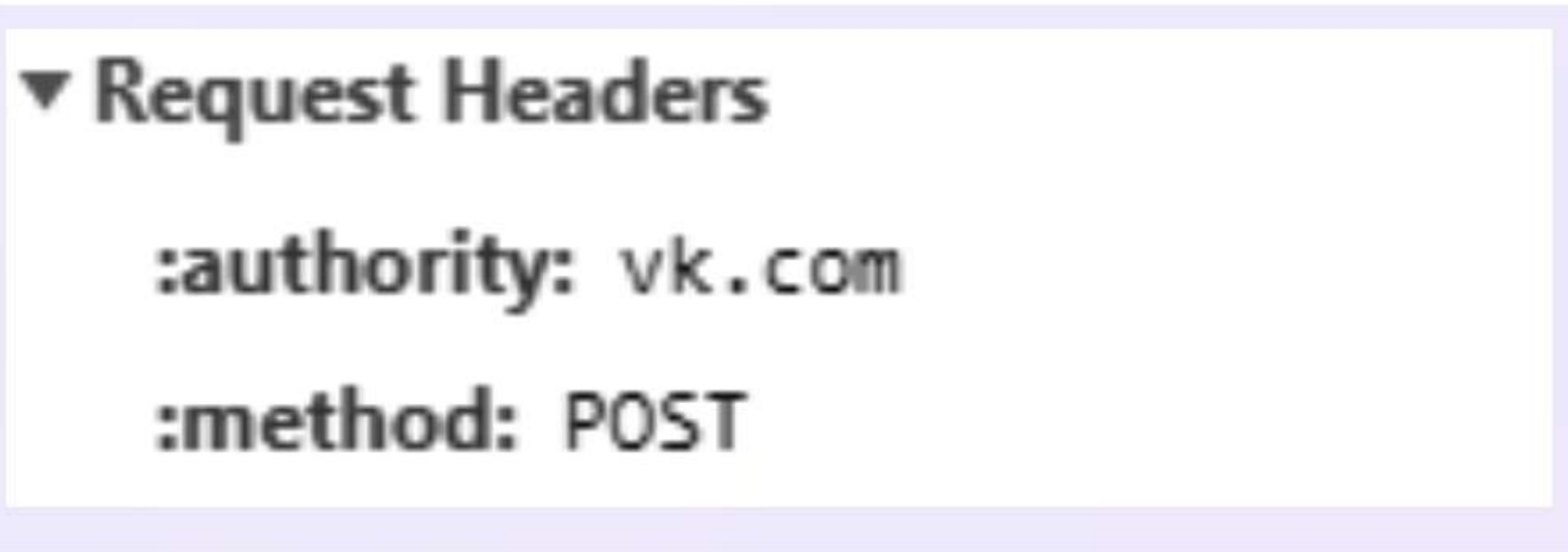
Заголовки запросов

Заголовки, это специальные строки используемые для передачи информации на сервер, в виде:

```
<имя заголовка>:<значение заголовка>
```

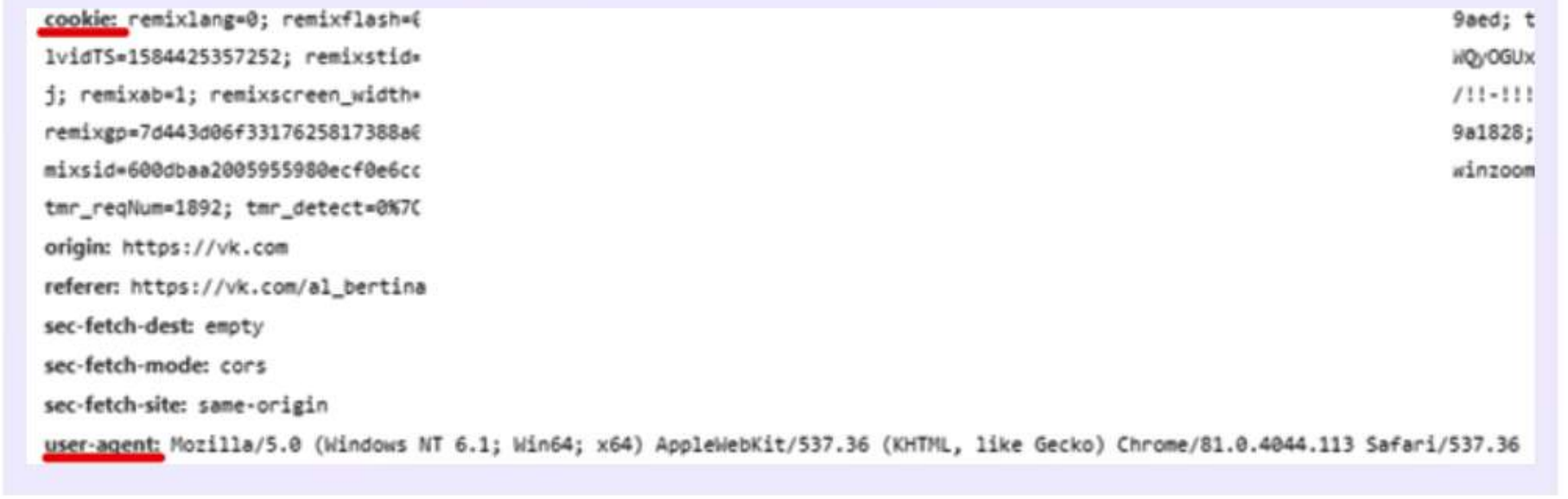
При отправке запросов на сервер, браузером отправляются специальные заголовки с данными, иногда эти данные не нужны серверу и не обязательно эмулировать отправку всех заголовков. Как определить, какие заголовки отправлять, а какие нет.

Данные, которые уходят вместе с запросом на сервер в виде заголовков находятся в разделе Request Header:



Если вы откроете этот раздел, то увидите много всяких данных:

Большая часть этих данных не критична для сервера, и здесь нужно определить какие данные нужны, а какие – нет. Как правило, сервер требует обязательным образом только уникальные данные, характерные для каждого конкретного пользователя:



В нашем случае, это куки и юзер-агент, так как эти данные различаются у всех пользователей. А остальные данные – статичны и носят скорее всего статистический характер. Отметим это у себя в голове или на листочке.

Данные из формы

Также, мы можем заметить раздел Form Data:

Это тоже данные которые всегда отправляются на сервер. Если вы видите в описании запроса блок Form Data, будьте уверены, что они уходят на сервер и их тоже необходимо передать на сервер.

Как видим, форма передаёт какие-то второстепенные статические параметры параметры, и два динамических (которые могут меняться в зависимости от условий), это некоторый параметр hash (какая-то защитная строка, сгенерированная сайтом) и параметр object – собственно, ссылка на пост, который мы лайкнули: `https://vk.com/wall-60114472_123633`

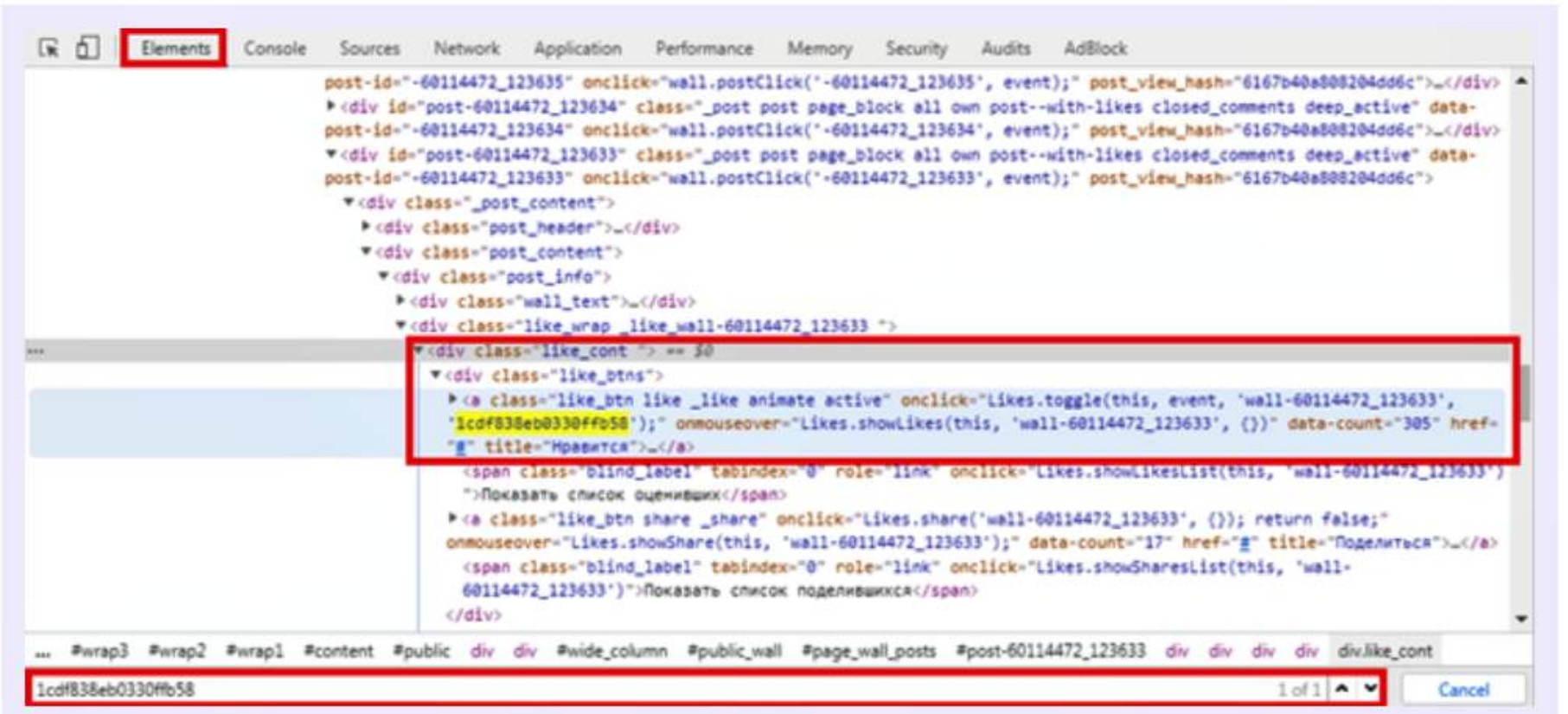
Только без самого домена `https://vk.com/`

Разберём все данные

Итак, нам нужно по адресу из графы Request URL (https://vk.com/like.php), отправить запрос методом из графы Request Method (POST). В заголовке запроса передать user agent и куки пользователя. А в данных формы защитный hash (1cdf838eb0330ffb58) и ссылку на пост (wall-60114472_123633).

Где взять юзер агент – мы знаем, в интернете полно баз откуда их можно скачать. Как получить куки мы тоже знаем (в прошлом уроке проходили). Остался один вопрос, что это за hash. Как правило, всякого рода непонятные последовательности символов, передаваемые на сервер содержатся в HTML-коде страницы.

Давайте попробуем узнать, где размещён этот hash. Для этого, сперва скопируем его. Далее, перейдём во вкладку Elements. Далее, нажимаем сочетание клавиш ctrl + f (поиск по коду), затем ctrl + v (вставить хеш, который вы скопировали), далее нажимаем клавишу enter (найти скопированный хеш в коде):



Как видите, у каждого поста свой уникальный хеш, который отправляется на сервер при совершении действия «Мне нравится», этот хеш можно будет вытащить при помощи специальных python-библиотек для парсинга HTML (о которых мы поговорим в следующих уроках).

Итог

Теперь у нас есть вся информация для совершения запроса. По адресу `vk.com/like.php` мы должны отправить POST-запрос, в заголовках которого мы передаём куки и юзер агент, а в данных формы – хеш (который лежит внутри HTML-кода страницы) и ссылку на пост, который мы лайкаем.

«Заголовки» и «данные формы» – это всего лишь два разных контейнера, которые передают данные на сервер.

О том, как всё это отправить на сервер мы поговорим в следующих уроках.

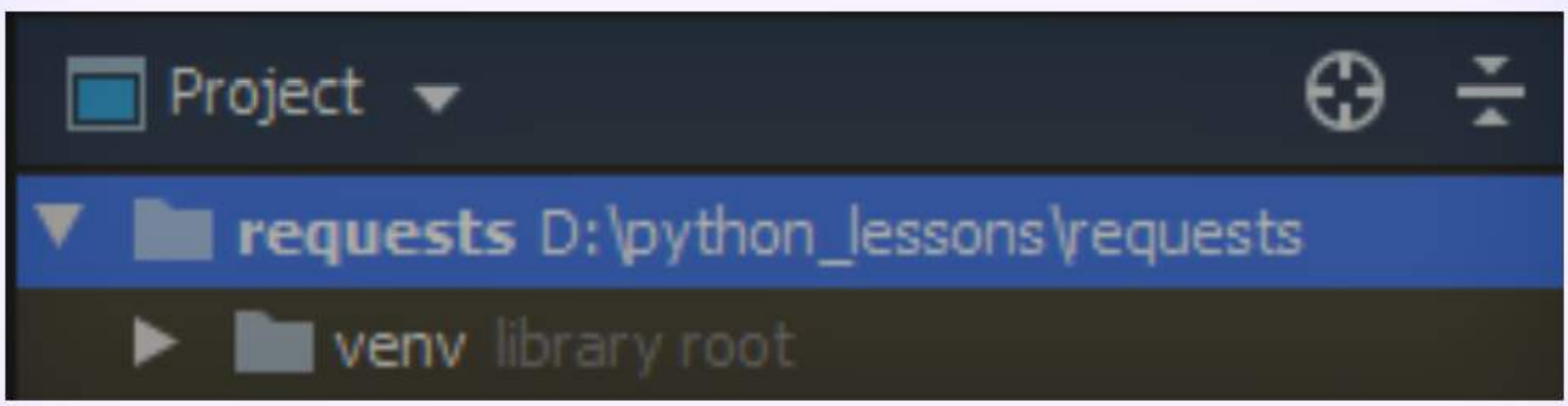
Отправляем запросы на сайт

Содержание урока

В прошлом уроке мы поговорили о том, что такое HTTP-запросы, из чего они состоят и как они отправляются. Теперь давайте научимся отправлять запросы на сервер при помощи Python. Для этого мы воспользуемся python-библиотекой Requests.

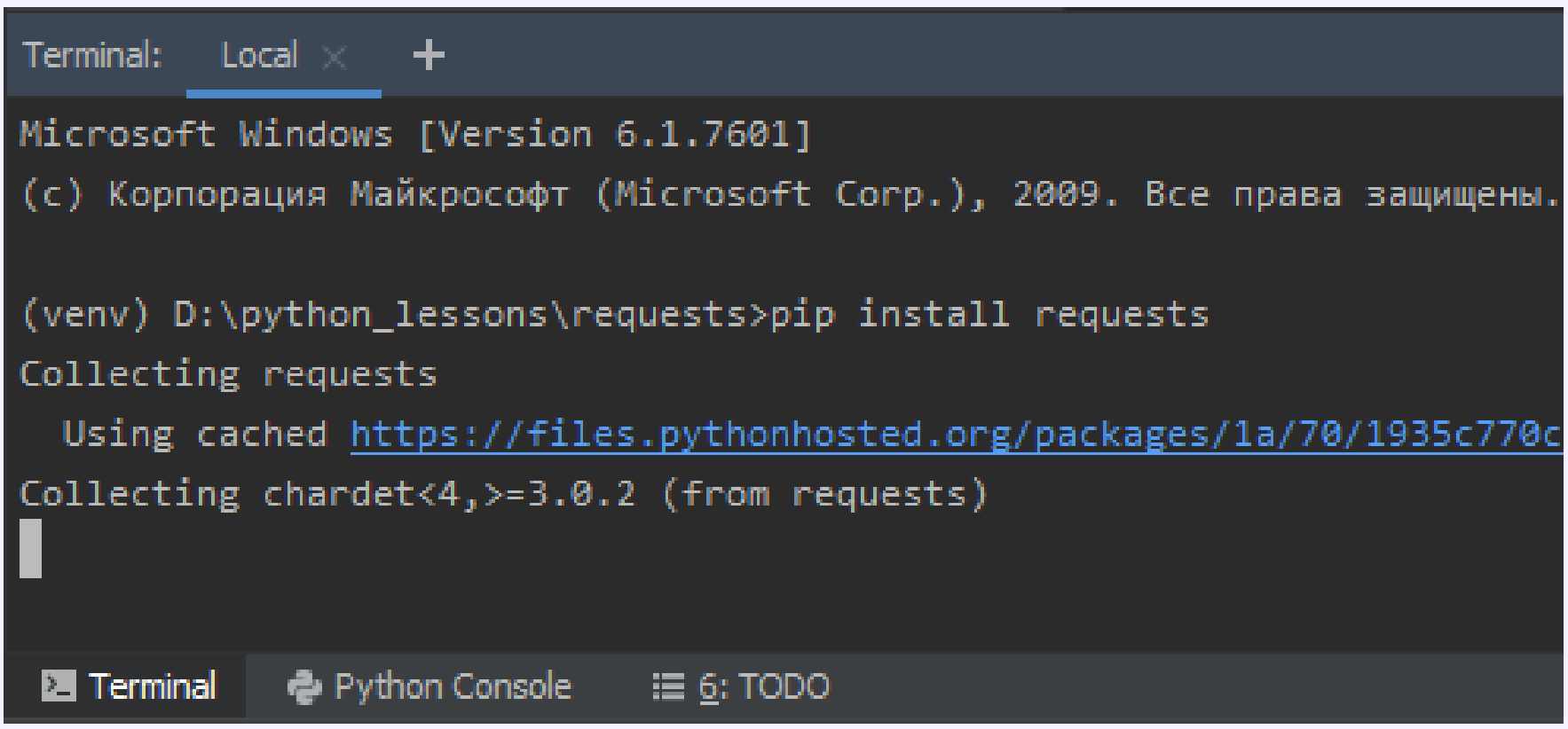
Документация Requests: https://requests.readthedocs.io/

Создадим новый проект, например, назовём его requests:



Далее, создаём в корневой директории нашего проекта файл, назовём его, например, index.py. В этом файле мы и будем писать код нашего проекта. Всё готово, теперь установим библиотеку Requests в наш проект. Введём в терминале нашего проекта команду:

```
pip install requests
```



После успешной установки импортируем библиотеку в наш рабочий файл index.py:

```
import requests
```

Теперь мы имеем возможность при помощи данной библиотеки отправлять запросы на сервер. Как мы уже знаем, запросы имеют разные типы, такие как:

Get
Post
Put
Delete
Head
Options

Мы рассмотрим два основных, это get и post.

Get-запросы в Requests

Для отправки get-запроса у модуля requests имеется одноимённая функция get:

```
import requests

r = requests.get('https://api.github.com/events')
```

В данном случае, мы отправили Get-запрос по адресу https://api.github.com/events, и в переменную r получили объект, с ответом сервера.

Давайте попробуем посмотреть, что нам ответил сервер:

```
import requests

r = requests.get('https://api.github.com/events')

# Получаем код ответа сервера
status = r.status_code
# Получаем текст ответа
text = r.text
```

```
print(status)
print(text)
```

Выводит:

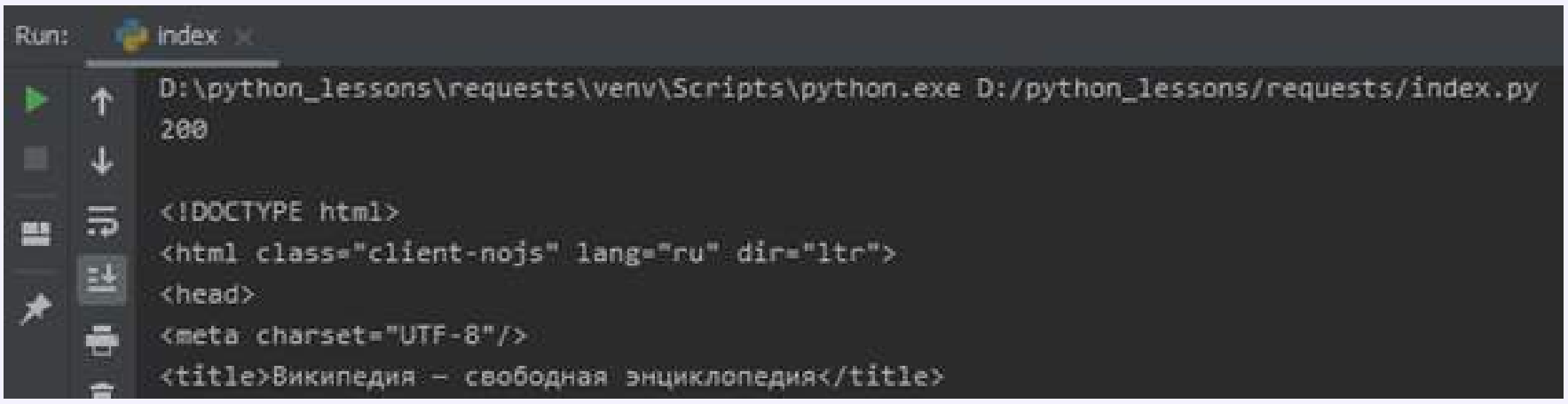
У объекта, который возвращает функция get есть переменная status_code (r.status_code), данная переменна содержит код ответа сервера, например, в нашем случае это 200 (значит, что всё хорошо). Код ответа сервера – это число, каждое число имеет какое-то значение. Например, ответ 200 указывает на то, что всё хорошо и сервер обработал входящий запрос. Ответ 404 значит что запрашиваемый ресурс – не найден.

Список кодов: https://developer.mozilla.org/ru/docs/Web/HTTP/Status

Переменная text (`r.text`) содержит данные, которые сервер прислал в качестве ответа на запрос. В нашем случае, это JSON-строка (о JSON мы поговорим позже), но может, например быть и изображение и HTML-код, например:

```
import requests

r = requests.get('https://ru.wikipedia.org/')

# Получаем код ответа сервера
status = r.status_code
# Получаем текст ответа
text = r.text

print(status)
print(text)
```

Отправим get-запрос по адресу https://ru.wikipedia.org и выведем HTML-код в консоль:



Как видите, в данном случае был выведен HTML-код, который в дальнейшем можно, например, спарсить и вытащить оттуда какие-либо данные (об этом в следующем уроке).

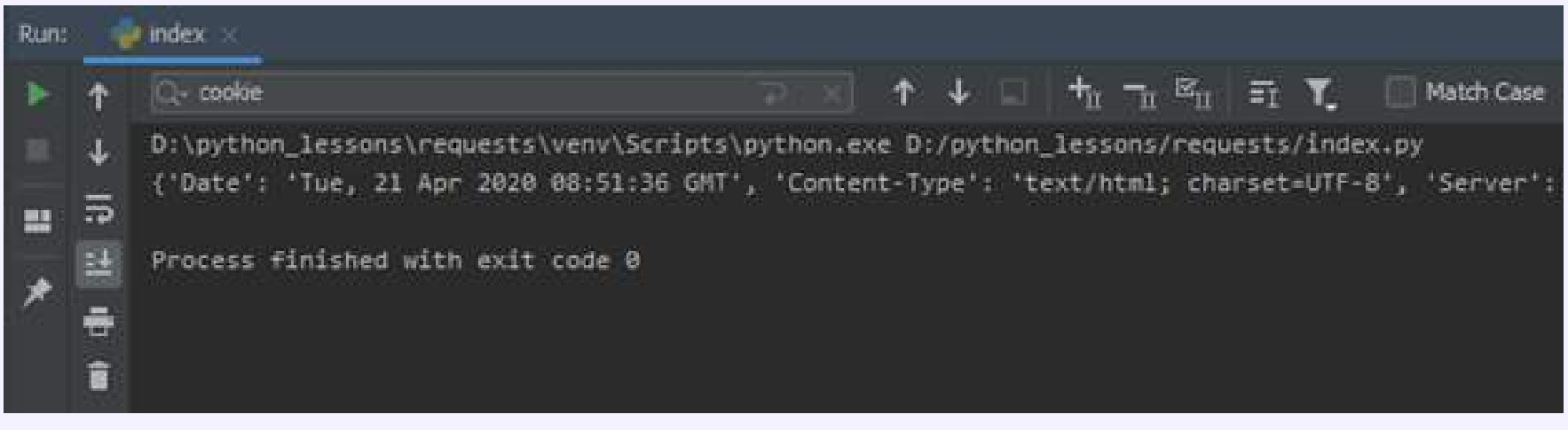
Получить заголовки

При запросе мы также можем получить список заголовков, которые были отправлены сервером при помощи переменной `headers`:

```
import requests

r = requests.get('https://ru.wikipedia.org')
```

```
# Получаем заголовки от сервера
headers = r.headers

print(headers)
```

Выводит словарь с заголовками:



Установить заголовки

Для того, чтобы установить заголовки, необходимо передать в именованный параметр headers словарь с заголовками (в формате <имя>:<значение>) в функцию отправки запроса:

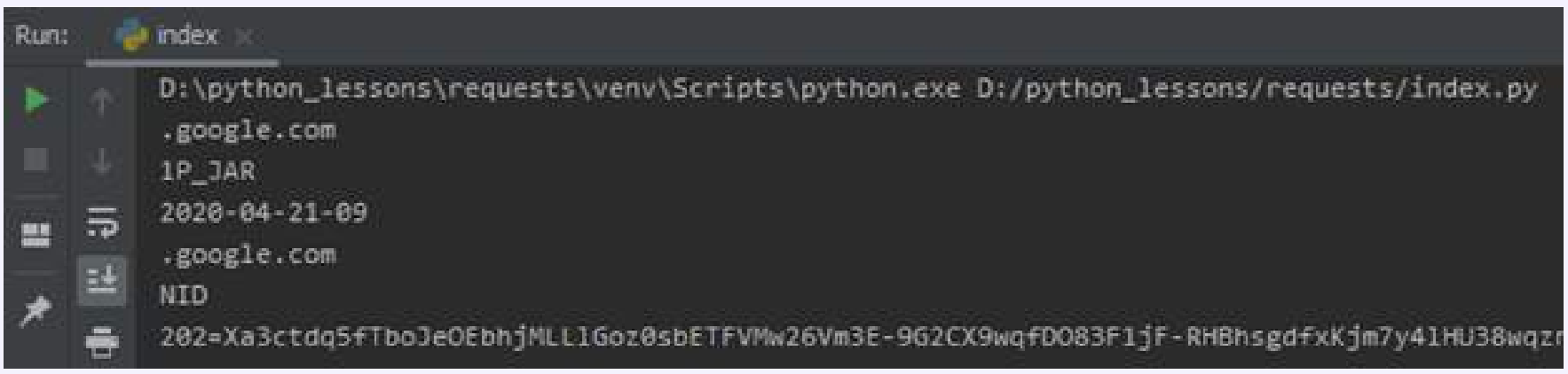
```
import requests

headers = {'user-agent': 'my-app/0.0.1'}
r = requests.get('https://www.google.com', headers=headers)
```

Получить Cookies

Получить куки которые прислал сервер можно при помощи переменной cookies:

```
import requests

r = requests.get('https://www.google.com')

# Получаем Cookies от сервера
cookies = r.cookies

# Перебираем массив с объектами cookie
for cookie in cookies:
    # Домен для которого присваивается Cookie
    print(cookie.domain)
    # Имя Cookie
    print(cookie.name)
    # Значение Cookie
```

Выводит:



Установить Cookies

Для того, чтобы установить заголовки, необходимо передать в именованный параметр cookies словарь с куками (в формате <имя>:<значение>) в функцию отправки запроса:

```
import requests

headers = {'user-agent': 'my-app/0.0.1'}

set_cookies = {
    "login": "admin",
    "password": "password",
    "name": "mark"
}
r = requests.get('https://www.google.com/', headers=headers, cookies=set_cookies)
```

Установка Get-параметров

Get-параметры, это параметры вставленные в URL, после знака «?»: https://www.google.com/search?`q=set+cookies+in+python+requests&oq=set+cookies`

Как правило, get-параметры указывают серверу каким образом отобразить страницу. Чтобы отправить get-запрос с параметрами необходимо передать в именованный параметр params словарь с get-параметрами (в формате <имя>:<значение>) в функцию отправки запроса:

```
import requests

set_params = {
    "q": "set cookies in python requests",
    "oq": "set cookies"
}

r = requests.get('https://www.google.com/search', params=set_params)
```

Установка proxy

Для установки proxy-сервера у методов отправки запросов есть параметр proxies, в который необходимо передать словарь с прокси серверами:

```
proxies = {
    'http': 'http://10.10.1.10:3128',
    'https': 'http://10.10.1.10:1080',
}

requests.get('https://vk.com', proxies=proxies)
```

Практикум

Давайте для примера скачаем HTML-код какой-нибудь статьи из интернета, например, отсюда: https://ru.wikipedia.org/wiki/Python

А затем, сохранить код в `*.html` файле.

Итак, отправляем Get-запрос:

```
import requests

# Устанавливаем переменную с юзерагентом
user_agent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36"
# Создаём словарь заголовков, передаём в гено юзерагент
headers = {"user-agent": user_agent}
# Формируем ссылку
link = "https://ru.wikipedia.org/wiki/Python"
# Отправляем запрос
r = requests.get(link, headers=headers)
```

Получаем HTML-код страницы:

```
html = r.text
```

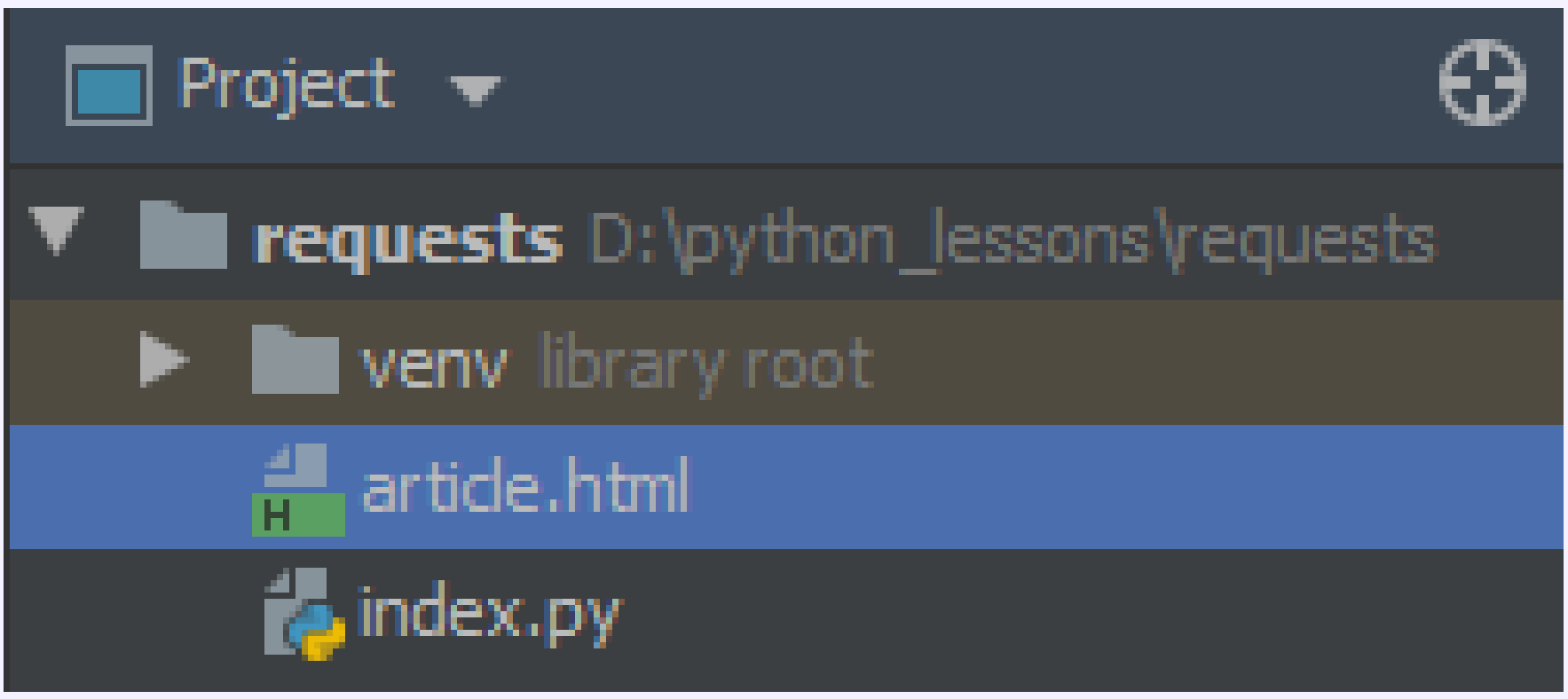
Сохраняем наш HTML в файл:

```
# Создаём новый файл
with open("article.html", "w", encoding="utf-8") as f:
    # Записываем HTML-код в файл
    f.write(html)
```

Всё вместе:

```
import requests

# Устанавливаем переменную с юзерагентом
user_agent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36"
# Создаём словарь заголовков, передаём в гено юзерагент
headers = {"user-agent": user_agent}
# Формируем ссылку
link = "https://ru.wikipedia.org/wiki/Python"
# Отправляем запрос
r = requests.get(link, headers=headers)
# Получаем HTML-код
html = r.text

# Создаём новый файл
with open("article.html", "w", encoding="utf-8") as f:
    # Записываем HTML-код в файл
    f.write(html)
```

Запускаем скрипт, и видим, что в нашей корневой директории был создан новый файл:



Откроем его в браузере:

Как видите, HTML-код был успешно загружен. В дальнейшем мы спокойной сможем с ним работать.

Post-запросы в Requests

Для отправки post-запроса у модуля requests имеется одноимённая функция post:

```
# Отправляем запрос
r = requests.post(link)
```

Установка Post-параметров

Параметры POST-запроса отправляются также, как и у функции get:

```
set_params = {
    "login": "admin",
    "password": "1111",
    "captcha": "uYt5Cz",
}

# Отправляем запрос
r = requests.post(link, params=set_params)
```

Все остальные настройки и функции у метода post, а именно:

Получить код ответа сервера
Получить текст ответа
Получить заголовки
Установить заголовки
Получит куки
Установить куки

Идентичны настройкам и функциям метода get. Для других типов запроса данные методы работают идентичным образом.

Извлекаем данные из HTML. BeautifulSoup

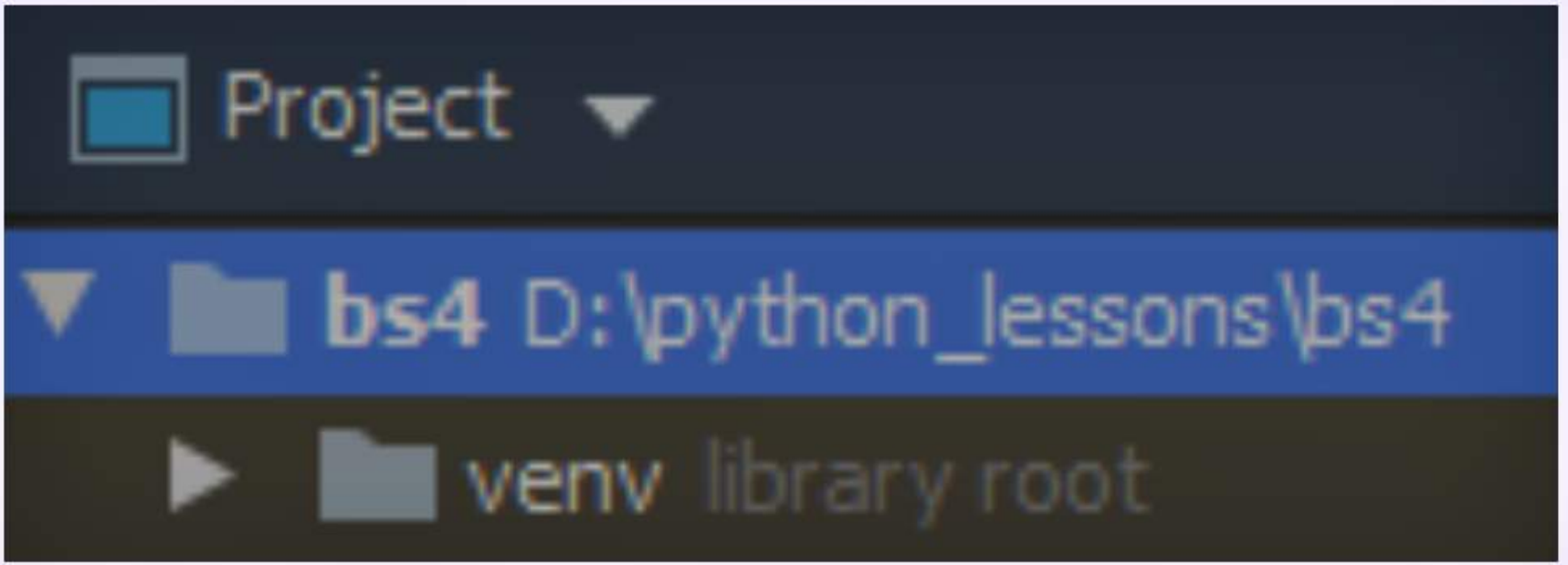
Содержание урока

Отправляя запросы на сайты мы получаем текст, иногда обычный текст, иногда в формате HTML, иногда в формате JSON или XML. Но чаще всего, работать приходится с текстом в формате HTML.

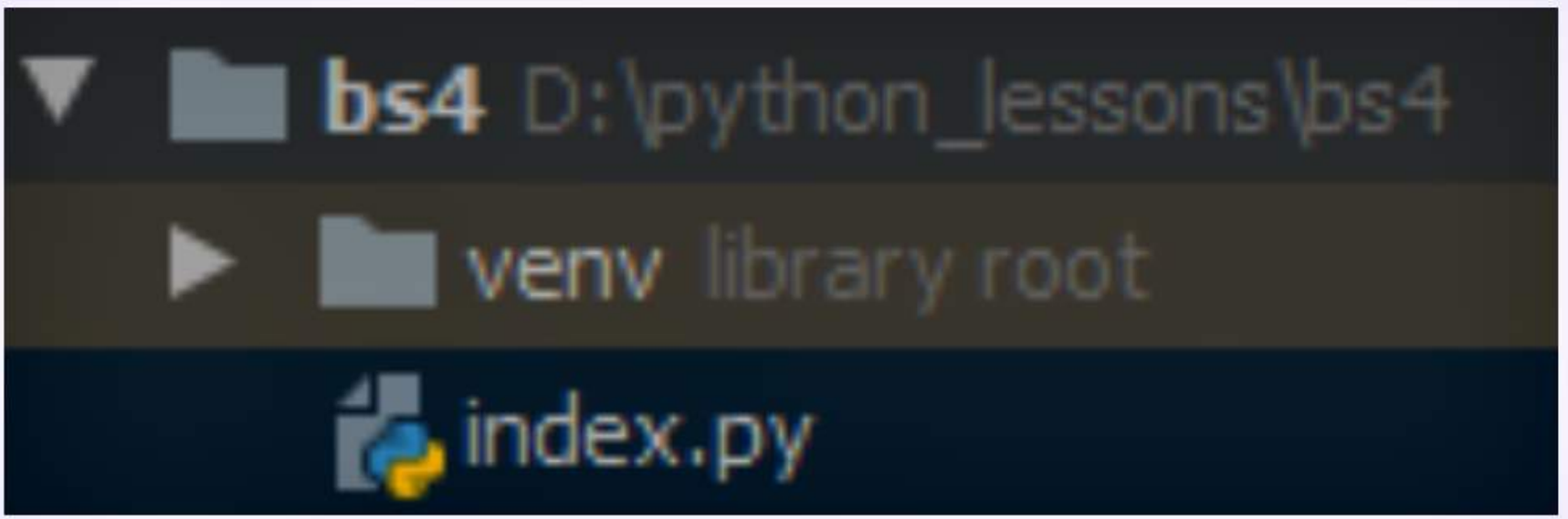
Мы получаем от сайта HTML код той или иной страницы, и нам нужно вытащить из этого кода некоторые данные. Для этого мы используем библиотеки для парсинга HTML-кода. Одна из самых удобных и мощных библиотек – BeautifulSoup.

Документация BeautifulSoup: https://www.crummy.com/software/BeautifulSoup/bs4/doc.ru/

Создадим новый проект, назовём его, например, bs4:



В корневой директории проекта создадим файл, в котором будем писать наш код, например, `index.py`:



Установка BeautifulSoup

Для установки BS вводим следующую команду в консоли:

```
pip install beautifulsoup4
```

Подключаем модуль BS:

```
from bs4 import BeautifulSoup as BS4
```

Для того, чтобы работать с HTML-кодом через BeautifulSoup необходимо загрузить этот исходный HTML-код непосредственно в объект BeautifulSoup. Соответственно, сперва получим HTML-код страницы, например, возьмём код загрузки страницы из прошлого урока.

Разумеется, в новый проект нужно также установить модуль requests для отправки запросов.

Получим исходный код страницы:

```
import requests
from bs4 import BeautifulSoup as BS4

r = requests.get("https://ru.wikipedia.org/wiki/Python")
# Получаем HTML-код
html = r.text
```

Далее, создаём объект BS и передаём в него этот код:

```
parser = BS4(html, "html.parser")
```

После этого, в переменной parser у нас будет объект, при помощи которого мы сможем работать с нашей страницей, извлекать оттуда разного рода данные.

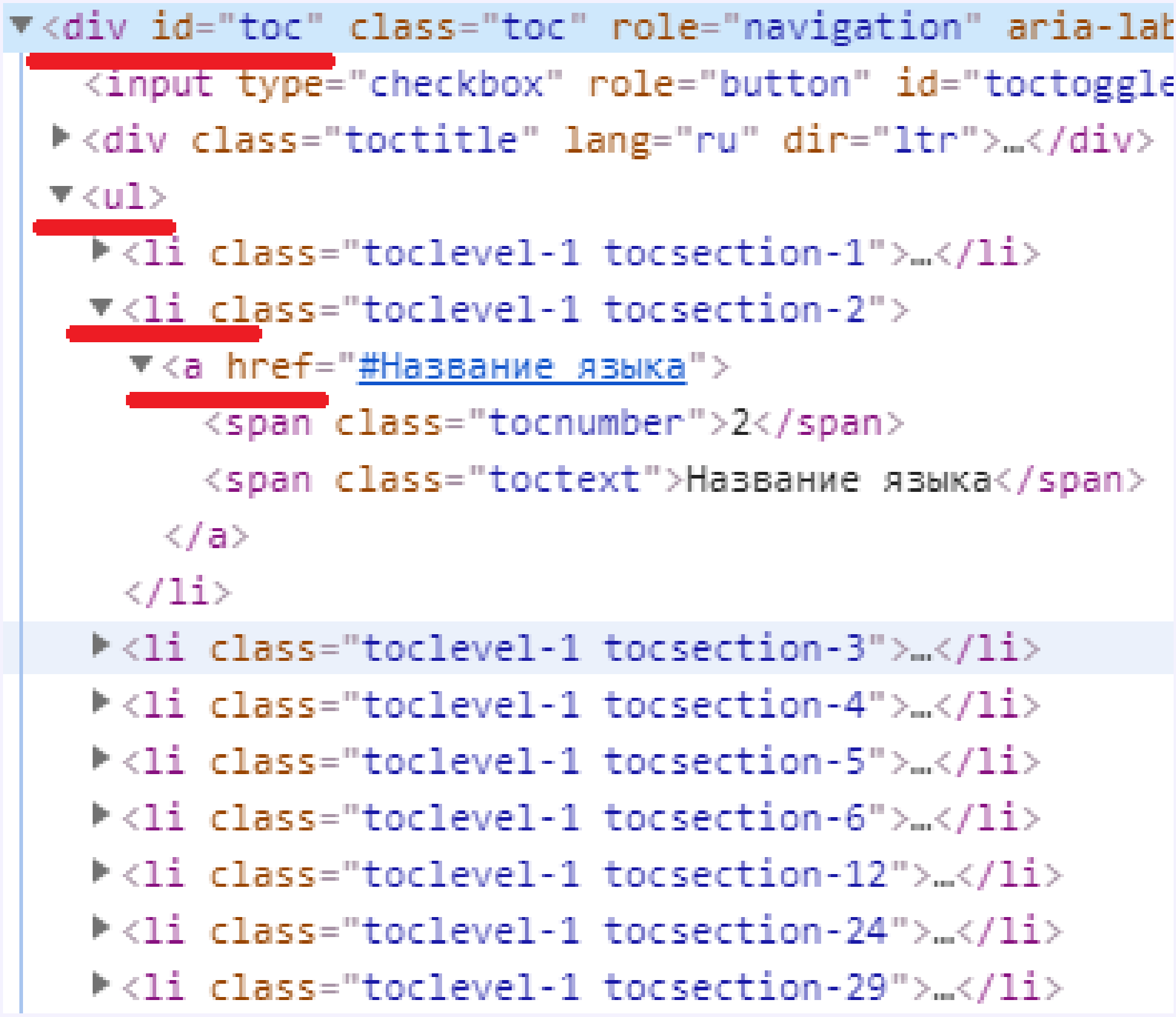
Поиск элементов на странице

У BS есть разные функции для поиска элементов на странице (со всеми можете ознакомится в документации), но я хочу посоветовать вам основной метод, который закроет ваши потребности в поиске элементов на странице на 99%.

Этот метод называется `select` и он ищет элементы на странице по CSS-селектору, ровно таким же образом, как это реализовано в Selenium. На странице со статьёй есть оглавление:

Содержание [скрыть]

Давайте при помощи метода select получим все заголовки и ссылки из оглавления. Давайте сперва проинспектируем элементы и найдём у них уникальные атрибуты:



Нам нужно получить доступ к ссылкам с заголовками. Наш список находится внутри div с уникальным атрибутом – `id="toc"`, а значит мы можем найти уникальный путь до всех ссылок:

```
elements = parser.select("div#toc ul li a")
```

Метод `select` возвращает список элементов, даже если элемент на странице один. Пройдёмся циклом по всем элементам и выведем название и ссылку на заголовок:

```
# Проходим по каждому элементу и забираем из него данные
for element in elements:
    # Извлекаем текст из элемента
    title = element.getText()
    # Извлекаем текст из атрибута href элемента
    link = element['href']
    # Выводим на экран
    print(f"Заголовок: {title}, ссылка: {link}")
```

Весь код целиком:

```python
import requests
from bs4 import BeautifulSoup as BS4

# Отправляем GET-запрос на сайт
r = requests.get("https://ru.wikipedia.org/wiki/Python")
# Получаем HTML-код
html = r.text

# Загружаем в BS наш HTML-код
parser = BS4(html, "html.parser")
# Находим все элементы с заголовками
elements = parser.select("div#toc ul li a")


# Проходим по каждому элементу и забираем из него данные
for element in elements:
    # Извлекаем текст из элемента
    title = element.getText()
    # Извлекаем текст из атрибута href элемента
    link = element['href']
    # Выводим на экран
    print(f"Заголовок: {title}, ссылка: {link}")
```

Выводит:

```
index
D:\python_lessons\bs4\venv\Scripts\python.exe D:/python_lessons/bs4/index.py
Заголовок: 1 Философия, ссылка: #Философия
Заголовок: 2 Название языка, ссылка: #Название_языка
Заголовок: 3 История, ссылка: #История
Заголовок: 4 Портируемость, ссылка: #Портируемость
Заголовок: 5 Типы и структуры данных, ссылка: #Типы_и_структуры_данных
Заголовок: 6 Синтаксис и семантика, ссылка: #Синтаксис_и_семантика
Заголовок: 6.1 Операторы, ссылка: #Операторы
Заголовок: 6.2 Выражения, ссылка: #Выражения
```

Как и планировалось, заголовки и ссылки из HTML-кода:

```
<a href="#Название_языка">
    <span class="tocnumber">2</span>
    <span class="toctext">Название языка</span> == $0
</a>
```

Итак, мы разобрались как находить элементы на странице. Но как извлекать из них данные?

Данные в HTML-коде хранятся в виде:

Текста внутри тега, например, Текст
Текста внутри атрибута тега, например, .

Извлечь текст из тега

Чтобы извлечь текст из тега в BS, у самого тега нужно вызвать функцию `getText()`, в нашем коде:

Например, у нас в коде есть тег: Text

Чтобы достать из этого тега текст, нужно написать следующий код:

```
element = parser.select("p#my_text")
element[0].getText()  # Text
```

Извлечь текст из атрибута

А чтобы вытащить текст из атрибута, у этого же тега, нужно в квадратных скобках прописать имя атрибута, например:

```
element = parser.select("p#my_text")
element[0]["id"]  # my_text
```

Сейчас может быть непонятно, но в практических уроках мы разберемся с этим подробнее.

Практика. Пишем парсер картинок

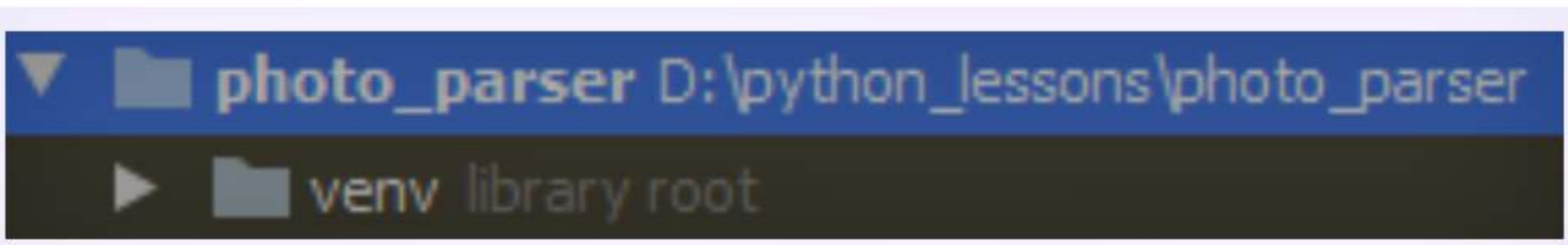
Содержание урока

Давайте закрепим полученные знания на практике и напишем парсер картинок с сайта: `istockphoto.com`. Это стоковый вебсайт, который отдаёт миниатюры фотографий без ватермарок. Фотографии как правило имеют разрешение 400 на 300 пикселей и выше. Соответственно, их можно использовать, например, для создания баннеров и для прочих маркетинговых нужд.

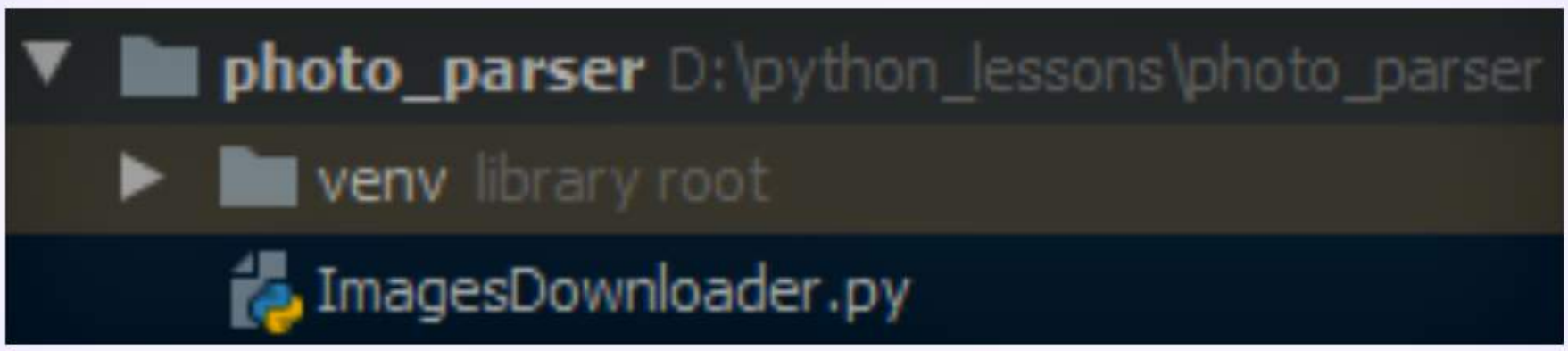
Функционал программы

В нашу программу мы будем передавать ключевой запрос и количество страниц выдачи, которые необходимо сохранить. Например, мы вводим запрос BMW, и количество страниц равное 5. Наша программа должна скачать фотографии с первых пяти страниц, по запросу BMW.

Создадим проект и назовём его `photo_parser`:



Создадим в проекте новый модуль, назовём его `ImagesDownloader.py`:



Установим в проект библиотеки Requests и Beautiful Soup, как мы делали это в прошлых уроках.

Импортируем в наш модуль библиотеки:

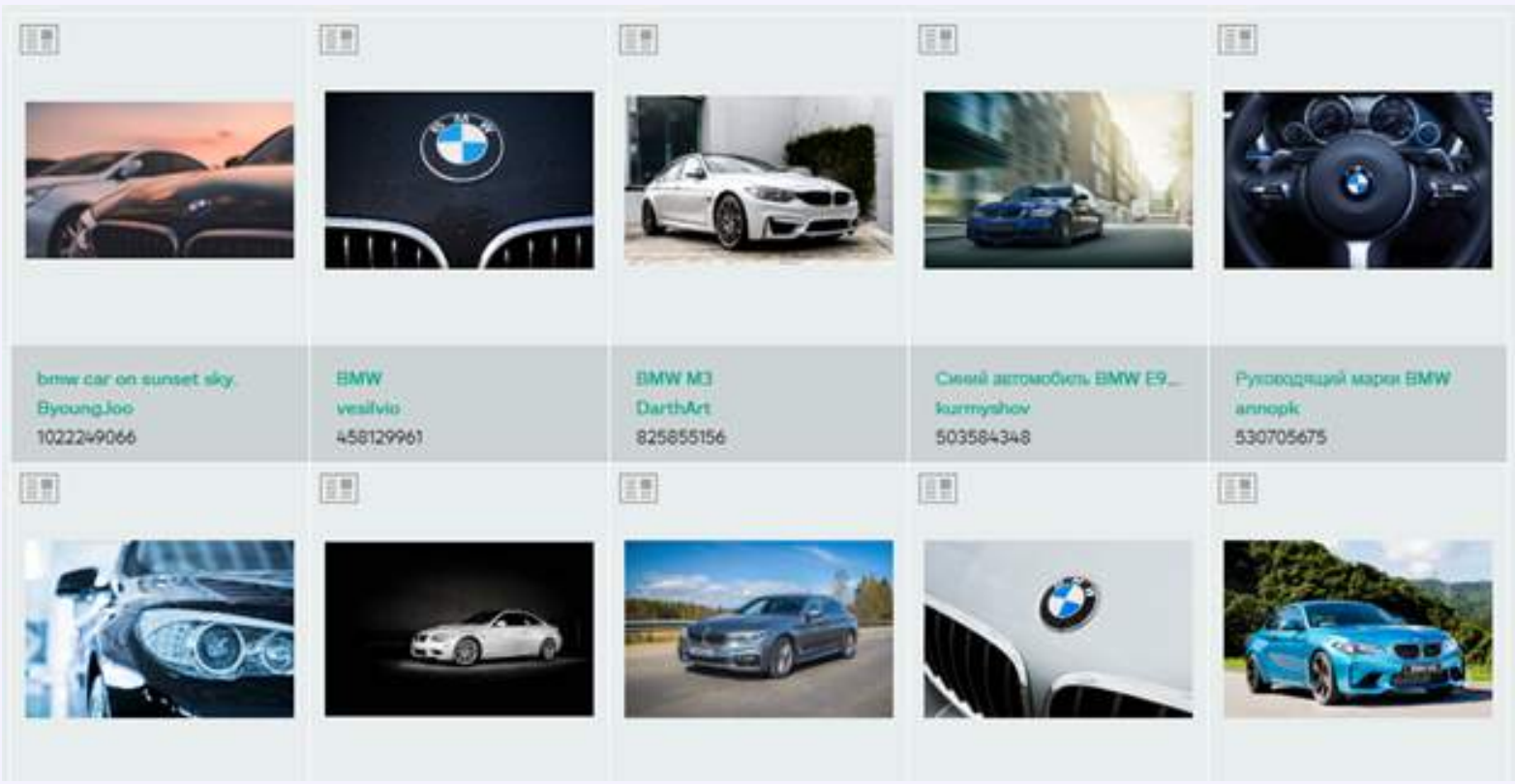
```
import shutil
import requests
import urllib.parse
from bs4 import BeautifulSoup as BS4
```

Библиотека `shutil` будет нужна для сохранения картинок, а `urllib` для того, чтобы обычные запросы, введённые пользователем специальным образом кодировались для передачи через `url`. Эти библиотеки уже включены в вашем виртуальном окружении, их устанавливать не нужно.

Пишем парсер

Сперва перейдём на сайт, введём запрос и посмотрим, как он работает. Введём некоторый запрос и нажмём на кнопку поиска.

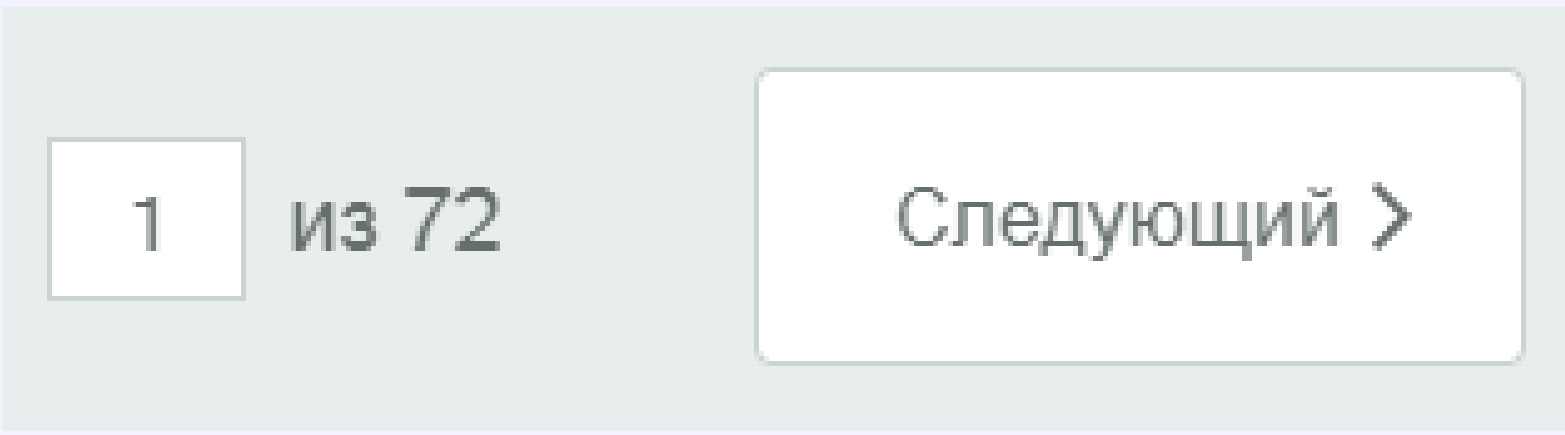
Откроется страница с изображениями:



Но мы обратим внимание на адрес страницы, данные с которой необходимо получить: https://www.istockphoto.com/ru/фотографии/bmw?phrase=bmw&sort=mostpopular

Как видите, основной динамический параметр здесь – это запрос: `bmw`, который тут дублируется, как часть url: `фотографии/bmw` и как часть get-параметра: `phrase=bmw`

Но как получить фотографии с других страниц? Давайте прокрутим страницу до конца, и перейдём на следующую страницу:



Открывается новая страница, смотрим на её адрес: https://www.istockphoto.com/ru/фотографии/bmw?page=2&phrase=bmw&sort=mostpopular

Видим, что появился новый параметр: `page=2`

Соответственно, можно сделать вывод, что чтобы получить заданное количество фотографий по ссылке, достаточно запустить цикл на количество итераций соответствующее количеству требуемых страниц, отправляя запрос для каждой страницы через `requests` получаем html-код страницы, затем извлекаем список ссылок на изображения через `Beautiful Soup` и сохраняем эти страницы.

Метод get_page. Получаем код страницы с изображениями

Данный метод будет получать html-код заданной страницы в выдаче по заданному запросу:

```
def get_page(phrase, page):
    # Определяем ссылку на страницу, к которой будем отправлять запрос
    link = "https://www.istockphoto.com/ru/%D1%84%D0%BE%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%B8/" + phrase
    # Определяем словарь с заголовками

    headers = {'user-agent': 'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.122 Safari/537.36'}
```

```
    # Определяем словарь с параметрами запроса
    params = {"page": page, "phrase": phrase, "sort": "mostpopular"}
    # Отправляем запрос на страницу с картинками
    request = requests.get(link, headers=headers, params=params)
    # Возвращаем HTML-код из функции
    return request.text
```

Тут всё стандартно, как и было в уроке по `requests`. Передаём параметры запроса, а именно номер страницы который нам нужен, фраза которую мы ищем и тип сортировки.

Далее, получаем ответ от сервера и возвращаем этот ответ из функции.

Метод is_404

Может сложится такая ситуация, что пользователь задал слишком большое количество страниц и на определённой странице уже нет фотографий. Поэтому, нам необходимо написать функцию проверки наличия фотографий на странице:

```
# Проверяет, есть ли на странице изображения
def is_404(html):
    page = BS4(html, "html.parser")
    # Если на странице ни одного изображения - возвращаем True
    if len(page.select("img.gallery-asset__thumb")) < 0:
        return True
    return False
```

При просмотре кода, мы можем увидеть, что ссылки на изображения содержатся в таких элементах:

```
<img alt="стильные нос в красный спортивный автомобиль-седан bmw - bmw стоковые фото и изображения" class="gallery-asset__thumb gallery-fixed-asset__thumb gallery-fixed-asset__thumb--null" ng-class="thumbClassNames" src="https://media.istockphoto.com/photos/stylish-nose-of-red-bmw-sports-se…6&m=157772682&s=612x612&w=0&h=8uhaU53p1q7EqgT8Qc3SAU3L1C2XRAv667HQFHyE42g=" style="opacity: 1;"> == $0
```

Если на странице нет ни одного элемента, `img` с классом `gallery-asset__thumb`, значит соответственно на этой странице нет фотографий. Этот метод мы будем использовать дальше.

Метод get_imgs_from_page

Данный метод будет извлекать все ссылки с отдельной страницы:

```
# Получает список ссылок с конкретной страницы из выдачи по запросу
def get_imgs_from_page(phrase, page):
    # Получаем HTML-код конкретной страницы выдачи
    html = get_page(phrase, page)

    # Определяем локальный список, в который будут помещаться ссылки на изображение
    images = []

    # Если вернуло страницу 404, значит прерываем выполнение функции, возвращая False
    if is_404(html) == True:
        return False

    # Загружаем код страницы полученной с сервера в парсер
    img_node = BS4(html, "html.parser")
    # Извлекаем все элементы со ссылками на картинки
    imgs = img_node.select("img.gallery-asset__thumb")
    # Извлекаем из элементов ссылки на картинки и добавляем их в список
    for img in imgs:
        if img.has_attr("src"):
            print(f"Получили ссылку на фото: {img['src']}")
            images.append(img["src"])

    # Возвращаем список из функции
    return images
```

1. При помощи функции get_page мы получаем html-код с картинками с конкретной страницы.
2. Проверяем, есть ли на этой странице вообще изображения. Если изображений нет – заканчиваем выполнение функции, возвращаем значение False.
3. Загружаем html-код страницы в парсер.
4. Извлекаем все элементы содержащие изображения.
5. Извлекаем из этих элементов ссылки на изображения и размещаем их в список.
6. Возвращаем список с изображениями из функции.

Наш софт теперь умеет извлекать ссылки со страницы выдачи, теперь нужен метод, который извлекает со всех заданных страниц выдачи.

Метод get_images

Проходит в цикле необходимое количество итераций и на каждую итерацию скачивает последующую страницу из выдачи по запросу:

```
# Получает список ссылок на изображения с заданного количества страниц по запросу
def get_images(query, pages):
    # Переводим ключевой запрос в приемлемый формат
    query = urllib.parse.quote(query)
    # Создаём список, в который будут помещаться все ссылки
    images = []
    # Запускаем цикл, для парсинга по страницам
    for i in range(pages):
        # Так как цикл начинается с 0, определяем номер страницы
        num_of_page = i + 1
        # Получаем все изображения со страницы
        img = get_imgs_from_page(query, num_of_page)

        # Если в переменной False, значит больше страниц с
        # фотографиями нет прерываем цикл
        if not img:
            break
        else:
            images += img

    return images
```

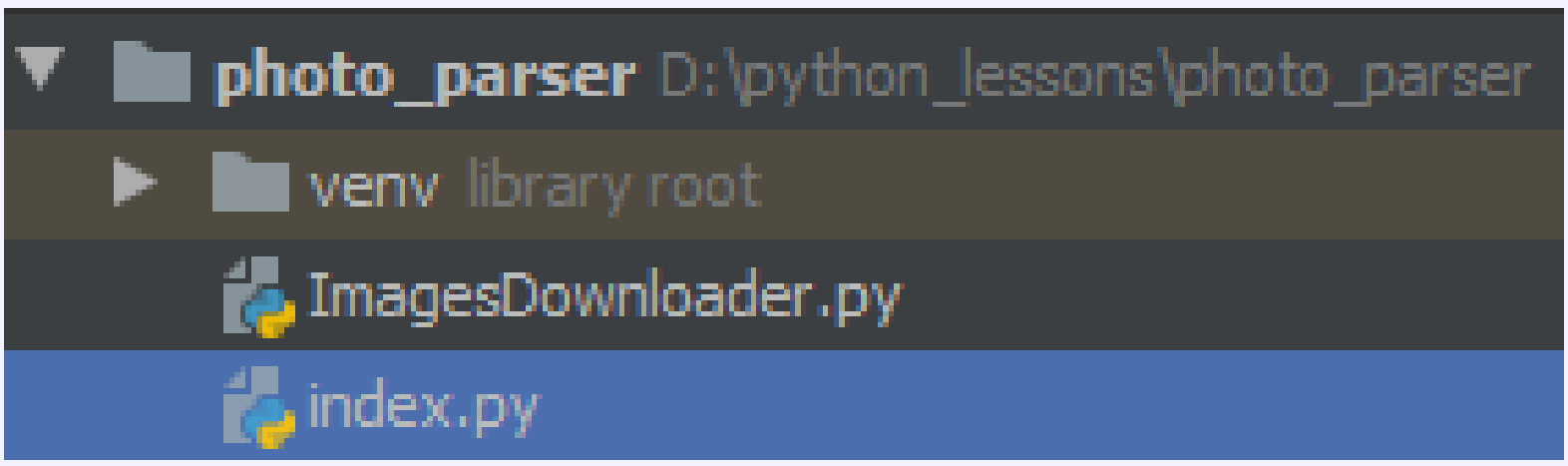
Сперва подготавливаем запрос. Когда мы формируем ссылку – текст, передаваемый в неё необходимо обработать, чтобы избежать шибок.

Запускаем цикл на заданное количество страниц, если мы задали спарсить 5 первых страниц из выдачи, значит задаём циклу отработать 5 итераций, и на каждой итерации скачиваем страницу, от страницы номер 1 до заданной. Так как цикл начинается с нуля, а нумерация страниц с единицы, на каждой итерации прибавляем к номеру текущей итерации одну единицу.

Проверяем, если на текущей итерации запрос вернул False, значит больше нет страниц с изображениями и необходимо прервать цикл.

Возвращаем все изображения, которые мы спарсили из выдачи.

Создадим ещё один модуль, назовём его index.py и вызовем в нём эту функцию, посмотрим, как она работает:



Импортируем наш модуль и вызовем метод `get_images`:

```
import ImagesDownloader
result = ImagesDownloader.get_images("money", 2)
print(f"Получили ссылки на: {len(result)} фотографий")
```

Выводит:

> … Получили ссылку на фото: https://media.istockphoto.com/vectors/vintage-money-font-with-shadow-vector-id500173082?k=6&m=500173082&s=612x612&w=0&h=bhIZiKhrXnu2snShd7cInAXYG7sfoxlZte6mSglKd2Q= Получили ссылку на фото: https://media.istockphoto.com/vectors/money-stack-of-us-dollars-vector-id909164260?k=6&m=909164260&s=612x612&w=0&h=yGgLOmvzzxf1rec71a2gg4breZNtN7ssnGUDdFOUuP8= Получили ссылку на фото: https://media.istockphoto.com/photos/business-audits-using-a-calculator-financial-data-investment-fund-at-picture-id1129810557?k=6&m=1129810557&s=612x612&w=0&h=uEl1W2RLjhjNrlRYNMBSGd5vxOVCdmP_VJVGPPiyQPA= Получили ссылки на: 120 фотографий

Вывел список всех фотографий в консоль, а также посчитал, сколько всего ссылок было получено.

Отлично, мы научили наш код получать ссылки на изображения. Теперь осталось, написать модуль для сохранения картинок.

Метод save_image

Сохраняет изображение в заданную папку. Мы указываем папку, в которую нужно сохранить изображение и ссылку на него:

```python
# Сохраняет отдельное изображение
def save_image(folder, link):
    # Получаем исходный код изображения
    image = requests.get(link, stream=True)

    # Извлекаем из ссылки, например:
    # https://media.istockphoto.com/vectors/finance-money-icons-vector-id1030878592?k=6&m=1030878592&s=612x612&w=0&h=HXSWx0FW-Re64cLJWL-LrbMx8Zx4rfDAC
    # Название изображения, например: finance-money-icons-vector-id1030878592
    filename = link.split("/")[4]
    filename = filename.split("?")[0]
    # Формируем путь до файла вида: папка/имя_файла.jpg
    path_to_file = f"{folder}/{filename}.jpg"
    print(f"Сохранили фото: {path_to_file}")
    # Сохраняем фото в файл
    with open(path_to_file, "wb") as file_obj:
        shutil.copyfileobj(image.raw, file_obj)
```

Передаём ссылку на изображение, скачиваем его с сервера в переменную. Извлекаем название изображения, далее сохраняем содержимое изображения в новый файл.

Теперь давайте напишем финальный метод, который будет скачивать все изображения по конкретному запросу в заданную папку.

Метод download_images

Получает список изображений по запросу, далее в цикле скачивает каждое изображение:

```python
# Скачивает все изображения по запросу в заданную папку
def download_images(folder, phrase, page):
    # Получаем все ссылки из выдачи по запросу
    images = get_images(phrase, page)
    # Сохраняем все ссылки
    for image in images:
        # Запускаем функцию сохранения в
        try:
            save_image(folder, image)
        except Exception as error:
            print(f"Проблема с записью файла: {error}")
```

Передаём папку в которую необходимо скачать изображение, далее запрос по которому эти изображения нужно спарсить и количество страниц по запросу, которые нужно спарсить.

Получаем все ссылки на изображения при помощи метода `get_images`. Далее проходим все ссылки в цикле и сохраняем каждую при помощи метода `save_image`.

Теперь, давайте запустим данный метод в нашем модуле `index.py`:

```python
import ImagesDownloader
ImagesDownloader.download_images("C:/Users/Moneymaker/Desktop/img", "money", 2)
```

После запуска в лог будет выведена информация о ходе выполнения скрипта. В папке, которую мы указали, начнут появляться изображения. Вот, собственно и все, наш парсер успешно работает. И теперь при помощи двух строчек кода можно скачивать большое количество стоковых изображений для ваших маркетинговых нужд.

Извлекаем и упаковываем Json. Стандартные средства Python

Содержание урока

Понятие JSON-объекта

Порой, данные с сайтов приходят не в виде привычного нам HTML-кода, а в виде JSON-объекта. Что на самом деле гораздо лучше и удобнее, так как работа с JSON куда удобнее, проще и быстрее.

Давайте сперва поговорим о том, что из себя представляет формат JSON. JSON – это легко читаемый формат обмена данными основанный на JavaScript.

JSON-объект, очень похож на обычный словарь:

```
{"name": "Oleg", "age": 19, "country": "Russia", "city": "Moscow"}
```

В то время как словарь в Python выглядит следующим образом:

```
{"name": "Oleg", "age": 19, "country": "Russia", "city": "Moscow"}
```

Хотя они и выглядят одинаково, но JSON-объект – это просто строка, сформированная по определённому принципу, а словарь в Python это сущность языка, объект.

В свою очередь, JSON объект, как и Python-массив может иметь вложенные в него подобъекты и массивы:

```
{"name":"Oleg","age":19,"country":"Russia","city":{"1":1,"2":2,"3":3,"4":4}}
```

Декодирование JSON

Работа с JSON-форматом крайне удобно реализована в Python «из коробки». Если вы получаете при запросе к сайту обычный JSON-объект, и вам нужно извлечь из него данные, вы можете при помощи функции loads встроенного модуля JSON преобразовать JSON-объект в некоторую структуру Python, в зависимости от формы JSON-объекта, данные из которого вы легко сможете извлечь.

```
import json
# Строка с JSON-объектом
json_string = '{"name":"Oleg","age":19,"country":"Russia","city":{"1":1,"2":2,"3":3,"4":4}}'
# Декодируем строку для работы с Python
dec_json = json.loads(json_string)
# Выводим тип объекта, который получился при десериализации
print(type(dec_json))
# Выводим сам объект и его структуру в виде строки
print(dec_json)
```

Выводит:

> <class 'dict'> {'name': 'Oleg', 'age': 19, 'country': 'Russia', 'city': {'1': 1, '2': 2, '3': 3, '4': 4}}

Как видите, строка была декодирована в список.

Давайте разберём пример. Допустим, сайт возвращает нам следующий JSON-объект:

```
{"_id":{"$oid":"5a3bbe1c3d8d2d1efa0c6ca0"},"id":334906,"name_site":"xvideos","url":"https://www.xvideos.com/video334906/roxxie_xxx_amari","id_video":334906,"name":"Roxxie XXX Amari","tags":[],"actors":[],"poster":"http://img-l3.xvideos.com/videos/thumbs169ll/22/2e/f9/222ef9def786f2e48ce68e83484dfed4/222ef9def786f2e48ce68e83484dfed4.11.jpg"}
```

И нам необходимо достать из данного объекта ссылку по ключу URL.

```
import json
# Строка с JSON-объектом
json_string ='{"_id":{"$oid":"5a3bbe1c3d8d2d1efa0c6ca0"},"id":334906,"name_site":"xvideos","url":"https://www.xvideos.com/video334906/roxxie_xxx_amari","id_video":334906,"name":"Roxxie XXX Amari","tags":[],"actors":[],"poster":"http://img-l3.xvideos.com/videos/thumbs169ll/22/2e/f9/222ef9def786f2e48ce68e83484dfed4/222ef9def786f2e48ce68e83484dfed4.11.jpg"}'
# Декодируем строку для работы с Python
dec_json = json.loads(json_string)
```

```
# Выводим тип объекта, который получился при десериализации
print(type(dec_json))
# Выводим сам объект и его структуру в виде строки
print(dec_json)
```

В итоге, массив dec_json будет содержать следующие данные:

```
{
      '_id': {'$oid': '5a3bbe1c3d8d2d1efa0c6ca0'},
      'id': 334906,
      'name_site': 'xvideos',
      'url': 'https://www.xvideos.com/video334906/roxxie_xxx_amari',
      'id_video': 334906,
      'name': 'Roxxie XXX Amari',
      'tags': [],
      'actors': [],
      'poster': 'http://img-l3.xvideos.com/videos/thumbs169ll/22/2e/f9/222ef9def786f2e48ce68e83484dfed4/222ef9def786f2e48ce68e83484dfed4.11.jpg'

}
```

Теперь просто обращаемся к ключу URL данного массива:

```
print(dec_json["url"])
```

Выводим на экран значение:

> https://www.xvideos.com/video334906/roxxie_xxx_amari

Кодирование JSON

Иногда напротив, нам необходимо передать некоторый массив данных на посторонний сервер, который принимает только JSON.

В таком случае, необходимо создать обычный массив данных, а затем упаковать его в JSON-объект. Делает это функция `json.dumps(data)`.

Создадим словарь с параметрами, которые нужно передать на сервер:

```
# Словарь с данными
data_dict = {"login": "user", "password": "ef4r446554", "api_key": "dcftrybyjty65765bf"}
```

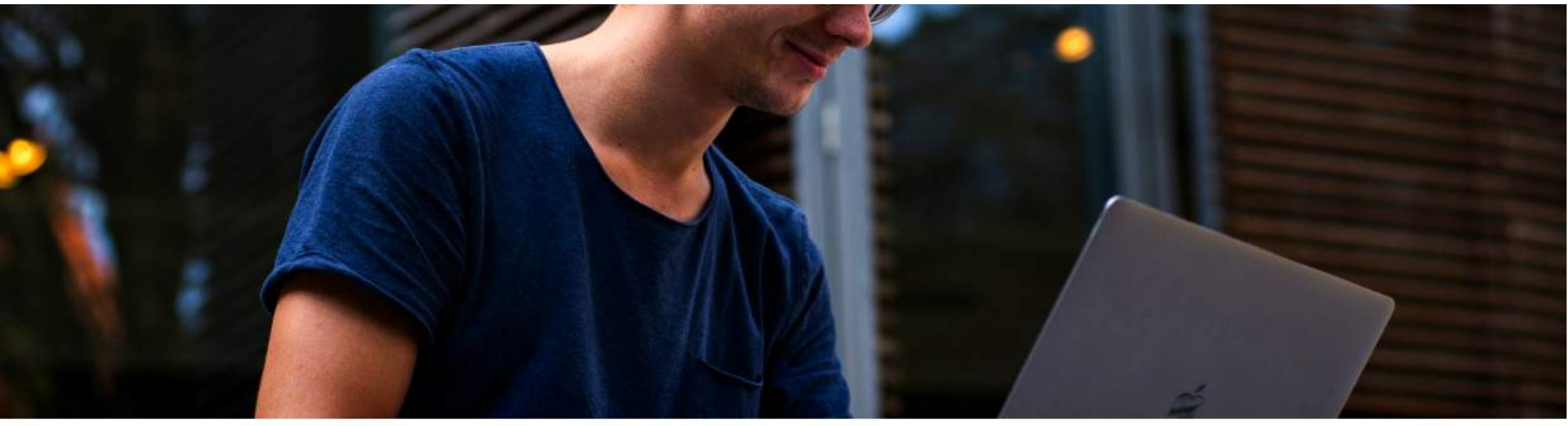
Допустим, эти данные нужны для авторизации в некотором сервисе. Теперь, при помощи метода json.dumps закодируем словарь в JSON и выведем на экран.

```
import json
# Словарь с данными
data_dict = {"login": "user", "password": "ef4r446554", "api_key": "dcftrybyjty65765bf"}
# Кодируем словарь в JSON-строку
json_string = json.dumps(data_dict)
# Выводим тип объекта, который получился при сериализации
print(type(json_string))
# Выводим сам объект и его структуру в виде строки
print(json_string)
```

Вот что мы увидим:

```
{"login": "user", "password": "ef4r446554", "api_key": "dcftrybyjty65765bf"}
```

Валидная JSON-строка, которую можно отправлять на сервер.



Парсим сайты на AJAX

Содержание урока

Понятие принципа работы AJAX

Процесс парсинга, как нам известно происходит в 2 этапа:

1. Отправка запросов и получение данных.
2. Приведение полученных данных к нужному формату.

Так вот, отправка запросов на страницы как мы привыкли (например, как когда мы отправляли запросы на страницу яндекса) не всегда возвращает закономерный результат.

Дело в том, что иногда данные на странице появляются не при нашем первом запросе, а при загрузке страницы. То есть, после того, как страница загрузилась, браузер находит JavaScript-код, который в свою очередь отправляет запрос на сервер, получает данные и выполняет этот код.

То есть, некоторые сайты отправляют запросы на получение информации только после того, как страница полностью будет загружена в браузере. А не тогда, когда мы переходим по ссылке или отправляем форму.

Это и есть технология AJAX, то есть, технология которая позволяет без перезагрузки страницы отправлять запросы на сервер, получать и обрабатывать ответы.

Но так как на работая с запросами нельзя открывать сайты в браузере, а можно лишь получить их исходный код, при парсинге таких сайтов данные в наш скрипт не приходят, но это легко исправить. не зависимости от того, классический сайт или сайт построенный на технологии AJAX он всё равно работает на обычных запросах, которые мы с вами можем легко имитировать.

Типичный сайт на AJAX при первичном запросе возвращает пользователю базовый шаблон страницы и некоторые данные, затем, при загрузке страницы запускается JavaScript-код, который берёт текущие данные со страницы и отправляет запрос на сервер. Затем получает ответ с данными которые нам нужны и размещает их на странице.

Нам просто достаточно просканировать все запросы, которые были отправлены с сайта и ответы на эти запросы, а затем при парсинге воспроизвести запросы с такой же последовательностью.

Анализируем запросы

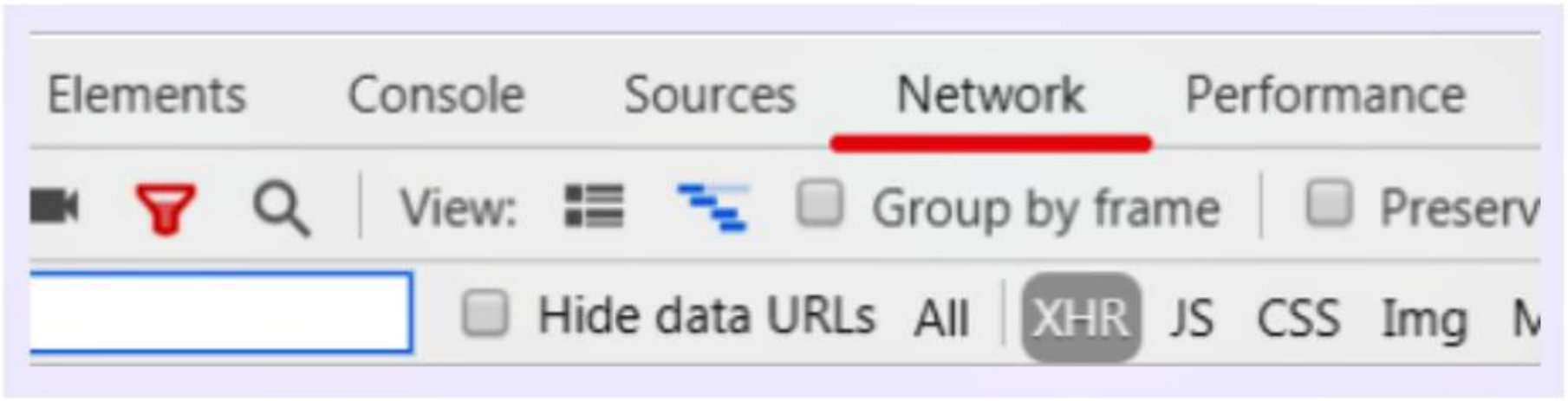
Теперь давайте попрактикуемся в написании парсера для сайта на AJAX, например, давайте напишем парсер сообществ с сайта https://allsocial.ru/

Очень удобный сайт, когда нужно собрать базы под размещение рекламы или под парсинг.

Наш парсер будет собирать заданное количество сообществ по конкретной категории или запросу.

Итак, начнём сканировать запросы. Переходим на страницу с категорией, например: https://allsocial.ru/communities/gadgets

Далее, ПКМ кликаем в любом месте веб-страницы, в выпавшем меню выбираем пункт просмотреть код. В панели инструментов выбираем вкладку Network:



Ниже, выбираем тип ресурса XHR, дабы все остальные ресурсы были не активны.

После этого, перезагружаем страницу и смотрим на какие адреса были отправлены запросы:

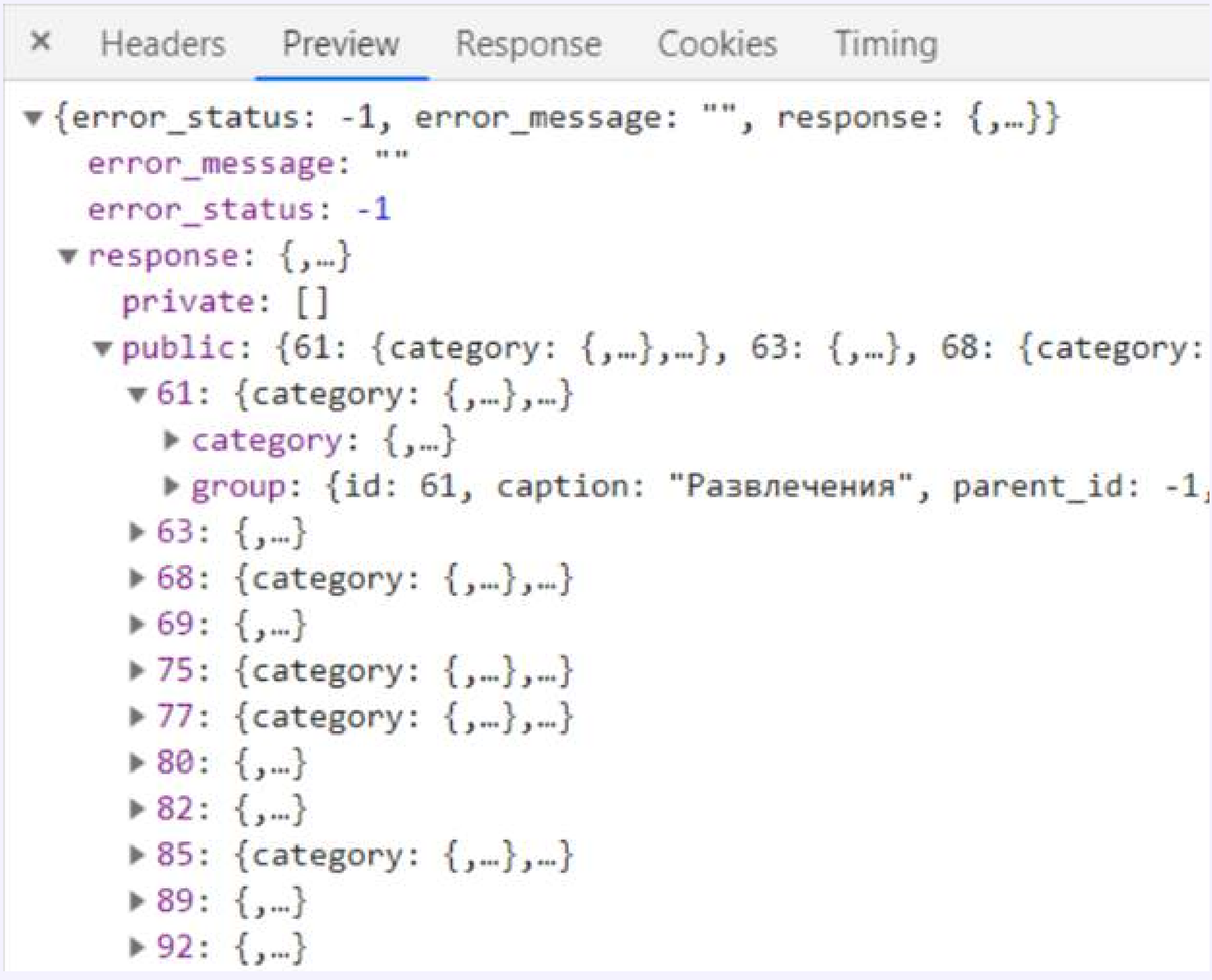
Кликаем на каждый элемент из списка и смотрим, какие данные пришли в ответе (вкладка preview):

По первому запросу, как вы видите только список категорий, нам это пока не нужно, идём дальше.

Доходим до этого запроса:

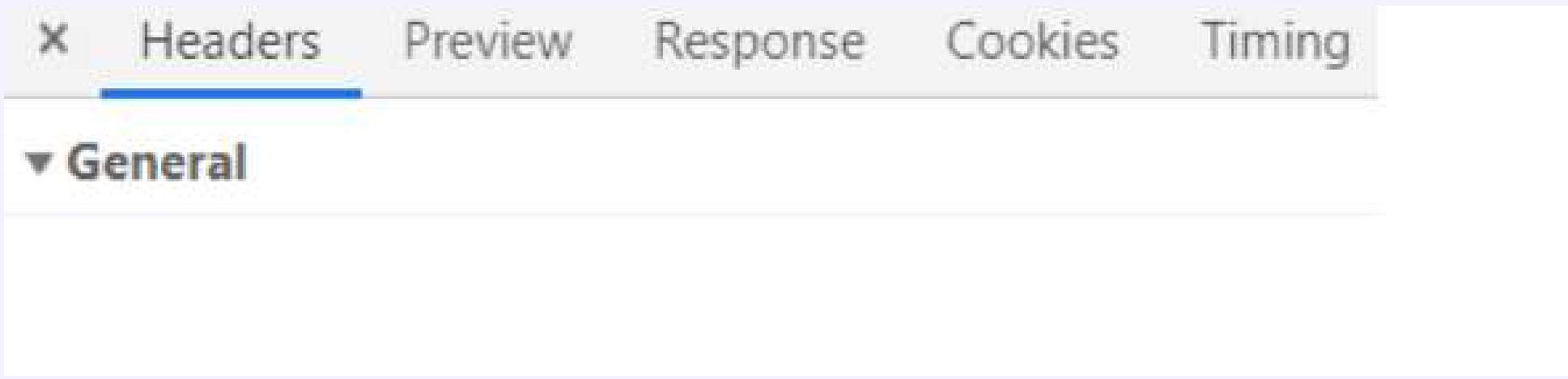
```
https://allsocial.ru/entity?
category_id=23&direction=1&is_closed=-1&is_verified=-1&list_type=3&order_by=diff_abs&period=day&platform=1&range=0:1200
0000&type_id=-1
```

И видим данные, которые он вернул:

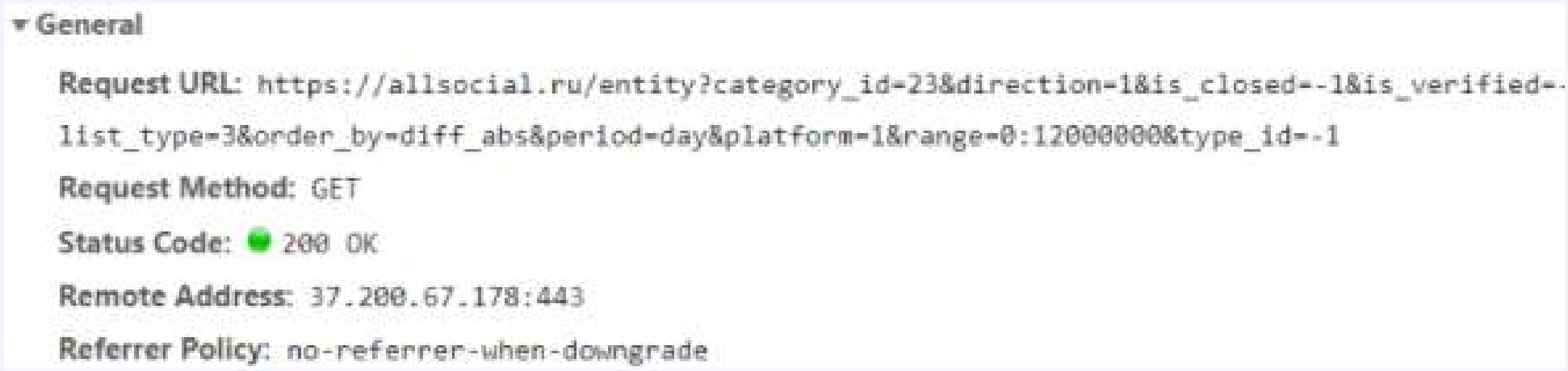


Да, это то, что нам нужно. Теперь мы знаем, какой первичный запрос отправляется на сервер, для получения данных. Как видите, данные в формате JSON.

Давайте научимся анализировать сам запрос, какими данными он оперирует и куда уходит. Для этого перейдём во вкладку Headers:



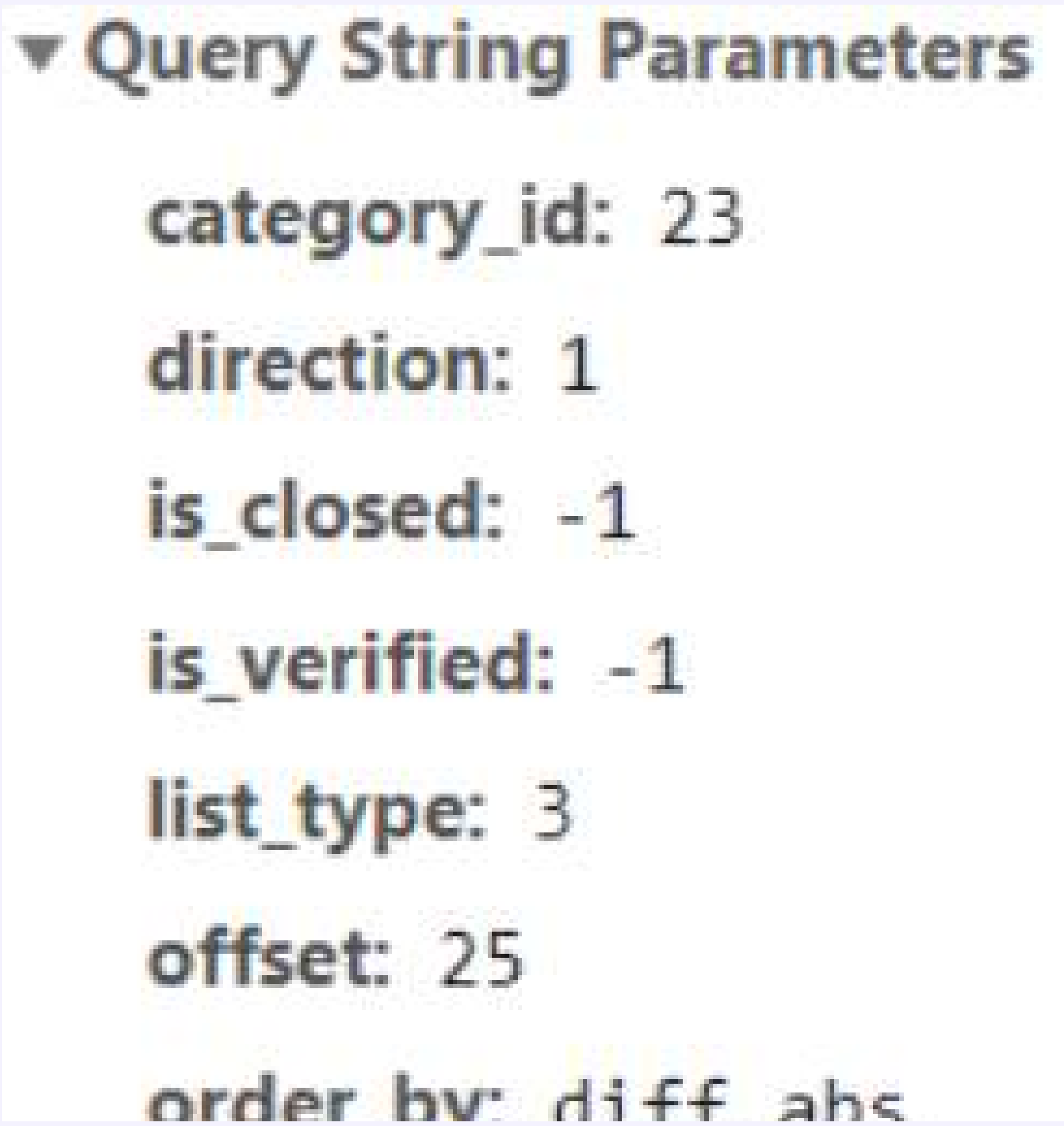
Начинаем с анализа самых основных данных General:



Здесь мы видим, во-первых, сам запрос, точнее адрес, на который этот запрос уходит, его метод (GET) и в нашем случае, так как это GET параметры в URL.

Как мы видим, это типичный GET-запрос, который возвращает данные. Однако, этот запрос возвращает только первые 25 записей. Но ведь их больше чем 25. Давайте посмотрим на параметры запроса, быть может в них есть параметр, который позволяет получить больше запросов.

Для этого листаем вкладку Headers вниз, до строки:



Здесь в удобочитаемом виде представлены все параметры запроса. Проанализируем их, учитывая, что в результате отправки запроса к нам пришло 25 элементов. И здесь мы видим параметр offset со значением 25, очевидно, что именно он отвечает за возвращаемые элементы.

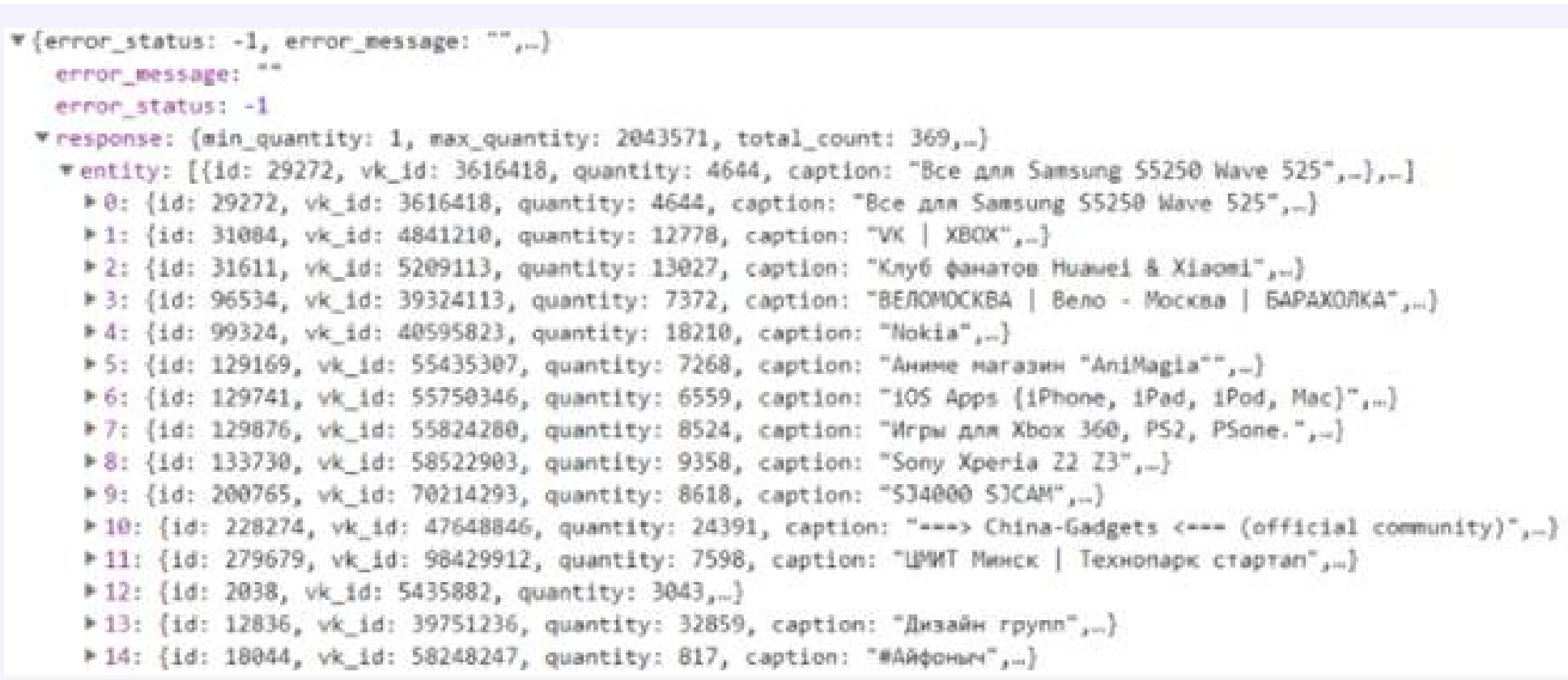
Теперь нужно протестировать нашу гипотезу, логически поразмыслив, можно предположить что чтобы вывести вторые 25 элементов, нужно просто прибавить к текущему параметру offset ещё 25.

Попробуем. Для этого просто копируйте сам запрос из строки:

```
Request URL: https://allsocial.ru/entity?category_id=23&direction=1&is_closed=-1&is_verified=-
list_type=3&offset=25&order_by=diff_abs&period=day&platform=1&range=0:12000000&type_id=-1
```

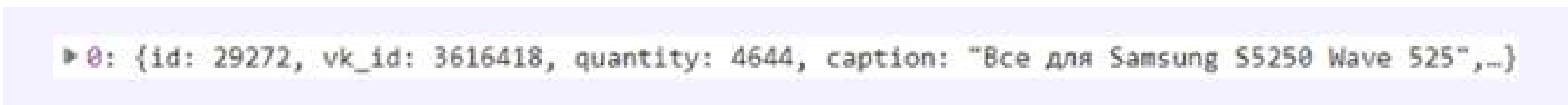
И подменяем цифру 25 на цифру 50. Затем вставляем в строку браузера и нажимаем `enter`.

После чего, снова открываем вкладку Network внутри которой, вкладку «Preview» и смотрим на результат:



А точнее, давайте сравним последнюю строку результата первого запроса:

И первую строку результата второго запроса:



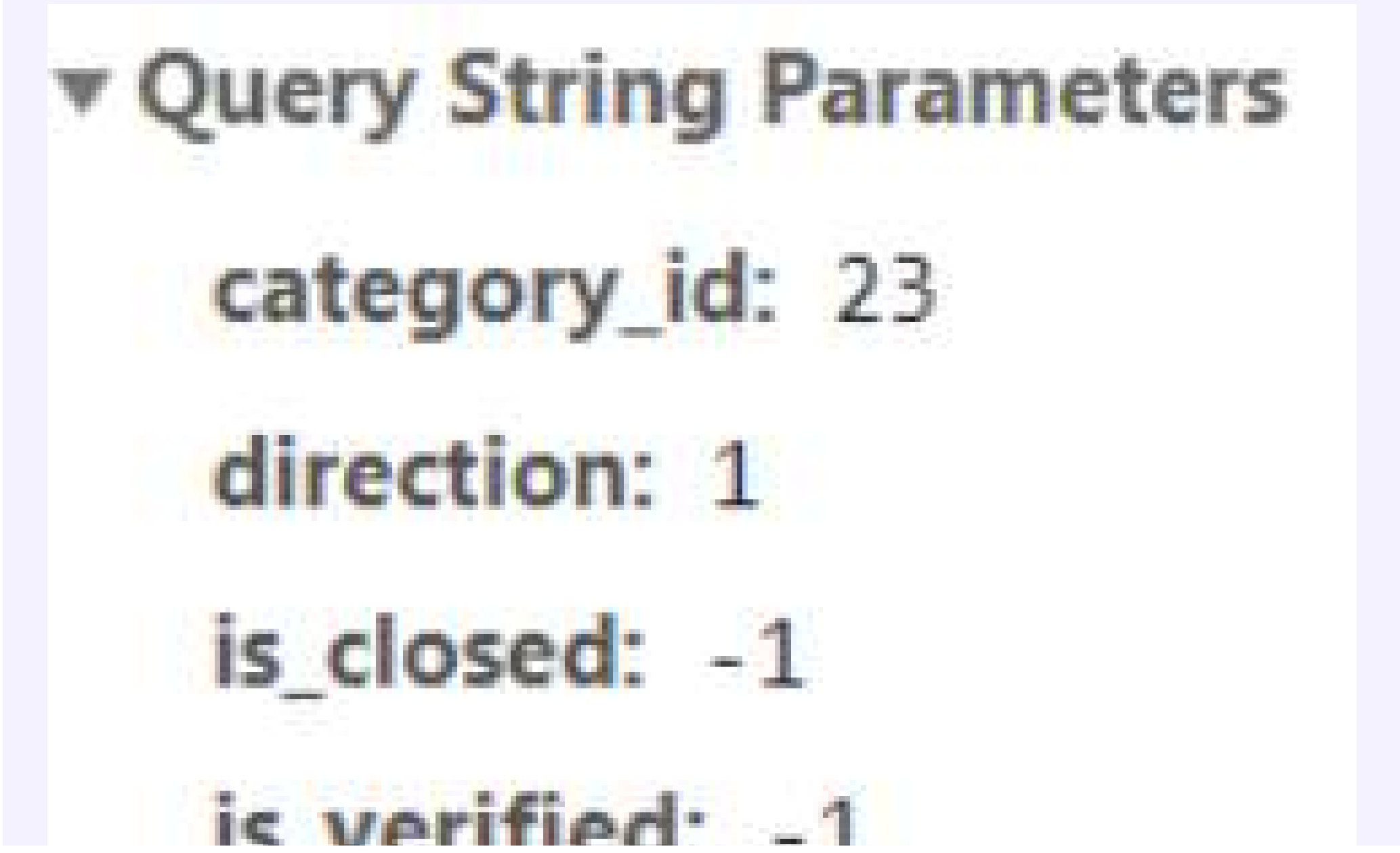
А теперь давайте посмотрим выдачу первых пятидесяти результатов и вот что мы обнаруживаем:

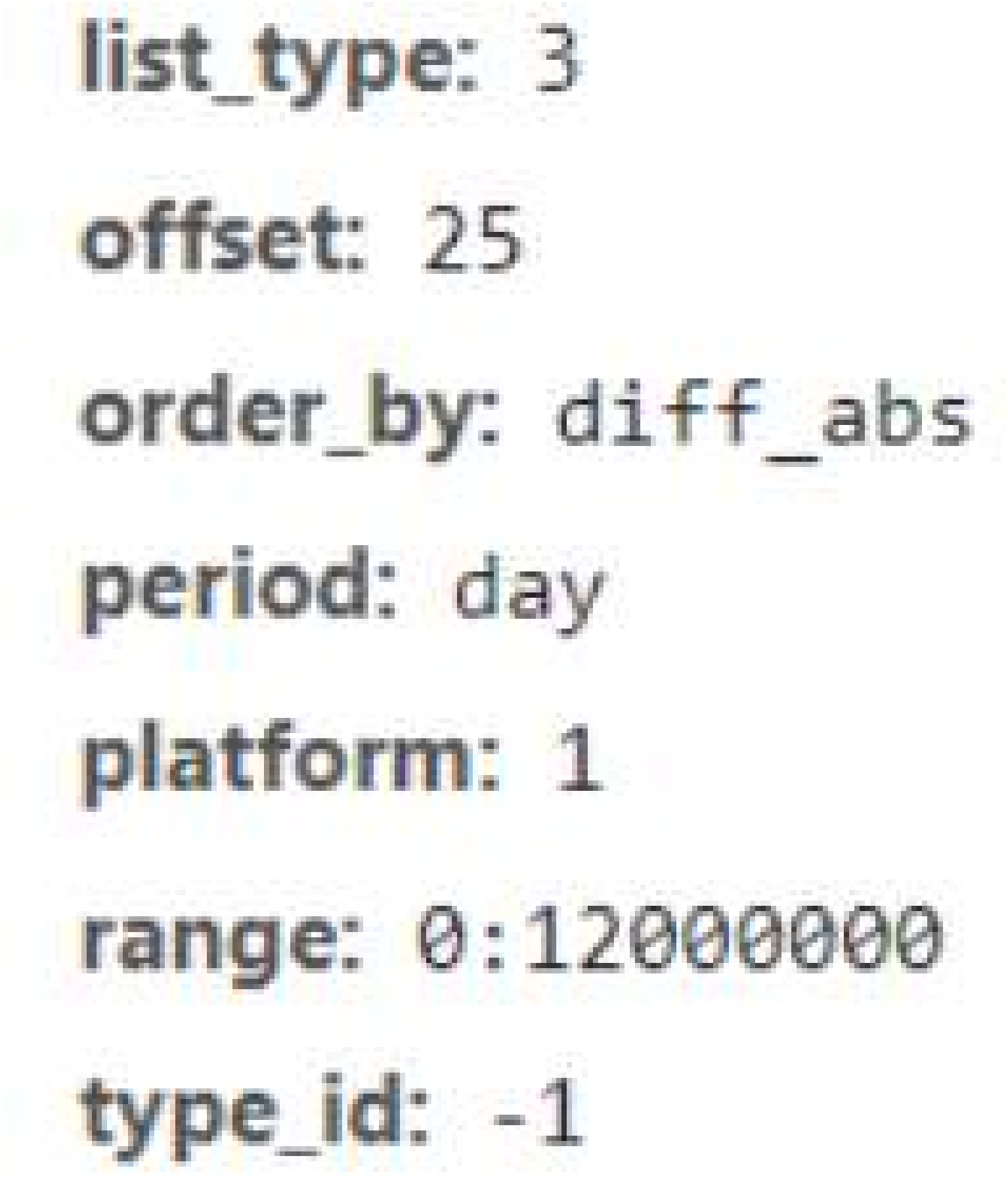
Как видите, последний элемент из первого запроса и первый элемент второго запроса идут по порядку.

Отсюда можно сделать вывод, что параметр `offset` позволяет выдавать `25` новых ответов, через промежуток в `25` единиц.

То есть, `offset = 25`, вернёт записи от 1 до 25, параметр `offset = 50`, вернёт записи с 26 по 50, `offset = 75`, вернёт записи с 51 по 75 и т.д.

Теперь давайте проанализируем остальные параметры запроса и выдвинем гипотезу:

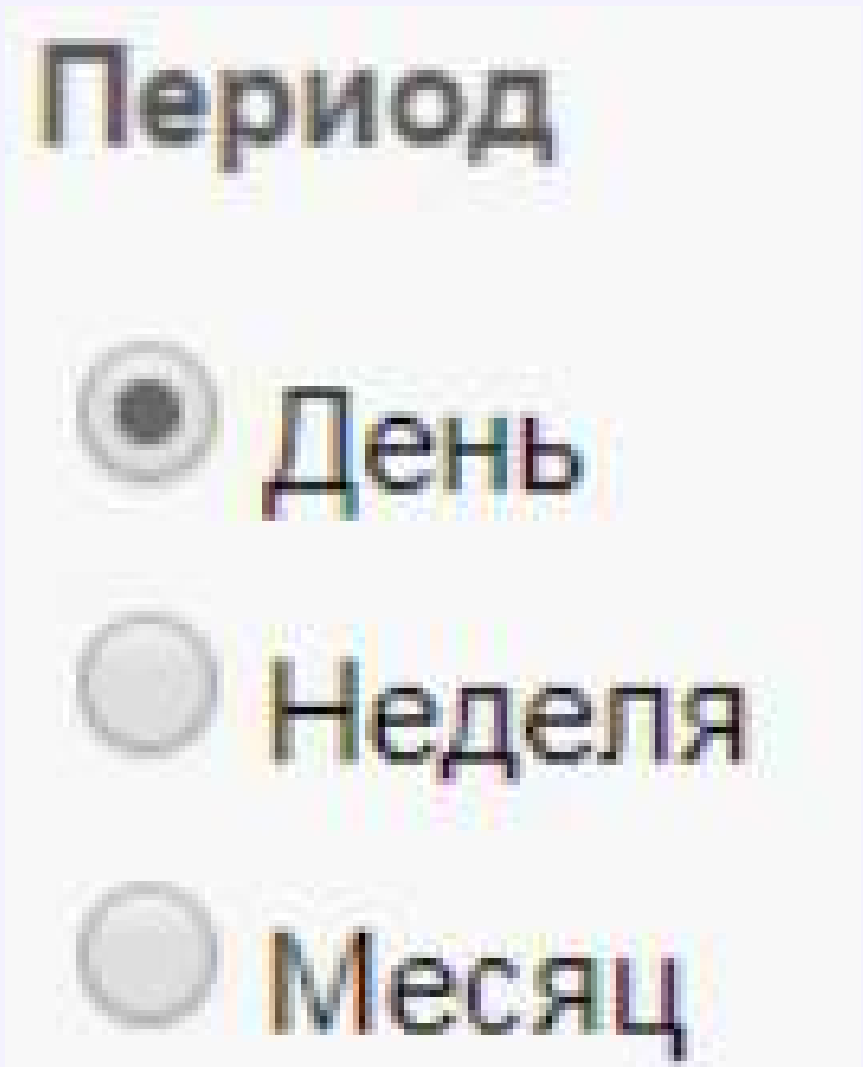




Параметр category_id отвечает за идентификатор категории, из которой мы берём группы.

Данные настройки скорее всего генерируются из формы настроек поиска, поэтому давайте поменяем каждый отдельный пункт настроек и посмотрим, как изменяются запросы.

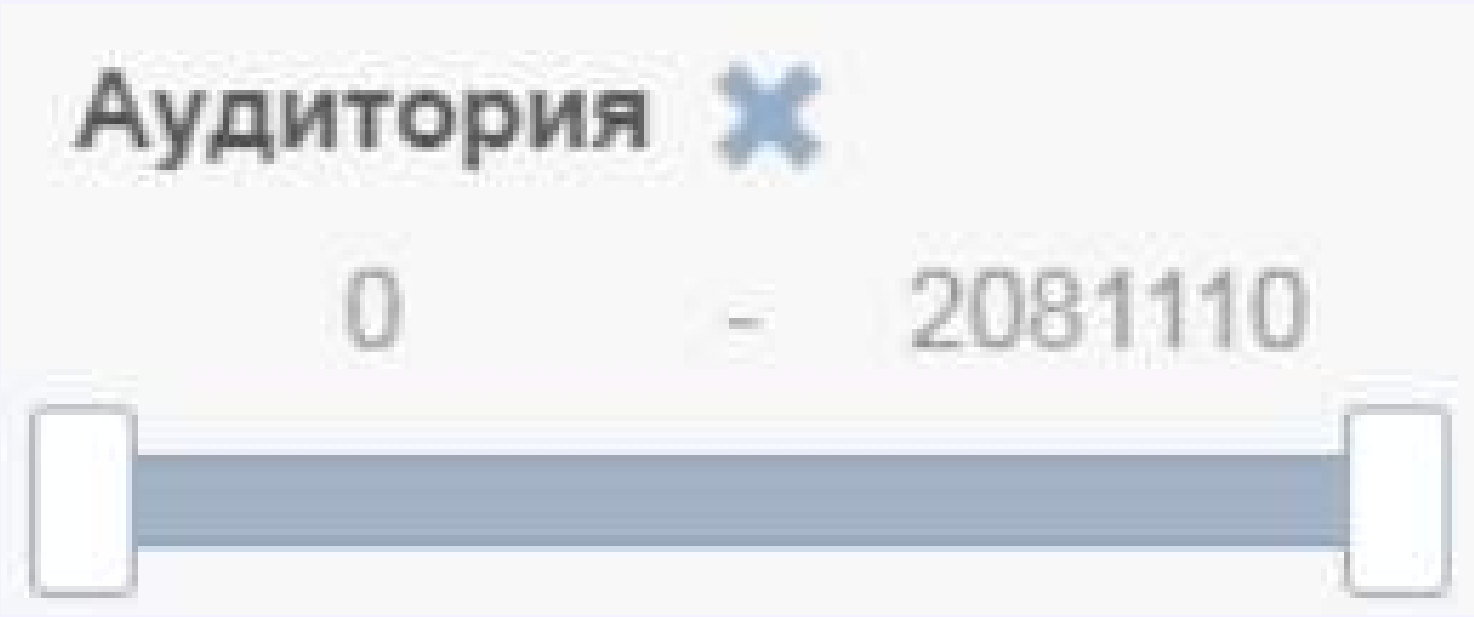
Например, изменение значений блока «Период»:



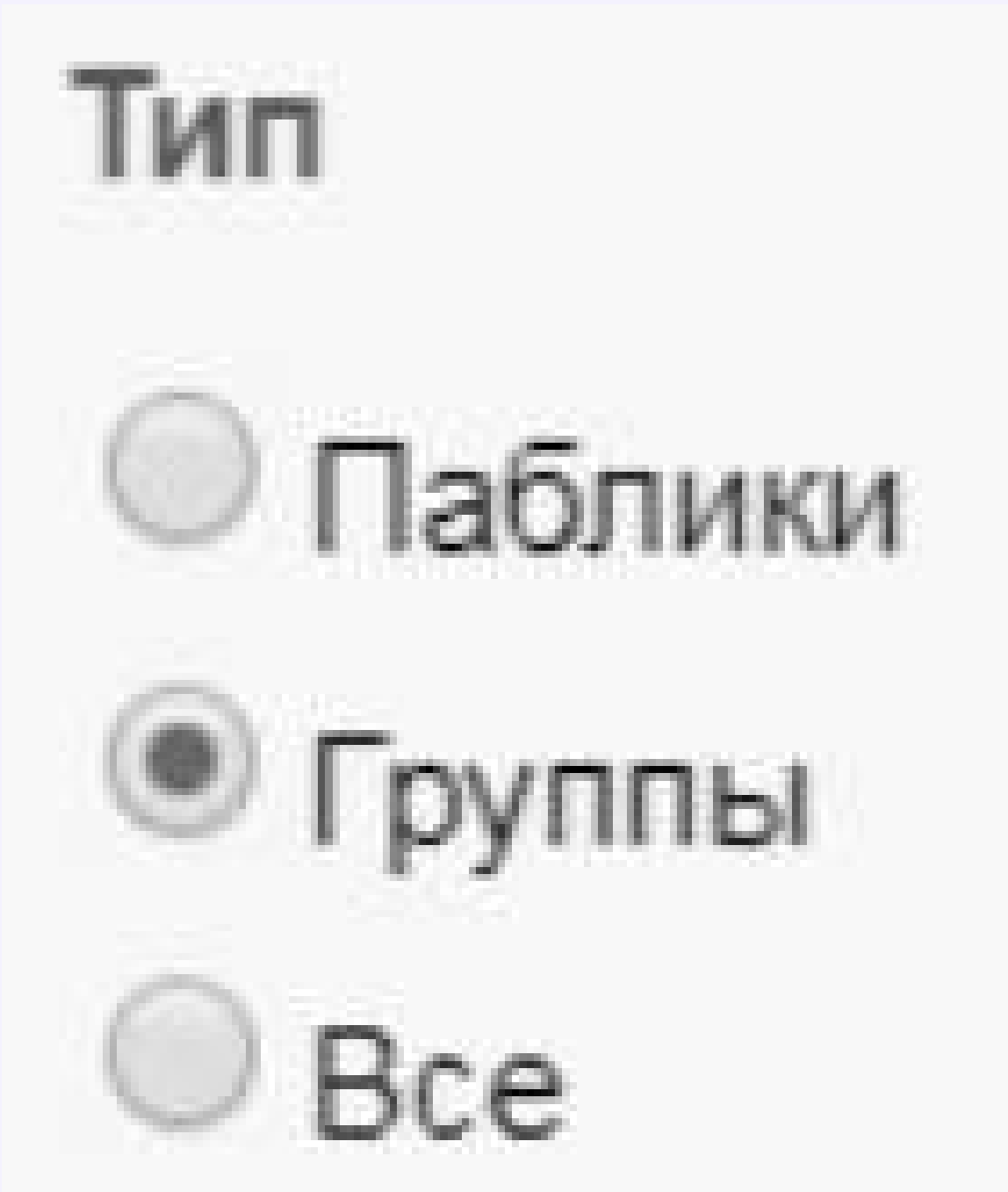
Показывает, что за этот блок отвечает параметр: `period`. В свою очередь данный параметр имеет 3 значения: `day`, `week`, `month`.

Поэтому принципу рассмотрим остальные элементы настроек и обнаружим что параметр range отвечает за диапазон групп по количеству подписчиков и имеет значение вида:

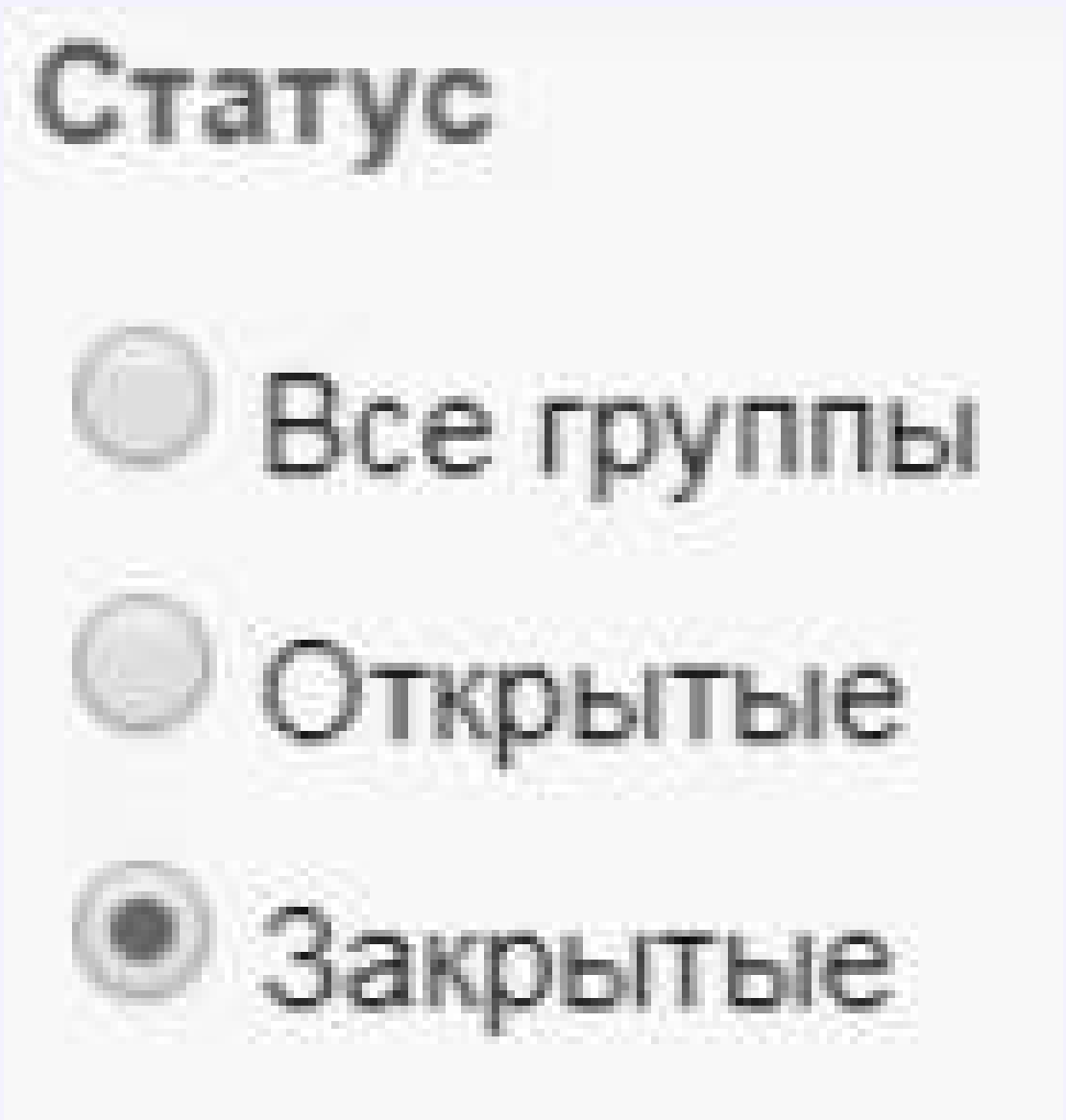
```
[минимальное количество подписчиков]:[максимальное количество подписчиков], например: range: 25000:500000
```



Параметр `type_id` отвечает за тип сообществ и имеет три параметра: паблики (1), сообщества (2), оба типа (3).



Параметр `is_closed` отвечает за статус сообщества и имеет три значения: открытые и закрытые (-1), открытые (1), закрытые (2).

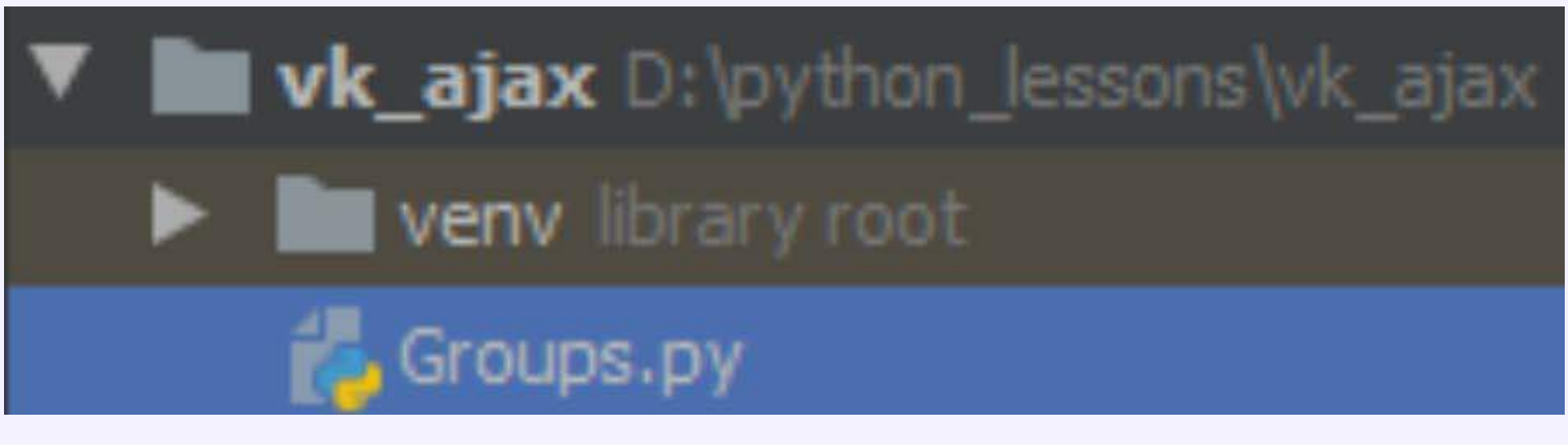


Параметр is_verified отвечает за тип сообщества по признаку верификации. Имеет два значения:

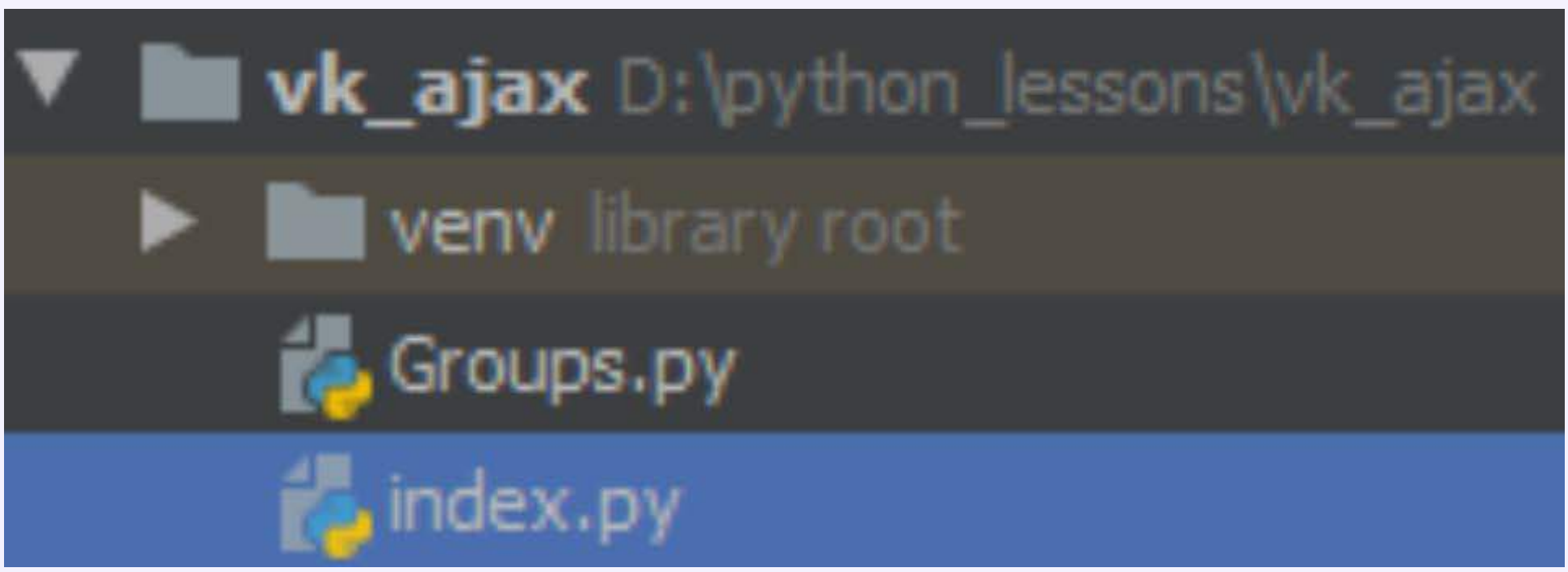
Парсим группы ВК

Создадим новый проект, назовём его `vk_ajax`:

Создадим первый модуль, назовём его `Groups.py`:



Создаём также модуль index.py, в нём будем запускать функционал нашего основного класса:



Модуль Groups

Что должен делать наш парсер? Ну, во-первых, регистрировать входящие данные, то есть он должен как минимум «заставить» пользователя ввести все необходимые данные.

Затем, парсер должен узнать, сколько всего сообществ по нашему запросу есть в выдаче. А затем, сравнить это количество с заданным нами количеством групп для парсинга.

Если количество которое задано нами больше чем фактическое, парсер будет парсить максимально возможное количество. Иначе – то, которое мы указали сами.

Соответственно, затем необходимо отправить запросы на сервер, получить и обработать ответы и возвратить их в виде массива.

Установим в наш проект библиотеку requests и импортируем её в модуль Groups. А также импортируем библиотеку json.

Метод get_max_count

Данный метод определит, сколько всего сообществ в выдаче, чтобы в случае если мы зададим количество сообществ большее, чем есть по запросу мы не отправляли запросы на несуществующие группы:

```python
import json
import math
import requests


def get_max_count(category_id, type, min_sub, max_sub, period, verified, status):
    # Определяем URL на который будет отправлен запрос
    base_url = "https://allsocial.ru/entity"
    # Определяем массив параметров, которые будут отправлены по запросу
    params = {
        "category_id": category_id,
        "type_id": type,
        "range": f"{min_sub}:{max_sub}",
        "period": period,
        "is_closed ": status,
        "is_verified": verified
    }
    # Отправляем запрос с параметрами на сервер
    result = requests.get(base_url, params=params)
    # Если мы получили ответ от сервера
    if result.status_code == 200:
        # Десериализуем JSON-ответ в словарь
        result_dict = json.loads(result.text)
        # Если ответ вернулся без ошибок
        if "response" in result_dict.keys():
            # Возвращаем количество сообществ
            return int(result_dict["response"]["total_count"])
    # Если возникла ошибка, значит не стоит парсить по этому запросу
    # возвращаем значение 0, как будто не существует сообществ по этому запросу
    return 0
```

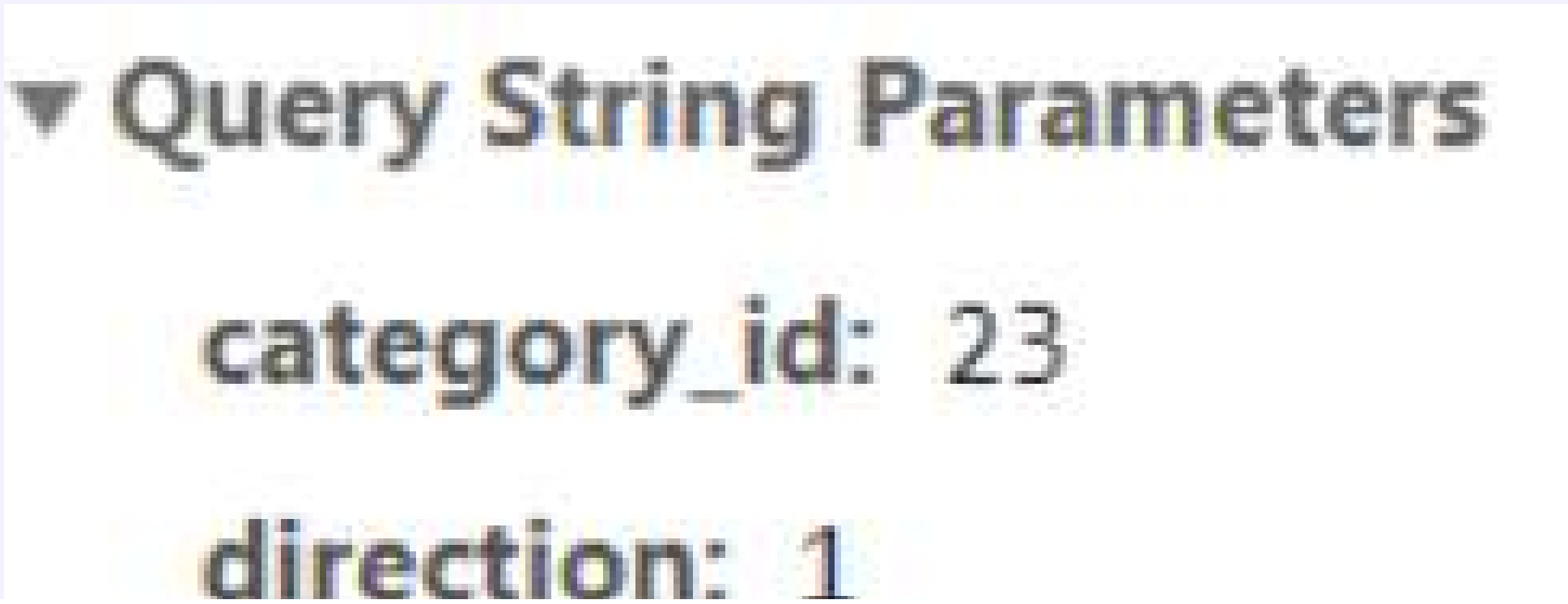
Запустим данную функцию в модуле `index`:

```python
import Groups

count_of_groups = Groups.get_max_count("12", "-1", "0", "500000", "day", "-1", "-1")
print(count_of_groups)
```

Выводит:

> 523752

Все данные для заполнения смотрим при сканировании запроса:



Метод get_list_groups

Этот метод просто возвращает список групп, с некоторым смещением, например, с 26 по 50 группу и т.д.:

```
def get_list_groups(category_id, type, min_sub, max_sub, period, verified, status, offset):
    # Определяем URL на который будет отправлен запрос
    base_url = "https://allsocial.ru/entity"
    # Определяем массив параметров, которые будут отправлены по запросу
```

```
    params = {
        "category_id": category_id,
        "type_id": type,
        "range": f"{min_sub}:{max_sub}",
        "period": period,
        "is_closed ": status,
        "is_verified": verified,
        "offset": offset
    }
    # Отправляем запрос с параметрами на сервер
    result = requests.get(base_url, params=params)
    # Если мы получили ответ от сервера
    if result.status_code == 200:
        # Десериализуем JSON-ответ в словарь
        result_dict = json.loads(result.text)
        # Если ответ вернулся без ошибок
        if "response" in result_dict.keys():
            # Возвращаем количество сообществ
            return result_dict["response"]["entity"]
    # Если возникла ошибка, значит не стоит парсить по этому запросу
    # возвращаем пустой список, как будто не существует сообществ по этому запросу
    return []
```

Запустим этот метод в нашем модуле index:

```
import Groups

list_of_groups = Groups.get_list_groups("12", "-1", "0", "500000", "day", "-1", "-1", 50)
print(len(list_of_groups))
print(list_of_groups)
```

Выводит:

> 25 {'id': 162865, 'vk_id': 65662998, 'quantity': 178013, 'caption': 'Рузке Кэжуал', 'avatar': 'https://sun9-57.userapi.com/c846416/v846416062/154c14/YsjYDJcbuFI.jpg?ava=1', 'diff_abs': -5055, 'diff_rel': -2.7612690366421, 'visitors': 251, 'reach': 1904, 'cpp': -1, 'in_search': 0, 'category': {'public': ['52'], 'private': []}, 'can_change_cpp': -1, 'is_closed': 0, 'is_verified': 0, 'type_id': 1, 'with_order_link': False, 'promoted': False}

Чтобы не забивать экран вывел только одну группу из списка, а всего их, как мы видим 25 штук.

Теперь объединим два наших метода, чтобы мы могли скачивать все необходимые нам группы по запросу.

Метод get

Данный метод автоматически расчитает количество запросов, и оффсеты которые необходимы чтобы скачать заданное количество сообществ:

```
def get(count, category_id, type, min_sub, max_sub, period, verified, status):
    # Получаем максимальное количество групп которые можно спарсить по запросу
    max_count = get_max_count(category_id, type, min_sub, max_sub, period, verified, status)
    # Определяем переменную, в которой будет храниться количесвто итераций
    iterations = 0
    # Определяем список, в котором будут храниться все собранные сообщества
    groups = []
    # Если запрашиваемое количество сообществ больше чем есть в выдаче
    if max_count < count:
        # Приравниваем запрашиваемое количество сообществ к максимально допустимому
        count = max_count
    # Если запрашиваемое или максимальное количество сообществ положительное
    if count > 0:
```

```
        # Определяем количество итераций, необходимое для сбора сообществ
        iterations = math.ceil(count / 25)
        # Запускаем в цикле нужное количество итераций
        for num in range(iterations):
            # Определяем запрос при итерации
            offset = num * 25
            # Добавляем в общий список сообщества полученные по текущему запросу
            groups += get_list_groups(category_id, type, min_sub, max_sub, period, verified, status, offset)
        # Возвращаем общий список сообществ
        return groups
    # В противном случае возвращаем  пустой массив
    return []
```

Запустим данный метод в нашем модуле index:

```
import Groups

list_of_groups = Groups.get(250, "12", "-1", "0", "500000", "day", "-1", "-1")
print(len(list_of_groups))
print(list_of_groups[125])
```

Выводит:

> 250 { 'id': 496453, 'vk_id': 14004071, 'quantity': 6621, 'caption': 'RIPNDIP Russia', 'avatar': 'https://sun9-6.userapi.com/c836131/v836131947/2826c/lQdMobGyKpU.jpg?ava=1', 'diff_abs': -1633, 'diff_rel': -19.784346983281, 'visitors': -1, 'reach': -1, 'cpp': -1, 'in_search': 1, 'category': {'public': [], 'private': []}, 'can_change_cpp': -1, 'is_closed': 0, 'is_verified': 0, 'type_id': 1, 'with_order_link': False, 'promoted': False }

Как мы и запросили, 250 групп из категории 12, с заданными параметрами. Одну из этих 250 групп мы вывели в лог.

Вывод

Парсинг JSON в сущности не отличается от парсинга HTML, мы также сканируем и парсим запросы, только в случае с JSON из него гораздо удобнее и проще извлекать данные, т.к. не нужно парсить HTML-код.

Работа с API

Содержание урока

Общаемся через API

Общение через API практически идентично обычному парсингу, оно точно так же базируется преимущественно на GET и POST запросах, способно выполнять определённые действия по запросам и возвращать некие данные.

Основная разница между обычным парсингом и общением через API – это стандартизация. Если вы общаетесь сайтом по API значит у него вероятнее всего есть документация, строгий формат возвращаемых значений, а сам принцип взаимодействия API основан на какой-то спецификации. Но главное отличие ещё и в том, что обычный парсинг, это не совсем правомерное действия, то есть владелец сайта однозначно против ваших действий.

В случае же с API, вы работаете с санкции администратора ресурса, поэтому никаких заградительных мер, как правило, не следует.

Давайте в рамках данного урока попробуем авторизоваться в ВК через API, а затем напишем парсер лайков к посту.

Документация по API VK: https://vk.com/dev/api_requests

Список методов API: https://vk.com/dev/methods

Прямая авторизация ВК

Для того, чтобы взаимодействовать с АПИ ВК, нужно авторизовать конкретный аккаунт ВК, от имени которого будут производиться запросы. При авторизации вы получите специальный ключ, который необходимо указывать при каждом запросе к АПИ ВК.

У ВК есть очень муторная, полуручная авторизация через приложения. Я же советую вам использовать некоторый баг, а точнее фичу API VK. Дело в том, что для некоторых авторизованных приложений, аля Kate Mobile или официальных приложений ВК есть возможность полностью автоматической авторизации лишь по логину и паролю.

Это делается при помощи метода прямой авторизации: https://vk.com/dev/auth_direct

Для того, чтобы можно было реализовать прямую авторизацию, необходимо передать в запросе в том числе id и секретный ключ согласованного с администрацией ВК приложения. Разумеется, вам никто не согласует приложение на прямую авторизацию.

Однако, есть ряд приложений, которым это доступно. Так вот, для прямой авторизации можно воспользоваться данными этих приложений. Как-бы «прикинуться» ими.

Для этого мы возьмём значения параметров `client_id` и `client_secret` этих приложений по данной ссылке: https://github.com/Kutabe/vk/blob/master/vk.go

Рассмотрим детальнее процесс авторизации. Чтобы авторизоваться и получить авторизационный токен для вашего аккаунта, необходимо отправить GET-запрос по адресу: https://oauth.vk.com/token

Со следующими параметрами:

1. `grant_type` (обязательный параметр) – тип авторизации, должен быть равен `password`.
2. `client_id` (обязательный параметр) – `id` Вашего приложения.
3. `client_secret` (обязательный параметр) – секретный ключ Вашего приложения.
4. `username` (обязательный параметр) – логин пользователя.
5. `password` (обязательный параметр) – пароль пользователя.
6. `scope` – права доступа, необходимые приложению (список прав доступа расположен на следующей странице: https://vk.com/dev/permissions).
7. `test_redirect_uri` – это 1 - инициировать тестовую проверку пользователя, используя `redirect_uri`. 0 — обычная авторизация (по умолчанию).
8. `v` (обязательный параметр) – версия API, которую Вы используете. При повышении версии пожалуйста ознакомьтесь со списком изменений. Актуальная версия: 5.92.
9. `2fa_supported` - передайте 1, чтобы включить поддержку двухфакторной аутентификации.

Пример запроса на авторизацию:

> https://oauth.vk.com/token?
> grant_type=password&client_id=1914441&client_secret=***&username=***&password=***&v=5.92&2fa_supported=1

Или если расшифровать:

> https://api.vk.com/oauth/token?grant_type=password&client_id=[наше значение]&scope=[наше значение]&client_secret=[наше значение]&username=[наше значение]&password=[наше значение]

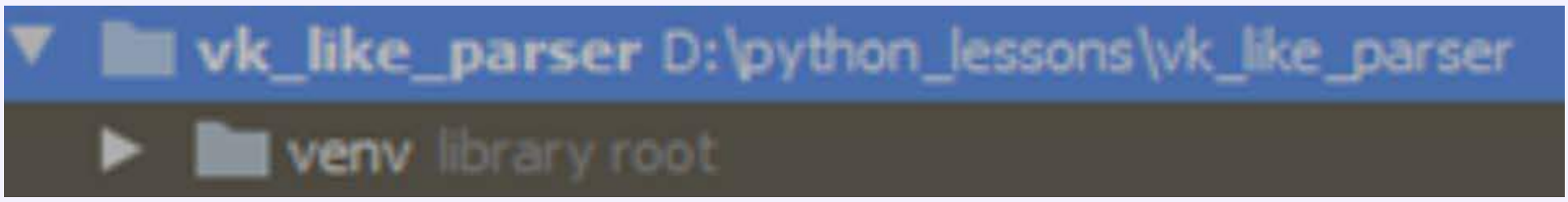
Таким образом, для прямой авторизации нам нужно знать 4 значения. `client_id` и `client_secret` приложения допущенного к прямой авторизации, `username` и `password` - данные для авторизации аккаунта.

В результате оправки данного запроса будет возвращён JSON-объект. С необходимым для отправки запросов токеном, например:

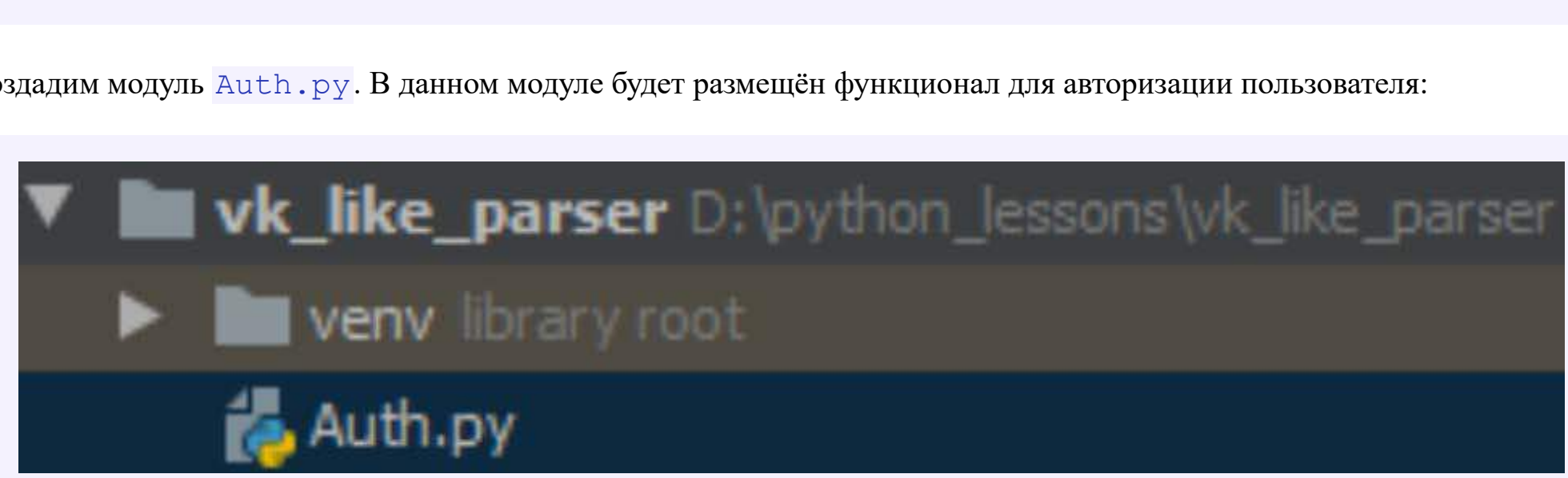
```
{"access_token":"f4b770c94dde0d563c3cd249ce8cad6e9f08c0c8cc376e3a055d51d21c3484fcb4920afb1e092e9dd88","expires_in":0,"
user_id":532443954}
```

Пишем модуль авторизации ВК

Для начала создадим новый проект. Назовём его `vk_like_parser`:



Создадим модуль `Auth.py`. В данном модуле будет размещён функционал для авторизации пользователя:



Устанавливаем в наш проект модуль `requests`. Импортируем в модуль `Auth` модули `requests` и `json`:

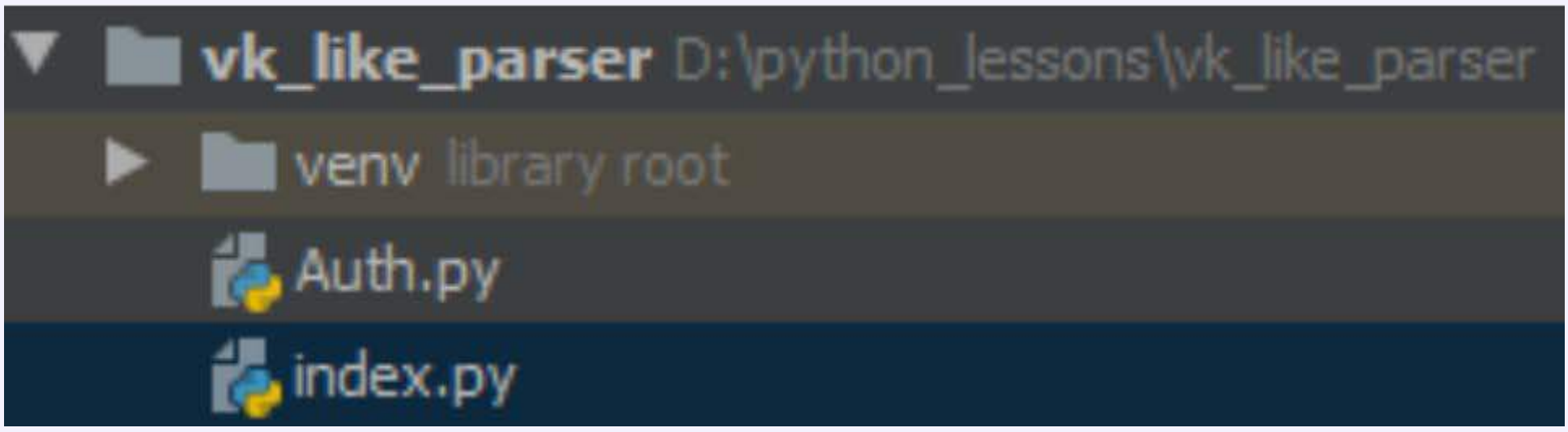
```
import json
import requests
```

Теперь напишем функцию которая авторирзует нас на сервере и вернёт авторизационный токен:

```
import json
import requests


def get_token(login, password):

    # Формируем массив параметров для авторизации
    params = {
        "grant_type": "password",
                "client_id": "2274003",
                "client_secret": "hHbZxrka2uZ6jB1inYsH",
                "username": login,
                "password": password,
                "2fa_supported": "1",
                "scope":
        "notify,friends,photos,audio,video,stories,pages,status,notes,messages,wall,ads,offline,docs,groups,notifications,stats,email,
        market,stories,photos,app_widget,messages,docs,manage",
                "v": "5.92",

    }
    # Отправляем запрос на сервер
    result = requests.get("https://oauth.vk.com/token", params=params)
    # Декодируем результат в словарь
    response = json.loads(result.text)
    # Если сервер ответил и в ответе есть токен - возвращаем его
    if result.status_code == 200 and "access_token" in response.keys():
        return response["access_token"]
```

```
    # Иначе возвращаем False
    return False
```

Теперь создадим привычный нам модуль `index`:



Импортируем в него модуль `Auth` и вызовем метод `get_token`:

```
import Auth

access_token = Auth.get_token("79506948825", "HMHYJswK1ws")
print(access_token)
```

Выводит:

> e13c52c25cec64174ae75aab728712933deaa6f247e81ea3a0135c111b59cb4eec44a58f480c6a795fd0e

Это и есть наш токен, далее при работе с API мы должны отправлять его при каждом запросе. Давайте попробуем использовать его на практике и при помощи API VK напишем парсер лайков к посту.

P.S. Этот токен вечный, то есть, его можно использовать до тех пор, пока вы повторно не авторизуетесь с этим логином и паролем и не сгенерируете новы токен. Поэтому, один раз получив такой токен, его имеет смысл сохранить в файл или БД и затем использовать, вместо того, чтобы каждый раз генерировать новый токен для пользователя.

Отправка запроса к API VK

Запрос к конкретному методу отправляется по адресу вида: `https://api.vk.com/method/имя_метода`

Или в нашем случае: https://api.vk.com/method/likes.getList

И соответственно необходимо передать параметры, необходимые для работы метода (они указываются в документации к функции).

Запрос можно отправлять как методом GET, так и методом POST. Разницы нет.

Пишем модуль для парсинга лайков у поста ВК

API VK, как, впрочем, и другие API состоят из методов. Каждый метод выполняет определённую функцию, например, поставить лайк на фото или получить список пользователей сообщества и т.д.

Список методов представлен по ссылке: https://vk.com/dev/methods

Для получения списка пользователей, лайкнувших некоторый пост необходимо воспользоваться методом: `likes.getList`

Документация метода: https://vk.com/dev/likes.getList

Метод принимает ряд обязательных параметров:

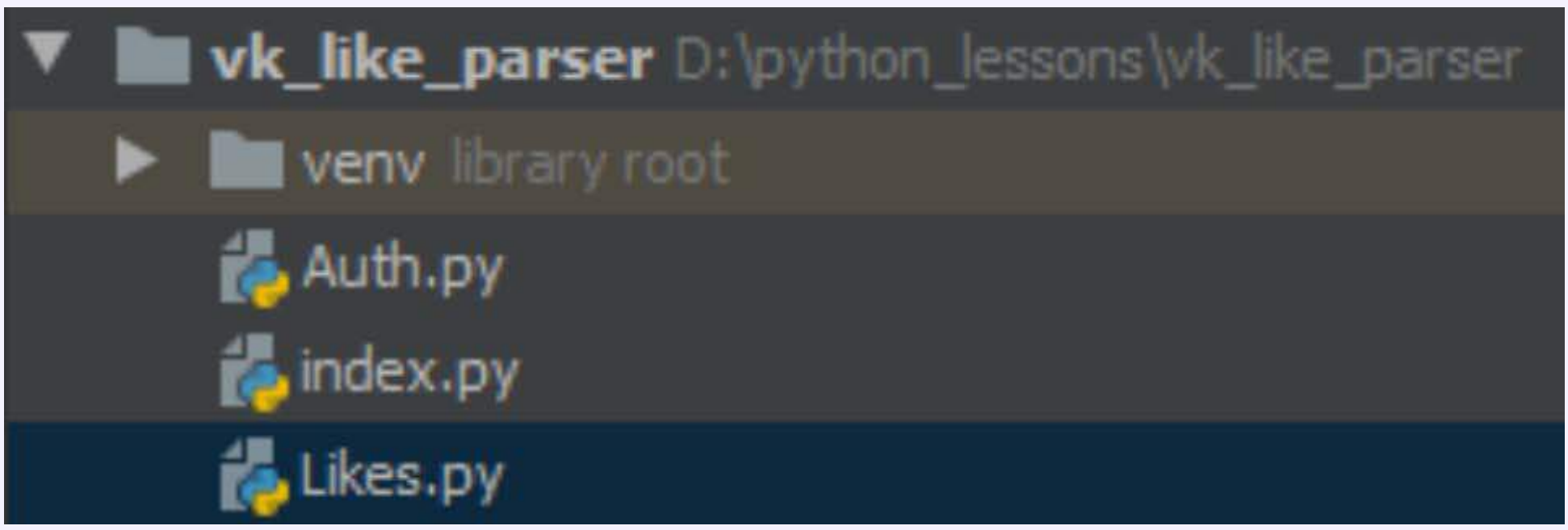
`type` – тип объекта, с которого собираем лайки (пост, фото, комментарий и т.д.)
`owner_id` – `id` сообщества или пользователя, лайки с поста которого мы парсим
`item_id` – `id` поста, лайки с которого мы парсим
`count` – сколько лайков с поста собирать (максимум 1000)
`access_token` – токен пользователя
`v` – версия API

ID сообщества и ID поста можно взять из самой ссылки на пост, например, ссылка: https://vk.com/feed?w=wall-87651722_52363

Здесь ID сообщества, это (всё что между «wall» и «_»): `-87651722`. ID поста (число после «_»): `52363`

Если у поста больше 1000 лайков, то чтобы собрать их все необходимо отправлять запросы с параметром `offset`.

Приступим к написанию модуля. В текущем проекте создадим модуль `Likes.py`:



Импортируем в него модули:

```
import json
import requests
```

Далее, напишем саму функцию по сбору пользователей лайкнувших пост:

```
def get(link, token):
    # Разбиваем ссылку вида: https://vk.com/feed?w=wall-87651722_52363
    # на 2 части (1) https://vk.com/feed?w= и (2) -87651722_52363
    # заносим вторую часть (-87651722_52363) в переменную
    id = link.split("wall")[1]
    # Разбиваем id поста вида: -87651722_52363
    # на 2 части (1) -87651722 и 52363
    # и помещаем их в две переменные с параметром
    owner_id, item_id = id.split("_")

    # Формируем словарь с параметрами для запроса
    params = {
        "type": "post",
        "owner_id": owner_id,
        "item_id": item_id,
        "count": 1000,
        "access_token": token,
        "v": "5.92",
    }
    # Отправляем запрос к API
    result = requests.post("https://api.vk.com/method/likes.getList", params=params)
```

```
    # Десериализуем ответ из JSON в словарь
    response = json.loads(result.text)
    # Если сервер успешно ответил и в ответе нет ошибок
    if result.status_code == 200 and "response" in response.keys():
        # Возвращаем список пользователей лайкнувших пост
        return response["response"]["items"]
    # В противном случае возвращаем False
    return False
```

Здесь всё в целом знакомо, кроме наверное работы с функцией `split`, с ней мы тоже работали, но разберём.

Наша задача из ссылки вида: `https://vk.com/feed?w=wall-87651722_52363`. Извлечь два значения:
(1) `-87651722` и `52363`

Поэтому сперва я извлёк из этой ссылки весь id: `-87651722_52363`, разбив эту строку по подстроке «wall» на две
(1) `https://vk.com/feed?w=` и (2) `-87651722_52363`

А затем просто разбил вторую строку ещё на две переменные по символу «_» получив список из двух значений:
(1) `-87651722` и (2) `52363` и сразу присвоил их двум переменным: `owner_id` и `item_id`.

Теперь попробуем запустить этот код в нашем модуле `index`:

```
import Auth
import Likes

# Получаем токен для авторизации
access_token = "e13c52c25cec64174ae75aab728712933deaa6f247e81ea3a0135c111b59cb4eec44a58f480c6a795fd0e"
# Получаем список пользователей лайкнувших определённые пост
users = Likes.get("https://vk.com/feed?w=wall-87651722_52363", access_token)

if users:
    # Выводим поличество полученных пользователей
    print(len(users))
    # Выводим список полученных пользователей
    print(users)
else:
    print("При парсинге произощла проблема")
```

Здесь я не стал повторно запускать модуль `Auth`, так как токен мной был уже получен в прошлом примере.

Выводит:

> 47 [338633920, 244174938, 154682025, 7476545, 470062015, 99609477, 265646351, 197159285, 86699339, 558385107, 153452434, 289111930, 474171893, 535521749, 319556926, 58895844, 299630906, 505380302, 171952781, 3092270, 459052298, 184341926, 151156614, 217512232, 154584060, 322710150, 195353264, 259025994, 7406544, 297371256, 226843831, 10256374, 86095960, 203983078, 29840, 353424427, 176432817, 173875875, 394830934, 76391742, 394268592, 3447275, 155354427, 95486563, 319166509, 530593530, 9819182]

Как видите, работа с API гораздо проще, у нас есть готовая документация которая указывает каким именно образом нам нужно отправить запрос, по запросу возвращаются удобные данные в формате JSON которые очень просто извлечь.

Классы в Python

Содержание урока

Почему стоит пользоваться классами?

В рамках данного курса я не буду раскрывать тему ООП, его концепций и паттернов. Однако я нахожу необходимым рассмотреть классы в Python, как минимум как крайне удобную и безопасную обёртку для ваших функций.

Что такое класс? Это просто некоторая конструкция языка, в которую помещается ваш код (переменные и функции). Разместив код внутрь класса у вашего кода расширяется функциональность, и вы можете гораздо продуктивнее его использовать.

Это не академическое толкование данного термина. А лишь та часть его значения, которая будет использоваться нами в повседневной деятельности.

При работе с классами наиважнейшей фичей для вас будет возможность создать, любое количество переменных, которые внутри класса будут доступны для всех функций, при этом извне вы не сможете на них повлиять (на самом деле сможете, но класс позволит вам не сломать программу по невнимательности).

Пример. Мы пишем модуль, который умеет с нами здороваться, прощаться и спрашивать, как наши дела.

Разумеется, данный функционал разумно вынести в отдельные функции, например:

```
say_hello
say_goodbye
say_whats_going
```

Для того, чтобы наш собеседник был более приятным, научим его обращаться по имени. Сперва создадим модуль, назовём его `my_friend`.

Выходит, для того, чтобы функции обращались к нам по имени, это имя необходимо в них передать. Самый очевидный вариант, передать в качестве аргумента:

```python
def say_hello(name):
    print(f"Hello {name}")

def say_goodbye(name):
    print(f"Goodbye {name}")

def say_whats_going(name):
    print(f"What`s going {name}?")
```

И вызвать:

```python
import my_friend

my_friend.say_hello("Korben")
my_friend.say_goodbye("Korben")
my_friend.say_whats_going("Korben")
```

Выводит:

> Hello Korben Goodbye Korben What`s going Korben?

Нудное занятие, каждый раз передавать один и тот же аргумент в разные функции. Можно сделать иначе:

```python
name = None

def say_hello():
    print(f"Hello {name}")

def say_goodbye():
    print(f"Goodbye {name}")

def say_whats_going():
    print(f"What`s going {name}?")
```

Вызовем:

```python
import my_friend

my_friend.name = "Korben"
my_friend.say_hello()
my_friend.say_goodbye()
my_friend.say_whats_going()
```

Выводит:

> Hello Korben Goodbye Korben What`s going Korben?

Данный принцип уже куда более приемлем и удобен. Он даже синтаксически идентичен подходу с использованием классов. Но есть одно, НО. Ко всем переменным и методом модуля вы можете обратиться в любой момент времени, изменить его или прочитать.

Если вы работаете с классом, то в нём можно также «залезть» в любой его член и всё сломать. Но только класс даёт всё же большую защищённость, он позволяет как минимум задать обязательные параметры, без которых работа остальных функций класса будет невозможна.

Например, в рамках кода, написанного выше, если мы не зададим имя для модуля, то получим бессмыслицу:

```
import my_friend

my_friend.say_hello()
my_friend.say_goodbye()
my_friend.say_whats_going()
```

Выводит:

> Hello None Goodbye None What`s going None?

Разумеется, код обращается ни к кому. А если, например, в эту переменную должен передавать какой-нибудь объект или структура данных, а мы просто забыли её передать?

Или, скажем по окончании работы класса нам необходимо всякий раз автоматически производить какие-то действия (это класс тоже умеет).

Словом, причин пользоваться классами даже не зная ООП куда больше, чем ими не пользоваться.

Понятие класса

Дальнейшее рассмотрение этой темы продолжим на уже реализованном напи примере модуля `my_friend`.

Класс – это специальным образом, описанный шаблон вашего кода. Вы объявляете класс и помещаете в этот класс ваши переменные и функции. Затем, создаёте из этого класса объект, который открывает вам доступ ко всем открытым членам класса.

Давайте для примера, разместим наши функции в классе. Перепишем содержание нашего модуля.

Класс объявляется следующим образом:

```
class имя_класса:
    содержание_класса
```

Например:

```
class MyFriend:
    # Переменная (сфойство)
    name = None
    # Функция (метод)
    def say_hello(self):
        print(f"Hello {self.name}")
    # Функция (метод)
    def say_goodbye(self):
        print(f"Goodbye {self.name}")
    # Функция (метод)
    def say_whats_going(self):
        print(f"What`s going {self.name}?")
```

Как видите, содержимое практически не изменилось, просто весь функционал и переменные засунули внутрь конструкции `class`. Но изменения всё же есть, можете заметить, что теперь внутри наших функций появилась ссылка

на объект `self`.

Только через это ключевое слово можно обратиться ко внутренним членам класса.

Это слово указывается в аргументе каждого нестатического метода, благодаря нему мы получаем доступ к переменным и другим методам класса. То есть, функция `say_goodbye` может обратиться к любой другой функции или методу только в том случае, если первым (он всегда должен быть на первом месте) аргументом был передан `self`.

Ещё раз, к нестатическим свойствам и методам внутри класса можно обращаться только через атрибут `self`. Если вы хотите, чтобы из конкретной функции можно было обратиться к другим членам класса в неё обязательно нужно передать атрибут `self`.

Внутри функции также можно создавать переменную. В таком случае это будет обычная локальная переменная, как в обычной функции модуля, например:

```
# Функция (метод)
def say_hello(self):
    age = 22
    print(f"Hello {self.name}! Your age: {age} years")
```

Эта функция недоступна для других функций внутри класса и вне класса к ней также нельзя получить доступ. Обращение ко внутренним, локальным функциям происходит без ссылки `self`.

Попробуем вызвать наш класс. Сперва, наш класс нужно импортировать. Делается это также, как и с модулем.

Из модуля в котором объявлен наш класс импортируем сам класс:

```
from my_friend import MyFriend
```

Теперь создаём объект класса, таким образом:

```
переменная_для_объекта = имя_класса()
```

Вот так:

```
from my_friend import MyFriend

friend = MyFriend()
friend.name = "Korben"
friend.say_hello()
friend.say_goodbye()
friend.say_whats_going()
```

Выводит:

```
Hello Korben
Goodbye Korben
What`s going Korben?
```

Выглядит также, как и с модулем.

Конструктор

Теперь давайте попробуем заставить любого пользователя класса в обязательном порядке указать имя:

```
class MyFriend:
    # Переменная (сфойство)
    name = None

    # Конструктор
    def __init__(self, name):
        self.name = name

    # Функция (метод)
    def say_hello(self):
        print(f"Hello {self.name}")

    # Функция (метод)
    def say_goodbye(self):
        print(f"Goodbye {self.name}")

    # Функция (метод)
    def say_whats_going(self):
        print(f"What`s going {self.name}?")
```

Как видите, у нас появилась функция `__init__`. Это так называемый конструктор. Данная функция будет запущена автоматически, во время создания объекта класса:

```
friend = MyFriend()
```

Мы можем сами определить, какие параметры будем в неё передавать, а также что данная функция будет делать, то есть произвольно определить содержимое этой функции. Одно условие, она не возвращает никаких значений, это правило.

Зачем нужен конструктор? Ну, например, он может выполнить некоторую обязательную работу перед тем, как начнут работать другие функции, например, создаст объекты других классов, которыми мы будем пользоваться в других функциях или присвоит некоторым переменным значения.

Если мы указали, что в функцию `__init__` передаются некоторые параметры, то мы обязаны их передать при создании объекта, вот так:

```
friend = MyFriend([здесь_параметры])
```

То есть, все аргументы указанные в функции __init__ передаются в неё при создании объекта класса:

```
переменная_для_объекта = имя_класса([параметры_передаваемые_в_init])
```

Например:

```
from my_friend import MyFriend

friend = MyFriend("Korben")

friend.say_hello()
```

```
friend.say_goodbye()
friend.say_whats_going()
```

Теперь мы обязательно должны передавать имя пользователя при создании объекта. Иначе возникнет ошибка:

```
from my_friend import MyFriend

# Не передали атрибут name
friend = MyFriend()

friend.say_hello()
friend.say_goodbye()
friend.say_whats_going()
```

Выводит:

> TypeError: __init__() missing 1 required positional argument: 'name'

Теперь при работе с классом мы точно не забудем передать нужные параметры, поскольку не получится создать объект класса. К тому же, данная переменна будет доступна всем остальным функциям внутри класса.

Деструктор

Деструктор подобно конструктору запускается автоматически, он не возвращает значений и не принимает аргументов. Данный метод запускается после того как класс закончил свою работу. Его используют для удаления разных ресурсов или выполнения каких-то действий по умолчанию после завершения работы.

Например, отправка какого-то отчёта по сети или сохранение данных о работе в лог-файл.

Деструктор определяется функцией `__del__`:

```
def __del__(self):
    print(f"I leave, {self.name}")
```

Внутри функции соответственно мы определяем некоторую логику, которая должна отработать после завершения работы класса.

Сохраним изменения и вызовем наш код:

```
from my_friend import MyFriend

friend = MyFriend("Korben")

friend.say_hello()
friend.say_goodbye()
friend.say_whats_going()
```

Выводит:

> Hello Korben! Your age: 22 years Goodbye Korben What`s going Korben? I leave, Korben

Явным образом последнюю строчку мы не вызывали, деструктор запустился автоматически.

Приватные и защищённые свойства и методы

Иногда, полезно скрыть некоторые функции и переменные от пользователя (или от себя) дабы случайно не влезть в жизненно важные части нашего кода. Таким образом, создав объект класса вы не сможете изменить некоторые данные или вызвать некоторые функции поскольку они будут закрыты для использования извне при помощи специальных модификаторов доступа.

Сразу следует отметить, что эти модификаторы носят лишь декларативный характер и не порождают настоящих ошибок интерпретатора, то есть к приватным и защищённым переменным всё же можно получить доступ извне, если это необходимо.

В данном случае, важно следовать этому правилу.

В Python существует два модификатора доступа: приватный и защищённый. Приватный модификатор позволяет вызов приватных функций и переменных только внутри класса. Защищённый позволяет вызов приватных функций и переменных как внутри класса, так и в дочерних (но в рамках данной темы мы не будем с ними работать).

Для того, чтобы создать переменную или функцию с приватным модификатором, достаточно написать перед именем переменной или функции два нижних подчёркивания:

```
class MyFriend:
    # Переменная (сфойство)
    __name = None
```

Теперь, попытавшись присвоить значение этой переменной ничего не произойдёт:

```
from my_friend import MyFriend

friend = MyFriend("Korben")

friend.__name = "Mark"

friend.say_hello()
friend.say_goodbye()
friend.say_whats_going()
```

Выводит:

> Mark Hello Korben! Your age: 22 years Goodbye Korben What`s going Korben? I leave, Korben

Имя осталось тем же, несмотря на то, что мы присвоили другое значение. Всё потому что изменить значение можно только внутри класса, либо первоначально присвоить его через конструктор (что тоже происходит внутри класса).

Если же мы попробуем получить значение приватной переменной всплывёт ошибка:

```
from my_friend import MyFriend

friend = MyFriend("Korben")
print(friend.__name)
friend.say_hello()
friend.say_goodbye()
friend.say_whats_going()
```

Выведет:

```
AttributeError: 'MyFriend' object has no attribute '__name'
```

Защищённый модификатор

Защищённым модификатором в рамках курса мы не будем пользоваться, но я всё же о нём расскажу. Он используется, когда мы хотим указать что переменная или функция могут использоваться только в текущем или дочернем классе.

Создаются защищённые модификаторы при помощи символа нижнего подчёркивания перед именем переменной или функции (`_name = None`). При попытке обратиться к члену класса с таким модификатором или присвоить ему значение – не возникнет никаких ошибок и значение будет присвоено:

```
from my_friend import MyFriend

friend = MyFriend("Korben")
friend._name = "Oleg"
print(friend._name)
friend.say_hello()
friend.say_goodbye()
friend.say_whats_going()
```

Выводит:

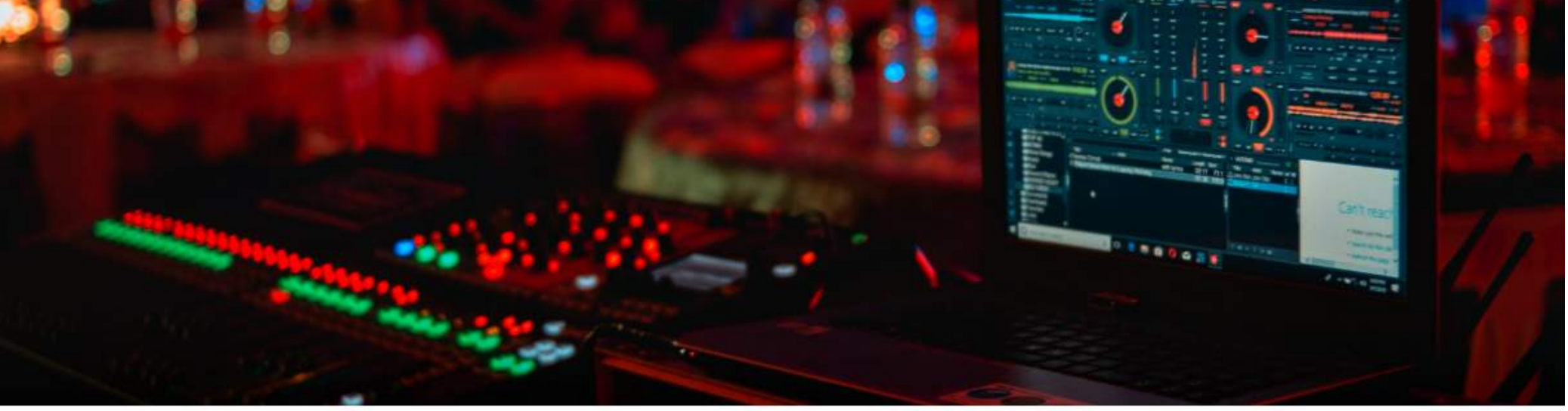
> Oleg Hello Oleg! Your age: 22 years Goodbye Oleg What`s going Oleg? I leave, Oleg

Как видите, значение переменной было изменено. Хотя оно и было установлено в конструкторе как Korben, на экран было выведено имя Oleg.

К тому же, при попытке получить значение этого защищённого члена не появилось ошибок.

Просто нужно знать, что, если вы хотите указать, что заданное свойство или метод нельзя вызывать извне, вы можете использовать приватные или защищённые модификаторы доступа.

На этом основы работы с классами будет достаточно. В дальнейших уроках я на примере разберу как с ними работать.



Масслайкер комментариев на Youtube

Содержание урока

Принцип работы программы

В данном уроке напишем масслайкер комментариев на Youtube. Данный скрипт будет принимать на вход ссылку на видео, а также количество комментариев, которые необходимо собрать, процент из них, на который необходимо поставить лайк и задержка между лайками.

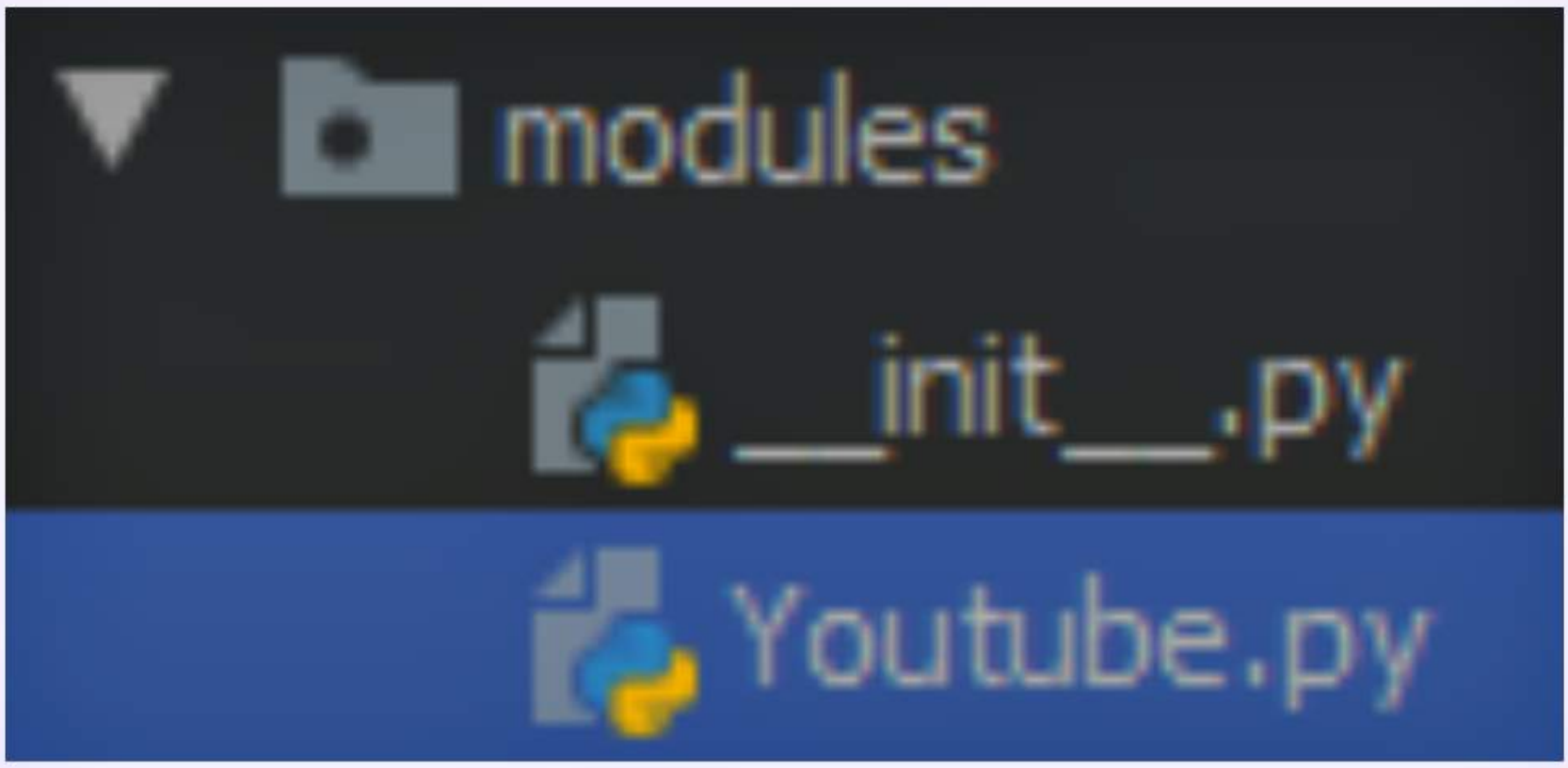
Например, мы указываем ссылку на видео, указываем количество комментариев, которые мы соберём для лайкинга и тот процент из них, которые необходимо пролайкать, а также задержку между лайками.

Алгоритм работы скрипта:

1. Устанавливаем куки для авторизации
2. Переходим на станицу с видео
3. Собираем заданное количество комментариев
4. Выбираем случайный заданный процент из них
5. Совершаем действие «лайк» на каждый из заданных комментариев

Пишем лайкер

Создадим новый проект, назовём его `youtube-liker`. Установим в проект `Selenium`. Создадим в проекте пакет `modules`, в нём модуль `Youtube.py`:



Далее, мы разместим в нашем модуле класс:

```
class Youtube:
```

Внутри которого и будут размещены функции и переменные нашего скрипта. Также, добавим в класс следующие модули, которые пригодятся нам в дальнейшей работе:

```
import random
import time

from selenium.common.exceptions import *
from selenium.webdriver import ActionChains
from selenium.webdriver.common.keys import Keys


class Youtube:
```

Сперва нам необходимо создать объект браузера, но внутри класса делать этого мы не будем. Сделаем так, чтобы уже настроенный по нашему разумению объект браузера можно было передать нашему классу. Для этого напишем функцию-конструктор `__init__`, в которую мы передадим объект браузера при создании:

```
# Передаём в класс объект браузера
def __init__(self, selenium_driver):
    self.__selenium_driver = selenium_driver
```

В конструкторе создаётся приватная переменная, доступная для всего класса. Далее, при помощи внутренних функций класса можно будет управлять браузером.

Теперь напишем функцию для перехода на страницу видео:

```
# Переходим на страницу с видео
def __go_to_video(self, url):
    self.__selenium_driver.get(url)
```

Поскольку мы работаем с динамическим сайтом, содержимое на нём может подгружаться не сразу, потому нам нужно создать функцию, которая будет искать заданный элемент на странице в течении некоторого времени.

Сперва, для поиска элемента на странице напишем функцию:

```
# Проверяем, существует ли заданный элемент в DOM-дереве страницы
def check_exists_by_css(self, css_selector):
    try:
        self.__selenium_driver.find_element_by_css_selector(css_selector)
    except NoSuchElementException:
        return False
    return True
```

Далее, напишем функцию которая будет искать элемент на странице в течении некоторого времени:

```
# Ищем элемент в течении заданного времени
def __find_and_wait(self, css_selector, wait_time=60):
    # Запускаем цикл, по количеству секунд ожидания
```

```
    for i in range(wait_time):
        # Если элемент найден -- возвращаем его
        if self.check_exists_by_css(css_selector):
            return self.__selenium_driver.find_elements_by_css_selector(css_selector)
        # Иначе -- ждём 1 секунду
        else:
            time.sleep(1)
```

Здесь мы просто запускаем цикл, с количеством итераций равному количеству секунд ожидания. На каждой итерации цикла ищем элемент, далее если он найден – возвращаем его, а противном случае ждём одну секунду и ищем заново.

Но что, если нужные элементы мы нашли, но нам также необходимо найти элементы внутри уже найденных элементов? Для этого напишем похожую функцию, но она будет искать элементы не на странице, а внутри другого элемента, который мы в неё передадим:

```
# Ищем элемент внутри заданного элемента в течении заданного времени
def __find_and_wait_in_el(self, element, css_selector, wait_time=60):
    # Запускаем цикл, по количеству секунд ожидания
    for i in range(wait_time):
        # Если элемент найден -- возвращаем его
        if self.check_exists_by_css(css_selector):
            return element.find_elements_by_css_selector(css_selector)
        # Иначе -- ждём 1 секунду
        else:
            time.sleep(1)
```

Теперь напишем функцию, которая ищет комментарии на странице:

```
# Собираем комментарии со страницы
def __load_comments(self, how_much, sleep=3):
    # Счётчик с количеством комментариев из предыдущей итерации
    current = 0
    # Список с объектами комментариев
    comments = []
    # Скролим до появление счётчика комментариев
    html = self.__find_and_wait("html")[0]
    html.send_keys(Keys.PAGE_DOWN)
    # Ожидаем загрузки
    time.sleep(sleep)
    # Пока количество комментариев в прошлом запросе меньше
    # чем требуется всего -- продолжаем сбор комментариев
    while current < how_much:
        # Скролим в конец страницы для подгрузки новых комментариев
        # При помощи нажатия кнопки END
        html = self.__find_and_wait("html")[0]
        html.send_keys(Keys.END)
        # Ожидаем загрузки комментариев
        time.sleep(sleep)
        # Получаем список комментариев
        comments = self.__find_and_wait("div#contents ytd-comment-thread-renderer")
        # Если количество комментариев из прошлого запроса равно
        # количеству комментариев текущего, значит новых комментариев нет
        if current == len(comments):
            break
        # Определяем количество комментариев собранных после запроса
        current = len(comments)
        print(f"Собрали комментариев: {current}")
    # Возвращаем список с комментариями собранными на странице
    return comments
```

Итак, чтобы загрузить новую порцию комментариев необходимо проскролить страницу вниз и получить все комментарии на странице. Для этого нужно завести две переменные, одна будет получать количество комментариев, собранных в предыдущем цикле (переменная current), вторая в начале цикла `comments` если количество комментариев на текущей итерации цикла равно количеству комментариев, собранных на прошлой итерации, значит комментарии на странице кончились и необходимо прервать цикл.

Далее возвращаем собранный список комментариев. Для лайкинга у нас всё готово. Мы написали методы которые ищут элементы на странице, написали функцию которая собирает заданное число комментариев, теперь напишем саму функцию для лайкинга:

```
def like_comments(self, url, how_much, percent, sleep=3):
    # Переходим на страницу с видеороликом
    self.__go_to_video(url)
    # Получаем заданное количество комментариевы
    comments = self.__load_comments(how_much)
    # Определяем количество комментариев котороре необходимо пролайкать
    count_of_likes = int(how_much / 100 * percent)

    # Получаем случайные комментарии из списка
    random.shuffle(comments)
    comments = comments[:count_of_likes:]

    # Перебираем список комментариев
    for comment in comments:
        try:
            # Получаем кнопку "Лайк" у комментария
            like = self.__find_and_wait_in_el(comment, "button#button yt-icon")
            # Если кнопка найдена
            if like is not None:
                # Проматываем страницу до кнопки
                actions = ActionChains(self.__selenium_driver)
                actions.move_to_element(like[0]).perform()
                # Ставим лайк
                like[0].click()
                print("Поставили лайк..")
            else:
                print("Не удалось поставить лайк")
        except ElementClickInterceptedException as error:
            print(f"Не удалось поставить поставить лайк: {error.msg}")
        # Задержка после действия
        time.sleep(sleep)
```

Данная функция, в сущности собирает все написанные нами функции в одном месте.

Сперва мы переходим на страницу с видеороликом, затем, получаем заданный список комментариев, далее рассчитываем, сколько комментариев из заданного процента нужно пролайкать.

Например, мы задали 15%, в этой строке мы просчитываем сколько будет 15% от заданного количества комментариев:

```
count_of_likes = int(how_much / 100 * percent)
```

Хорошо, выяснили допустим, что нам нужно сделать 30 случайных лайков из 200 собранных комментариев. Соответственно, нам нужно собрать 30 случайных комментариев, для этого перемешаем значение списка с комментариями:

```
# Получаем случайные комментарии из списка
random.shuffle(comments)
```

После этого, все комментарии в списке перемешанны случайным образом, и мы просто можем обрезать (при помощи слайсов) список до 30 комментариев, таким образом мы и получим список из 30 случайных:

```
comments = comments[:count_of_likes:]
```

Теперь осталось только пролайкать заданные комментарии:

```
for comment in comments:
    try:
        # Получаем кнопку "Лайк" у комментария
        like = self.__find_and_wait_in_el(comment, "button#button yt-icon")
        # Если кнопка найдена
        if like is not None:
            # Проматываем страницу до кнопки
            actions = ActionChains(self.__selenium_driver)
            actions.move_to_element(like[0]).perform()
            # Ставим лайк
            like[0].click()
            print("Поставили лайк..")
        else:
            print("Не удалось поставить лайк")
    except ElementClickInterceptedException as error:
        print(f"Не удалось поставить поставить лайк: {error.msg}")
    # Задержка после действия
    time.sleep(sleep)
```

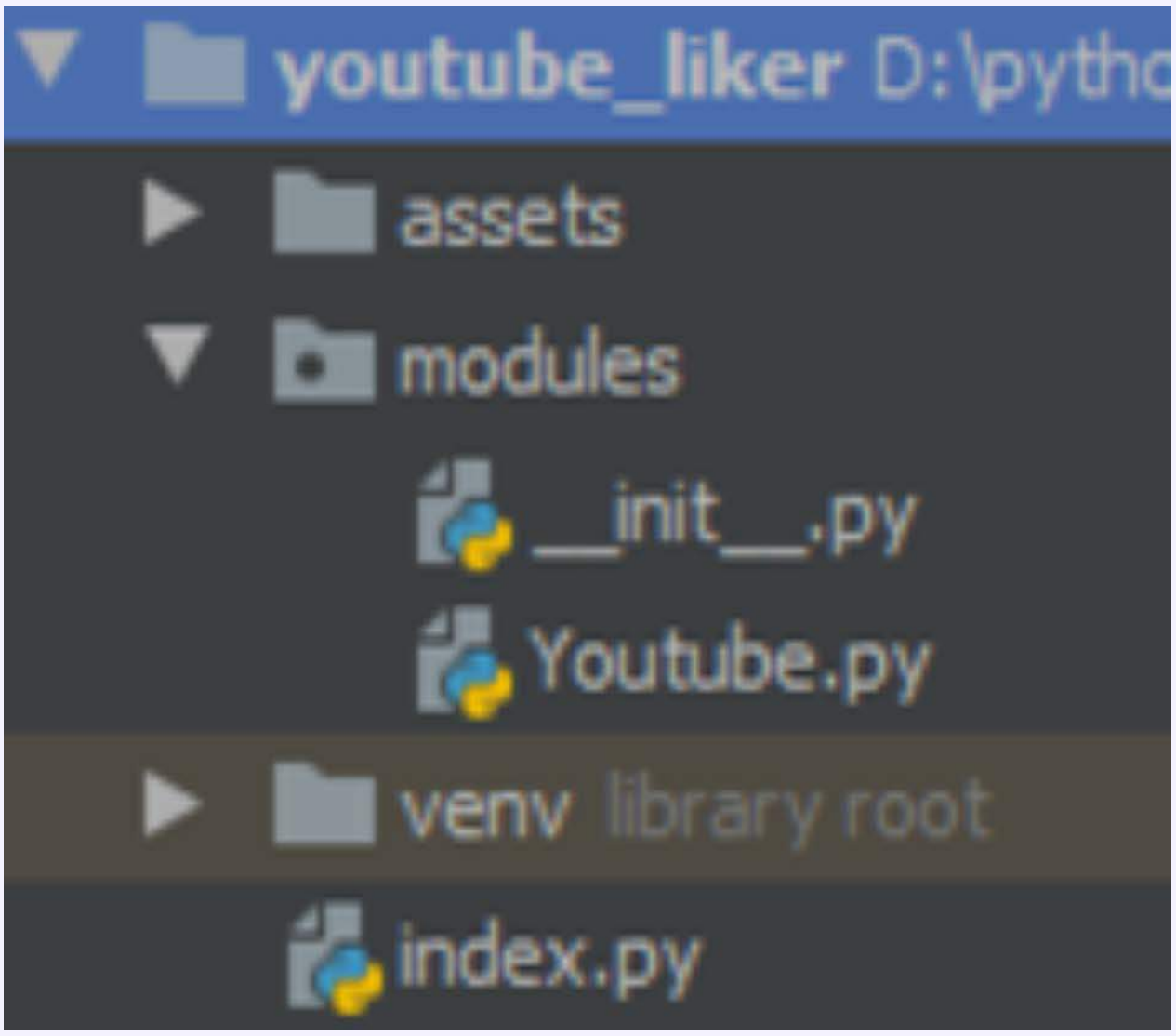
Всё, класс для лайкинга комментариев готов. Приведу его полный листинг:

```
import random
import time

from selenium.common.exceptions import *
from selenium.webdriver import ActionChains
from selenium.webdriver.common.keys import Keys


class Youtube:

    # Передаём в класс объект браузера
    def __init__(self, selenium_driver):
        self.__selenium_driver = selenium_driver

    # Переходим на страницу с видео
    def __go_to_video(self, url):
        self.__selenium_driver.get(url)

    def like_comments(self, url, how_much, percent, sleep=3):
        # Переходим на страницу с видеороликом
        self.__go_to_video(url)
        # Получаем заданное количество комментариевы
        comments = self.__load_comments(how_much)
        # Определяем количество комментариев котороре необходимо пролайкать
        count_of_likes = int(how_much / 100 * percent)

        # Получаем случайные комментарии из списка
        random.shuffle(comments)
        comments = comments[:count_of_likes:]

        # Перебираем список комментариев
        for comment in comments:
```

```
            try:
                # Получаем кнопку "Лайк" у комментария
                like = self.__find_and_wait_in_el(comment, "button#button yt-icon")
                # Если кнопка найдена
                if like is not None:
                    # Проматываем страницу до кнопки
                    actions = ActionChains(self.__selenium_driver)
                    actions.move_to_element(like[0]).perform()
                    # Ставим лайк
                    like[0].click()
                    print("Поставили лайк..")
                else:
                    print("Не удалось поставить лайк")
            except ElementClickInterceptedException as error:
                print(f"Не удалось поставить поставить лайк: {error.msg}")
            # Задержка после действия
            time.sleep(sleep)

    # Собираем комментарии со страницы
    def __load_comments(self, how_much, sleep=3):
        # Счётчик с количеством комментариев из предыдущей итерации
        current = 0
        # Список с объектами комментариев
        comments = []
        # Скролим до появление счётчика комментариев
        html = self.__find_and_wait("html")[0]
        html.send_keys(Keys.PAGE_DOWN)
        # Ожидаем загрузки
        time.sleep(sleep)
        # Пока количество комментариев в прошлом запросе меньше
        # чем требуется всего -- продолжаем сбор комментариев
        while current < how_much:
            # Скролим в конец страницы для подгрузки новых комментариев
            html = self.__find_and_wait("html")[0]
            html.send_keys(Keys.END)
            # Ожидаем загрузки
            time.sleep(sleep)
            # Получаем список комментариев
            comments = self.__find_and_wait("div#contents ytd-comment-thread-renderer")
            # Если количество комментариев из прошлого запроса равно
            # количеству комментариев текущего, значит новых комментариев нет
            if current == len(comments):
                break
            # Определяем количество комментариев собранных после запроса
            current = len(comments)
            print(f"Собрали комментариев: {current}")
        # Возвращаем список с комментариями собранными на странице
        return comments

    # Ищем элемент в течении заданного времени
    def __find_and_wait(self, css_selector, wait_time=60):
        # Запускаем цикл, по количеству секунд ожидания
        for i in range(wait_time):
            # Если элемент найден -- возвращаем его
            if self.check_exists_by_css(css_selector):
                return self.__selenium_driver.find_elements_by_css_selector(css_selector)
            # Иначе -- ждём 1 секунду
            else:
                time.sleep(1)

    # Ищем элемент внутри заданного элемента в течении заданного времени
    def __find_and_wait_in_el(self, element, css_selector, wait_time=60):
        # Запускаем цикл, по количеству секунд ожидания
        for i in range(wait_time):
            # Если элемент найден -- возвращаем его
            if self.check_exists_by_css(css_selector):
                return element.find_elements_by_css_selector(css_selector)
            # Иначе -- ждём 1 секунду
```

```
            else:
                time.sleep(1)

    # Проверяем, существует ли заданный элемент в DOM-дереве страницы
    def check_exists_by_css(self, css_selector):
        try:
            self.__selenium_driver.find_element_by_css_selector(css_selector)
        except NoSuchElementException:
            return False
        return True
```

Осталось запустить полученный код. Для этого в корневой директории проекта создадим модуль `index.py`:



Импортируем наш класс и модуль selenium:

```
from modules.Youtube import Youtube
from selenium import webdriver
```

Теперь в конструкции `try..finally` запустим наш класс. Сперва создадим объект хрома и перейдём на главную страницу Youtube:

```
# Создаём новый объект хрома с установленным расширением
driver = webdriver.Chrome(executable_path='assets/chromedriver.exe')

driver.get("https://www.youtube.com")
```

Далее, создаём список кукисов (я скопировал со своего аккаунта из браузера) и загружаем в селениум:

```
cookies = [
 {"name": "HSID", "domain": ".youtube.com", "value": "Ac7NaiU4kQ"},
    {"name": "SSID", "domain": ".youtube.com", "value": "A-pUmx4otRBx"},
    {"name": "APISID", "domain": ".youtube.com", "value": "Rwud_s1INxNu/AzDfLaT-SYveKy-Ux"},
    {"name": "SAPISID", "domain": ".youtube.com", "value": "cRo2pIfeKLQQbIr6/Ar2f2Y-N6secQ9w5K"},
    {"name": "__Secure-HSID", "domain": ".youtube.com", "value": "Ac7NaicPoae7YU4kQ"},
    {"name": "__Secure-SSID", "domain": ".youtube.com", "value": "A-pUmkb7WTx4otRBx"},
    {"name": "__Secure-APISID", "domain": ".youtube.com", "value": "Rwud_s1INV9eZxNu/AzDT-SYveKy-Ux"},
    {"name": "__Secure-3PAPISID", "domain": ".youtube.com", "value": "cRo2pIfeKLQQbIr6/Ar2f2Y-N6secQ9w5K"},
    {"name": "VISITOR_INFO1_LIVE", "domain": ".youtube.com", "value": "wR5mmPJk"},
    {"name": "PREF", "domain": ".youtube.com", "value": "f6"},
    {"name": "1P_JAR", "domain": ".youtube.com", "value": "2020-05-04-11"},
    {"name": "YSC", "domain": ".youtube.com", "value": "qlR2vWpVaFQ"},
    {"name": "SID", "domain": ".youtube.com", "value": "vwdrKZESHN0F_tec15WSLBfvnjFu56iAQ3YFMlgSFzC4wkQo2rr5Uj4CBBYTQ."},
    {"name": "__Secure-3PSID", "domain": ".youtube.com", "value":
"vwdrKZESHN0F_teuufDjcWY8c15WSLBfvnjFu56iAQ3YFMlgTDdmrNyvzhQU9M_ScD8RYQ."},
    {"name": "LOGIN_INFO", "domain": ".youtube.com", "value":
"AFmmF2swRQIhAPB04o6S68ZsgOCUOzVbGikDP1P7NUNT4DVrRvVji9gdAiAR1f8lbms71Hp_gs4bvuRRkjSItwltfInpzNWDG_m6-
w:QUQ3MjNmem83Mm44dWdIM2doRlVVcVBZWFdaQngtZ0Q3OVFxc0J3VURacVZ4NnJaY3lWZGpQb1FBUm5DTkw5TGhDSDd5OGRTRkx1UVdTdXZhQjZYRGM0RkhZdlBhNHZBeHBWV
HJ2MU1xcGVHY3Q2Y2FKQVBPUzdkU0xvTldadldlZ1dfR19BRlExSWWhvTGt5eWdGemliWElaeDh2WTFaeWQw"},
    {"name": "SIDCC", "domain": ".youtube.com", "value": "AJi4QfG3L2_tyV-6g1BNzJnjV2oZeUUqECQ0VVcSXf4RqD0LGRtHZTKEWLrdGNw"},
]
for cookie in cookies:
    driver.add_cookie(cookie)
```

Далее, создаём объект нашего лайкера и передаём ему объект браузера:

```
youtube_worker = Youtube(driver)
```

Далее запускаем лайкинг комментариев, передавая все заданные параметры для лайкинга:

```
youtube_worker.like_comments("https://www.youtube.com/watch?v=0EHwJcBNxSE&list=PLTlunBkXD9gtUf2_fd-
zpxSCDsn7UR4Zh&index=13", 50, 15)
```

В блоке finally прописываем инструкцию закрытия браузера:

```
driver.close()
```

Полный листинг контроллера:

```
from modules.Youtube import Youtube
from selenium import webdriver

try:

    # Создаём новый объект хрома с установленным расширением
```

```
    driver = webdriver.Chrome(executable_path='assets/chromedriver.exe')

    driver.get("https://www.youtube.com")

    cookies = [
                {"name": "HSID", "domain": ".youtube.com", "value": "Ac7NaicPoae7YU4kQ"},
                {"name": "SSID", "domain": ".youtube.com", "value": "A-pUmkb7WTx4otRBx"},
                {"name": "APISID", "domain": ".youtube.com", "value": "Rwud_s1INV9eZxNu/AzDfLaT-SYveKy-Ux"},
                {"name": "SAPISID", "domain": ".youtube.com", "value": "cRo2pIfeKLQQbIr6/Ar2f2Y-N6secQ9w5K"},
                {"name": "__Secure-HSID", "domain": ".youtube.com", "value": "Ac7NaicPoae7YU4kQ"},
                {"name": "__Secure-SSID", "domain": ".youtube.com", "value": "A-pUmkb7WTx4otRBx"},
                {"name": "__Secure-APISID", "domain": ".youtube.com", "value": "Rwud_s1INV9eZxNu/AzDfLaT-SYveKy-Ux"},
                {"name": "__Secure-3PAPISID", "domain": ".youtube.com", "value": "cRo2pIfeKLQQbIr6/Ar2f2Y-N6secQ9w5K"},
                {"name": "VISITOR_INFO1_LIVE", "domain": ".youtube.com", "value": "wR5ImMmmPJk"},
                {"name": "PREF", "domain": ".youtube.com", "value": "f6"},
                {"name": "1P_JAR", "domain": ".youtube.com", "value": "2020-05-04-11"},
                {"name": "YSC", "domain": ".youtube.com", "value": "qlR2vWpVaFQ"},
                {"name": "SID", "domain": ".youtube.com", "value": "vwdrKZESHN0F_teuufDjcWY8c15WSLBfvnjFu56iAQ3YFMlgSFzC4wkQo2rr5Uj4CBBYTQ."},
                {"name": "__Secure-3PSID", "domain": ".youtube.com", "value":
            "vwdrKZESHN0F_teuufDjcWY8c15WSLBfvnjFu56iAQ3YFMlgTDdmrNyvzhQU9M_ScD8RYQ."},
                {"name": "LOGIN_INFO", "domain": ".youtube.com", "value":
            "AFmmF2swRQIhAPB04o6S68ZsgOCUOzVbGikDP1P7NUNT4DVrRvVji9gdAiAR1f8lbms71Hp_gs4bvuRRkjSItwltfInpzNWDG_m6-
            w:QUQ3MjNmem83Mm44QnkwYTM0dWdIM2doRlVVcVBZWFdaQngtZ0Q3OVFxc0J3VURacVZ4NnJaY3lWZGpQb1FBUm5DTkw5TGhDSDd5OGRTRkx1UVdTdXZhQjZYRGM0RkhZdlBhNHZBeH
            BWVHJ2MU1xcGVHY3Q2Y2FKQVBPUzdkU0xvTldadldlZ1dfR19BRlExSWQ3OGhwbXBNRGZWalRXbWhvTGt5eWdGemliWElaeDh2WTFaeWQw"},
                {"name": "SIDCC", "domain": ".youtube.com", "value": "AJi4QfG3L2_tyV-6g1BNzJnjV2oZeUUqEGlcICQ0VVcSXf4RqL0I3xwKQD0LGRtHZTKEWLrdGNw"},
    ]

        for cookie in cookies:
            driver.add_cookie(cookie)

        youtube_worker = Youtube(driver)
             youtube_worker.like_comments("https://www.youtube.com/watch?v=0EHwJcBNxSE&list=PLTlunBkXD9gtUf2_fd-zpxSCDsn7UR4Zh&index=13", 50, 15)
    finally:
        # Закрываем браузер
        driver.close()
```

Поиск битых ссылок и незанятых доменов на сайтах

Содержание урока

Сегодня будет интересный практикум. Большая часть работы в сфере автоматизации проходит за пределами крупных ресурсов. Поскольку для тех уже есть автоматизация всех цветов и расцветок.

Очень часто приходится работать, хоть и с достаточно крупными, но тем не менее с массовыми веб-сайтами. Этот урок – один из примеров работы с такими сайтами.

Идея программы

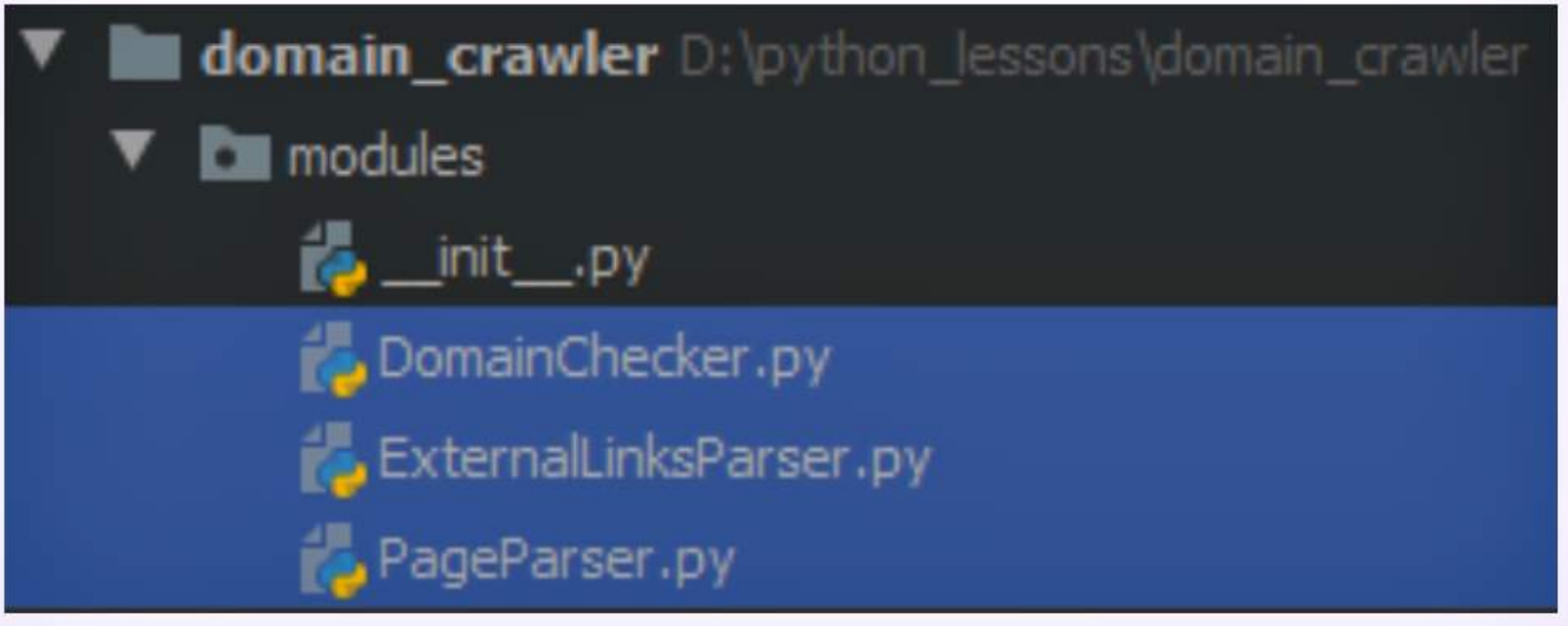
Идея продукта следующая. На сайте https://gogetlinks.net/ находим трастовый сайт, с большим количеством страниц. Собираем все внутренние страницы. Извлекаем из них внешние ссылки. Проверяем домены на возможность регистрации.

Если среди них есть домены с истёкшим сроком регистрации, то мы можем получить его, восстановить его страницы, а вместе с тем и получить беклинки с крупных сайтов. Если ссылки на сайт закупались на крупных площадках, значит над сайтом работали.

Создадим проект, назовём его `domain_crawler`. Установим в проект модули `requests` и `beautiful soup`.

Создадим пакет `modules`, а в нём три модуля:

PageParser – собирает ссылки на страницы конкретного сайта
ExternalLinksParser – на вход принимает ссылки на страницы с сайта, с каждой страницы извлекает внешние ссылки
DomainChecker – проверяет внешние домены на занятость

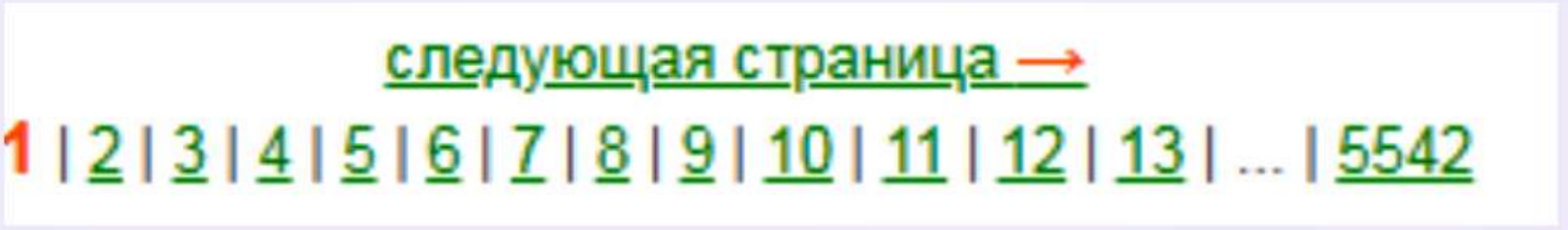


В качестве донора был выбран сайт: http://www.smartphone.ua/

В этом сайте есть раздел новости: http://www.smartphone.ua/news/

Здесь содержатся статьи, в которых могут находиться ссылки.

На странице новостей, в самом низу видно ссылки на страницы пагинации:



Здесь мы видим, что их на данный момент `5542`.

Если мы перейдём на какую-нибудь другую страницу в выдаче, то увидим следующий адрес: http://www.smartphone.ua/news/_page7.html

Получается, что каждая страница из выдачи новостей должна включать в себя цифру новости. Дела простые, вытаскиваем из кода текущее количество страниц и запускаем цикл по их количеству. На каждой итерации цикла парсим все ссылки на новости и собираем их в список.

Пишем парсер

Импортируем модули для работы с сайтом в модуль `PageParser`:

```
import requests
from bs4 import BeautifulSoup as BS4
```

Теперь создадим класс:

```
class PageParser:
```

Так как все запросы будут отправляться к одному и тому же сайту, определим приватную переменную, с адресом сайта:

```
# Определяем базовый адрес сайта
__base_url = "http://www.smartphone.ua"
```

В первую очередь нам нужна функция для получения количества страниц с новостями:

```
def __get_count_of_pages(self):
    # Получаем код страницы с новостями
    page = requests.get(f"{self.__base_url}/news/")
    # Если страница доступна
    if page.status_code == 200:
        # Создаём объект парсера исходного кода страницы
        html_parser = BS4(page.text, "html.parser")
        # Получаем ссылку на последеюю страницу из выдачи
        page_element = html_parser.select("div.pages a.digit")[-1]
        # Получаем номер страницы из ссылки
        count_of_pages = page_element.getText()
        # Возвращаем номер страницы с типом int
        return int(count_of_pages)
    return 0
```

Ссылка на последнюю страницу выдачи:

Соответственно, получаем список ссылок:

```
page_element = html_parser.select("div.pages a.digit")
```

Нам возвращается список и мы получаем у него последний элемент, передавая индекс -1. Индекс -1 всегда возвращает последний элемент из списка:

```
page_element = html_parser.select("div.pages a.digit")[-1]
```

Предположим, у нас есть количество страниц, и мы можем начать парсить их в цикле. Давайте напишем метод, который будет парсить ссылки на новости, с заданной страницы выдачи:

```
def __get_news_from_page(self, num_of_page):
    try:
        # Получаем исходный код страницы из выдачи
        page = requests.get(f"{self.__base_url}/news/_page{num_of_page}.html")
        # Если страница доступна
        if page.status_code == 200:
            # Создаём объект парсера для этой страницы
            html_parser = BS4(page.text, "html.parser")
            # Возвращаем список ссылок со страницы
            return html_parser.select("ul#news_list a.green")
        # В противном случае
        else:
            # Возвращаем пустой список
            return []
    # Если при парсинге возникла ошибка
    except:
        # Возвращаем пустой список
        return []
```

Формируем ссылку вида: http://www.smartphone.ua/news/_page1.html

Подставляя цифру из аргумента функции `num_of_page`:

```
f"{self.__base_url}/news/_page{num_of_page}.html")
```

На странице выдаче, мы получаем ссылки на страницы новостей:

Соответственно, чтобы получить эти элементы мы прописываем селектор:

```
html_parser.select("ul#news_list a.green")
```

Теперь нам нужен метод, который будет собирать ссылки на новости со всех страниц из выдачи:

```
def get_news(self):
    # Получаем список страниц, и прибавляем единицу
    count_of_pages = self.__get_count_of_pages() + 1
    # Определяем список для ссылок на новости
    news_list = []
    # Проходим в списке все страницы выдачи
```

```
    for i in range(1, count_of_pages):
        # У каждой страницы получаем ссылки на новости и добавляем
        # их в список news_list
        [news_list.append(f"{self.__base_url}{news['href']}") for news in self.__get_news_from_page(i)]
    # Возвращаем список news_list
    return news_list
```

Мы получаем число страниц, а затем используем его в функции range. Мы указываем, что итерация начнётся от 1 и закончится содержимым переменной `count_of_pages`. Но когда мы указываем крайнюю границу, то есть до какого числа продолжать итерацию, итерация доходит до предпоследней позиции.

Например, когда мы пишем следующий код:

```
for i in range(1, 10):
```

Он породит только 9 итераций, от 1 до 9 включительно. Поэтому, если мы хотим, чтобы было инициализировано 10 итераций, от одного до десяти, мы должны передать крайнюю границу на 1 больше, как в данной инструкции:

```
count_of_pages = self.__get_count_of_pages() + 1
```

Мы получили количество страниц, и увеличили его на 1. Вы можете также обратить внимание на неизвестную доселе конструкцию:

```
[news_list.append(f"{self.__base_url}{news['href']}") for news in self.__get_news_from_page(i)]
```

Это такая бонусная фича для вас. Данная конструкция является сокращённым синтаксисом написания цикла, и выглядит вот так:

```
[одна_инструкция for item in items if условие]
```

И является эквивалентом такой записи:

```
for item in items:
    if условие:
        одна_инструкция
```

Если инструкция прописывается в одну строку, то это нередко бывает очень удобно.

Ещё есть такой синтаксис:

```
[одна_инструкция for item in items]
```

И он является эквивалентом такой записи:

```
for item in items:
    одна_инструкция
```

В нашем случае, мы в цикле прошли все элементы со ссылками, и добавили их в новый список.

Полный листинг класса:

```
import requests
from bs4 import BeautifulSoup as BS4


class PageParser:
    # Определяем базовый адрес сайта
    __base_url = "http://www.smartphone.ua"

    def __get_count_of_pages(self):
        # Получаем код страницы с новостями
        page = requests.get(f"{self.__base_url}/news/")
        # Если страница доступна
        if page.status_code == 200:
            # Создаём объект парсера исходного кода страницы
            html_parser = BS4(page.text, "html.parser")
            # Получаем ссылку на последеюю страницу из выдачи
            page_element = html_parser.select("div.pages a.digit")[-1]
            # Получаем номер страницы из ссылки
            count_of_pages = page_element.getText()
            # Возвращаем номер страницы с типом int
            return int(count_of_pages)
        return 0

    def __get_news_from_page(self, num_of_page):
        try:
            # Получаем исходный код страницы из выдачи
            page = requests.get(f"{self.__base_url}/news/_page{num_of_page}.html")
            # Если страница доступна
            if page.status_code == 200:
                # Создаём объект парсера для этой страницы
                html_parser = BS4(page.text, "html.parser")
                # Возвращаем список ссылок со страницы
                return html_parser.select("ul#news_list a.green")
            # В противном случае
            else:
                # Возвращаем пустой список
                return []
        # Если при парсинге возникла ошибка
        except:
            # Возвращаем пустой список
            return []

    def get_news(self):
        # Получаем список страниц, и прибавляем единицу
        count_of_pages = self.__get_count_of_pages() + 1
        # Определяем список для ссылок на новости
        news_list = []
        # Проходим в списке все страницы выдачи
        for i in range(1, count_of_pages):
            # У каждой страницы получаем ссылки на новости и добавляем
            # их в список news_list
            [news_list.append(f"{self.__base_url}{news['href']}") for news in self.__get_news_from_page(i)]
        # Возвращаем список news_list
        return news_list
```

Мы научились получать ссылки на новости сайта, теперь нужно написать функционал по получению внешних ссылок из новостей.

Для этого в модуль `ExternalLinksParser` импортируем модули:

```
import requests
from bs4 import BeautifulSoup as BS4
```

Объявим класс:

```
class ExternalLinksParser:
```

Данный класс будет принимать список ссылок на страницы новостей, открывать каждую и извлекать из неё внешние ссылки.

Реализуем конструктор. В него будет передаваться список ссылок:

```
def __init__(self, list_of_news):
    # Инициализируем список ссылок на новости
    self.__list_of_news = list_of_news
```

Теперь нам нужно научить софт понимать, внешняя ссылка или внутренняя:

```
def __is_external_link(self, link):
    # Если ссылка начинается с символа "/"
    # значит это внутренняя ссылка
    if link.startswith("/"):
        return False
    # Если ссылка начинается с символа "#"
    # значит это внутренняя ссылка
    if link.startswith("#"):
        return False
    # Если ссылка содержит имя сайта
    # значит это внутренняя ссылка
    if "smartphone.ua" in link:
        return False
    # В остальных случаях - это внешняя ссылка
    return True
```

Мы передаём ссылку в данную функцию, если она возвращает `True` – значит ссылка внешняя. Если `False` – значит внутренняя, соответственно.

Напишем метод, который извлекает все ссылки со страницы:

```
def __get_links_from_page(self, link):
    # Определяем список для ссылок
    links = []
    try:
        # Отправляем запрос к странице
        page = requests.get(link)
        # Если страница доступна
        if page.status_code == 200:
            # Создаём объект парсера этой страницы
```

```
            html_parser = BS4(page.text, "html.parser")
            # Получаем все элементы со ссылками
            list_of_links = html_parser.select("a")
            # Добавляем ссылка в список inks
            [links.append(link["href"]) for link in list_of_links]
            # Возвращаем список
            return links
        # Иначе, если страница недоступна
        # возвращаем пустой список
        return []
    # Если возникла ошибка
    except:
        # Возвращаем пустой список
        return []
```

Теперь напишем метод, который извлекает только внешние ссылки:

```
def __get_external_links_from_page(self, link):
    # Определяем список для внешних ссылок
    external_links = []
    # Получаем все ссылки со страницы
    links = self.__get_links_from_page(link)
    # Если текущая ссылка внешняя, тогда добавляем её в список external_links
    [external_links.append(link) for link in links if self.__is_external_link(link)]
    # Возвращаем список со внешними ссылками
    return external_links
```

Как раз используем новую сокращённую конструкцию цикла с условием.

```
[external_links.append(link) for link in links if self.__is_external_link(link)]
```

Она эквивалентна:

```
for link in links:
    if self.__is_external_link(link):
        external_links.append(link)
```

Теперь напишем открытый сервисный метод, который извлекает список внешних ссылок из страниц с новостями:

```
def get(self):
    # Определяем список для внешних ссылок
    external_links = []
    # Проходим в цикле каждую ссылку на новость
    for news_link in self.__list_of_news:
        # Извлекаем из каждой новости внешние ссылки
        ex_links = self.__get_external_links_from_page(news_link)
        # Добавляем все ссылки из новости в список external_links
        [external_links.append(ex_link) for ex_link in ex_links]
    # Возвращаем список без дублей
    return list(set(external_links))
```

Полный листинг класса:

```python
import requests
from bs4 import BeautifulSoup as BS4


class ExternalLinksParser:

    def __init__(self, list_of_news):
        # Инициализируем список ссылок на новости
        self.__list_of_news = list_of_news

    def __is_external_link(self, link):
        # Если ссылка начинается с символа "/"
        # значит это внутренняя ссылка
        if link.startswith("/"):
            return False
        # Если ссылка начинается с символа "#"
        # значит это внутренняя ссылка
        if link.startswith("#"):
            return False
        # Если ссылка содержит имя сайта
        # значит это внутренняя ссылка
        if "smartphone.ua" in link:
            return False
        # В остальных случаях - это внешняя ссылка
        return True

    def __get_links_from_page(self, link):
        # Определяем список для ссылок
        links = []
        try:
            # Отправляем запрос к странице
            page = requests.get(link)
            # Если страница доступна
            if page.status_code == 200:
                # Создаём объект парсера этой страницы
                html_parser = BS4(page.text, "html.parser")
                # Получаем все элементы со ссылками
                list_of_links = html_parser.select("a")
                # Добавляем ссылка в список inks
                [links.append(link["href"]) for link in list_of_links]
                # Возвращаем список
                return links
            # Иначе, если страница недоступна
            # возвращаем пустой список
            return []
        # Если возникла ошибка
        except:
            # Возвращаем пустой список
            return []

    def __get_external_links_from_page(self, link):
        # Определяем список для внешних ссылок
        external_links = []
        # Получаем все ссылки со страницы
        links = self.__get_links_from_page(link)
        # Если текущая ссылка внешняя, тогда добавляем её в список external_links
        [external_links.append(link) for link in links if self.__is_external_link(link)]
        # Возвращаем список со внешними ссылками
        return external_links

    def get(self):
        # Определяем список для внешних ссылок
        external_links = []
        # Проходим в цикле каждую ссылку на новость
        for news_link in self.__list_of_news:
            # Извлекаем из каждой новости внешние ссылки
            ex_links = self.__get_external_links_from_page(news_link)
            # Добавляем все ссылки из новости в список external_links
```

```
            [external_links.append(ex_link) for ex_link in ex_links]
        # Возвращаем список без дублей
        return list(set(external_links))
```

Итак, внешние ссылки извлекать научились. Теперь научимся проверять домены на занятость. Для этого в модуль `DomainChecker` импортируем следующий модуль: `import socket`

Объявим класс:

Проверка домена на занятость, это крайне непростой и трудозатратный процесс. Поэтому даже регистраторы доменов, в своих публичных проверках предоставляют не всегда достоверны данные. По-настоящему надёжная проверка на занятость производится только в момент регистрации.

Более-менее пригодный и универсальный способ проверки — это работа с библиотекой `socket`, а точнее с методом `gethostbyname_ex`, у свободных доменов, при попытке получить имя хоста выбрасывается ошибка. Поэтому, передавая домены в этот метод и отлавливая ошибки, можно дать относительно большую вероятность что домен свободен.

Нередко ошибки выбиваются и у зарегистрированных доменов, поэтому после проверки получив список, мы должны повторно проверить домены на занятость уже на сайте регистратора.

Напишем метод, который будет проверять, занят домен или свободен:

```
def __this_domain_is_available(self, link):
    try:
        print(f"Проверяем домен: {link}")
        # Если сокет не выбросил ошибку,
        # значит домен занят
        socket.gethostbyname_ex(link)
    # Если ошибка была выброшена,
    # значит домен свободен
    except:
        return True
    return False
```

Нужно также не забыть, что метод принимает домены вида vk.com без «слешей» и «протоколов».

Поэтому напишем метод, который делает ссылку пригодной для проверки:

```
def __make_clear_domain(self, url):
    # Извлекаем только хост: vk.com,
    # вместо https://vk.com/
    first = url.split("/")[2]
    return first
```

В списке может быть несколько одинаковых доменов. Их нужно «почистить» от дубликатов. Для этого напишем простую функцию:

```
def __make_uniq_domains(self, links):
    # Определяем список для
    # неповторяющихся доменов
    clear_links = []
    # Проходим все ссылки в цикле
    for link in links:
        # Извлекаем из ссылки хост
        clear_links.append(self.__make_clear_domain(link))
    # Возвращаем очищенный
```

```
        # от дублей список ссылок
        return list(set(clear_links))
```

Теперь напишем сервисный метод, который будет производить проверку доменов на занятость:

```
def check(self, links):
    # Определяем список для свободных доменов
    available_domains = []
    # Получаем список уникальных хостов
    links = self.__make_uniq_domains(links)
    # Проходим по списку хостов в цикле
    for link in links:
        # Если хост свободен
        if self.__this_domain_is_available(link):
            print(f"Домен свободен: {link}")
            # Добавляем его в список свободных доменов
            available_domains.append(link)
        else:
            print(f"Домен занят: {link}")
    # Возвращаем список свободных доменов
    return available_domains
```

Полный листинг класса:

```
import socket


class DomainChecker:

    def __this_domain_is_available(self, link):
        try:
            print(f"Проверяем домен: {link}")
            # Если сокет не выбросил ошибку,
            # значит домен занят
            socket.gethostbyname_ex(link)
        # Если ошибка была выброшена,
        # значит домен свободен
        except:
            return True
        return False

    def __make_clear_domain(self, url):
        # Извлекаем только ност: vk.com,
        # вместо https://vk.com/
        first = url.split("/")[2]
        return first

    def __make_uniq_domains(self, links):
        # Определяем список для
        # неповторяющихся доменов
        clear_links = []
        # Проходим все ссылки в цикле
        for link in links:
            # Извлекаем из ссылки хост
            clear_links.append(self.__make_clear_domain(link))
        # Возвращаем очищенный
        # от дублей список ссылок
        return list(set(clear_links))

    def check(self, links):
        # Определяем список для свободных доменов
        available_domains = []
```

```
        # Получаем список уникальных хостов
        links = self.__make_uniq_domains(links)
        # Проходим по списку хостов в цикле
        for link in links:
            # Если хост свободен
            if self.__this_domain_is_available(link):
                print(f"Домен свободен: {link}")
                # Добавляем его в список свободных доменов
                available_domains.append(link)
            else:
                print(f"Домен занят: {link}")
        # Возвращаем список свободных доменов
        return available_domains
```

Осталось написать контроллер, который будет запускать все эти классы. В корневой директории проекта создадим модуль index.py:

Пропишем логику контроллера:

```
    from modules.PageParser import PageParser
from modules.ExternalLinksParser import ExternalLinksParser
from modules.DomainChecker import DomainChecker

# Объект парсера новостей сайта
page_parser_obj = PageParser()
# Список ссылок на новости сайта
news_links = page_parser_obj.get_news()

# Объект парсера внешних ссылок
external_links_obj = ExternalLinksParser(news_links)
# Список внешних ссылок
external_links = external_links_obj.get()

# Объект чекера доменов
domain_checker_obj = DomainChecker()
# Список свободных доменов
free_domains = domain_checker_obj.check(external_links)

# Выводим свободные домены на экран
print(free_domains)
```

Здесь всё просто:

1. Парсим ссылки на новости сайта
2. Парсим внешние ссылки из новостей
3. Чекаем домены из внешних ссылок на занятость
4. Выводим свободные домены на экран

Также отмечу, что на данном сайте около 50 000 новостей, поэтому, чтобы проверить как работает данный скрипт, можно в классе `PageParser`, в методе `get_news` заменить строку:

```
count_of_pages = self.__get_count_of_pages() + 1
```

Например, на:

```
count_of_pages = 11
```

Чтобы не парсить 5000+ страниц из выдачи, а скажем 10, и увидеть как работает парсер.

Рассыльщик email через SMTP-сервер

Содержание урока

Многие знакомы с таким явлением, как формы обратной связи или формы заявок на CPA-сайтах. Все эти заявки при помощи сервера отправляются или в базу данных, или что бывает часто (мои заказчики часто просят такой вариант) производить отправку заявок на почту.

Таким образом, часто требуется реализовать функционал, который при происхождении на сайте некоторого события отправлял об этом информацию на почту заказчика.

Сама по себе отправка сообщений на почту не такое уж тривиальное действо, поскольку существует несколько протоколов для отправки почты, а также существуют сильные спам-фильтры у разных почтовых сервисов.

Но к нашему счастью, существуют библиотеки, которые упростили отправку почты при помощи Python, а также существуют специальные сервера (как платные, так и бесплатные) которые позволяют отправлять сообщения на почту пользователей.

Понятие SMTP-сервера

В данном уроке мы напишем простой скрипт, который при помощи бесплатного SMTP-сервера от google будет рассылать сообщения на почты других пользователей.

Для начала определим, что такое SMTP-сервер. Это такой сервер, который реализует отправку писем, при помощи специального протокола, именуемого как SMTP (Simple Mail Transfer Protocol). Условно говоря, это сервер который берёт на себя сложные технические моменты отправки писем, а тебе предоставляет простой интерфейс.

Есть специальные коммерческие SMTP-сервера, которые за плату предоставляют определённый лимит писем, которые вы можете разослать. Есть и бесплатные или пробные SMTP-сервера.

Для удобства, давайте рассмотрим вариант работы с бесплатным SMTP-сервером от гугла.

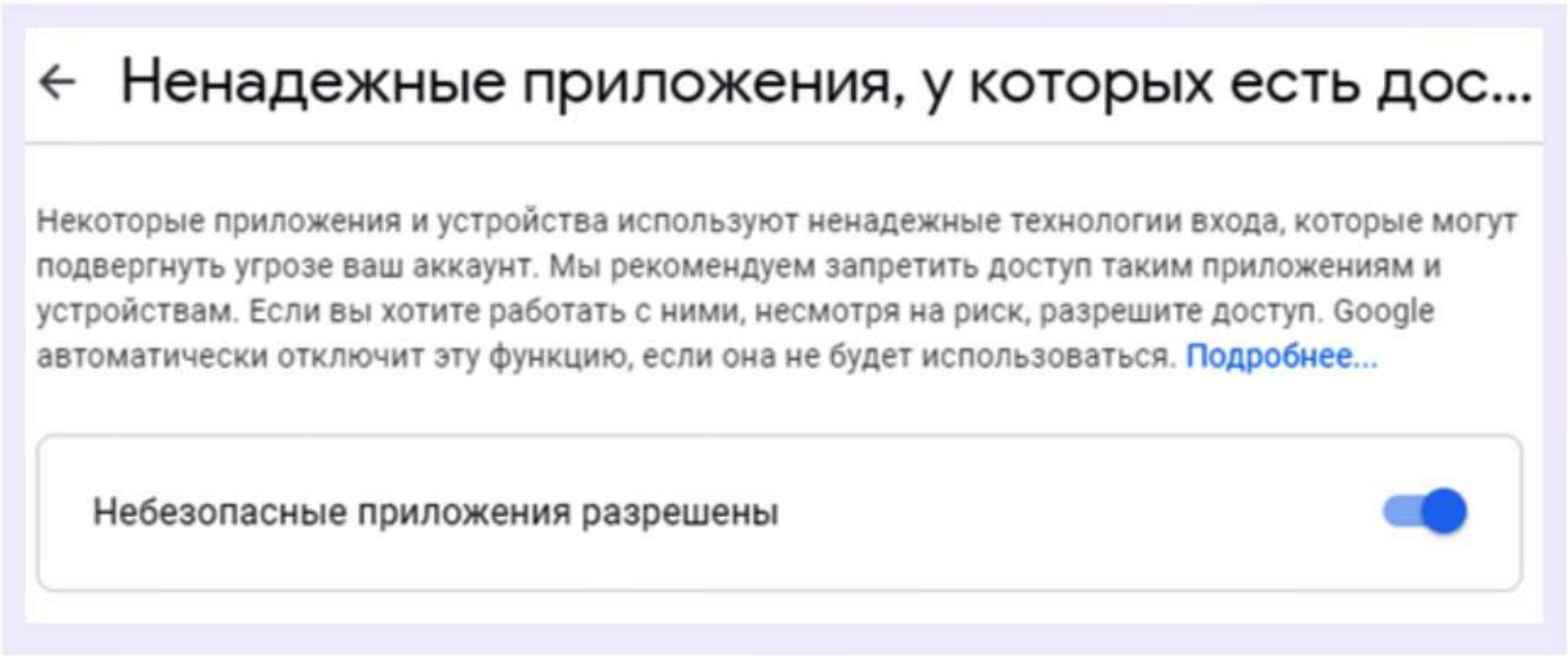
Настройка SMTP от Google

Если у вас есть почтовый ящик гугла, значит у вас автоматически есть и бесплатный SMTP-сервер. Логин и пароль от вашего ящика будут также логином и паролем для SMTP-сервера.

Гугл предоставляет 100 бесплатных писем в сутки для каждого ящика. Но для работы с его сервером, нужно открыть доступ для работы с посторонними приложениями.

Переходите по ссылке: https://myaccount.google.com/u/0/lesssecureapps

Выставляем флажок в такое положение:



Теперь мы можем приступить к написанию программы для рассылки email через SMTP-сервер.

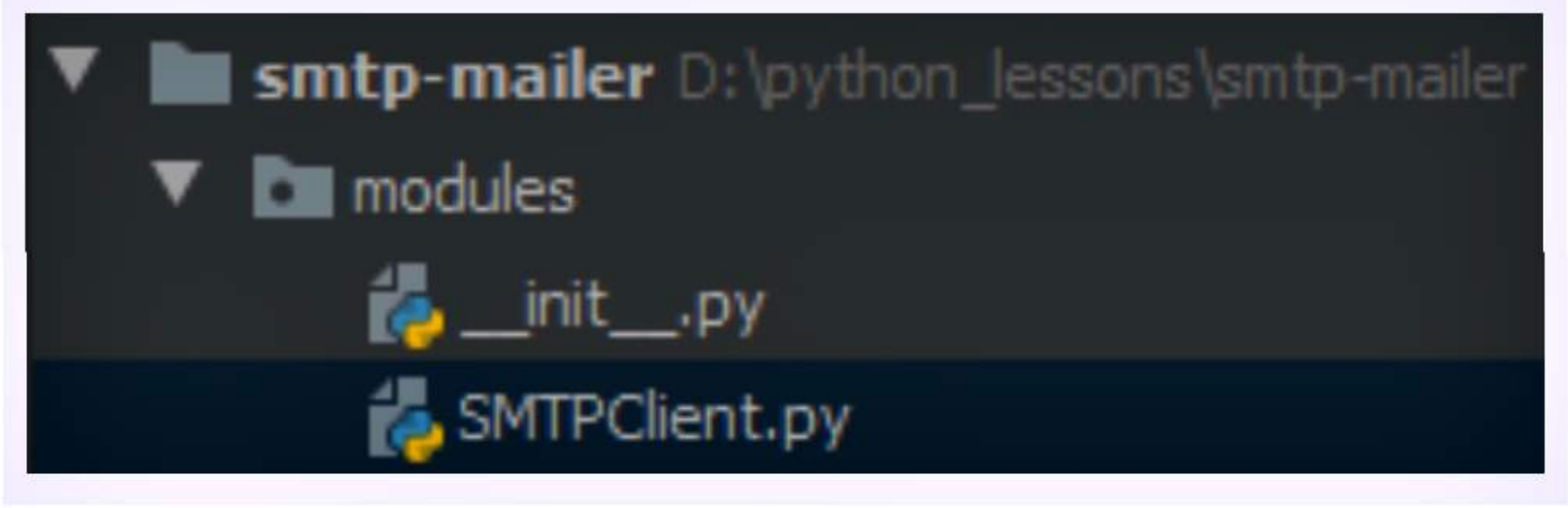
Общий принцип отправки сообщений через SMTP-сервер

Отправка будет производиться через библиотеку smtplib, устанавливать её не нужно, она уже по умолчанию поставляется с интерпретатором.

Алгоритм отправки следующий:

1. Создаём объект SMTP-сервера
2. Создаём объект сообщения
3. Настраиваем объект сообщения
4. Авторизуемся на сервере
5. Отправляем сообщение
6. Закрываем соединение с сервером

Создадим проект, назовём его `smtp-mailer`. Внутри проекта создадим пакет `modules`, в нём модуль `SMTPClient`:



Импортируем в `SMTPClient` следующие модули:

```
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
import smtplib
```

Создадим класс:

```
class SMTPClient:
```

Для подключения к SMTP-серверу необходимо знать его адрес, порт, а также логин и пароль к серверу. У гугла адрес сервера: `smtp.gmail.com`, порт: `587`, а логин и пароль – это логин и пароль от вашего аккаунта.

Соответственно, это основные данные которые необходимо передать серверу. Потому определим их инициализацию через конструктор:

```
def __init__(self, login, password, host="smtp.gmail.com", port=587):
    # Инициализируем логин и пароль от SMTP-сервера
    self.__login, self.__password = login, password
    # Создадим объект SMTP-сервера
    self.__smtp_server = smtplib.SMTP(host, port)
    self.__smtp_server.starttls()
```

Сперва мы должны создать объект, а затем уже в него добавить наше сообщение и некоторые его настройки.

Как я уже говорил, отправка сообщения производится при помощи объекта для сообщения, поэтому нам нужно его заполнить. Напишем функцию для этого:

```
def __make_msg(self, subject, message, email):
    # Инициализируем объект сообщения, которое будет отправлено пользователю
    self.__msg = MIMEMultipart()
    # От кого. Указываем наш емейл от гугла
    self.__msg['From'] = self.__login
    # Кому. Указываем адрес пользователя, которому будет отправлено сообщение
    self.__msg['To'] = email
    # Тема. Заголовок сообщения
    self.__msg['Subject'] = subject
    # Текст сообщения пропущенный в виде специального объекта
    self.__msg.attach(MIMEText(message, 'plain'))
```

В данную функцию мы передаём заголовок и текст сообщение, остальные составляющие были инициализированы до этого.

Отправка сообщения

Сообщение готово, соединение с сервером установлено осталось только его отправить, напишем для этого функцию-контроллер, которая будет отправлять сообщение конкретному пользователю:

```
def __send_message(self, subject, message, email):
    # Создаём сообщение для отправки
    self.__make_msg(subject, message, email)
    # Авторизуемся на сервере
    self.__smtp_server.login(self.__login, self.__password)
    # Отправляем сообщение
```

```
        self.__smtp_server.sendmail(self.__msg['From'], self.__msg['To'], self.__msg.as_string())
        print(f"Отправили сообщение по адресу: {email}")
```

В функцию мы передаём заголовок, текст сообщения и емейл-адрес пользователя. Далее мы просто вызываем уже написанные нами функции для подготовки сообщения к отправке.

Осталось написать итоговую функцию-контроллер, которая будет отправлять сообщения целому списку пользователей:

```
def send_messages(self, subject, message, emails):
    try:
        for email in emails:
            self.__send_message(subject, message, email)
    finally:
        # Закрываем соединение с сервером
        self.__smtp_server.quit()
```

Здесь мы передаём список с емейлами, и в цикле отправляем сообщение на каждый из них.

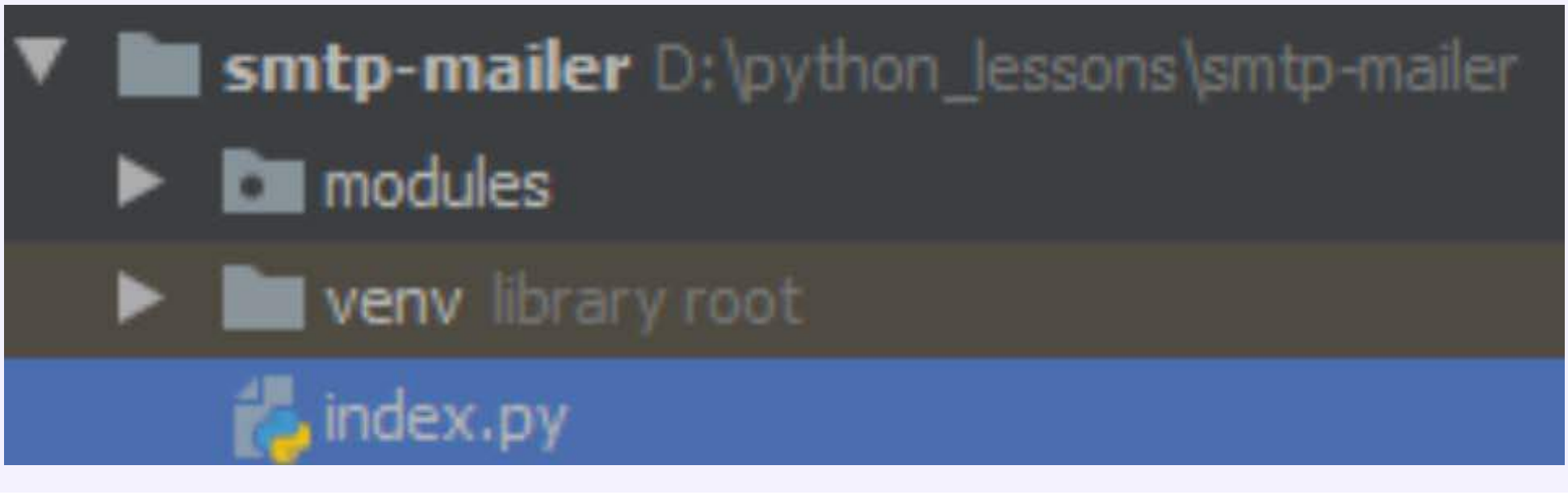
Функцию отправки запускаем в конструкции `try..finally`, чтобы в случае ошибки можно было гарантированно закрыть соединение с SMTP-сервером.

Полный листинг класса:

```
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
import smtplib


class SMTPClient:

    def __init__(self, login, password, host="smtp.gmail.com", port=587):
        # Инициализируем логин и пароль от SMTP-сервера
        self.__login, self.__password = login, password
        # Создадим объект SMTP-сервера
        self.__smtp_server = smtplib.SMTP(host, port)
        self.__smtp_server.starttls()

    def __make_msg(self, subject, message, email):
        # Инициализируем объект сообщения, которое будет отправлено пользователю
        self.__msg = MIMEMultipart()
        # От кого. Указываем наш емейл от гугла
        self.__msg['From'] = self.__login
        # Кому. Указываем адрес пользователя, которому будет отправлено сообщение
        self.__msg['To'] = email
        # Тема. Заголовок сообщения
        self.__msg['Subject'] = subject
        # Текст сообщения пропущенный в виде специального объекта
        self.__msg.attach(MIMEText(message, 'plain'))

    def __send_message(self, subject, message, email):
        # Создаём сообщение для отправки
        self.__make_msg(subject, message, email)
        # Авторизуемся на сервере
        self.__smtp_server.login(self.__login, self.__password)
        # Отправляем сообщение
        self.__smtp_server.sendmail(self.__msg['From'], self.__msg['To'], self.__msg.as_string())
        print(f"Отправили сообщение по адресу: {email}")

    def send_messages(self, subject, message, emails):
        try:
            for email in emails:
                self.__send_message(subject, message, email)
```

```
        finally:
            # Закрываем соединение с сервером
            self.__smtp_server.quit()
```

Теперь напишем контроллер, который вызовет весь наш код. Создадим в корневой директории проекта модуль `index.py`:



Содержание контроллера:

```
from modules.SMTPClient import SMTPClient
# Список емейлов для рассылки
emails = ["mail1@rambler.ru", "mail2@rambler.ru", "mail3@lenta.ru"]
# Создаём объект клиента
SMTPClient = SMTPClient("mymail@gmail.com", "mypass")
# Отправляем сообщение
SMTPClient.send_messages("Заголовок сообщения", "Текст сообщения", emails)
```

Теперь отправка сообщения на разные емейл-адреса сводится к 4 строкам кода.

Бот ВК для автоматического слива трафика через ЛС

Содержание урока

Идея программы

В данном уроке мы напишем программу, которая будет автоматически отвечать на некоторые сообщения, направленные к сообществу ВК. Проще говоря, напишем простого чат-бота для сообщества ВК.

Для написания нашего бота используем библиотеку `vk_api`. Документация: https://vk-api.readthedocs.io/en/latest/

Логика чат бота будет простая. Мы создаём словарь, в котором каждому ключу соответствует определённое сообщение. Если ключ из словаря присутствует в сообщении – мы отправляем его значение.

Например:

```
msg_dict = {
    "Привет": "Здравствуй, как дела?",
    "Сколько стоит": "500 рублей",
    "цена": "500 рублей"
}
```

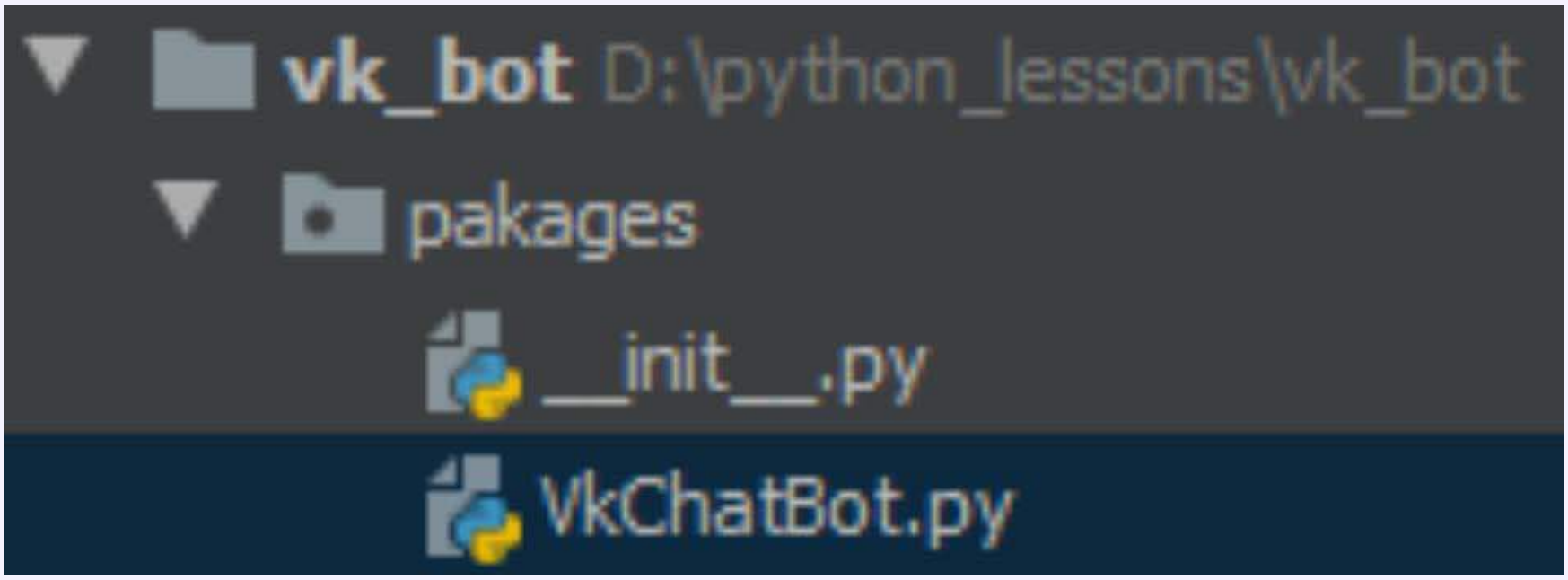
Если в сообщении пользователя будет найдено слово «Привет», бот ответит ему сообщением «Здравствуй, как дела?» и т.д.

Создаём бота

Создадим проект, назовём его `vk_bot`. Установим в него библиотеку `vk_api`:

```
pip install vk_api
```

Создадим пакет, назовём его `pakages`. В нём создадим модуль под названием `VkChatBot`:



Импортируем в него модули:

```
import time
import vk_api
```

Создадим класс:

```
class VkChatBot:
```

Создадим переменную, в которой будет храниться объект с авторизованным пользователем, через которую мы будем работать с АПИ:

```
# Поле для хранения объекта VK-клиента
__vk_client = None
```

Определяем конструктор. При создании объекта мы будем передавать объекту чат-бота токен сообщества, от имени которого будет вестись общение и словарь с сообщениями.

Инструкция по получению токена сообщества

Конструктор:

```
# Определяем словарь с сообщениями
def __init__(self, token, message_dict):
```

```
    # Инициализируем переменную со словарём сообщений
    self.__messages_dict = message_dict
    # Инициализируем переменную со статусом авторизации
    # Если авторизация успешна передаём в переменную со
    # статусом True, иначе - False
    self.auth_status = self.__auth(token)
```

Пишем функцию для авторизации и создания объекта АПИ:

```
# Авторизуемся при помощи токена
def __auth(self, token):
    try:
        self.__vk_client = vk_api.VkApi(token=token)
    except vk_api.AuthError as error_msg:
        print(f"Авторизация не удалась: {error_msg}")
        return False
    return True
```

В неё мы передаём токен, она создаёт объект VkApi и если он был создан без ошибок, то объект записывается в переменную для клиента `self.__vk_client`, и после этого функция возвращает `True`, иначе – возвращает `False`.

Когда мы получаем сообщение от пользователя, нам нужно на него как-то ответить. Для этого воспользуемся методом `messages.send`

Документация: https://vk.com/dev/messages.send

Напишем функцию для данного метода:

```
# Отправляем сообщение
def __write_msg(self, peer_id, message):
    self.__vk_client.method("messages.send",
        {
            "peer_id": peer_id,
            "message": message,
            "random_id": peer_id + time.time()
        }
    )
```

Данная функция должна передавать на сервер минимум 3 параметра: id сообщения на которое мы отвечаем, текст нашего ответного сообщения и параметр случайного ответа, который должен быть уникален для нашего сообщения, чтобы его потом можно было легче идентифицировать (для этого я просто складываю ID сообщения и текущее время, выраженное в секундах, прошедших с 1 января 1970 года).

Мы авторизовались в ВК, написали метод, который умеет отправлять сообщения в ответ. Осталось написать метод, который будет «слушать» пришли ли сообществу новые сообщения.

Для этого нам нужно непрерывно получать все новые сообщения для нашего сообщества, с помощью функции `messages.getConversations`

Документация: https://vk.com/dev/messages.getConversations

Для этого создадим функцию:

```
# Получить сообщения от сервера
def listen(self):
    # Запускаем вечный цикл
    while True:
        # Получаем список сообщений
        conversations = self.__vk_client.method("messages.getConversations", {"offset": 0, "count": 100, "filter": "unread"})
        # Если сообщения есть
        if conversations["count"] > 0:
            # Получаем ID последнего сообщения
            peer_id = conversations["items"][0]["last_message"]["from_id"]
            # Получаем текст последнего сообщения
            msg = conversations["items"][0]["last_message"]["text"]
            # Проверяем, есть ли в сообщении пользователя
```

```
            # слова состветствующие нашему словарю
            for key in self.__messages_dict:
                # Если есть
                if key.lower() in msg.lower():
                    # Отправляем соответствующий ответ на заданное сообщение
                    self.__write_msg(peer_id, self.__messages_dict[key])
        # Ожидаем секунду
        time.sleep(1)
```

Что делает данная функция. Мы создаём вечный цикл, который непрерывно раз в секунду отправляет запросы на сервер. Каждый запрос – это получение списка всех переписок с новыми сообщениями. Если есть хотя бы одна переписка с новым сообщением – получаем это сообщение.

Проходимся циклом по нашему словарю с сообщениями и смотрим, есть ли в сообщении пользователя слова или словосочетания, которые соответствуют нашему шаблону. Важно, приводим все ключи нашего словаря и сообщение пользователя в нижний регистр, чтобы не вышло так, что пользователь что-то написал со строчной буквы, а в нашем словаре оно с заглавной и из-за этого не вышло ему ответить.

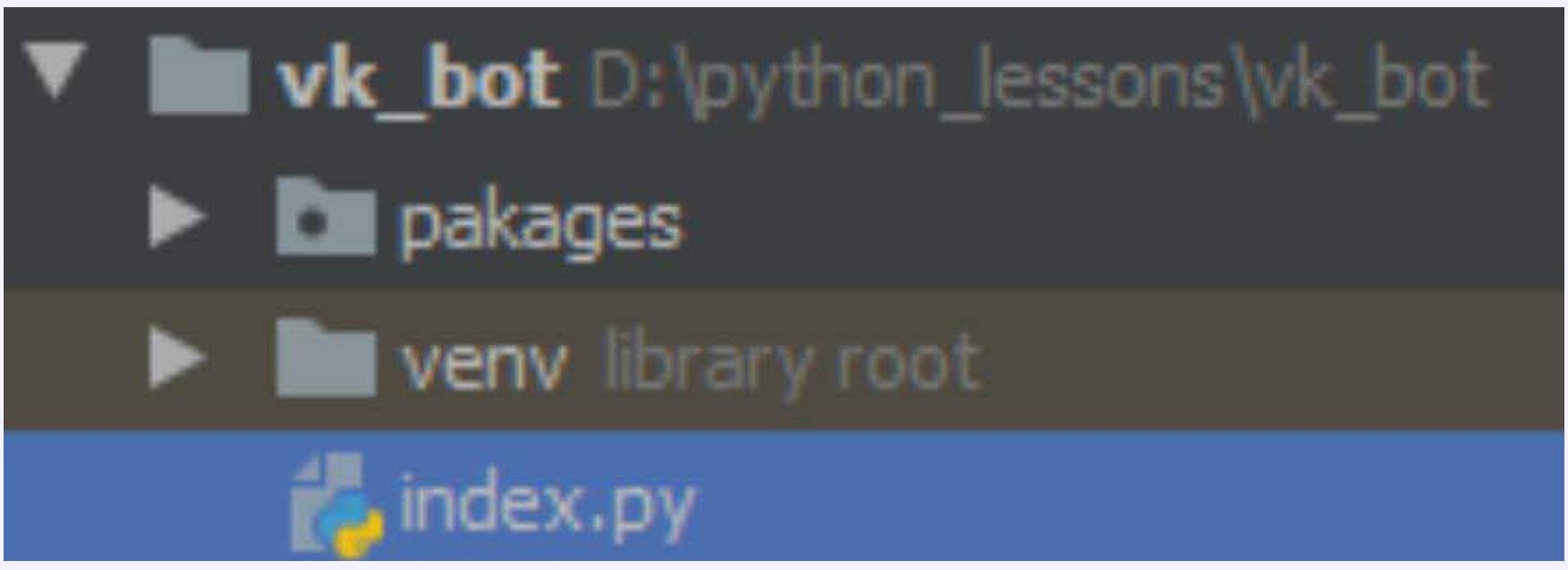
Если есть соответствия – отправляем сообщение в ответ. Полный листинг класса:

```
import time
import vk_api


class VkChatBot:
    # Поле для хранения объекта VK-клиента
    __vk_client = None

    # Определяем словарь с сообщениями
    def __init__(self, token, message_dict):
        # Инициализируем переменную со словарём сообщений
        self.__messages_dict = message_dict
        # Инициализируем переменную со статусом авторизации
        # Если авторизация успешна передаём в переменную со
        # статусом True, иначе - False
        self.auth_status = self.__auth(token)

    # Авторизуемся при помощи токена
    def __auth(self, token):
        try:
            self.__vk_client = vk_api.VkApi(token=token)
        except vk_api.AuthError as error_msg:
            print(f"Авторизация не удалась: {error_msg}")
            return False
        return True

    # Отправляем сообщение
    def __write_msg(self, peer_id, message):
        self.__vk_client.method("messages.send",
            {
                "peer_id": peer_id,
                "message": message,
                "random_id": peer_id + time.time()
            }
        )

    # Получить сообщения от сервера
    def listen(self):
        # Запускаем вечный цикл
        while True:
            # Получаем список сообщений
            conversations = self.__vk_client.method("messages.getConversations", {"offset": 0, "count": 100, "filter": "unread"})
            # Если сообщения есть
            if conversations["count"] > 0:
                # Получаем ID последнего сообщения
                peer_id = conversations["items"][0]["last_message"]["from_id"]
                # Получаем текст последнего сообщения
                msg = conversations["items"][0]["last_message"]["text"]
                # Проверяем, есть ли в сообщении пользователя
                # слова состветствующие нашему словарю
                for key in self.__messages_dict:
                    # Если есть
```

```
                if key.lower() in msg.lower():
                    # Отправляем соответствующий ответ на заданное сообщение
                    self.__write_msg(peer_id, self.__messages_dict[key])
        # Ожидаем секунду
        time.sleep(1)
```

Осталось проверить как работает наш класс. Для этого сперва необходимо создать сообщество в ВК и получить для него API-ключ (ссылку на инструкцию по получению ключа я уже указывал).

Теперь напишем контроллер, который и будет вызывать весь функционал. Создадим в корневой директории проекта модуль `index.py`:



Импортируем в модуль наш класс чат-бота:

```
from pakages.VkChatBot import VkChatBot
```

Создадим словарь с сообщениями для ответа:

```
# Создаём словарь с сообщениями
msg_dict = {
    "Привет": "Здравствуй, как дела?",
    "Сколько стоит": "500 рублей",
    "цена": "500 рублей"
}
```

Создадим переменную с ключом сообщества:

```
# Получаем токен сообщества
token = "тут ключ сообщества"
```

Создадим объект нашего чат-бота:

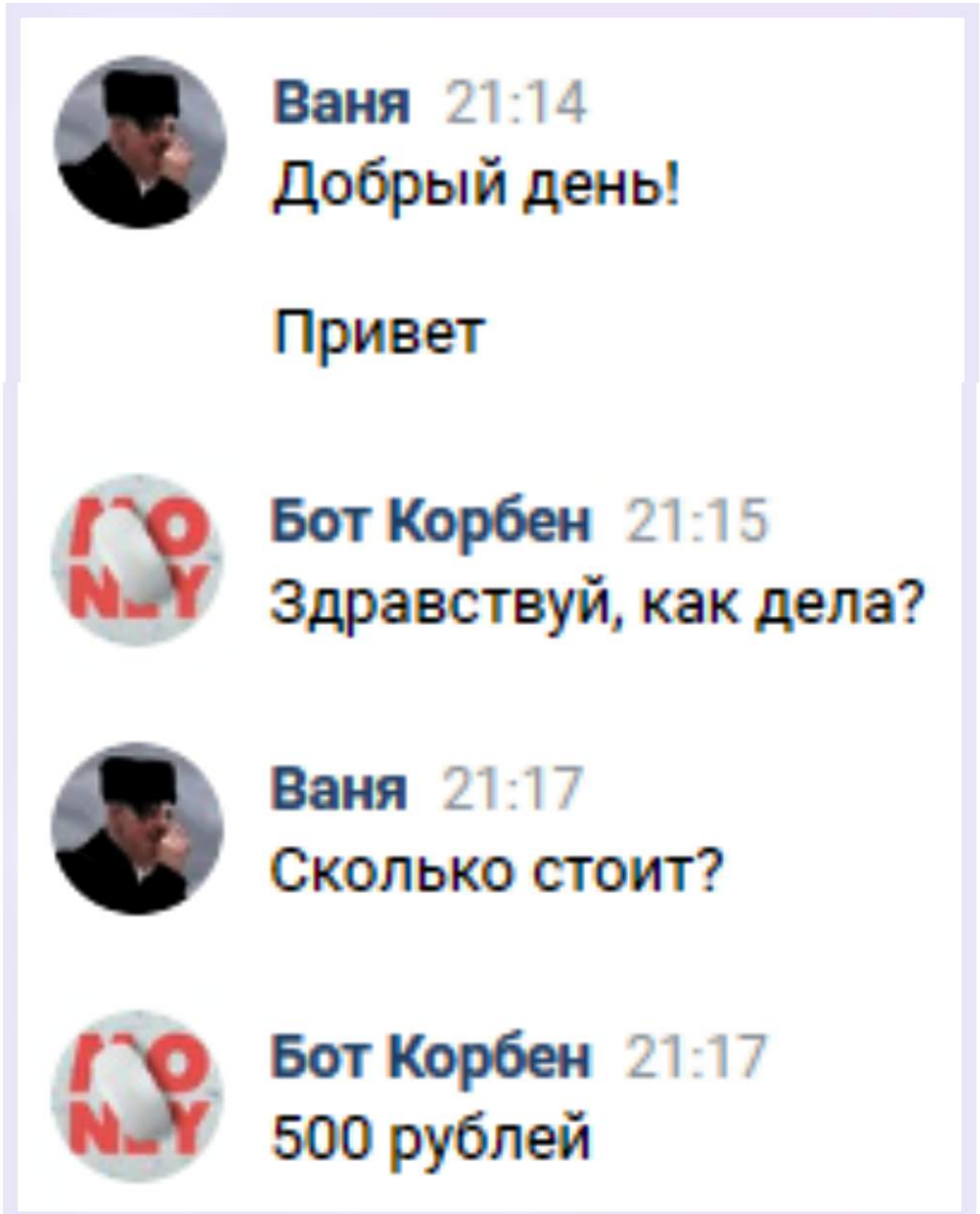
```
# Создаём объект чат-бота
vk_chat_bot = VkChatBot(token, msg_dict)
```

Проверяем, авторизация прошла успешно?

```
# Если авторизация прошла
if vk_chat_bot.auth_status:
    # Подклчаемся к серверу и отвечаем на сообщения
    vk_chat_bot.listen()
```

Если да – запускаем бота.

Осталось проверить его работу. Переходим в сообщество, которое вы подключили к боту через токен. И пишем ему сообщения:



Как видите, если ключ нашего словаря присутствует в сообщении пользователя, то ему отправляется сообщение, соответствующее этому ключу.

Софт для автоматического сбора и анализа пабликов ВК под рекламу

Содержание урока

Идея программы и формула

Поработаем немного со сферой SMM, а точнее, с сообществами ВК. Все прекрасно знают, что в сообществах ВК нередко размещают рекламу на платной основе. Успешность размещения зависит в основном от 3 факторов:

1. Качество рекламного поста
2. Удачно подобранная аудитория
3. Выгодное размещение

Вот над третьим пунктом сегодня мы и поработаем. Выгодное размещение, в нашем случае, сводится к максимально низкой себестоимости поста.

Однако, сама по себе стоимость поста – находится в абсолютном выражении и как таковая ни о чём не говорит.

Пост за 100 000 может дать больше профита, чем пост за 5 000, а может и наоборот. Поэтому специалисты давно измеряют себестоимость поста по критерию «CPM» - `Cost Per Mille`, проще говоря, по стоимости за 1000 показов.

Если пост стоит 500 рублей, а охват равен 5000 пользователей, значит стоимость 1000 охвата составляет (500 / (5000 / 1000)) = 100 рублей. Таким образом, получается более точно определить, в каком сообществе закупаться выгоднее. Разумеется, в том, что имеет наименьший показатель «CPM».

Я предлагаю пойти чуть дальше, и рассчитывать не «CPM», а «CPMT» - `Cost Per Mille Targeted` (стоимость 1000 показов целевая). То есть, мы будем считать не стоимость 1000 охваченных пользователей, а стоимость 1000 целевых охваченных пользователей.

Например, наша целевая аудитория – это мужчины 18-24 лет, из Москвы. Соответственно, более корректно рассчитывать именно по этому показателю, поскольку остальные пользователи являются для нас «побочными».

Таким образом, для расчёта «CPMT» мы будем делить на стоимость размещения не общий показатель охвата, а только целевой. Благодаря этому, мы узнаем сколько нам будет стоить охват нашей целевой аудитории.

Если пост стоит 500 рублей, а охват равен 5000 пользователей (но целевых при этом всего 2000 человек), значит стоимость 1000 охвата составляет (500 / (2000 / 1000)) = 250 рублей.

Данный показатель всегда будет больше чем простой CPM, но при этом он будет более корректным.

Пишем скрипт

Определим алгоритм нашего скрипта:

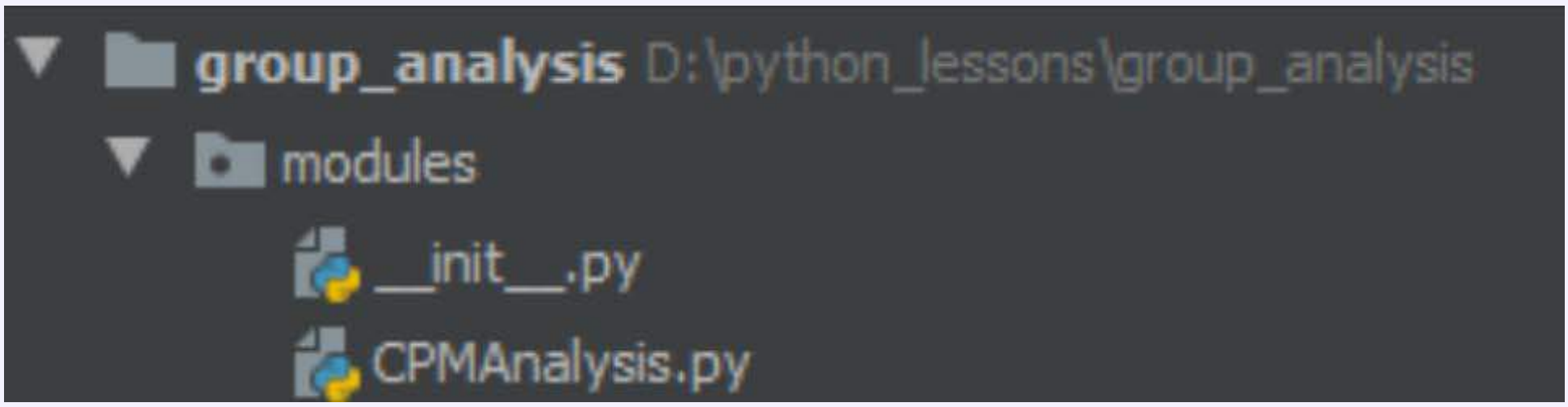
1. Получаем статистику сообщества за прошлый месяц
2. Получаем охват по определённой возрастной категории
3. Получаем % пользователей из целевого города
4. Получаем % пользователей целевого пола
5. Умножаем охват по целевому возрасту на структуру по полу и ГЕО (получаем целевой охват)
6. Определяем стоимость за 1000 целевого охвата

Создадим проект, назовём его `group_analysis`. Установим в него библиотеку `vk_api`:

```
pip install vk-api
```

Документация: https://vk-api.readthedocs.io/en/latest/

Создадим пакет `modules`, а в нём модуль `CPMAnalysis`:



Импортируем в него модуль `vk_api`:

```
import vk_api
```

Создадим класс:

```
class CPMAnalysis:
```

Для получения статистики группы мы будем использовать метод: https://vk.com/dev/stats.get

Данный метод возвращает статистику группы за определённый интервал, например, за день, неделю, месяц и т.д. Также можно получить несколько интервалов, например, n-дней, n-недель, n-месяцев.

Для работы нам обязательно нужен токен для авторизации, а также `id` сообщества, статистику которого мы должны получить в первую очеред. Поэтому создадим соответствующий конструктор:

```
def __init__(self, token, group_id):
    # Авторизуемся в ВК
    vk_session = vk_api.VkApi(token=token)
    # Получаем статистику за предыдущий месяц
    self.__group_stat = vk_session.method("stats.get", {
        "group_id": group_id,
        "interval": "month",
        "intervals_count": 2,  # Получаем за 2 месяца
    })[1]  # Возвращаем статистику второго месяца
```

Здесь мы при помощи токена авторизуемся в ВК, а затем отправляем запрос к методу stats.get, мы отправляем запрос на получение статистики заданной группы, за месяц, и хотим получить последние 2 месяца – текущий и предыдущий, чтобы анализировать статистику по прошлому, уже завершённому периоду.

И весь объект статистики помещаем в переменную `__group_stat`.

Отлично, статистику мы получили, теперь нужно получить охват по тому или иному возрастному периоду за один день:

```python
def __reach_of_age_per_day(self, age_group):
    # Получаем средний дневной охват за прошлый месяц по возрвсту
    return self.__group_stat["reach"]["age"][age_group]["count"] / 30
```

Из документации я взял структуру ответа, в ней нашёл элемент содержащий охват по возрасту. Выглядит он в виде списка со словарями по возрастам:

```
[
  {'value': '12-18', 'count': 109069},
  {'value': '18-21', 'count': 133722},
  {'value': '21-24', 'count': 116057},
  {'value': '24-27', 'count': 109933},
  {'value': '27-30', 'count': 118993},
  {'value': '30-35', 'count': 223775},
  {'value': '35-45', 'count': 218469},
  {'value': '45-100', 'count': 65730}
],
```

Соответственно, для получения сегмента 21-24 нужно обратиться ко 2 элементу списка. Далее, во вложенном в нём словаре получить значение элемента `count`. А если нужен сегмент 45-100, то выбираем элемент списка номер 7.

Но так как данные приходят за месяц, нужно это значение поделить на 30, чтобы вышло среднедневное.

Хорошо, охват по возрасту мы получили. Теперь нужно узнать, какой процент пользователей в структуре охвата из заданного города:

```python
def __city_structure(self, id_of_city):
    # Охват по всем городам
    reach = 0
    # Охват заданного города
    reach_of_city = 0
    # Перебираем охваты всех городов
    for city in self.__group_stat["reach"]["cities"]:
        # Получаем охват заданного города
        if city["value"] == id_of_city:
            reach_of_city = city["count"]
        # Суммируем охват каждого города в общий счётчик
        reach += city["count"]
    # Определяем сколько процентов от общего охвата в заданном городе
    return reach_of_city / reach * 100
```

Здесь мы перебираем охват всех городов и суммируем его. Далее, определяем охват заданного города и высчитываем, сколько процентов от общего охвата он составляет.

В качестве параметра мы передаём `id` города. Получить `id` города можно по этой ссылке: https://vk.com/dev/database.getCities

В конце страницы можно увидеть список ID крупных городов РФ (можно и других стран):

```
"response": {
   "count": 18,
   "items": [{
      "id": 2,
      "title": "Saint Petersburg",
      "important": 1
   }, {
      "id": 1,
      "title": "Moscow",
      "important": 1
   }, {
      "id": 10,
      "title": "Volgograd"
   }, {
```


Также делаем с половой структурой:

```python
def __sex_structure(self, sex):
    # Общий охват обоих полов
    count = self.__group_stat["reach"]["sex"][0]["count"] + self.__group_stat["reach"]["sex"][1]["count"]
    # Определяем сколько процентов от общего охвата у заданного пола
    return self.__group_stat["reach"]["sex"][sex]["count"] / count * 100
```

Определяем охват у двух полов, а затем определяем процент вхождения в общий охват, охват того или иного пола. При передаче параметра нужно передать пол в виде 0 (женский) или 1 (мужской).

Теперь нам нужно определить непосредственно целевой охват. Это делается следующим образом: мы получаем охват по возрасту, затем умножаем его на процент заданного нами города, а полученное значение умножаем на процент заданного пола.

Таким образом, мы получаем примерное количество охвата пользователей заданного возраста, пола и города.

```
def get_target_coverage(self, age_group, city, sex):
    # Определяем общий дневной охват пользователей по возрвсту
    reach = self.__reach_of_age_per_day(age_group)
    # Определяем сколько из них из заданного города
    reach = reach / 100 * self.__city_structure(city)
    # Определяем сколько из них заданного пола
    reach = reach / 100 * self.__sex_structure(sex)
    # Возвращам "целевой" охват по полу, возрасту и городу
    return reach
```

Теперь у нас есть всё, чтобы определить стоимость 1000 целевого охвата:

```
def get_target_cost(self, age_group, city, sex, cpm):
    # Определяем цену за 1000 "целевых просмотров"
    return cpm / (self.get_target_coverage(age_group, city, sex) / 1000)
```

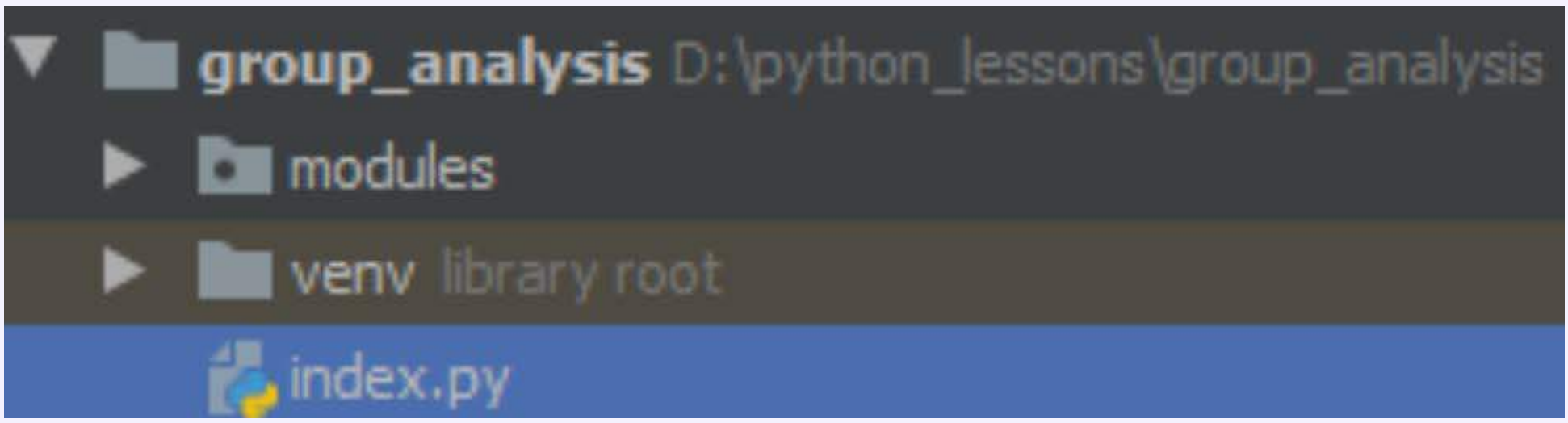
Здесь мы просто делим стоимость на охват, в свою очередь поделенный на 1000, таким образом определяя стоимость 1000 целевого охвата.

Полный листинг класса:

```
import vk_api


class CPMAnalysis:
    def __init__(self, token, group_id):
        # Авторизуемся в ВК
        vk_session = vk_api.VkApi(token=token)
        # Получаем статистику за предыдущий месяц
        self.__group_stat = vk_session.method("stats.get", {
            "group_id": group_id,
            "interval": "month",
            "intervals_count": 2,  # Получаем за 2 месяца
        })[1]  # Возвращаем статистику второго месяца

    def __reach_of_age_per_day(self, age_group):
        # Получаем средний дневной охват за прошлый месяц по возрвсту
        return self.__group_stat["reach"]["age"][age_group]["count"] / 30

    def __city_structure(self, id_of_city):
        # Охват по всем городам
        reach = 0
        # Охват заданного города
        reach_of_city = 0
        # Перебираем охваты всех городов
        for city in self.__group_stat["reach"]["cities"]:
            # Получаем охват заданного города
```

```
            if city["value"] == id_of_city:
                reach_of_city = city["count"]
            # Суммируем охват каждого города в общий счётчик
            reach += city["count"]
        # Определяем сколько процентов от общего охвата в заданном городе
        return reach_of_city / reach * 100

    def __sex_structure(self, sex):
        # Общий охват обоих полов
        count = self.__group_stat["reach"]["sex"][0]["count"] + self.__group_stat["reach"]["sex"][1]["count"]
        # Определяем сколько процентов от общего охвата у заданного пола
        return self.__group_stat["reach"]["sex"][sex]["count"] / count * 100

    def get_target_coverage(self, age_group, city, sex):
        # Определяем общий дневной охват пользователей по возрвсту
        reach = self.__reach_of_age_per_day(age_group)
        # Определяем сколько из них из заданного города
        reach = reach / 100 * self.__city_structure(city)
        # Определяем сколько из них заданного пола
        reach = reach / 100 * self.__sex_structure(sex)
        # Возвращам "целевой" охват по полу, возрасту и городу
        return reach

    def get_target_cost(self, age_group, city, sex, cpm):
        # Определяем цену за 1000 "целевых просмотров"
        return cpm / (self.get_target_coverage(age_group, city, sex) / 1000)
```

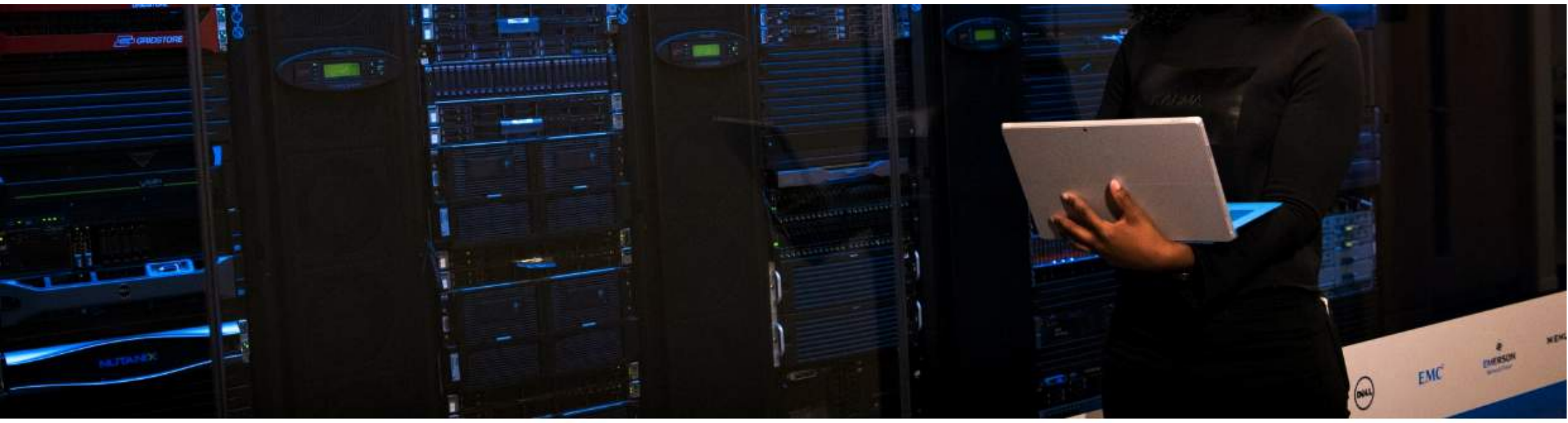
Напишем контроллер, который будет запускать наш код и анализировать сообщества. В корневой директории создадим модуль `index.py`:



Содержание контроллера:

```
from modules.CPMAnalysis import CPMAnalysis

# Словарь по возрастным группам
age ={
    "12-18":  0,
    "18-21":  1,
    "21-24":  2,
    "24-27":  3,
    "27-30":  4,
    "30-35":  5,
    "35-45":  6,
    "45-100": 7,
}

sex = {
    "female": 0,
    "male": 1,

}
```

```python
# Указываем токен
token = "ваш_апи_токен"
# Создаём объект анализатора
cpm_analysis = CPMAnalysis(token=token, group_id=29246653)
# Расчитываем цену целевого охвата 1000 пользователей
target_cost = cpm_analysis.get_target_coverage(age["21-24"], 1, sex["male"])
# Выводим на экран
print(target_cost)
```



Софт для автоматического сбора и анализа аккаунтов Инстаграм под рекламу

Содержание урока

Идея программы и формула расчёта

Как известно, чем больше вовлеченность аудитории в контент площадки, тем больше отдача от рекламы. Соответственно, при подборе площадок для рекламы, в частности аккаунтов Instagram важно учитывать и уровень вовлеченности аудитории (ER).

ER в Instagram рассчитывается по следующей формуле: (количество лайков + количество комментариев) / количество подписчиков * 100

Лайки и комментарии берутся у последнего поста. Я предлагаю пойти чуть дальше, и слегка подкорректировать формулу. Мы будем считать среднее количество лайков и комментариев у последних 10 постов, для большей репрезентативности.

То есть, сумму лайков у последних N постов разделить на N, сумма комментариев у последних N постов разделить на N.

Для этого напишем нехитрый класс, который будет выполнять заданную задачу, а именно:

1. Соберёт и просуммирует статистику по последним постам
2. Получит количество подписчиков аккаунта
3. Рассчитает ER на основе этих данных

Принцип парсинга данных из Instagram

Чтобы получить информацию о аккаунте и последних постах нужно оправить get-запрос вида: `https://www.instagram.com/имя_аккаунта/?__a=1`

Например: https://www.instagram.com/kingjames/?__a=1

В ответ придёт вот такая мешанина из JSON:

{"logging_page_id":"profilePage_19410587","show_suggested_profiles":true,"show_follow_dialog":f
{"biography":"","blocked_by_viewer":false,"restricted_by_viewer":false,"country_block":false,"e
{"count":66232751},"followed_by_viewer":false,"edge_follow":{"count":345},"follows_viewer":fals
James","has_ar_effects":false,"has_channel":false,"has_blocked_viewer":false,"highlight_reel_co
recently":false,"business_category_name":null,"category_id":null,"overall_category_name":null,"
{"count":0,"edges":[]},"profile_pic_url":"https://instagram.fhel4-1.fna.fbcdn.net/v/t51.2885-19
1.fna.fbcdn.net&_nc_ohc=5IhgT5OVNCEAX_AUc9B&oh=ecef47036ea720cbf2d91ba64335edc5&oe=5F0AE7B8","p
19/s320x320/83700920_480186686267142_2503895191161667584_n.jpg?_nc_ht=instagram.fhel4-
1.fna.fbcdn.net&_nc_ohc=5IhgT5OVNCEAX_AUc9B&oh=e2d23b09e6aa9ad2a898461821318e36&oe=5F0B70C8","r
_timeline":{"count":3,"page_info":{"has_next_page":false,"end_cursor":null},"edges":[{"node":{"
{"height":853,"width":480},"display_url":"https://instagram.fhel4-1.fna.fbcdn.net/v/t51.2885-15
1.fna.fbcdn.net&_nc_cat=1&_nc_ohc=WXejGEpQOtIAX9H_IF0&oh=80ea6b856b90ec819cde9c98ea9d315f&oe=5E
media_overlay_info":null,"media_preview":"ABcq2QeDzjAzn0rnL3O7JJJ963zH5qlM7Scc4z+VYd+P3x4KggcHH
XoR29s9PpRRcVhqwM5BIIQ5+YDP6d6BD5YEg3N6gKR6gnPt+tQO7BuCRj3oaVwBhiOPU1RJYa9CjaTgj1B6dqKyZSWJJ5PH
{"id":"19410587","username":"kingjames"},"is_video":true,"accessibility_caption":null,"edge_med
#GodPleaseProtectUs\ud83d\ude4f\ud83c\udffe"}}]},"edge_media_to_comment":{"count":17574},"comme
{"count":1126480},"edge_media_preview_like":{"count":1126480},"location":null,"thumbnail_src":"
15/e35/c0.186.480.480a/101512822_140943670861127_5890011604673781608_n.jpg?_nc_ht=instagram.fhe
1.fna.fbcdn.net&_nc_cat=1&_nc_ohc=WXejGEpQOtIAX9H_IF0&oh=393a8b0762754ed5e4f93f55b39dedbd&oe=5E
15/e35/p150x150/101512822_140943670861127_5890011604673781608_n.jpg?_nc_ht=instagram.fhel4-
1.fna.fbcdn.net&_nc_cat=1&_nc_ohc=WXejGEpQOtIAX9H_IF0&oh=c4f2cf7d01a318e32bdcbd4f351240a4&oe=5E
1.fna.fbcdn.net/v/t51.2885-15/e35/p240x240/101512822_140943670861127_5890011604673781608_n.jpg?
1.fna.fbcdn.net&_nc_cat=1&_nc_ohc=WXejGEpQOtIAX9H_IF0&oh=9423785e1c9039afd1057dead7d5eff2&oe=5E
1.fna.fbcdn.net/v/t51.2885-15/e35/p320x320/101512822_140943670861127_5890011604673781608_n.jpg?
1.fna.fbcdn.net&_nc_cat=1&_nc_ohc=WXejGEpQOtIAX9H_IF0&oh=c032c8d5f978f2dc635877e4a20ce1ab&oe=5E
1.fna.fbcdn.net/v/t51.2885-15/e35/101512822_140943670861127_5890011604673781608_n.jpg?_nc_ht=in
1.fna.fbcdn.net&_nc_cat=1&_nc_ohc=WXejGEpQOtIAX9H_IF0&oh=80ea6b856b90ec819cde9c98ea9d315f&oe=5E
1.fna.fbcdn.net/v/t51.2885-15/e35/101512822_140943670861127_5890011604673781608_n.jpg?_nc_ht=in
1.fna.fbcdn.net&_nc_cat=1&_nc_ohc=WXejGEpQOtIAX9H_IF0&oh=80ea6b856b90ec819cde9c98ea9d315f&oe=5E
status":null,"is_published":true,"product_type":"igtv","title":"I LOVE YOU KID! I LOVE US!
\ud83d\ude22\ud83d\ude4f\ud83c\udffe\ud83d\udcaa\ud83c\udffe\u2764\ufe0f\u270a\ud83c\udffe\ud83
{"__typename":"GraphVideo","id":"2318564356230627676","shortcode":"CAtMf9qAkFc","dimensions":{"
15/e35/101458664_705308750011107_4011284243742164649_n.jpg?_nc_ht=instagram.fhel4-
1.fna.fbcdn.net&_nc_cat=103&_nc_ohc=NjkMYRqa7dQAX96oQny&oh=a761cc07d6587a189047230afd4f0b7a&oe=
,"media_overlay_info":null,"media_preview":"ABcq2QeDzjAzn0rnL3O7JJJ963zH5qlM7Scc4z+VYd+P3x4Kggc
5PXoR29s9PpRRcVhqwM5BIIQ5+YDP6d6BD5YEg3N6gKR6gnPt+tQO7BuCRj3oaVwBhiOPU1RJYa9CjaTgj1B6dqKyZSWJJ5
{"id":"19410587","username":"kingjames"},"is_video":true,"accessibility_caption":null,"edge_med
{"text":"#StayWoke\ud83d\udc41"}}]},"edge_media_to_comment":{"count":329},"comments_disabled":f
{"count":16420},"edge_media_preview_like":{"count":16420},"location":null,"thumbnail_src":"http

Естественно непонятно где здесь хранятся нужные нам данные и как их извлечь. Но есть сервисы, которые делают JSON более читаемым, например: https://jsonformatter.curiousconcept.com/

Вставляете туда ваш json-код, жмёте на кнопку «Process» и вам показывается качественно отформатированный JSON:

{
"logging_page_id":"profilePage_19410587",
"show_suggested_profiles":true,
"show_follow_dialog":false,
"graphql":{
"user":{ }
},
"toast_content_on_load":null
}

Уже куда понятнее и видно, что вся информация хранится по ключу `["graphql"]["user"]`:


```
"user":{
    "biography":"",
    "blocked_by_viewer":false,
    "restricted_by_viewer":false,
    "country_block":false,
    "external_url":null,
    "external_url_linkshimmed":null,
    "edge_followed_by":{
        "count":66232751
    },
    "followed_by_viewer":false,
    "edge_follow":{
        "count":345
```


Полистав содержимое этого ключа я увидел, что данные постах хранятся здесь:


```
"edge_felix_video_timeline":{ },
"edge_owner_to_timeline_media":{
    "count":2017,
    "page_info":{ },
    "edges":[
        { },
        { },
        {
            "node":{
                "__typename":"GraphImage",
                "id":"2322435810651862057",
                "shortcode":"CA68xCjjPQp",
                "dimensions":{
```


То есть по ключу:

```
user_info["graphql"]["user"]["edge_owner_to_timeline_media"]["edges"]
```

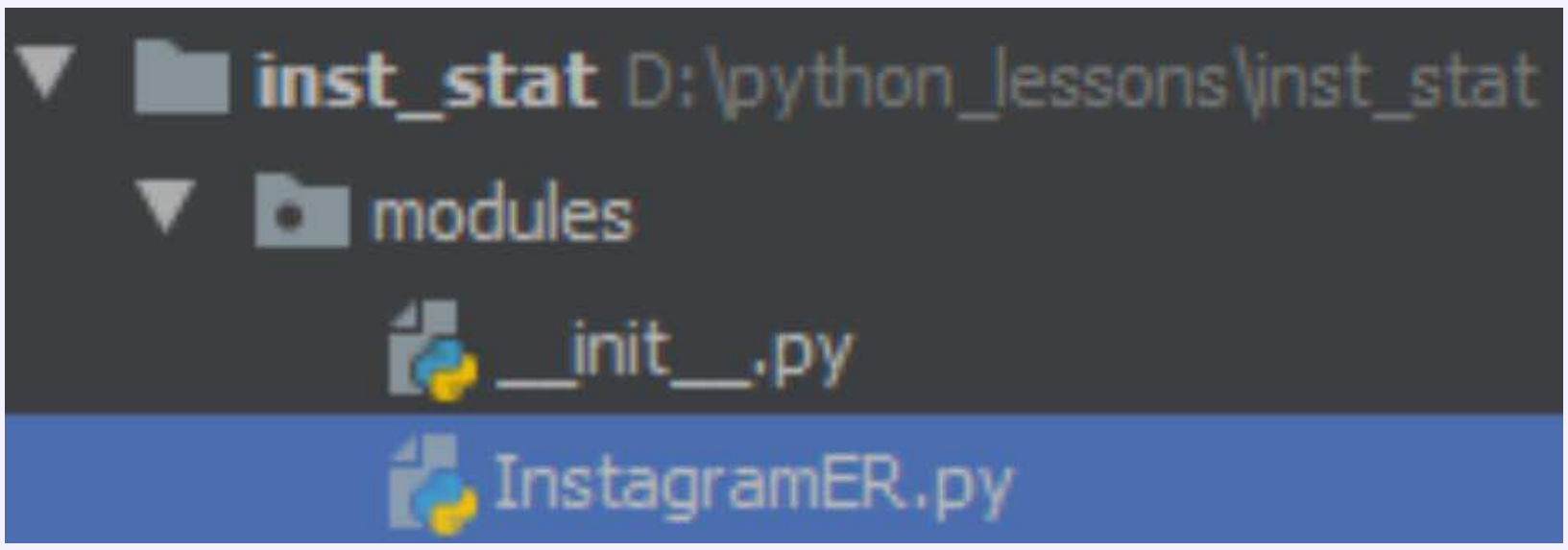
Лежит список с данными о постах. А именно:

Количество комментариев и количество лайков.

Напишем в таком случае проект, который автоматически рассчитает наш расширенный показатель ER.

Пишем парсер

Создадим проект, назовём его `inst_stat`. В нём создадим пакет `modules`, в котором создадим модуль `InstagramER`:



Импортируем в него следующие модули:

```
import json
import requests
```

Объявим класс:

```
class InstagramER:
```

Объявим в конструктор. В него передадим в виде необязательных параметров заголовки запроса, прокси и куки (если это нужно):

```
def __init__(self, headers=None, proxy=None, cookies=None):
    # Инициализируем заголовки
    self.__headers = headers
    # Инициализируем прокси
    self.__proxy = proxy
    # Инициализируем куки
    self.__cookies = cookies
```

Теперь напишем метод, который будет отправлять запрос на сервер и получать нужную информацию:

```
def __get_user_info(self, url):
    # Получаем данные пользователя, в т.ч. посты
    user_info = requests.get(url, headers=self.__headers, proxies=self.__proxy, params={"__a": 1})
    # Если сервер ответил
    if user_info.status_code == 200:
        # Получаем JSON-ответ
        json_info = user_info.text
        print(json_info)
        # Десертализуем JSON в словарь
        return json.loads(json_info)
```

Теперь напишем методы, которые извлекают все нужные нам данные из кода. Сперва метод, который возвращает количество подписчиков.

```
def __get_count_of_subscribers(self, user_info):
    # Из словаря с данными возвращаем
    return user_info["graphql"]["user"]["edge_followed_by"]["count"]
```

Теперь, метод который возвращает суммарный средний показатель по лайкам и комментариям за последние посты:

```
def __get_summary_likes_and_comments(self, user_info):
    # Счётчик комментариев
    count_of_comments = 0
    # Счётчик лайков
    count_of_likes = 0
    # Со скольки постов собираем статистику
    count_of_posts = len(user_info["graphql"]["user"]["edge_owner_to_timeline_media"]["edges"])
    # Обходим в цикле все посты
    for key in user_info["graphql"]["user"]["edge_owner_to_timeline_media"]["edges"]:
        # Суммируем количество комментариев поста к общему счётчику
        count_of_comments += key["node"]["edge_media_to_comment"]["count"]
        # Суммируем количество лайков поста к общему счётчику
        count_of_likes += key["node"]["edge_liked_by"]["count"]
    # Суммируем среднее количество лайков и среднее количество комментариев у одного поста
    return (count_of_comments / count_of_posts) + (count_of_likes / count_of_posts)
```

Тут мы определяем среднее количество лайков и комментариев за последние 10 постов и суммируем этот показатель.

Итак, все данные у нас имеются, остаётся только посчитать ER по приведённой выше формуле:

```
def get(self, url):
    # Получаем информацию о пользователей
    user_info = self.__get_user_info(url)
    # Получаем количество подписчиков
    count_of_subscribers = self.__get_count_of_subscribers(user_info)
    # Получаем суммарное количество лайков и репостов у одного поста (в среднем)
    summary_likes_and_comments = self.__get_summary_likes_and_comments(user_info)
    # Определяем ER
    return summary_likes_and_comments / count_of_subscribers * 100
```

Полный листинг класса:

```
import json
import requests
```

```python
class InstagramER:

    def __init__(self, headers=None, proxy=None, cookies=None):
        # Инициализируем заголовки
        self.__headers = headers
        # Инициализируем прокси
        self.__proxy = proxy
        # Инициализируем куки
        self.__cookies = cookies

    def __get_user_info(self, url):
        # Получаем данные пользователя, в т.ч. посты
        user_info = requests.get(url, headers=self.__headers, proxies=self.__proxy, params={"__a": 1})
        # Если сервер ответил
        if user_info.status_code == 200:
            # Получаем JSON-ответ
            json_info = user_info.text
            # Десертализуем JSON в словарь
            return json.loads(json_info)

    def __get_count_of_subscribers(self, user_info):
        # Из словаря с данными возвращаем
        return user_info["graphql"]["user"]["edge_followed_by"]["count"]

    def __get_summary_likes_and_comments(self, user_info):
        # Счётчик комментариев
        count_of_comments = 0
        # Счётчик лайков
        count_of_likes = 0
        # Со скольки постов собираем статистику
        count_of_posts = len(user_info["graphql"]["user"]["edge_owner_to_timeline_media"]["edges"])
        # Обходим в цикле все посты
        for key in user_info["graphql"]["user"]["edge_owner_to_timeline_media"]["edges"]:
            # Суммируем количество комментариев поста к общему счётчику
            count_of_comments += key["node"]["edge_media_to_comment"]["count"]
            # Суммируем количество лайков поста к общему счётчику
            count_of_likes += key["node"]["edge_liked_by"]["count"]
        # Суммируем среднее количество лайков и среднее количество комментариев у одного поста
        return (count_of_comments / count_of_posts) + (count_of_likes / count_of_posts)

    def get(self, url):
        # Получаем информацию о пользователей
        user_info = self.__get_user_info(url)
        # Получаем количество подписчиков
        count_of_subscribers = self.__get_count_of_subscribers(user_info)
        # Получаем суммарное количество лайков и репостов у одного поста (в среднем)
        summary_likes_and_comments = self.__get_summary_likes_and_comments(user_info)
        # Определяем ER
        return summary_likes_and_comments / count_of_subscribers * 100
```

Всё, данная функция возвращает ER.

Напишем контроллер, который запустит данный код. В корневой директории проекта создадим модуль `index.py`, в нём пропишем:

```python
from modules.InstagramER import InstagramER

headers = {
    "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
    "user-agent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.97 Safari/537.36",
}

# Список аккаунтов для определения ER
accounts = [
    "https://www.instagram.com/leonardodicaprio/",
    "https://www.instagram.com/sarwendah29/",
    "https://www.instagram.com/michelle_lewin/",
    "https://www.instagram.com/usainbolt/",
]

# Создаём объект анализатора
inst_er = InstagramER(headers=headers)
# Получаем ER заданнх аккаунтов
for account in accounts:
    er_of_account = inst_er.get(account)
    print(f"Account: {account}, ER: {er_of_account}")
```

Выводит:

```
Account: https://www.instagram.com/leonardodicaprio/, ER: 0.8119506446561444
Account: https://www.instagram.com/sarwendah29/, ER: 1.205529314023479
Account: https://www.instagram.com/michelle_lewin/, ER: 0.7551046935635846
Account: https://www.instagram.com/usainbolt/, ER: 1.4677711483743003
```

Таким образом, мы видим какие аккаунты более предпочтительны по вовлеченности.

Регер аккаунтов Mail

Содержание урока

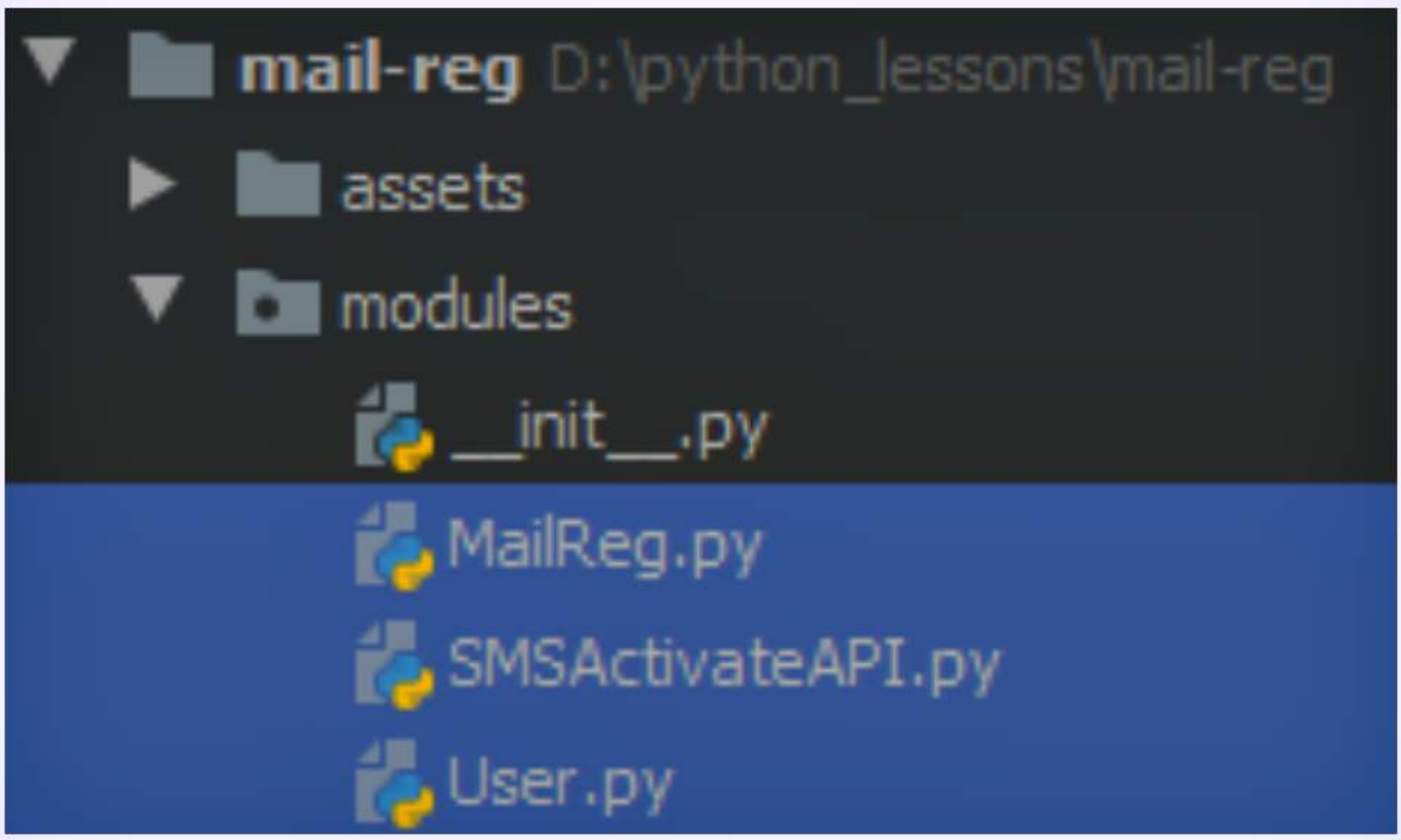
При регистрации практически любой сервис требует указания валидной почты с последующим подтверждением. Напишем скрипт, который будет регистрировать почты Mail.ru в автоматическом режиме.

Пишем модуль регистрации

Создадим проект, назовём его: `mail-reg`

Установим в него `Selenium`, так как регистрацию будем осуществлять при помощи браузера. Также установим библиотеку `requests`.

Создадим директорию `modules`, в неё скопируем класс `SMSActivate`, который мы написали в уроке номер 10. В этой же директории создадим ещё 2 модуля:



Начнём с написания модуля и класса MailReg. Импортируем в него следующие модули:

```
import time
from modules.SMSActivateAPI import SMSActivateAPI
from selenium.common.exceptions import NoSuchElementException
```

Объявим класс:

```
class MailReg:
```

Так как класс будет работать с Selenium необходимо инициализировать его объект, однако внутри класса мы не будем создавать его объект. Гораздо удобнее передавать уже «готовый» объект Selenium со всеми настройками.

Поэтому определим конструктор, который в качестве обязательного параметра принимает объект Selenium:

```
def __init__(self, chrome_driver, sms_activate_key):
    # Инициализируем объект chrome-драйвера
    self.__chrome_driver = chrome_driver
    # Инициализируем объект SMS-активации
    self.__sms_activate = SMSActivateAPI(sms_activate_key)
```

При регистрации каждый почтовый ящик необходимо подтверждать по СМС, поэтому обязательно также передавать API-ключ сервиса SMS активации.

Класс для работы с сервисом смс-активаций выше мы скопировали в наш проект и импортировали его.

Напишем функцию, которая закрывает сам браузер:

```
def close_browser(self):
    # Закрываем драйвер
    self.__chrome_driver.close()
```

Её мы будем использовать для закрытия браузера в блоке `try..finally`

Большая часть логики скрипта сводится к тому, чтобы искать элементы на странице и производить над ними некоторые действия. Потому создадим специальный метод, который будет искать элементы на странице в течении некоторого времени (ведь не всегда элемент отображается на странице сразу, порой он отображается через некоторое время).

Сперва нужно написать метод, который будет искать элемент на странице, если элемент найден – возвращает `True`, иначе – `False`:

```
def __check_exists_by_css(self, css_selector):
    try:
        # Пробуем найте элемент
        print(f"Ищем элемент: {css_selector}")
        self.__chrome_driver.find_element_by_css_selector(css_selector)
    except NoSuchElementException:
        # Если элемента на странице нет - возвращаем False
        print(f"Элемент: {css_selector} Не найден!")
        return False
    # Иначе возвращаем True
    return True
```

Если выскакивает ошибка о том, что элемента нет на странице – возвращаем `False`. Иначе – `True`.

Наконец, напишем метод который ищет элемент в коде страницы в течении некоторого времени:

```
# Ищем элемент в течении заданного времени
def __find_and_wait(self, css_selector, wait_time=60):
    # Запускаем цикл, по количеству секунд ожидания
    for i in range(wait_time):
        # Если элемент найден -- возвращаем его
        if self.__check_exists_by_css(css_selector):
            return self.__chrome_driver.find_elements_by_css_selector(css_selector)
        # Иначе -- ждём 1 секунду
        else:
            time.sleep(1)
```

Если элемент найден при помощи функции __check_exists_by_css возвращаем этот элемент. Иначе, ждём одну сикунду и запускаем поиск заново. И так до тех пор, пока элемент не будет найден или не истечёт заданное время поиска.

Мы научились искать элементы на странице, а что если нам нужно найти элемент внутри элемента, давайте напишем методы и для этого:

```
def __check_exists_by_css_in_element(self, element, css_selector):
    try:
        # Пробуем найте элемент
        element.find_element_by_css_selector(css_selector)
    except NoSuchElementException:
        # Если элемента на странице нет - возвращаем False
        return False
    # Иначе возвращаем True
    return True

def __find_and_wait_in_element(self, element, css_selector, wait_time=60):
    for i in range(wait_time):
        # Если элемент найден -- возвращаем его
        if self.__check_exists_by_css_in_element(element, css_selector):
            return element.find_elements_by_css_selector(css_selector)
        # Иначе -- ждём 1 секунду
        else:
            time.sleep(1)
```

Методы похожие на предыдущие, единственное отличие в том, что поиск осуществляется не по всей странице:

```
self.__chrome_driver.find_elements_by_css_selector
```

А по конкретному элементу:

```
element.find_elements_by_css_selector
```

Отлично, научились искать элементы, теперь давайте научимся с ними взаимодействовать. А именно, кликать по ним и вводить в них текст:

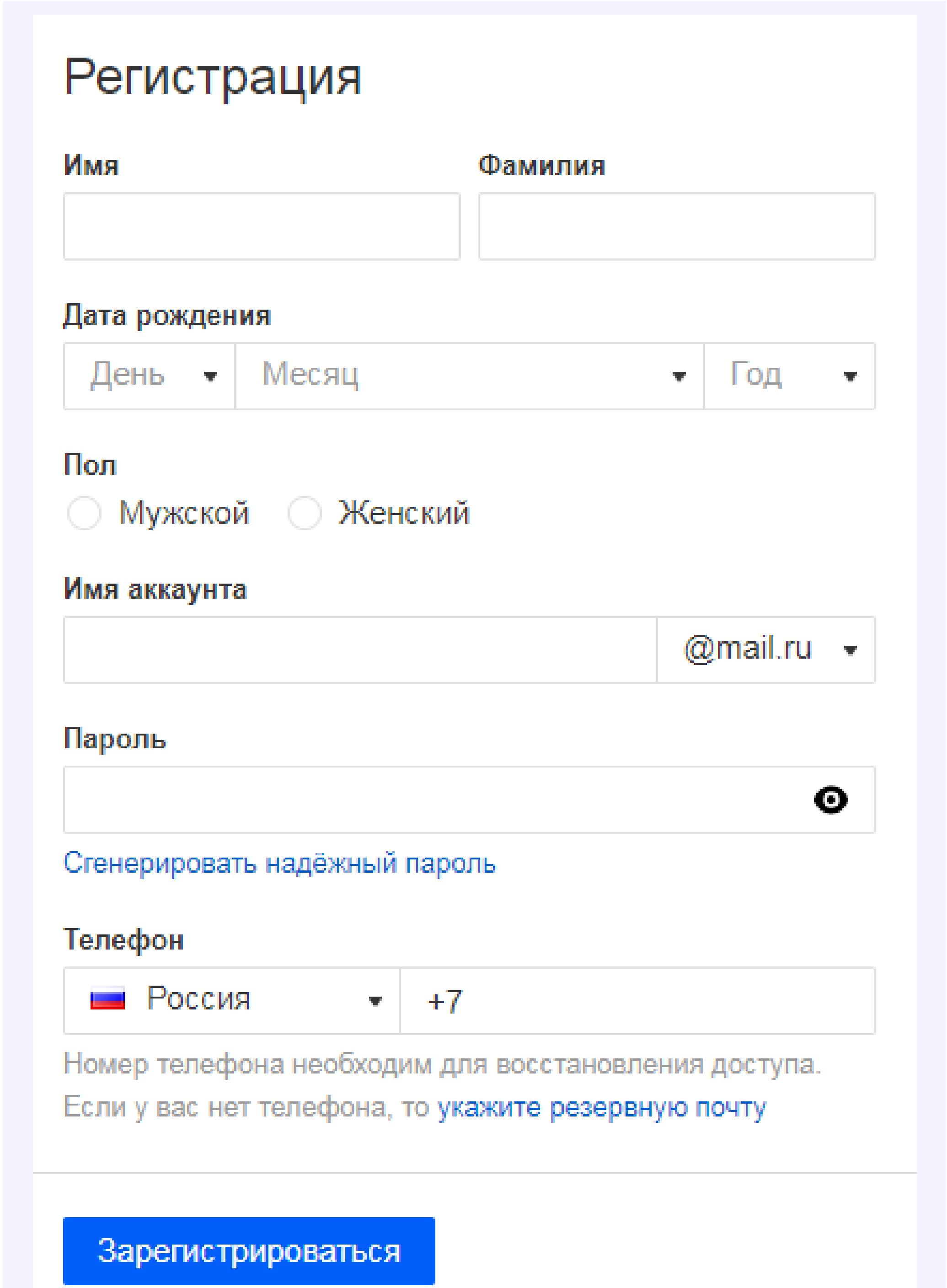
```
def __input(self, value, css_selector, position=0):
    # Ищем элемент
```

```
        input = self.__find_and_wait(css_selector)
        # Кликаем по инпуту
        input[position].click()
        # Вводим логин в форму
        input[position].send_keys(value)

    def __click(self, css_selector, position=0):
        # Ищем элемент
        elem = self.__find_and_wait(css_selector)
        # Кликаем по инпуту
        elem[position].click()
```

При поиске, нам возвращается список элементов, даже если он один. Поэтому мы передаём номер элемента, который нам нужен. По умолчанию мы передаём первый элемент (`position=0`), но если нужно взаимодействовать с другим элементом в списке (если их несколько) то можно передать его позицию.

Регистрация происходит на странице: https://account.mail.ru/signup?from=main&rf=auth.mail.ru

Сама страница:



Давайте напишем функции для заполнения основных полей. Сперва нужно перейти на страницу регистрации:

```
def __go_to_page(self):
    # Переходим на страницу регистрации
    self.__chrome_driver.get("https://account.mail.ru/signup?from=main&rf=auth.mail.ru")
```

Далее, заполнить первые поля Имя и Фамилия:

```
def __input_name(self, name):
    # Вводим имя пользователя
    self.__input(name, "input#fname")

def __input_surname(self, lname):
    # Вводим фамилию пользователя
    self.__input(lname, "input#lname")
```

Далее, ввести дату рождения. Она реализована в виде выпадающего списка, поэтому нужно сперва кликнуть на список, чтобы появились вариант для выбора, а затем уже на соответствующий элемент:

```
def __input_birth_day(self, day):
    # Открываем список дней
    self.__find_and_wait("div[data-test-id='birth-date__day']")[0].click()
    # Получаем список дней
    list_of_days = self.__find_and_wait("div.Select-menu-outer")[0]
    days = self.__find_and_wait_in_element(list_of_days, "div.Select-option")
    # Кликаем по заданному дню
    days[day-1].click()
```

Здесь вы можете заметить новую конструкцию CSS-селектора:

```
div[data-test-id='birth-date__day']
```

Данная конструкция расшифровывается так:

```
имя_тега[имя_атрибуда='содержимое атрибута']
```

Если вы хотите найти тег `input` с атрибутом name равным `login`:

```
<input name='login'>
```

То вам нужно написать следующее выражение:

```
input[name='login']
```

Это удобно, когда у элемента нет уникального класса или `id`, но есть какой-то уникальный атрибут.

Мы ищем сам список, кликаем на него, затем получаем элементы выпавшего списка, их в случае с днями 30-31, соответственно и элементов списка 30-31, поэтому кликаем на заданный элемент, 0 – 1 число, 15 – 14 число и т.д.

По такой же логике и остальные выпадающие списки, месяц и год рождения:

```
def __input_birth_month(self, month):
    # Открываем список месяцев
    self.__find_and_wait("div[data-test-id='birth-date__month']")[0].click()
    # Получаем список месяцев
    list_of_days = self.__find_and_wait("div.Select-menu-outer")
    days = self.__find_and_wait_in_element(list_of_days[0], "div.Select-option")
    # Выбираем соответствующий месяц
    days[month-1].click()

def __input_birth_year(self, year):
    # Открываем список по годам
    self.__find_and_wait("div[data-test-id='birth-date__year']")[0].click()
    # Получаем список по годам
    list_of_days = self.__find_and_wait("div.Select-menu-outer")
    days = self.__find_and_wait_in_element(list_of_days[0], "div.Select-option")
    # Выбираем нужный год. 1 - 2020, 2 - 2019, 3 - 2018 и т.д.
    days[year-1].click()
```

При заполнении года рождения есть одна особенность, текущий год имеет индекс 1 (2020), предыдущий индекс 2 (2019), предыдущий за ним индекс 3 (2018) и далее по аналогии.

Выбираем пол:

```
def __input_sex(self, sex):
    # Если пол равен True
    if sex:
        # Выбираем мужской
        self.__find_and_wait("label[data-test-id='gender-male']")[0].click()
    # Иначе
    else:
        # Выбираем женский
        self.__find_and_wait("label[data-test-id='gender-female']")[0].click()
```

Далее, вводим логин, пароль и повторяем пароль:

```
def __input_login(self, login):
    # Вводим логин
    self.__input(login, "input#aaa__input")

def __input_password(self, password):
    # Вводим пароль
    self.__input(password, "input#password")

def __input_repeat_password(self, password):
    # Повторяем ввод пароля
    self.__input(password, "input#repeatPassword")
```

Теперь нам нужно ввести номер телефона для получения СМС-активации. Напишем функцию, которая при помощи написанного нами ранее класса для СМС-активации получит номер телефона:

```
def __get_num(self):
    # Получаем статистику по свободным номерам
```

```
    count_of_num = self.__sms_activate.get_numbers_status(country=0, operator="any")
    # Узнаём баланс
    balance = self.__sms_activate.get_balance()
    # Если номера для заданного сервиса есть
    if int(count_of_num["ma_0"]) > 0 and balance >= 1:
        # Заказываем номер
        return self.__sms_activate.get_number("ma", "any")
```

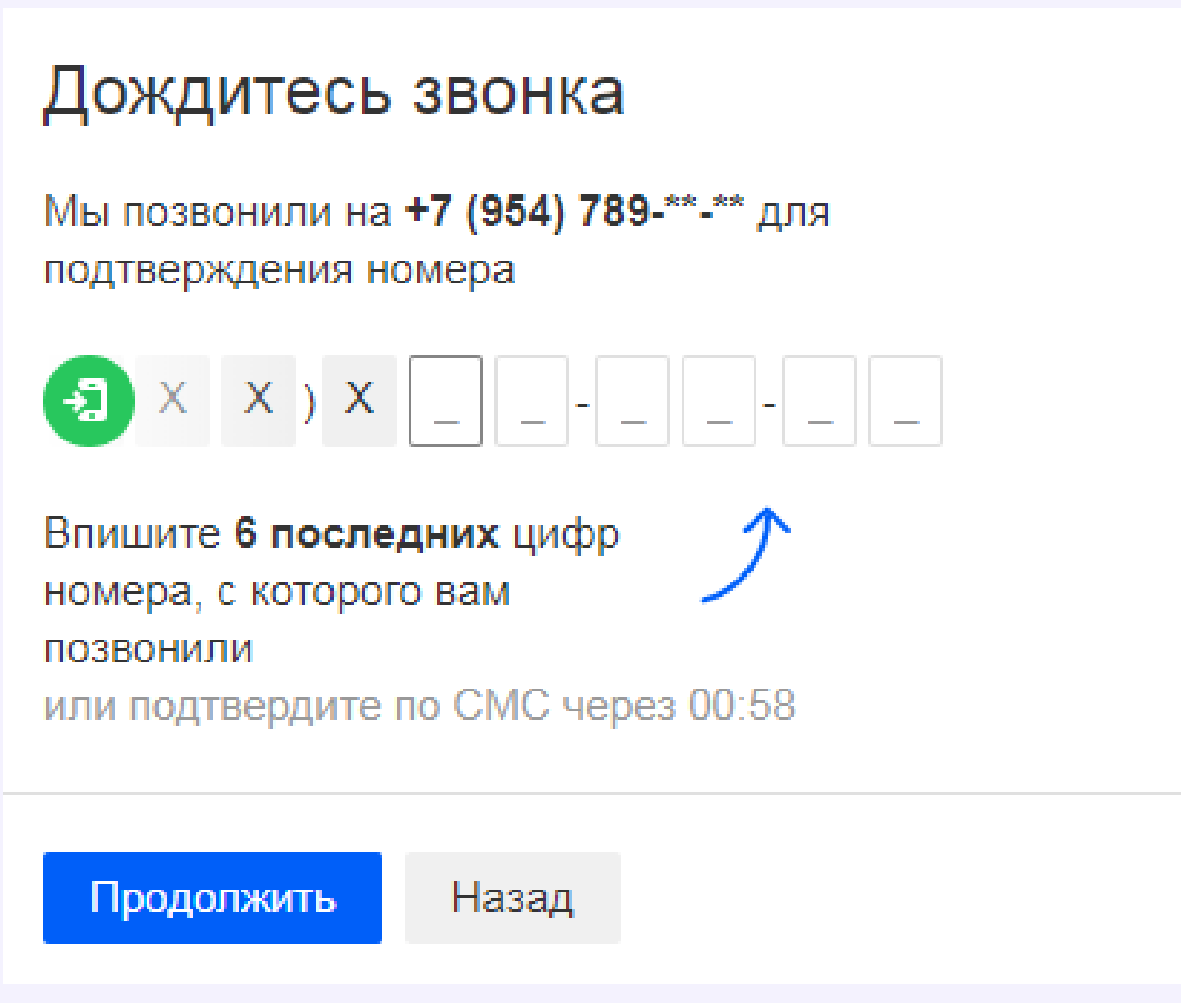
Введём полученный номер в форму:

```
def __input_phone(self, phone):
    # Вводим номер телефона для активации
    self.__input(phone, "input#phone-number__phone-input")
```

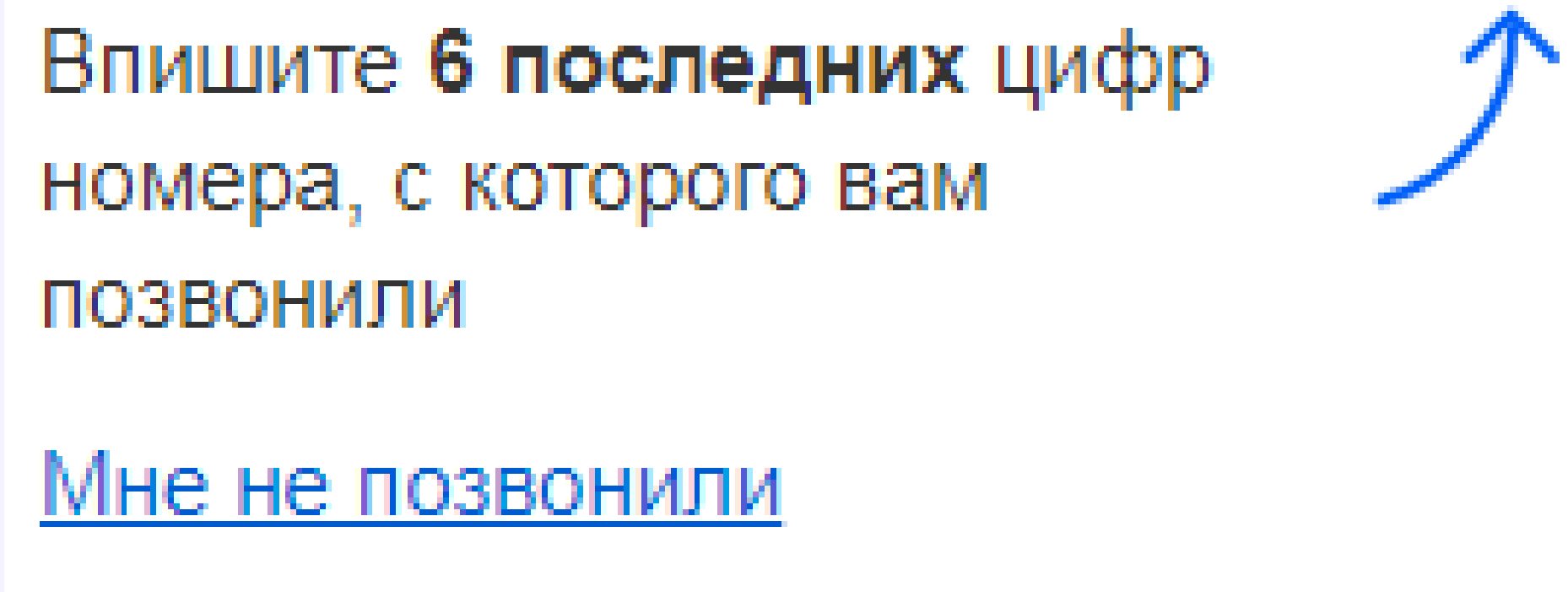
Отправим форму:

```
def __send_form(self):
    # Отправляем форму
    self.__click("button[data-test-id='first-step-submit']")
```

После отправки формы появится такое окно:

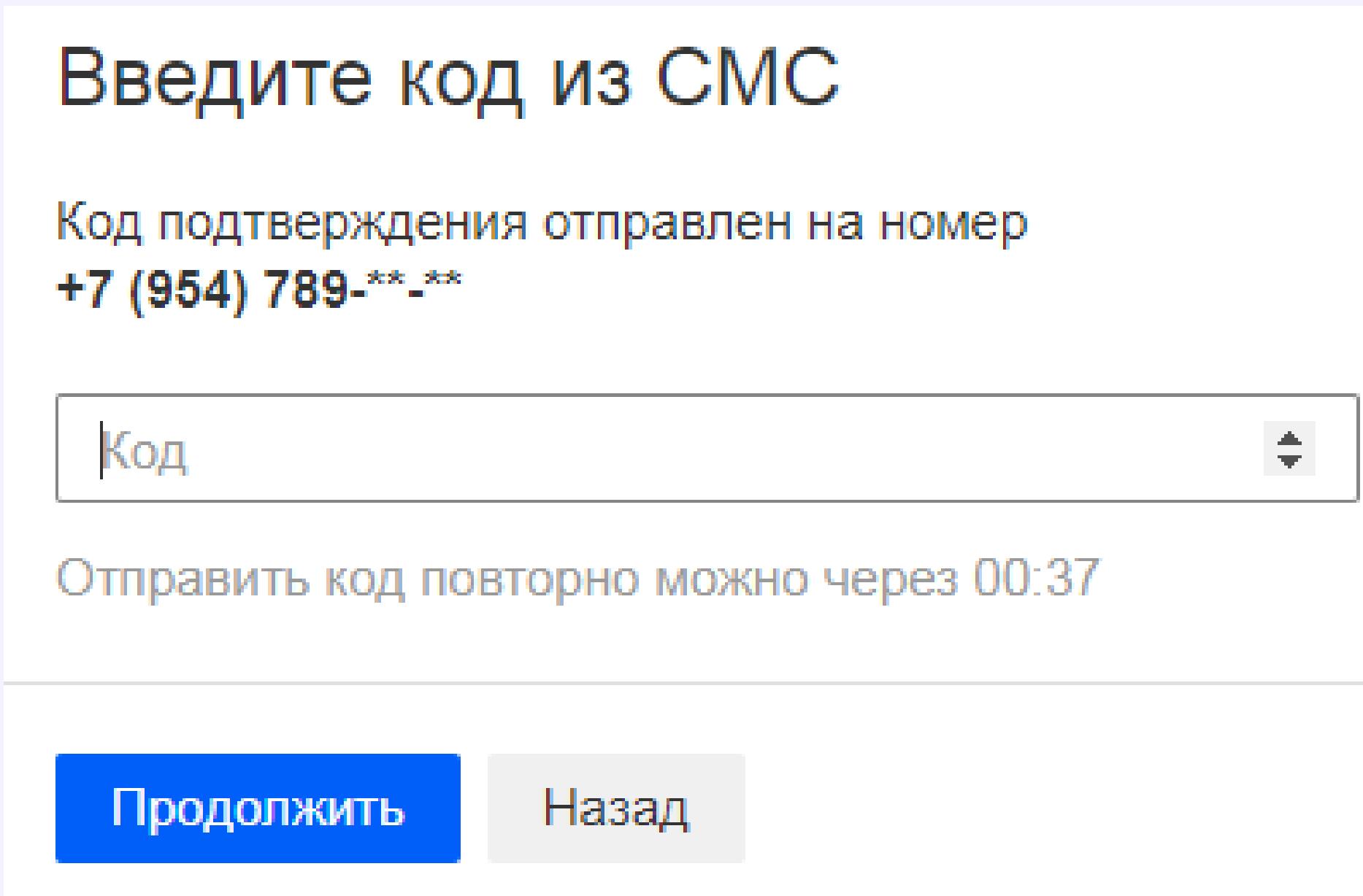


Но нам нужно принять СМС, а не звонок. Через 60 секунд появится кнопка, которая инициирует отправку СМС:



Поэтому, после нажатия на отправку формы, сразу запускаем поиск этой кнопки в течении 250 секунд. Таким образом, после того как пройдёт 60 секунд эта кнопка появится на странице и произойдёт нажатие.

После нажатия на неё откроется такое окно:



Осталось запросить получение кода, ввести его в форму и завершить регистрацию:

```
def __confirm_phone(self, id):
    # Указываем, что код отправлен
    self.__sms_activate.set_status(id, 1)
    # Ждём прихода СМС-кода
    code = self.__sms_activate.get_code(id, max_wait=250)
    # Если код успешно вернулся
    if code is not None and code[0] == "STATUS_OK":
        # Вводим код
```

```
            self.__input_code(code[1])
            # Уведомляем сервис о том, что код получен
            self.__sms_activate.set_status(id, status=6)
            # Отправляем код подтверждения mail.ru
            self.__send_code()
```

Всё. Регистрация на этом завершится.

Осталось написать сервисные методы, которые вызовут все те функции, которые мы написали.

Заполнение основных полей формы:

```
def __input_base_form(self, user):
    # Переходим на страницу регистрации
    self.__go_to_page()
    # Вводим имя
    self.__input_name(user.name)
    # Вводим фамилию
    self.__input_surname(user.surname)
    # Вводим день рождения
    self.__input_birth_day(user.birth_day)
    # Вводим месяц рождения
    self.__input_birth_month(user.birth_month)
    # Вводим год рождения
    self.__input_birth_year(user.age)
    # Выбираем пол
    self.__input_sex(user.sex)
    # Вводим логин
    self.__input_login(user.login)
    # Вводим пароль
    self.__input_password(user.password)
    # Повторяем ввод пароля
    self.__input_repeat_password(user.password)
```

Как видно, здесь функция принимает объект user, который содержит все данные для заполнения. Зачем это нужно? Ведь можно просто передать все параметры в функцию. Да, запросто, но вот сигнатура функции разрастётся.

Проще и правильнее создать отдельный класс, содержащий все поля для заполнения, и затем передавать его объект в функцию. Его мы напишем после этого класса.

Получение и введение номера телефона:

```
def __set_phone(self):
    # Получаем номер
    num = self.__get_num()
    # Вводим номер телефона
    self.__input_phone(num[1][1::])
    # Отправляем форму
    self.__send_form()
    # Просим mail.ru отправить код по СМС
    self.__take_sms_code()

    return num
```

Инициализация отправки СМС, ввод СМС-кода и завершения регистрации:

```
def __confirm_phone(self, id):
    # Указываем, что код отправлен
```

```
    self.__sms_activate.set_status(id, 1)
    # Ждём прихода СМС-кода
    code = self.__sms_activate.get_code(id, max_wait=250)
    # Если код успешно вернулся
    if code is not None and code[0] == "STATUS_OK":
        # Вводим код
        self.__input_code(code[1])
        # Уведомляем сервис о том, что код получен
        self.__sms_activate.set_status(id, status=6)
        # Отправляем код подтверждения mail.ru
        self.__send_code()
```

Осталось запустить все 3 функции в итоговой функции-контроллере:

```
def reg(self, user):
    # Заполняем форму регистрации
    self.__input_base_form(user)
    # Вводим номер телефона в форму
    num = self.__set_phone()
    # Подтверждаем регистрацию по СМС
    self.__confirm_phone(num[0])
```

Здесь функция принимает переменную user, которая содержит объект класса User.

Листинг класса:

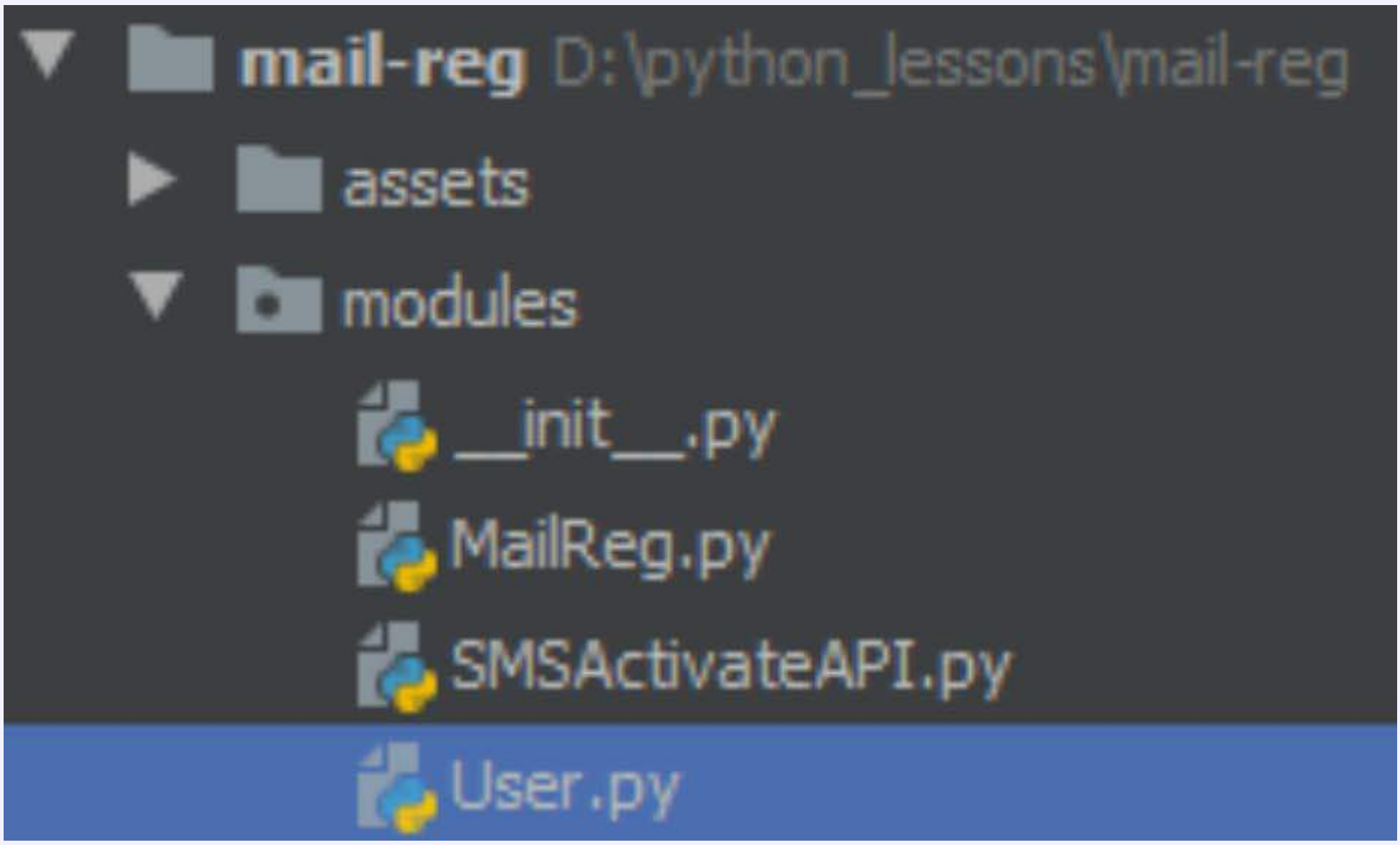
```
import time
from modules.SMSActivateAPI import SMSActivateAPI
from selenium.common.exceptions import NoSuchElementException


class MailReg:

    def __init__(self, chrome_driver, sms_activate_key):
        # Инициализируем объект chrome-драйвера
        self.__chrome_driver = chrome_driver
        # Инициализируем объект SMS-активации
        self.__sms_activate = SMSActivateAPI(sms_activate_key)

    def close_browser(self):
        # Закрываем драйвер
        self.__chrome_driver.close()

    def __check_exists_by_css(self, css_selector):
        try:
            # Пробуем найте элемент
            print(f"Ищем элемент: {css_selector}")
            self.__chrome_driver.find_element_by_css_selector(css_selector)
        except NoSuchElementException:
            # Если элемента на странице нет - возвращаем False
            print(f"Элемент: {css_selector} Не найден!")
            return False
        # Иначе возвращаем True
        return True

    # Ищем элемент в течении заданного времени
    def __find_and_wait(self, css_selector, wait_time=60):
        # Запускаем цикл, по количеству секунд ожидания
        for i in range(wait_time):
            # Если элемент найден -- возвращаем его
            if self.__check_exists_by_css(css_selector):
                return self.__chrome_driver.find_elements_by_css_selector(css_selector)
```

```
            # Иначе -- ждём 1 секунду
            else:
                time.sleep(1)

    def __check_exists_by_css_in_element(self, element, css_selector):
        try:
            # Пробуем найте элемент
            element.find_element_by_css_selector(css_selector)
        except NoSuchElementException:
            # Если элемента на странице нет - возвращаем False
            return False
        # Иначе возвращаем True
        return True

    def __find_and_wait_in_element(self, element, css_selector, wait_time=60):
        for i in range(wait_time):
            # Если элемент найден -- возвращаем его
            if self.__check_exists_by_css_in_element(element, css_selector):
                return element.find_elements_by_css_selector(css_selector)
            # Иначе -- ждём 1 секунду
            else:
                time.sleep(1)

    def __input(self, value, css_selector, position=0):
        # Ищем элемент
        input = self.__find_and_wait(css_selector)
        # Кликаем по инпуту
        input[position].click()
        # Вводим логин в форму
        input[position].send_keys(value)

    def __click(self, css_selector, position=0):
        # Ищем элемент
        elem = self.__find_and_wait(css_selector)
        # Кликаем по инпуту
        elem[position].click()

    def __go_to_page(self):
        # Переходим на страницу регистрации
        self.__chrome_driver.get("https://account.mail.ru/signup?from=main&rf=auth.mail.ru")

    def __input_name(self, name):
        # Вводим имя пользователя
        self.__input(name, "input#fname")

    def __input_surname(self, lname):
        # Вводим фамилию пользователя
        self.__input(lname, "input#lname")

    def __input_birth_day(self, day):
        # Открываем список дней
        self.__find_and_wait("div[data-test-id='birth-date__day']")[0].click()
        # Получаем список дней
        list_of_days = self.__find_and_wait("div.Select-menu-outer")[0]
        days = self.__find_and_wait_in_element(list_of_days, "div.Select-option")
        # Кликаем по заданному дню
        days[day-1].click()

    def __input_birth_month(self, month):
        # Открываем список месяцев
        self.__find_and_wait("div[data-test-id='birth-date__month']")[0].click()
        # Получаем список месяцев
        list_of_days = self.__find_and_wait("div.Select-menu-outer")
        days = self.__find_and_wait_in_element(list_of_days[0], "div.Select-option")
        # Выбираем соответствующий месяц
        days[month-1].click()

    def __input_birth_year(self, year):
```

```
        # Открываем список по годам
        self.__find_and_wait("div[data-test-id='birth-date__year']")[0].click()
        # Получаем список по годам
        list_of_days = self.__find_and_wait("div.Select-menu-outer")
        days = self.__find_and_wait_in_element(list_of_days[0], "div.Select-option")
        # Выбираем нужный год. 1 - 2020, 2 - 2019, 3 - 2018 и т.д.
        days[year-1].click()

    def __input_sex(self, sex):
        # Если пол равен 0
        if sex:
            # Выбираем мужской
            self.__find_and_wait("label[data-test-id='gender-male']")[0].click()
        # Иначе
        else:
            # Выбираем женский
            self.__find_and_wait("label[data-test-id='gender-female']")[0].click()

    def __input_login(self, login):
        # Вводим логин
        self.__input(login, "input#aaa__input")

    def __input_password(self, password):
        # Вводим пароль
        self.__input(password, "input#password")

    def __input_repeat_password(self, password):
        # Повторяем ввод пароля
        self.__input(password, "input#repeatPassword")

    def __get_num(self):
        # Получаем статистику по свободным номерам
        count_of_num = self.__sms_activate.get_numbers_status(country=0, operator="any")
        # Узнаём баланс
        balance = self.__sms_activate.get_balance()
        # Если номера для заданного сервиса есть
        if int(count_of_num["ma_0"]) > 0 and balance >= 1:
            # Заказываем номер
            return self.__sms_activate.get_number("ma", "any")

    def __input_phone(self, phone):
        # Вводим номер телефона для активации
        self.__input(phone, "input#phone-number__phone-input")

    def __send_form(self):
        # Отправляем форму
        self.__click("button[data-test-id='first-step-submit']")

    def __take_sms_code(self):
        # Ожидаем пока появится кнопка для получения СМС
        self.__find_and_wait("a[data-test-id='resend-callui-link']", wait_time=250)[0].click()

    def __input_code(self, code):
        # Вводим код из СМС
        self.__input(code, "input[data-test-id='code']")

    def __send_code(self):
        # Отправляем код на сервер
        self.__click("button[data-test-id='verification-next-button'")

    def __input_base_form(self, user):
        # Переходим на страницу регистрации
        self.__go_to_page()
        # Вводим имя
        self.__input_name(user.name)
        # Вводим фамилию
        self.__input_surname(user.surname)
        # Вводим день рождения
```

```
        self.__input_birth_day(user.birth_day)
        # Вводим месяц рождения
        self.__input_birth_month(user.birth_month)
        # Вводим год рождения
        self.__input_birth_year(user.age)
        # Выбираем пол
        self.__input_sex(user.sex)
        # Вводим логин
        self.__input_login(user.login)
        # Вводим пароль
        self.__input_password(user.password)
        # Повторяем ввод пароля
        self.__input_repeat_password(user.password)

    def __set_phone(self):
        # Получаем номер
        num = self.__get_num()
        # Вводим номер телефона
        self.__input_phone(num[1][1::])
        # Отправляем форму
        self.__send_form()
        # Просим mail.ru отправить код по СМС
        self.__take_sms_code()

        return num

    def __confirm_phone(self, id):
        # Указываем, что код отправлен
        self.__sms_activate.set_status(id, 1)
        # Ждём прихода СМС-кода
        code = self.__sms_activate.get_code(id, max_wait=250)
        # Если код успешно вернулся
        if code is not None and code[0] == "STATUS_OK":
            # Вводим код
            self.__input_code(code[1])
            # Уведомляем сервис о том, что код получен
            self.__sms_activate.set_status(id, status=6)
            # Отправляем код подтверждения mail.ru
            self.__send_code()

    def reg(self, user):
        # Заполняем форму регистрации
        self.__input_base_form(user)
        # Вводим номер телефона в форму
        num = self.__set_phone()
        # Подтверждаем регистрацию по СМС
        self.__confirm_phone(num[0])
```

Теперь напишем класс User. Для этого в пакете modules создадим модуль User:

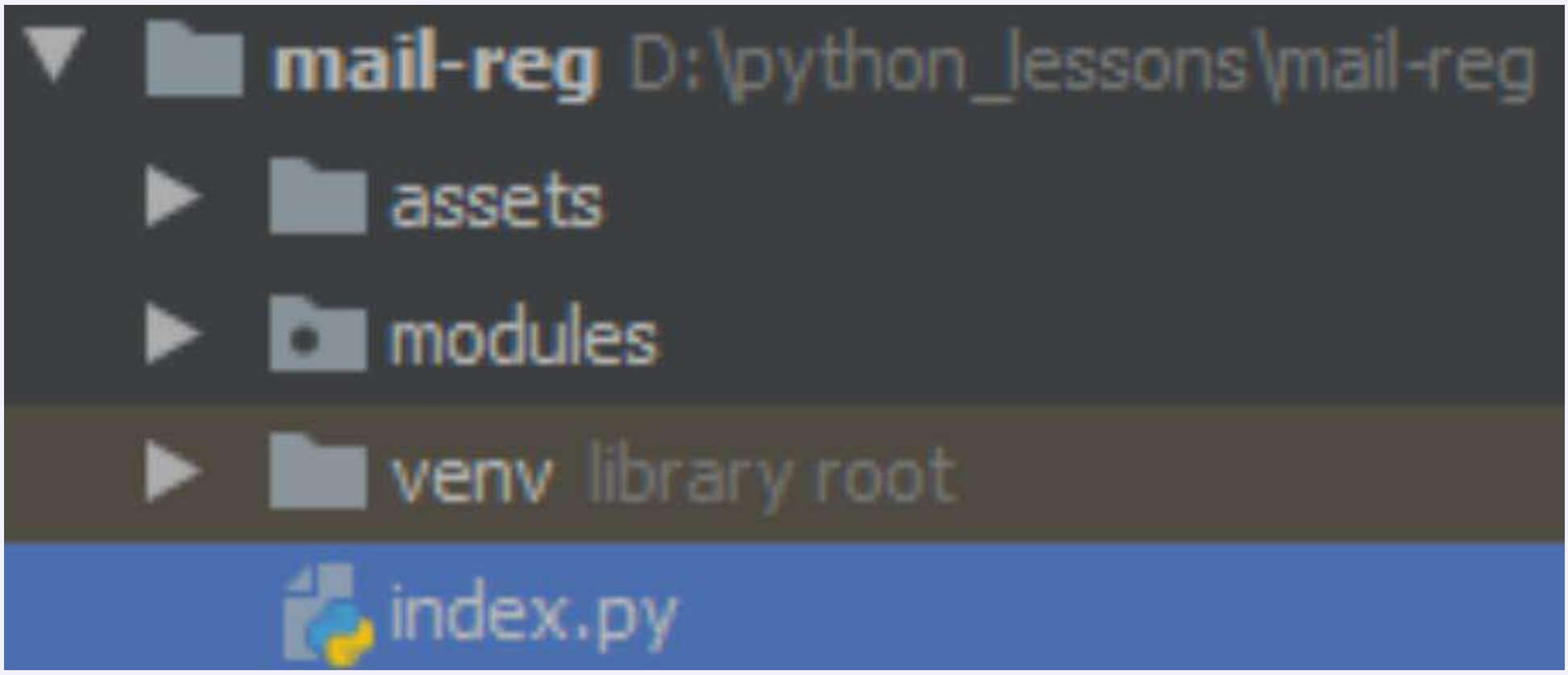


В нём объявим класс User и конструктор, в котором обязательно инициализируем все переменные, которые должны быть у пользователя:

```
class User:
    def __init__(self, name, surname, sex, login, password, birth_day, birth_month, age):
        # Имя
        self.name = name
        # Фамилия
        self.surname = surname
        # Пол. True - мужской, False - женский
        self.sex = sex
        # Логин
        self.login = login
        # Пароль
        self.password = password
        # День рождения
        self.birth_day = birth_day
        # Месяц рождения
        self.birth_month = birth_month
        # Год рождения.
        self.age = age
```

Пишем контроллер

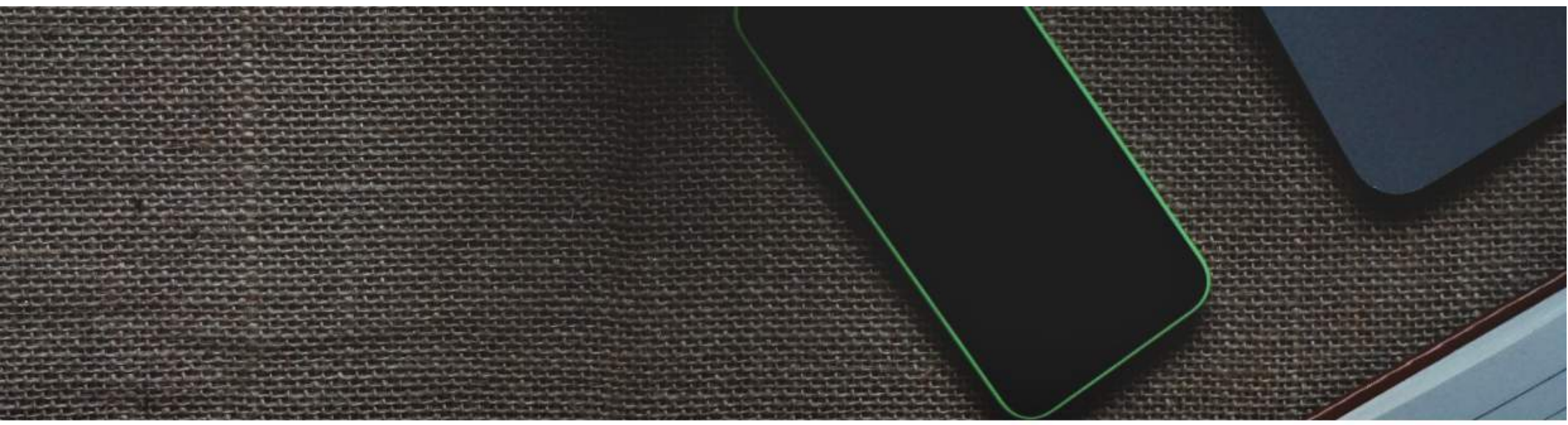
Отлично. Осталось только написать модуль-контроллер, который запустит весь наш код. В корневой директории создадим модуль `index`:



Содержимое контроллера:

```
from modules.MailReg import MailReg
from modules.User import User
from selenium.webdriver import Chrome

# Создаём объект браузера
chrome_obj = Chrome(executable_path="assets/drivers/chromedriver.exe")
# Объявляем API-ключ сервиса СМС-активации
sms_api_key = "A764A0Ad1412340f181c786240b53c4d"

# Создаём объект с данными пользователя для регистрации
user = User(
    name="Марк",
    surname="Аврелий",
    sex=True,
    login="Логин",
    password="Пароль",
    birth_day=18,
    birth_month=8,
    age=2
)

try:
    # Создаём объект регистратора почты
    mail_reg = MailReg(chrome_obj, sms_api_key)
    # Запускаем процесс регистрации
    mail_reg.reg(user)
finally:
    mail_reg.close_browser()
```

Запустив данный код, мы увидим как достаточно быстро будет зарегистрирован новый аккаунт mail.ru.



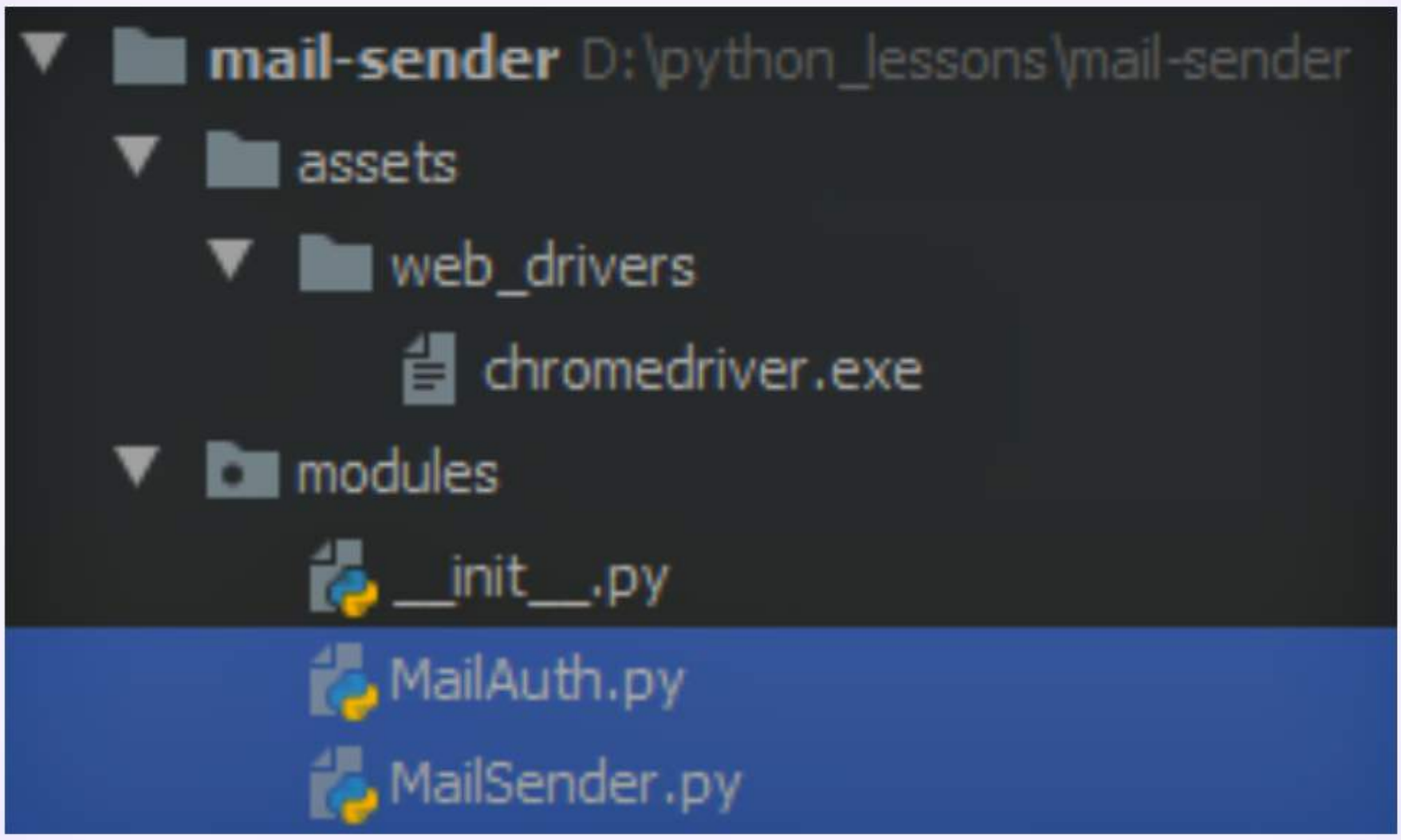
Email-рассыльщик через Mail

Содержание урока

Мы научились регистрировать аккаунты, теперь нам стоит научиться их использовать. Давайте напишем программу для рассылки сообщений через веб-интерфейс `mail.ru`

Создадим проект, назовём его `mail-sender`. Установим в него `selenium`.

Создадим пакет `modules`. В нём создадим два модуля, а именно: `MailAuth` и `MailSender`.



Модуль `MailAuth` будет отвечать за авторизацию пользователя по логину и паролю, а модуль `MailSender` будет отправлять письма с авторизованного аккаунта.

Модуль авторизации

Начнём с модуля авторизации. Импортируем в него следующие модули:

```
import time
from selenium.common.exceptions import NoSuchElementException
```

Создадим класс:

```
class MailAuth:
```

Для авторизации по логину и паролю через браузер, нам нужно три вещи: объект браузера, логин и пароль. Создадим конструктор:

```
def __init__(self, chrome_obj, login, password):
    # Получаем объуект юбраузера
    self.chrome_driver = chrome_obj
    # Получаем логин
    self.__login = login
    # Получаем пароль
    self.__password = password
    # Авторизуемся
    self.__auth()
```

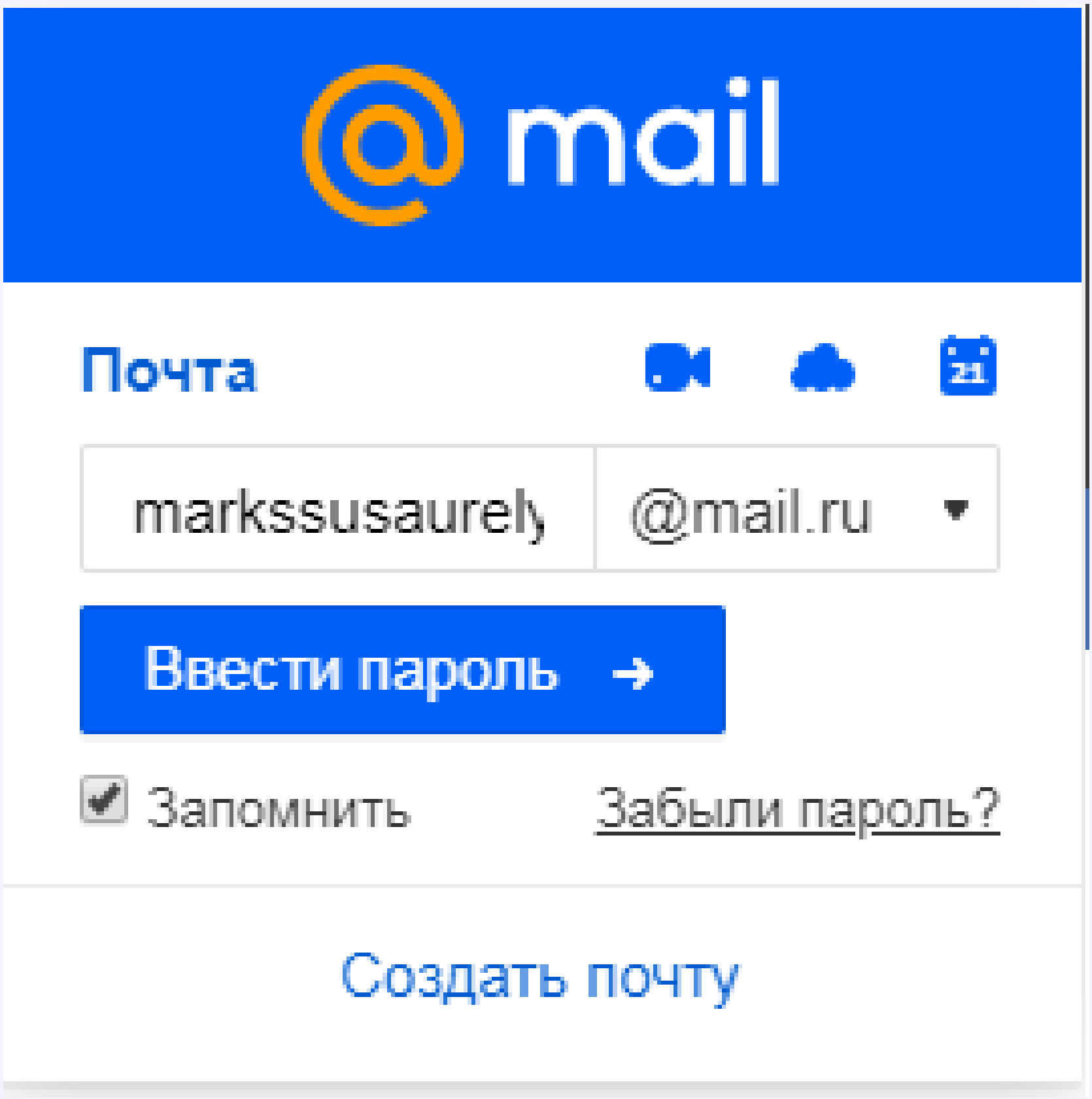
Нижеследующие методы я скопировал из прошлого урока:

```
def close_browser(self):
    # Закрываем браузер
    self.chrome_driver.close()

def __check_exists_by_css(self, css_selector):
    try:
        # Пробуем найте элемент
        print(f"Ищем элемент: {css_selector}")
        self.chrome_driver.find_element_by_css_selector(css_selector)
    except NoSuchElementException:
        # Если элемента на странице нет - возвращаем False
        print(f"Элемент: {css_selector} Не найден!")
        return False
    # Иначе возвращаем True
    return True

# Ищем элемент в течении заданного времени
def __find_and_wait(self, css_selector, wait_time=60):
    # Запускаем цикл, по количеству секунд ожидания
    for i in range(wait_time):
        # Если элемент найден -- возвращаем его
        if self.__check_exists_by_css(css_selector):
            return self.chrome_driver.find_elements_by_css_selector(css_selector)
        # Иначе -- ждём 1 секунду
        else:
            time.sleep(1)

def __check_exists_by_css_in_element(self, element, css_selector):
    try:
        # Пробуем найте элемент
        element.find_element_by_css_selector(css_selector)
    except NoSuchElementException:
        # Если элемента на странице нет - возвращаем False
        return False
    # Иначе возвращаем True
    return True

def __find_and_wait_in_element(self, element, css_selector, wait_time=60):
    for i in range(wait_time):
        # Если элемент найден -- возвращаем его
        if self.__check_exists_by_css_in_element(element, css_selector):
            return element.find_elements_by_css_selector(css_selector)
        # Иначе -- ждём 1 секунду
        else:
            time.sleep(1)
```

```python
def __input(self, value, css_selector, position=0):
    # Ищем элемент
    input = self.__find_and_wait(css_selector)
    # Кликаем по инпуту
    input[position].click()
    # Вводим логин в форму
    input[position].send_keys(value)

def __click(self, css_selector, position=0):
    # Ищем элемент
    elem = self.__find_and_wait(css_selector)
    # Кликаем по инпуту
    elem[position].click()
```

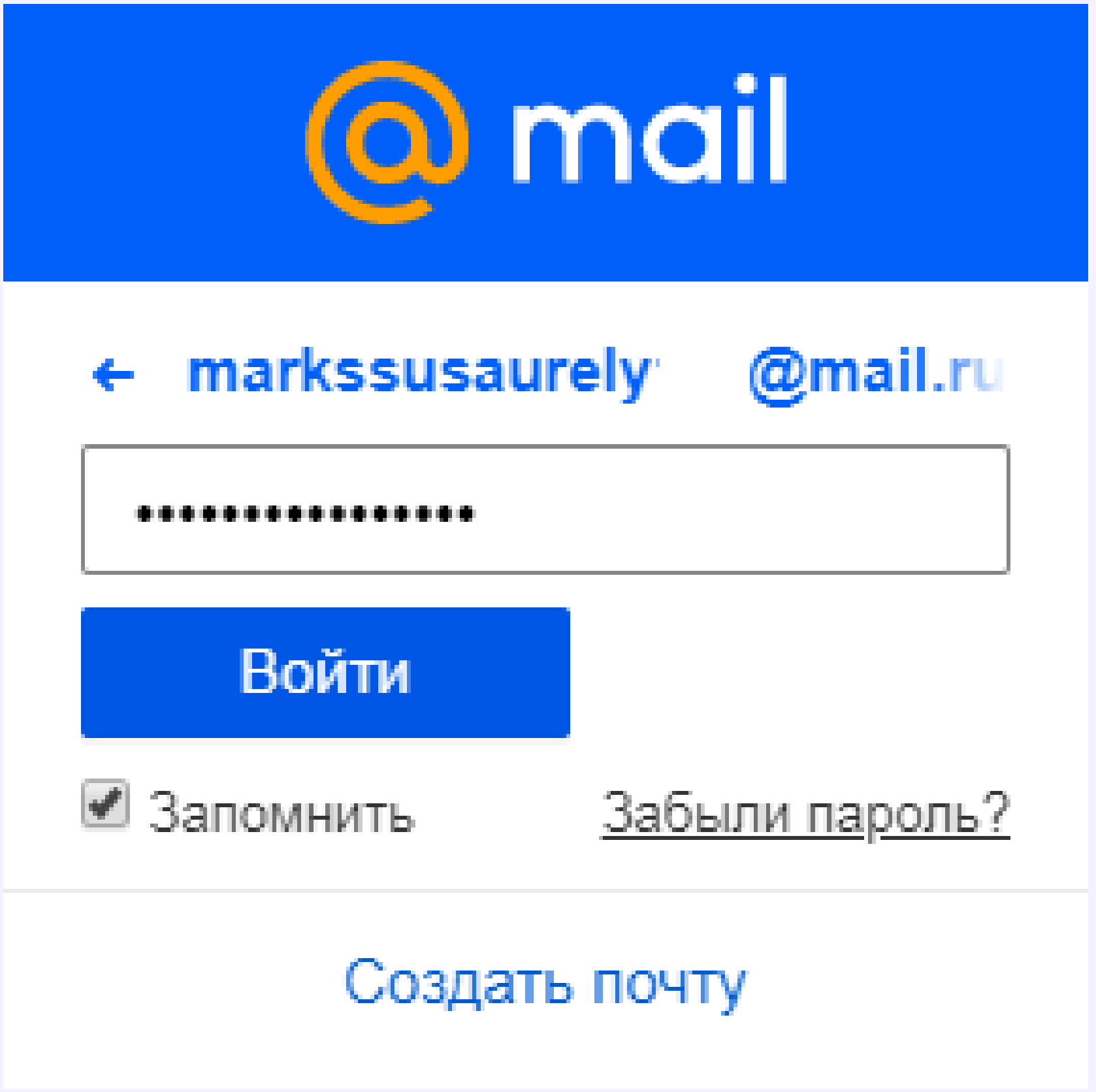
Они нужны для взаимодействия со страницей.

Форма авторизации в `mail.ru` следующая:



Нужно:

1. Ввести логин
2. Нажать кнопку «Ввести пароль»
3. Ввести пароль
4. Нажать кнопку «Войти»



Переходим на страницу авторизации:

```
def __go_to_login_page(self):
    # Переходим на страницу регистрации
    self.chrome_driver.get("https://mail.ru")
```

Вводим логин:

```
def __input_login(self, login):
    # Вводим логин
    self.__input(login, "input[name='login']")
```

Нажимаем «Ввести пароль»:

```
def __click_submit(self):
    # Отправляем форму
    self.__click("input[value='Ввести пароль']")
```

Вводим пароль:

```
def __input_password(self, password):
    # Вводим пароль
```

```
    self.__click("div.mailbox__step_second", 1)
    self.__input(password, "input[name='password']")
```

Пишем сервисный метод:

```
def __auth(self):
    # Переходим на страницу
    self.__go_to_login_page()
    time.sleep(1)
    # Вводим логин
    self.__input_login(self.__login)
    time.sleep(1)
    # Отправляем логин
    self.__click_submit()
    time.sleep(1)
    # Вводим пароль
    self.__input_password(self.__password)
    time.sleep(1)
    # Отправляем пароль
    self.__click_submit()
    # Ждём пока не произойдёт авторизация
    self.__find_and_wait("a[href='/compose/']")
```

Данный метод вводит данные в форму авторизации, затем отправляет её на сервер и ждёт, пока не откроется страница, следующая после авторизации.

После того, как вы авторизовались можно использовать текущий объект браузера с авторизованным аккаунтом.

Этот объект мы передадим в свою очередь в `MailSender`. Полный листинг класса:

```
import time
from selenium.common.exceptions import NoSuchElementException


class MailAuth:

    def __init__(self, chrome_obj, login, password):
        # Получаем обьуект юбраузера
        self.chrome_driver = chrome_obj
        # Получаем логин
        self.__login = login
        # Получаем пароль
        self.__password = password
        # Авторизуемся
        self.__auth()

    def close_browser(self):
        # Закрываем браузер
        self.chrome_driver.close()

    def __check_exists_by_css(self, css_selector):
        try:
            # Пробуем найте элемент
            print(f"Ищем элемент: {css_selector}")
            self.chrome_driver.find_element_by_css_selector(css_selector)
        except NoSuchElementException:
            # Если элемента на странице нет - возвращаем False
            print(f"Элемент: {css_selector} Не найден!")
            return False
        # Иначе возвращаем True
        return True

    # Ищем элемент в течении заданного времени
    def __find_and_wait(self, css_selector, wait_time=60):
        # Запускаем цикл, по количеству секунд ожидания
        for i in range(wait_time):
            # Если элемент найден -- возвращаем его
            if self.__check_exists_by_css(css_selector):
                return self.chrome_driver.find_elements_by_css_selector(css_selector)
            # Иначе -- ждём 1 секунду
            else:
                time.sleep(1)
```

```
    def __check_exists_by_css_in_element(self, element, css_selector):
        try:
            # Пробуем найте элемент
            element.find_element_by_css_selector(css_selector)
        except NoSuchElementException:
            # Если элемента на странице нет - возвращаем False
            return False
        # Иначе возвращаем True
        return True

    def __find_and_wait_in_element(self, element, css_selector, wait_time=60):
        for i in range(wait_time):
            # Если элемент найден -- возвращаем его
            if self.__check_exists_by_css_in_element(element, css_selector):
                return element.find_elements_by_css_selector(css_selector)
            # Иначе -- ждём 1 секунду
            else:
                time.sleep(1)

    def __input(self, value, css_selector, position=0):
        # Ищем элемент
        input = self.__find_and_wait(css_selector)
        # Кликаем по инпуту
        input[position].click()
        # Вводим логин в форму
        input[position].send_keys(value)
```

```
    def __click(self, css_selector, position=0):
        # Ищем элемент
        elem = self.__find_and_wait(css_selector)
        # Кликаем по инпуту
        elem[position].click()

    def __go_to_login_page(self):
        # Переходим на страницу регистрации
        self.chrome_driver.get("https://mail.ru")

    def __input_login(self, login):
        # Вводим логин
        self.__input(login, "input[name='login']")

    def __click_submit(self):
        # Отправляем форму
        self.__click("input[value='Ввести пароль']")

    def __input_password(self, password):
        # Вводим пароль
        self.__click("div.mailbox__step_second", 1)
        self.__input(password, "input[name='password']")

    def __auth(self):
        # Переходим на страницу
        self.__go_to_login_page()
        time.sleep(1)
        # Вводим логин
        self.__input_login(self.__login)
        time.sleep(1)
        # Отправляем логин
        self.__click_submit()
        time.sleep(1)
        # Вводим пароль
        self.__input_password(self.__password)
        time.sleep(1)
        # Отправляем пароль
        self.__click_submit()
        # Ждём пока не произойдёт авторизация
        self.__find_and_wait("a[href='/compose/']")
```

Модуль для отправки почты

В модуль `MailSender` импортируем следующие модули:

```
import time
from selenium.common.exceptions import NoSuchElementException
```

Создадим класс:

```
class MailSender:
```

Создадим конструктор:

```
def __init__(self, chrome_obj):
    # Принимаем объект браузера
    self.__chrome_driver = chrome_obj
```

Он принимает объект браузера, в котором присутствует авторизованный пользователь.

Нижеследующие методы взяты из предыдущего урока:

```
def close_browser(self):
    # Закрываем браузер
    self.__chrome_driver.close()
```

```
def __check_exists_by_css(self, css_selector):
    try:
        # Пробуем найте элемент
        print(f"Ищем элемент: {css_selector}")
        self.__chrome_driver.find_element_by_css_selector(css_selector)
    except NoSuchElementException:
        # Если элемента на странице нет - возвращаем False
        print(f"Элемент: {css_selector} Не найден!")
        return False
    # Иначе возвращаем True
    return True

# Ищем элемент в течении заданного времени
def __find_and_wait(self, css_selector, wait_time=60):
    # Запускаем цикл, по количеству секунд ожидания
    for i in range(wait_time):
        # Если элемент найден -- возвращаем его
        if self.__check_exists_by_css(css_selector):
            return self.__chrome_driver.find_elements_by_css_selector(css_selector)
        # Иначе -- ждём 1 секунду
        else:
            time.sleep(1)

def __check_exists_by_css_in_element(self, element, css_selector):
    try:
        # Пробуем найте элемент
        element.find_element_by_css_selector(css_selector)
    except NoSuchElementException:
        # Если элемента на странице нет - возвращаем False
        return False
    # Иначе возвращаем True
    return True

def __find_and_wait_in_element(self, element, css_selector, wait_time=60):
    for i in range(wait_time):
        # Если элемент найден -- возвращаем его
        if self.__check_exists_by_css_in_element(element, css_selector):
            return element.find_elements_by_css_selector(css_selector)
        # Иначе -- ждём 1 секунду
        else:
            time.sleep(1)

def __input(self, value, css_selector, position=0):
    # Ищем элемент
    input = self.__find_and_wait(css_selector)
    # Кликаем по инпуту
    input[position].click()
    # Вводим логин в форму
    input[position].send_keys(value)

def __click(self, css_selector, position=0):
    # Ищем элемент
    elem = self.__find_and_wait(css_selector)
```

```
    # Кликаем по инпуту
    elem[position].click()
```

Они нужны для того, чтобы манипулировать страницей. Перейдём на страницу для отправки сообщения:

```
def __go_to_send_page(self):
    # Переходим на страницу регистрации
    self.__chrome_driver.get("https://e.mail.ru/compose/")
```

Введём `email` пользователя, которому будем отправлять сообщение:

```
def __input_mail(self, mail):
    # Находим контейнер для письма
    head_container = self.__find_and_wait("div[class^='head_container']")[0]
    # В контейнре ищем элемент для ввода
    input = self.__find_and_wait_in_element(head_container, "input")[0]
    # Кликаем по форме ввода
    input.click()
    # Вводим почту
    input.send_keys(mail)
```

Здесь вы можете заметить CSS-селектор с новым символом:

```
div[class^='head_container']
```

Символ `^` обозначает, что мы ищем элемент в атрибуте которого присутствует вхождение заданной строки. Поскольку у `mail.ru` класс элемента генерируется случайным образом, по типу:

```
<div class='head_container-YeB2'>
```

Мы ищем не по полному соответствию, а по вхождению строки: `head_container`

Введём тему сообщения:

```
def __input_subject(self, subject):
    # Находим контейнер для ввода темы сообщения
    subject_container = self.__find_and_wait("div[class^='subject__container']")[0]
    # Ищем форму ввода
    input = self.__find_and_wait_in_element(subject_container, "input")[0]
    # Кликаем по форме ввода
    input.click()
    # Вводим почту
    input.send_keys(subject)
```

Введём сообщение:

```
def __input_editor(self, content):
    # Находим контейнер для ввода сообщения
    editor_container = self.__find_and_wait("div[class^='editor_container']")[0]
    # Находим вложенный контейнер для ввода темы сообщения
    editable_container = self.__find_and_wait_in_element(editor_container, "div[class^='editable-container']")[0]
    # Находим обёртку для формы ввода
    input_wrapper = self.__find_and_wait_in_element(editable_container, "div[role='textbox']")[0]
    # Находим форму ввода
    input = self.__find_and_wait_in_element(input_wrapper, "div")[0]
    # Кликаем по форме ввода
    input.click()
    # Вводим почту
    input.send_keys(content)
```

Нажмём кнопку «Отправить сообщение»:

```
def __send_message(self):
    # Кликаем по кнопке для отправки сообщения
    self.__click("span[title='Отправить']")
```

Напишем сервисный метод, отправляющий сообщение:

```
def send(self, mail, subject, message):
    # Переходим на страницу для отправки сообщения
    self.__go_to_send_page()
    # Вводим почту
    self.__input_mail(mail)
    # Вводим тему сообщения
    self.__input_subject(subject)
    # Вводим текст сообщения
    self.__input_editor(message)
    # Отправляем сообщение
    self.__send_message()
    # Ожидаем отправки
    self.__find_and_wait("div.layer-sent-page")
```

Отлично, теперь у нас есть метод который переходит на страницу отправки сообщения, вводит email адресата, тему сообщения, текст сообщения и отправляет письмо.

Полный листинг класса:

```
import time
from selenium.common.exceptions import NoSuchElementException


class MailSender:

    def __init__(self, chrome_obj):
        # Принимаем объект браузера
        self.__chrome_driver = chrome_obj

    def close_browser(self):
        # Закрываем браузер
        self.__chrome_driver.close()

    def __check_exists_by_css(self, css_selector):
        try:
            # Пробуем найте элемент
            print(f"Ищем элемент: {css_selector}")
            self.__chrome_driver.find_element_by_css_selector(css_selector)
        except NoSuchElementException:
            # Если элемента на странице нет - возвращаем False
            print(f"Элемент: {css_selector} Не найден!")
            return False
        # Иначе возвращаем True
        return True

    # Ищем элемент в течении заданного времени
    def __find_and_wait(self, css_selector, wait_time=60):
        # Запускаем цикл, по количеству секунд ожидания
        for i in range(wait_time):
            # Если элемент найден -- возвращаем его
            if self.__check_exists_by_css(css_selector):
                return self.__chrome_driver.find_elements_by_css_selector(css_selector)
            # Иначе -- ждём 1 секунду
            else:
                time.sleep(1)

    def __check_exists_by_css_in_element(self, element, css_selector):
        try:
            # Пробуем найте элемент
            element.find_element_by_css_selector(css_selector)
        except NoSuchElementException:
            # Если элемента на странице нет - возвращаем False
```

```
            return False
        # Иначе возвращаем True
        return True

    def __find_and_wait_in_element(self, element, css_selector, wait_time=60):
        for i in range(wait_time):
            # Если элемент найден -- возвращаем его
            if self.__check_exists_by_css_in_element(element, css_selector):
                return element.find_elements_by_css_selector(css_selector)
            # Иначе -- ждём 1 секунду
            else:
                time.sleep(1)

    def __input(self, value, css_selector, position=0):
        # Ищем элемент
        input = self.__find_and_wait(css_selector)
        # Кликаем по инпуту
        input[position].click()
        # Вводим логин в форму
        input[position].send_keys(value)

    def __click(self, css_selector, position=0):
        # Ищем элемент
        elem = self.__find_and_wait(css_selector)
        # Кликаем по инпуту
        elem[position].click()

    def __go_to_send_page(self):
        # Переходим на страницу регистрации
        self.__chrome_driver.get("https://e.mail.ru/compose/")

    def __input_mail(self, mail):
        # Находим контейнер для письма
        head_container = self.__find_and_wait("div[class^='head_container']")[0]
        # В контейнре ищем элемент для ввода
        input = self.__find_and_wait_in_element(head_container, "input")[0]
        # Кликаем по форме ввода
        input.click()
        # Вводим почту
        input.send_keys(mail)

    def __input_subject(self, subject):
        # Находим контейнер для ввода темы сообщения
        subject_container = self.__find_and_wait("div[class^='subject__container']")[0]
        # Ищем форму ввода
        input = self.__find_and_wait_in_element(subject_container, "input")[0]
        # Кликаем по форме ввода
        input.click()
        # Вводим почту
        input.send_keys(subject)

    def __input_editor(self, content):
        # Находим контейнер для ввода сообщения
        editor_container = self.__find_and_wait("div[class^='editor_container']")[0]
        # Находим вложенный контейнер для ввода темы сообщения
        editable_container = self.__find_and_wait_in_element(editor_container, "div[class^='editable-container']")[0]
        # Находим обёртку для формы ввода
        input_wrapper = self.__find_and_wait_in_element(editable_container, "div[role='textbox']")[0]
        # Находим форму ввода
        input = self.__find_and_wait_in_element(input_wrapper, "div")[0]
        # Кликаем по форме ввода
        input.click()
        # Вводим почту
        input.send_keys(content)

    def __send_message(self):
        # Кликаем по кнопке для отправки сообщения
        self.__click("span[title='Отправить']")

    def send(self, mail, subject, message):
        # Переходим на страницу для отправки сообщения
        self.__go_to_send_page()
        # Вводим почту
        self.__input_mail(mail)
        # Вводим тему сообщения
        self.__input_subject(subject)
        # Вводим текст сообщения
        self.__input_editor(message)
        # Отправляем сообщение
```

```
        self.__send_message()
        # Ожидаем отправки
        self.__find_and_wait("div.layer-sent-page")
```

Напишем теперь контроллер, который всё это будет запускать. В корневой директории проекта создадим модуль index. Пропишем в нём следующий код:

```
from selenium.webdriver import Chrome
from modules.MailAuth import MailAuth
from modules.MailSender import MailSender

# Создаём объект браузера
driver = Chrome(executable_path="assets/web_drivers/chromedriver.exe")

try:
    # Авторизуемся
    auth = MailAuth(driver, "логин_от_почты", "пароль")
    # Создаём объект рассыльщика, передаём объект авторизованного браузера
    sender = MailSender(auth.chrome_driver)
    # Отправляем сообщение
    sender.send("korben-mail@gmail.com", "Привет, Корбен!", "Пишу тебе по делу. Должна приехать моя тётушка, не мог бы ты её встретить?")
finally:
    # Закрываем соединение с браузером
    driver.close()
```

Сперва мы авторизуемся при помощи класса `MailAuth`, затем получаем объект браузера этого класса, в котором мы авторизовались в mail.ru, далее, передаём этот объект классу `MailSender`, который в свою очередь использует объект с авторизованным пользователем для отправки сообщения.

Можно, например, запустить рассылку в цикле на несколько пользователей сразу:

```
from selenium.webdriver import Chrome
from modules.MailAuth import MailAuth
from modules.MailSender import MailSender

# Список емейлов для рассылки
emails = [
    "mail1@mail.ru",
    "mail2@mail.ru",
    "mail3@mail.ru",
    "mail4@mail.ru",
    "mail5@mail.ru",
]

# Заголовок сообщения
subject = "Test subject"
# Текст сообщения
message = "Test message"


# Создаём объект браузера
driver = Chrome(executable_path="assets/web_drivers/chromedriver.exe")

try:
    # Авторизуемся
    auth = MailAuth(driver, "логин_от_почты", "пароль")
    # Создаём объект рассыльщика, передаём объект авторизованного браузера
    sender = MailSender(auth.chrome_driver)

    for email in emails:
        # Отправляем сообщение
        sender.send(email, subject, message)

finally:
    # Закрываем соединение с браузером
    driver.close()
```

Пишем модуль подтверждения SMS для регистрации

Содержание урока

Идея программы

Нередко для работы с определёнными ресурсами требуется подтверждение по СМС. Давайте напишем такое решение. Подтверждение по СМС производят через специальные сервисы активации. Одним из таких сервисов является сервис sms-activate: https://sms-activate.ru/

У данного сервиса есть API: https://sms-activate.ru/ru/api2

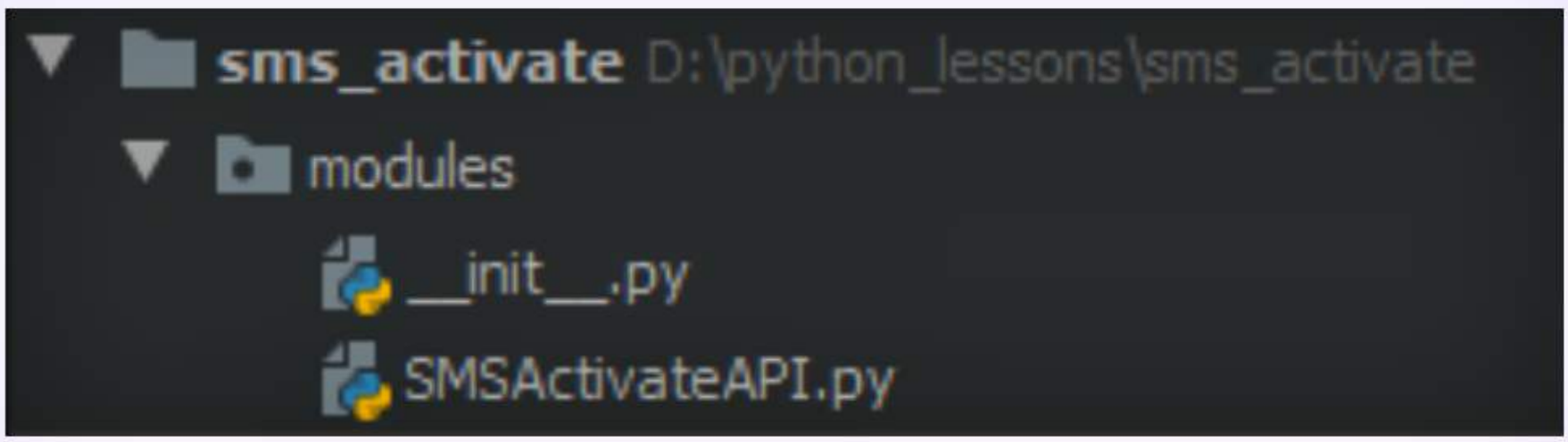
Поэтому будем работать через него. Логика работы скрипта следующая:

1. Проверяем баланс
2. Если его достаточно, заказываем номер для подтверждения
3. Вводим номер на заданном ресурсе и отправляем на него СМС
4. Проверяем статус СМС у сервиса до тех пор, пока не придёт СМС с кодом
5. Когда пришло СМС с кодом – подтверждаем получение

Пишем модуль для подтверждения

Создадим проект, назовём его `sms_activate`. Установим в проект модуль `requests`.

Создадим пакет `modules`. В нём создадим модуль под названием `SMSActivateAPI.py`:



Подключим модули по работе со временем, `JSON` и `requests`:

```
import json
import time
import requests
```

Теперь создадим класс для работы с `API SMS Activate`:

```
class SMSActivateAPI:
```

Итак, исходя из документации, для работы с API необходим специальный ключ, который находится в личном кабинете сервиса. Также, там написано, что все запросы к API будут направляться по адресу: **https://sms-activate.ru/stubs/handler_api.php**

Поэтому, давайте определим переменную с базовым адресом API:

```
# Определяем базовый URL-адрес до API
__base_url = "https://sms-activate.ru/stubs/handler_api.php"
```

Также, давайте определим в конструкторе инициализацию переменной с нашим ключом, который впоследствии будет использоваться при каждом запросе:

```
# Инициализируем API-ключ
def __init__(self, api_key):
    self.__api_key = api_key
```

Отлично, теперь при создании объекта мы будем обязаны указать API ключ.

Сперва напишем метод для определения баланса. Документация: https://sms-activate.ru/ru/api2#balans

Метод принимает:

API-key аккаунта, у которого мы получаем баланс
Имя метода, к которому мы обращаемся

Реализация:

```
# Получить баланс
def get_balance(self):
    # Определяем параметры запроса
    params = {
        "api_key": self.__api_key,
        "action": "getBalance",
    }
    # Отправляем запрос на сервер
    res = requests.post(self.__base_url, params=params)
    # Если пришёл положительный ответ
    if res.status_code == 200:
        # Возвращаем значение
        # Ответ приходет в формате вида ACCESS_BALANCE:'баланс на счету'
        # Разбивем его по символу ":" и возвращаем числовое значение
        return float(res.text.split(":")[1])
    # В противном случае возвращаем статус-код
    return res.status_code
```

Далее, напишем метод который получает количество свободных номеров. Документация: https://sms-activate.ru/ru/api2#quantity

Метод принимает:

API-key аккаунта
Имя метода, к которому мы обращаемся
Страну номера (не обязательно)
Оператора номера (не обязательно)

Реализация:

```
# Получить количество свободных номеров
def get_numbers_status(self, country=None, operator=None):
    # Определяем параметры запроса
    params = {
        "api_key": self.__api_key,
        "action": "getNumbersStatus",
        "country": country,
        "operator": operator,
    }
    # Отправляем запрос на сервер
    res = requests.post(self.__base_url, params=params)
    # Если пришёл положительный ответ
    if res.status_code == 200:
        # Возвращаем значение
        return json.loads(res.text)
    # В противном случае возвращаем статус-код
    return res.status_code
```

Заказать номер. Документация: https://sms-activate.ru/ru/api2#number

Метод принимает:

API-key аккаунта
Имя метода, к которому мы обращаемся
Сервис, который необходимо активировать
Оператор (необязательно)
Переадресация (необязательно)
Страна

Реализация:

```
# Заказать номер
def get_number(self, service, operator="any", forward=0, country=0):
    # Определяем параметры запроса
    params = {
        "api_key": self.__api_key,
        "action": "getNumber",
        "service": service,
        "operator": operator,
        "forward": forward,
        "country": country,
    }
    # Отправляем запрос на сервер
    res = requests.post(self.__base_url, params=params)
    # Если пришёл положительный ответ
    if res.status_code == 200 and "ACCESS_NUMBER" in res.text:
        # Возвращаем номер
        # Ответ приходет в формате вида ACCESS_NUMBER:$id:$number , где ($id - id операции,$number - номер телефона)
        # Разбивем его по символу ":" и возвращаем список ез первого элемента
        list = res.text.split(":")[1:3]
        return list
    return res.status_code
```

Установить статус. Документация: https://sms-activate.ru/ru/api2#changeStatus

Метод нужен, чтобы оповестить сервис активации о том, что сайт на котором вы регистрируетесь отправил СМС на ваш номер, а также, что необходимо завершить активацию.

Метод принимает:

API-key аккаунта
Имя метода, к которому мы обращаемся
Id-активации
Номер, на который необходимо выполнить переадресацию (необязательно)
Статус, который вы хотите передать сервису

Реализация:

```python
# Установить статус
def set_status(self, activation_id, status, forward=0):
    # Определяем параметры запроса
    params = {
        "api_key": self.__api_key,
        "id": activation_id,
        "status": status,
        "action": "setStatus",
    }
    # Отправляем запрос на сервер
    res = requests.post(self.__base_url, params=params)
    # Если пришёл положительный ответ
    if res.status_code == 200:
        # Возвращаем значение
        return res.text
    # Иначе, возвращаем код ответа
    return res.status_code
```

Получить статус. Документация: https://sms-activate.ru/ru/api2#getStatus

Метод необходим для того, чтобы узнать, пришло СМС от сервиса или нет.

Метод принимает:

API-key аккаунта
Имя метода, к которому мы обращаемся
Id активации

```python
# Получить статус
def get_status(self, activation_id):
    # Определяем параметры запроса
    params = {
        "api_key": self.__api_key,
        "id": activation_id,
        "action": "getStatus",
    }
    # Отправляем запрос на сервер
    res = requests.post(self.__base_url, params=params)
    # Если пришёл положительный ответ
    if res.status_code == 200:
        # Возвращаем значение
        return res.text.split(":")
    # Иначе, возвращаем код ответа
    return res.status_code
```

Получить код. После того, как мы отправили СМС на полученный в сервисе номер мы должны раз в несколько секунд отправлять запросы к сервису, дабы узнать, дошло до них СМС и, если оно дошло, получить его.

Для этого запустим цикл, в котором раз в три секунды будем получать статус СМС. Если ответ будет содержать заданный статус, сигнализирующий о том, что СМС получено – возвращаем содержимое СМС.

В противном случае, ждём 3 секунды и снова отправляем запрос. До тех пор, пока СМС не придёт, или не истечёт заданный промежуток времени.

```
# Получаем код
def get_code(self, id, wait=3, max_wait=60):
    # Инициализируем счётчик времени
    wait_time = 0
    # Иначе, запускаем цикл для ожидания СМС кода
    for i in range(max_wait):
        # Получаем статус СМС
        status = self.get_status(id)
        print(f"Статус активации: {status[0]}")
        # Если код был прислан
        if "STATUS_OK" in status:
            # Возвращаем код
            return status
        # Ожидаем 1 секунду
        time.sleep(wait)
        # Увеличиваем счётчик ожидания
        wait_time += wait
        # Если время, которое мы прождали больше,
        # либо равно максимально допустимому времени
        # ожидания, тогда возвращаем None
        if wait_time >= max_wait:
            print(f"Timeout: {max_wait} seconds")
            # Возвращаем None
            return None
```

Таким образом, наш класс был написан. Приведу полный листинг:

```
import json
import time
import requests


class SMSActivateAPI:

    # Определяем базовый URL-адрес до API
    __base_url = "https://sms-activate.ru/stubs/handler_api.php"

    # Инициализируем API-ключ
    def __init__(self, api_key):
        self.__api_key = api_key

    # Получить баланс
    def get_balance(self):
        # Определяем параметры запроса
        params = {
            "api_key": self.__api_key,
            "action": "getBalance",
        }
        # Отправляем запрос на сервер
        res = requests.post(self.__base_url, params=params)
        # Если пришёл положительный ответ
        if res.status_code == 200:
            # Возвращаем значение
            # Ответ приходет в формате вида ACCESS_BALANCE:'баланс на счету'
            # Разбивем его по символу ":" и возвращаем числовое значение
            return float(res.text.split(":")[1])
        # В противном случае возвращаем статус-код
        return res.status_code
```

```
    # Получить количество свободных номеров
    def get_numbers_status(self, country=None, operator=None):
        # Определяем параметры запроса
        params = {
            "api_key": self.__api_key,
            "action": "getNumbersStatus",
            "country": country,
            "operator": operator,
        }
        # Отправляем запрос на сервер
        res = requests.post(self.__base_url, params=params)
        # Если пришёл положительный ответ
        if res.status_code == 200:
            # Возвращаем значение
            return json.loads(res.text)
        # В противном случае возвращаем статус-код
        return res.status_code

    # Заказать номер
    def get_number(self, service, operator="any", forward=0, country=0):
        # Определяем параметры запроса
        params = {
            "api_key": self.__api_key,
            "action": "getNumber",
            "service": service,
            "operator": operator,
            "forward": forward,
            "country": country,
        }
        # Отправляем запрос на сервер
        res = requests.post(self.__base_url, params=params)
        # Если пришёл положительный ответ
        if res.status_code == 200 and "ACCESS_NUMBER" in res.text:
            # Возвращаем номер
            # Ответ приходет в формате вида ACCESS_NUMBER:$id:$number , где ($id - id операции,$number - номер телефона)
            # Разбивем его по символу ":" и возвращаем список ез первого элемента
            list = res.text.split(":")[1:3]
            return list

    # Установить статус
    def set_status(self, activation_id, status, forward=0):
        # Определяем параметры запроса
        params = {
            "api_key": self.__api_key,
            "id": activation_id,
            "status": status,
            "action": "setStatus",
        }
        # Отправляем запрос на сервер
        res = requests.post(self.__base_url, params=params)
        # Если пришёл положительный ответ
        if res.status_code == 200:
            # Возвращаем значение
            return res.text
        # Иначе, возвращаем код ответа
        return res.status_code

    # Получить статус
    def get_status(self, activation_id):
        # Определяем параметры запроса
        params = {
            "api_key": self.__api_key,
            "id": activation_id,
            "action": "getStatus",
        }
        # Отправляем запрос на сервер
        res = requests.post(self.__base_url, params=params)
        # Если пришёл положительный ответ
        if res.status_code == 200:
            # Возвращаем значение
```

```
            return res.text.split(":")
        # Иначе, возвращаем код ответа
        return res.status_code

    # Получаем код
    def get_code(self, id, wait=3, max_wait=60):
        # Инициализируем счётчик времени
        wait_time = 0
        # Получаем статус СМС
        status = self.get_status(id)
        # Если код был прислан
        # Иначе, запускаем цикл для ожидания СМС кода
        for i in range(max_wait):
            # Получаем статус СМС
            status = self.get_status(id)
            print(status[0])
            # Если код был прислан
            if "STATUS_OK" in status:
                # Возвращаем код
                return status
            # Ожидаем 1 секунду
            time.sleep(wait)
            # Увеличиваем счётчик ожидания
            wait_time += wait
            # Если время, которое мы прождали больше,
            # либо равно максимально допустимому времени
            # ожидания, тогда возвращаем None
            if wait_time >= max_wait:
                print(f"Timeout: {max_wait} seconds")
                # Возвращаем None
                return None
```

Теперь напишем контроллер, который будет отправлять СМС и получать код для активации:

```
from modules.SMSActivateAPI import SMSActivateAPI

# Создаём объект СМС-активации
sms_activate = SMSActivateAPI("A764A0Ad1412340f181c786240b53c4d")
# Получаем статистику по количеству номеров возвращается
# словарь с количеством номеров по каждому сервису
count_of_num = sms_activate.get_numbers_status(country=0, operator="any")
# Узнаём баланс
balance = sms_activate.get_balance()

# Если номера для заданного сервиса есть
if int(count_of_num["ma_0"]) > 0 and balance >= 5:
    # Заказываем номер
    id, num = sms_activate.get_number("ma", "any")
    print(f"Получили номер: {num}")
    # Здесь прописываем код, который вводит полученный номер на заданном
    # сайте (например при регистрации) и отправляет на него СМС

    # Говорим сервису о том, что СМС на номер было отправлено
    # и мы ожидаем от него код
    sms_activate.set_status(id, 1)
    # Ожидаем пока код отправленный на заданный номер код
    # будет возвращен сервисом активации
    code = sms_activate.get_code(id, max_wait=250)
    # Если код успешно вернулся
    if code is not None and code[0] == "STATUS_OK":
        # Выводим его на экран
        print(sms_activate.set_status(id, status=6))
```

Тут всё просто:

1. Подключаем наш класс к контроллеру
2. Создаём объект класса для активаций
3. Получаем список номеров доступных для активации
4. Получаем наш баланс
5. Если есть номер для активации нашего сервиса и баланс соответствует требованиям получаем номер
6. Вводим этот номер на сайт и отправляем на него СМС
7. Уведомляем сервис СМС-активаций о том, что СМС было отправлено
8. Получаем код из СМС
9. Выводим его на экран

Пишем модуль для разгадывания каптчи через сервис

Содержание урока

Принцип работы программыа

При работе с разного рода онлайн-ресурсами нередко требуется ввод каптчи. Поэтому давайте напишем свой модуль, который нам в этом поможет.

Данный модуль будет работать с API сервиса anti-captcha.com. Документация: https://anti-captcha.com/apidoc/image

Что позволит нам разгадывать каптчу в автоматическом режиме.

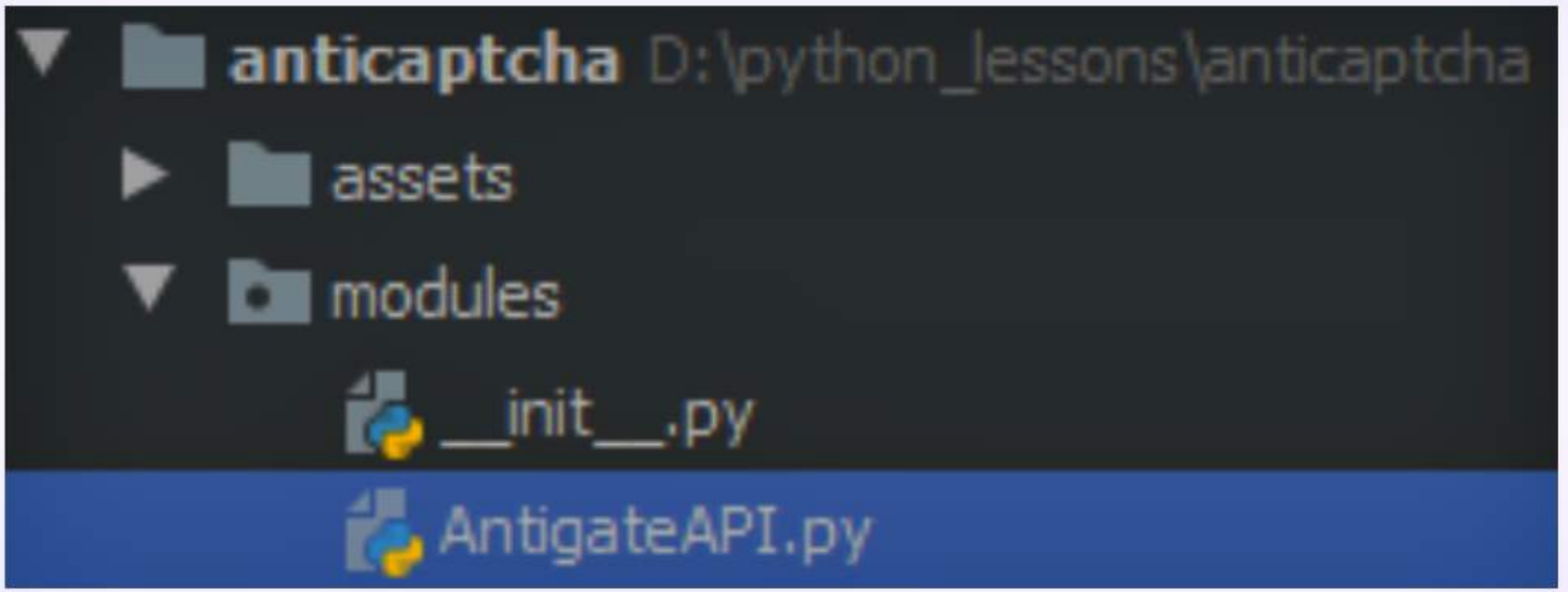
Алгоритм работы программы:

1. Получаем ссылку на изображение с каптчей
2. Скачиваем и кодируем в `base64`
3. Отправляем на сервер для разгадывания
4. Раз в несколько секунд проверяем, разгадали каптчу или нет
5. Если разгадали – получаем текст каптчи

Пишем модуль разгадывания каптчи

Создадим новый проект. Назовём его `anticaptcha`. Установим в проект библиотек `requests`. Создадим папку `assets`, в ней папку `img`:

В этой папке будут храниться скачанные нами изображения с каптчей. Создадим также пакет `modules`, а в нём модуль `AntigateAPI`:



Импортируем в модуль следующие библиотеки:

```
import base64
import json
import os
import shutil
import time

import requests
```

Создадим класс:

```
class ImageToTextTask:
```

В классе инициируем базовый URL до API anti-captcha.com:

```
__base_url = "https://api.anti-captcha.com"
```

Из документации понятно, что для работы с данным сервисом нам нужно как минимум передавать ему API-ключ, который лежит по адресу: https://anti-captcha.com/clients/settings/apisetup

Поэтому, пропишем инициализацию API-ключа через конструктор, т.к. без этого работать с сервисом будет невозможно. А также нам обязательно нужно сохранять изображения локально, поэтому также определим инициализацию папки, в которой будет сохраняться изображения каптчи:

```
def __init__(self, api_key, img_dir):
    self.__api_key = api_key
    self.__img_dir = img_dir
```

Из документации мы знаем, что для разгадки каптчи нам необходимо отправить изображение в виде строки, в формате base64. Для этого нам необходимо скачать изображение.

Напишем метод для этого:

```
# Сохраняет отдельное изображение
def __save_image(self, url_to_img):
    filename = url_to_img.split("?")[0]
    # Получаем исходный код изображения
    image = requests.get(filename, stream=True)

    # Извлекаем из ссылки, например:
    # https://media.istockphoto.com/vectors/finance-money-icons-vector-id1030878592?k=6&m=1030878592&s=612x612&w=0&h=HXSWx0FW-Re64cL
    # Название изображения, например: finance-money-icons-vector-id1030878592
    filename = url_to_img.split("/")[2]
    # Формируем путь до файла вида: папка/имя_файла.jpg
    path_to_file = f"{self.__img_dir}/{filename}.jpg"
    # Сохраняем фото в файл
    with open(path_to_file, "wb") as file_obj:
        shutil.copyfileobj(image.raw, file_obj)
        return path_to_file
```

Далее, это изображение нужно прочитать в строку и перекодировать в `base64`:

```
# Кодируем изображение в base64
def __img_to_base(self, url_to_img):
    # Скачиваем изображение и получаем путь до него
    path = self.__save_image(url_to_img)
    # Открываем изображение для чтения
```

```
    with open(path, "rb") as img_obj:
        # Читаем изображение
        image_read = img_obj.read()
        # Кодируем в base64
        b64_img = base64.encodebytes(image_read)
        # Возвращаем список из 2 элементов: (1) код изображения
        # в формате base64 (2) путь до изображения на вашем ПК
        return [b64_img.decode("UTF-8"), path]
```

У нас есть всё, чтобы отправить каптчу для разгадывания на сервер. Напишем метод для этого:

```
# Отправить каптчу на разгадывание
def __create_task(self, url_to_img, phrase=False, case=False, numeric=0, math=False):
    # Конвертируем каптчу в формат ase64
    b64_img = self.__img_to_base(url_to_img=url_to_img)
    # Получили из каптчи её base64 отображение, удяляем её локальную копию
    os.remove(b64_img[1])
    # Устанавливаем заголовки для запроса
    headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
    # Формируем параметры запроса
    task = {
        "clientKey": self.__api_key,
        "task": {
            "type": "ImageToTextTask",
            "body": b64_img[0],
            "phrase": phrase,
            "case": case,
            "numeric": numeric,
            "math": math,
        }
    }
    # Отправляем зпрос на сервер
    response = requests.post(f"{self.__base_url}/createTask", json=task, headers=headers)
    # Если получили успешный ответ
    if response.status_code == 200:
        # Декодируем JSON и возвращаем его
        return json.loads(response.text)
```

Создали задание на разгадывание каптчи, теперь нужно проверить, есть разгадали ли каптчу. Напишем метод проверяющий статус нашего задания по разгадыванию каптчи.

Каптчу разгадывают не сразу, потому нужно в течении некоторого времени отправлять запросы на сервер, с целью проверки её статуса и получения решённой каптчи:

```
# Получить разгаданную каптчу после создания
def __get_captcha(self, task_id, wait_resolve, delay):
    # Счётчик ожидания
    waiting = 0
    # Определяем словарь, в котором будет находится
    # статус задания на разгадываение каптчи
    status = {}
    # Запускем цикл,
    while waiting <= wait_resolve:
        # Получаем статус нашего задания
        status = self.__get_task_result(task_id)
        # Если задание решено
        if "solution" in status:
            # Возвращаем текст каптчи
            return status["solution"]["text"]
        # Иначе, ждём 3 секунды и продолжаем
        # отправлять запросы на сервер
        time.sleep(delay)
        waiting += delay
        print(f"Каптча не решена, ожидаем секунд: {delay}")
```

Здесь всё просто. Мы в цикле отправляем запросы на сервер, раз в несколько секунд, пока не истечёт время ожидания.

Нам осталось только объединить написанный нами код. Напишем итоговый контроллер:

```
# Контроллер для разгадывания каптчи
def resolve(self, url_to_img, wait_resolve=60, delay=3, phrase=False, case=False, numeric=0, math=False):
    # Создаём задание для разгадывания каптчи
    task = self.__create_task(url_to_img, phrase, case, numeric, math)
    # Если задание создано успешно
    if "taskId" in task:
        # Запускаем метод для получения каптчи после создания задания
        captcha = self.__get_captcha(task["taskId"], wait_resolve, delay)
        # Если каптча решена - возвращаем
        if captcha is not None:
            return captcha
    # В остальных случаях возвращаем информацию
    # о задании вероятно с ним какая-то проблема
    return task
```

Алгоритм:

1. Принимаем основные настройки для создания задания по разгадыванию каптчи
2. Создаём задание для разгадывания каптчи, получаем id задания
3. Далее, запускаем отслеживание задания
4. Если задание решено – возвращаем разгаданную каптчу
5. Иначе, возвращаем данные задания

Полный листинг класса:

```
import base64
import json
import os
import shutil
import time

import requests


class ImageToTextTask:
    __base_url = "https://api.anti-captcha.com"

    def __init__(self, api_key, img_dir):
        self.__api_key = api_key
        self.__img_dir = img_dir

    # Сохраняет отдельное изображение
    def __save_image(self, url_to_img):
        filename = url_to_img.split("?")[0]
        # Получаем исходный код изображения
        image = requests.get(filename, stream=True)

        # Извлекаем из ссылки, например:
        # https://media.istockphoto.com/vectors/finance-money-icons-vector-id1030878592?k=6&m=1030878592&s=612x612&w=0&h=HXSWx0FW-Re
        # Название изображения, например: finance-money-icons-vector-id1030878592
        filename = url_to_img.split("/")[2]
        # Формируем путь до файла вида: папка/имя_файла.jpg
        path_to_file = f"{self.__img_dir}/{filename}.jpg"
        # Сохраняем фото в файл
        with open(path_to_file, "wb") as file_obj:
            shutil.copyfileobj(image.raw, file_obj)
            return path_to_file

    # Кодируем изображение в base64
    def __img_to_base(self, url_to_img):
        # Скачиваем изображение и получаем путь до него
        path = self.__save_image(url_to_img)
        # Открываем изображение для чтения
        with open(path, "rb") as img_obj:
            # Читаем изображение
            image_read = img_obj.read()
            # Кодируем в base64
            b64_img = base64.encodebytes(image_read)
```

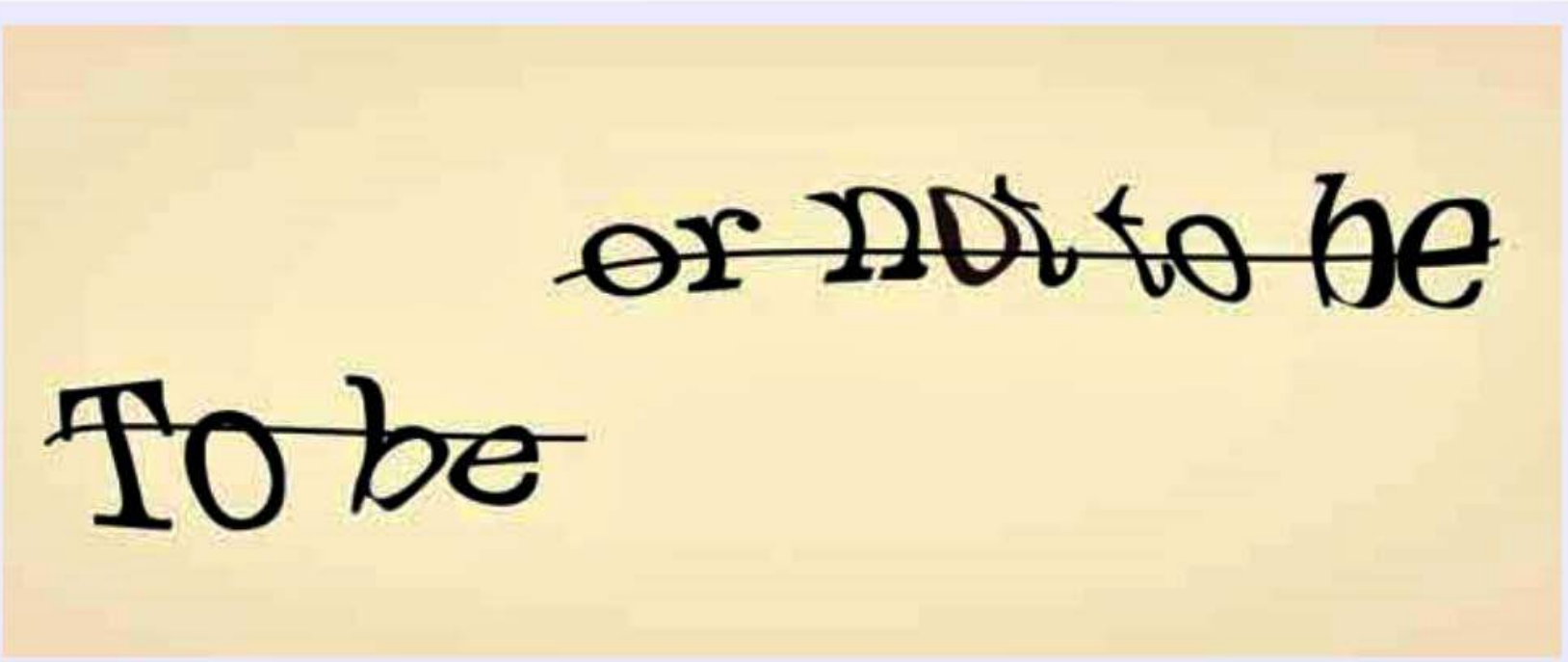
```
            # Возвращаем список из 2 элементов: (1) код изображения
            # в формате base64 (2) путь до изображения на вашем ПК
            return [b64_img.decode("UTF-8"), path]

    # Отправить каптчу на разгадывание
    def __create_task(self, url_to_img, phrase=False, case=False, numeric=0, math=False):
        # Конвертируем каптчу в формат ase64
        b64_img = self.__img_to_base(url_to_img=url_to_img)
        # Получили из каптчи её base64 отображение, удяляем её локальную копию
        os.remove(b64_img[1])
        # Устанавливаем заголовки для запроса
        headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
        # Формируем параметры запроса
        task = {
            "clientKey": self.__api_key,
            "task": {
                "type": "ImageToTextTask",
                "body": b64_img[0],
                "phrase": phrase,
                "case": case,
                "numeric": numeric,
                "math": math,
            }
        }
        # Отправляем запрос на сервер
        response = requests.post(f"{self.__base_url}/createTask", json=task, headers=headers)
        # Если получили успешный ответ
        if response.status_code == 200:
            # Декодируем JSON и возвращаем его
            return json.loads(response.text)

    # Получить статус задания
    def __get_task_result(self, task_id):
        # Параметры запроса
        task = {
            "clientKey": self.__api_key,
            "taskId": task_id
        }
        # Устанавливаем заголовки для запроса
        headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
        # Отправляем запрос на сервер
        response = requests.post(f"{self.__base_url}/getTaskResult", json=task, headers=headers)
        # Если получили успешный ответ
        if response.status_code == 200:
            # Декодируем JSON и возвращаем его
            return json.loads(response.text)

    # Получить разгаданную каптчу после создания
    def __get_captcha(self, task_id, wait_resolve, delay):
        # Счётчик ожидания
        waiting = 0
        # Запускем цикл,
        while waiting <= wait_resolve:
            # Получаем статус нашего задания
            status = self.__get_task_result(task_id)
            # Если задание решено
            if "solution" in status:
                # Возвращаем текст каптчи
                return status["solution"]["text"]
            # Иначе, ждём 3 секунды и продолжаем
            # отправлять запросы на сервер
            time.sleep(delay)
            waiting += delay
            print(f"Каптча не решена, ожидаем секунд: {delay}")

    # Контроллер для разгадывания каптчи
    def resolve(self, url_to_img, wait_resolve=60, delay=3, phrase=False, case=False, numeric=0, math=False):
        # Создаём задание для разгадывания каптчи
        task = self.__create_task(url_to_img, phrase, case, numeric, math)
        # Если задание создано успешно
        if "taskId" in task:
            # Запускаем метод для получения каптчи после создания задания
            captcha = self.__get_captcha(task["taskId"], wait_resolve, delay)
            # Если каптча решена - возвращаем
            if captcha is not None:
```

```
            return captcha
        # В остальных случаях возвращаем информацию
        # о задании вероятно с ним какая-то проблема
        return task
```

Давайте напишем контроллер для этого класса. Создадим как всегда файл `index.py` и пропишем в нём следующий код:

```
from modules.AntigateAPI import ImageToTextTask

# Создаём объект. Указываем API-key и путь до папки с изображениями
ImageToTextTask = ImageToTextTask("ЗДЕСЬ ВАШ API-KEY", "assets/img")
# Запускаем метод для разгадывания. Указываем ссылку на изображение
captcha = ImageToTextTask.resolve("https://redbackwebdesign.com.au/wp-content/uploads/2018/03/Captcha-test-507192307_931x377.jpg")
# Выводим  результат на страницу
print(captcha)
```

Изображение с капчей следующее:



На экран будет выведено:

> To be or not to be

Капча разгадана!

Практика. Пишем парсер картинок

Содержание урока

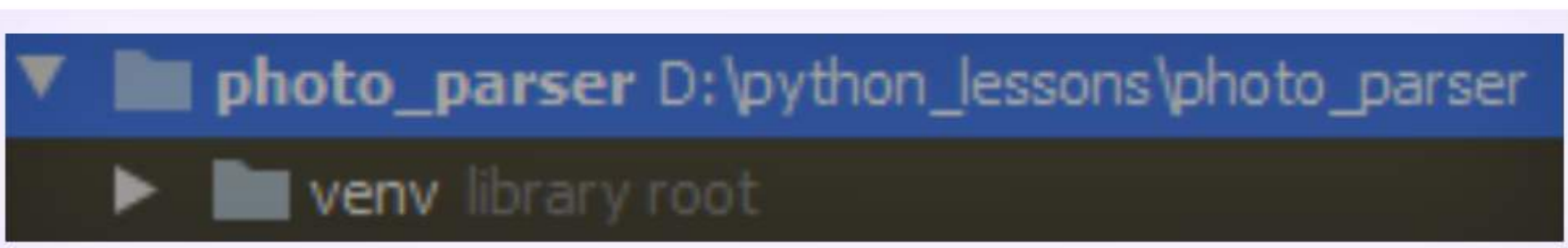
Данный парсер уже был вами пройден в качестве практики в уроке 4 главы 6, поэтому в рамках данной главы продублирую его в качестве повторения.

Давайте закрепим полученные знания на практике и напишем парсер картинок с сайта: `istockphoto.com`. Это стоковый вебсайт, который отдаёт миниатюры фотографий без ватермарок. Фотографии как правило имеют разрешение 400 на 300 пикселей и выше. Соответственно, их можно использовать, например, для создания баннеров и для прочих маркетинговых нужд.

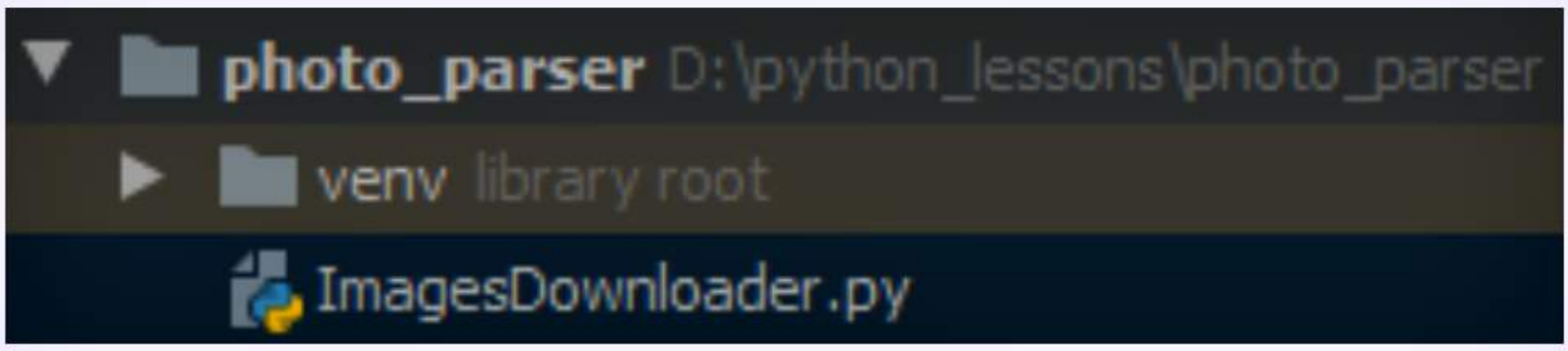
Функционал программы

В нашу программу мы будем передавать ключевой запрос и количество страниц выдачи, которые необходимо сохранить. Например, мы вводим запрос BMW, и количество страниц равное 5. Наша программа должна скачать фотографии с первых пяти страниц, по запросу BMW.

Создадим проект и назовём его `photo_parser`:



Создадим в проекте новый модуль, назовём его `ImagesDownloader.py`:



Установим в проект библиотеки Requests и Beautiful Soup, как мы делали это в прошлых уроках.

Импортируем в наш модуль библиотеки:

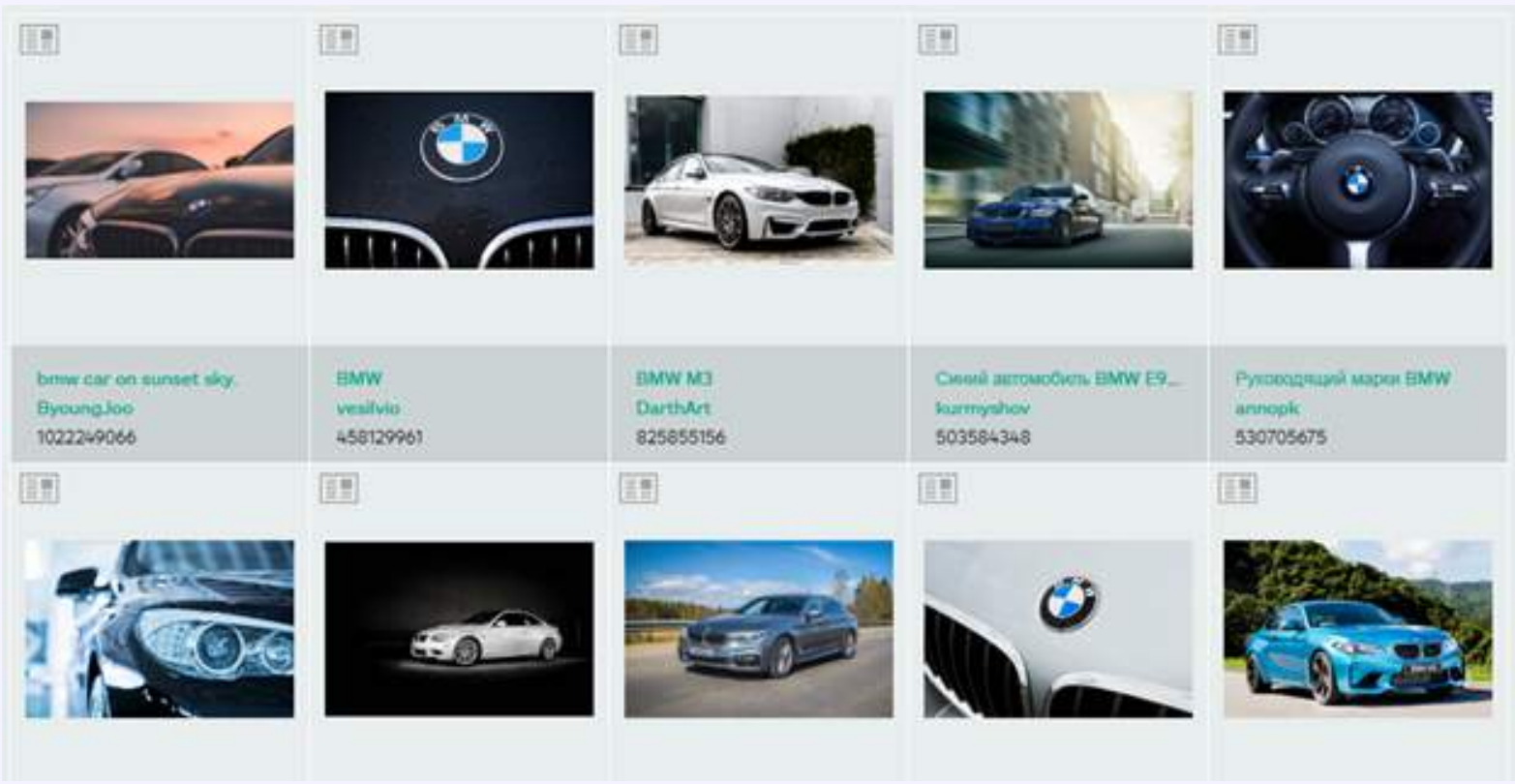
```
import shutil
import requests
import urllib.parse
from bs4 import BeautifulSoup as BS4
```

Библиотека `shutil` будет нужна для сохранения картинок, а `urllib` для того, чтобы обычные запросы, введённые пользователем специальным образом кодировались для передачи через `url`. Эти библиотеки уже включены в вашем виртуальном окружении, их устанавливать не нужно.

Пишем парсер

Сперва перейдём на сайт, введём запрос и посмотрим, как он работает. Введём некоторый запрос и нажмём на кнопку поиска.

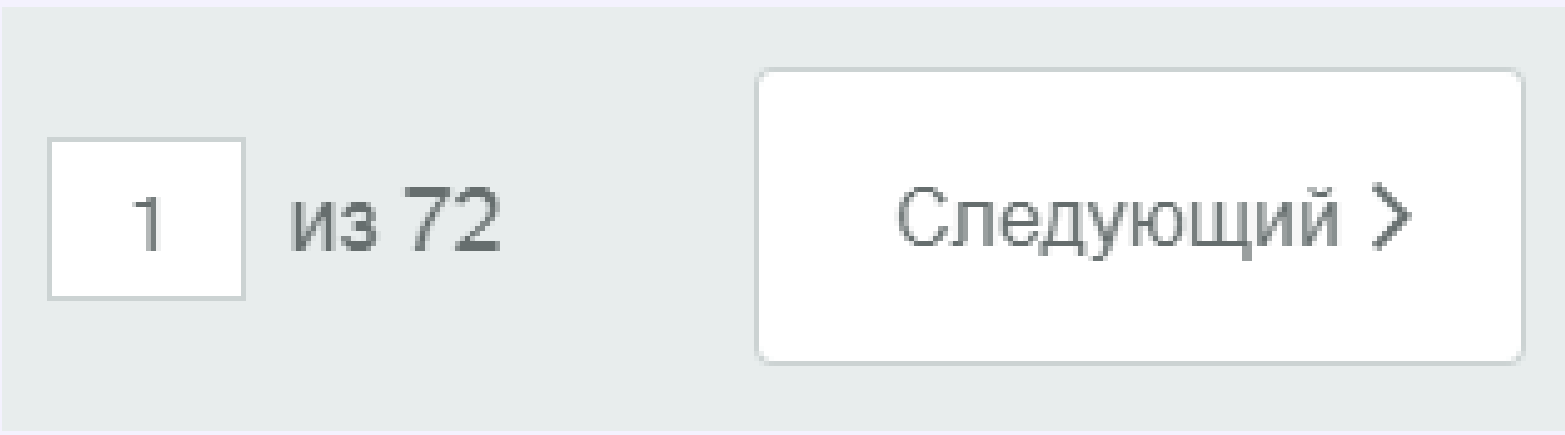
Откроется страница с изображениями:



Но мы обратим внимание на адрес страницы, данные с которой необходимо получить: https://www.istockphoto.com/ru/фотографии/bmw?phrase=bmw&sort=mostpopular

Как видите, основной динамический параметр здесь – это запрос: `bmw`, который тут дублируется, как часть url: `фотографии/bmw` и как часть get-параметра: `phrase=bmw`

Но как получить фотографии с других страниц? Давайте прокрутим страницу до конца, и перейдём на следующую страницу:



Открывается новая страница, смотрим на её адрес: https://www.istockphoto.com/ru/фотографии/bmw?page=2&phrase=bmw&sort=mostpopular

Видим, что появился новый параметр: `page=2`

Соответственно, можно сделать вывод, что чтобы получить заданное количество фотографий по ссылке, достаточно запустить цикл на количество итераций соответствующее количеству требуемых страниц, отправляя запрос для каждой страницы через `requests` получаем html-код страницы, затем извлекаем список ссылок на изображения через `Beautiful Soup` и сохраняем эти страницы.

Метод get_page. Получаем код страницы с изображениями

Данный метод будет получать html-код заданной страницы в выдаче по заданному запросу:

```
def get_page(phrase, page):
    # Определяем ссылку на страницу, к которой будем отправлять запрос
    link = "https://www.istockphoto.com/ru/%D1%84%D0%BE%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%B8/" + phrase
    # Определяем словарь с заголовками

    headers = {'user-agent': 'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.122 Safari/537.36'}
```

```
    # Определяем словарь с параметрами запроса
    params = {"page": page, "phrase": phrase, "sort": "mostpopular"}
    # Отправляем запрос на страницу с картинками
    request = requests.get(link, headers=headers, params=params)
    # Возвращаем HTML-код из функции
    return request.text
```

Тут всё стандартно, как и было в уроке по `requests`. Передаём параметры запроса, а именно номер страницы который нам нужен, фраза которую мы ищем и тип сортировки.

Далее, получаем ответ от сервера и возвращаем этот ответ из функции.

Метод is_404

Может сложится такая ситуация, что пользователь задал слишком большое количество страниц и на определённой странице уже нет фотографий. Поэтому, нам необходимо написать функцию проверки наличия фотографий на странице:

```
# Проверяет, есть ли на странице изображения
def is_404(html):
    page = BS4(html, "html.parser")
    # Если на странице ни одного изображения - возвращаем True
    if len(page.select("img.gallery-asset__thumb")) < 0:
        return True
    return False
```

При просмотре кода, мы можем увидеть, что ссылки на изображения содержатся в таких элементах:

```
<img alt="стильные нос в красный спортивный автомобиль-седан bmw - bmw стоковые фото и изображения" class="gallery-asset__thumb gallery-fixed-asset__thumb gallery-fixed-asset__thumb--null" ng-class="thumbClassNames" src="https://media.istockphoto.com/photos/stylish-nose-of-red-bmw-sports-se…6&m=157772682&s=612x612&w=0&h=8uhaU53p1qZEqgT8Qc3SAU3L1C2XRAv667HQFHyE42g=" style="opacity: 1;"> == $0
```

Если на странице нет ни одного элемента, `img` с классом `gallery-asset__thumb`, значит соответственно на этой странице нет фотографий. Этот метод мы будем использовать дальше.

Метод get_imgs_from_page

Данный метод будет извлекать все ссылки с отдельной страницы:

```
# Получает список ссылок с конкретной страницы из выдачи по запросу
def get_imgs_from_page(phrase, page):
    # Получаем HTML-код конкретной страницы выдачи
    html = get_page(phrase, page)

    # Определяем локальный список, в который будут помещаться ссылки на изображение
    images = []

    # Если вернуло страницу 404, значит прерываем выполнение функции, возвращая False
    if is_404(html) == True:
        return False

    # Загружаем код страницы полученной с сервера в парсер
    img_node = BS4(html, "html.parser")
    # Извлекаем все элементы со ссылками на картинки
    imgs = img_node.select("img.gallery-asset__thumb")
    # Извлекаем из элементов ссылки на картинки и добавляем их в список
    for img in imgs:
        if img.has_attr("src"):
            print(f"Получили ссылку на фото: {img['src']}")
            images.append(img["src"])

    # Возвращаем список из функции
    return images
```

1. При помощи функции get_page мы получаем html-код с картинками с конкретной страницы.
2. Проверяем, есть ли на этой странице вообще изображения. Если изображений нет – заканчиваем выполнение функции, возвращаем значение False.
3. Загружаем html-код страницы в парсер.
4. Извлекаем все элементы содержащие изображения.
5. Извлекаем из этих элементов ссылки на изображения и размещаем их в список.
6. Возвращаем список с изображениями из функции.

Наш софт теперь умеет извлекать ссылки со страницы выдачи, теперь нужен метод, который извлекает со всех заданных страниц выдачи.

Метод get_images

Проходит в цикле необходимое количество итераций и на каждую итерацию скачивает последующую страницу из выдачи по запросу:

```
# Получает список ссылок на изображения с заданного количества страниц по запросу
def get_images(query, pages):
    # Переводим ключевой запрос в приемлемый формат
    query = urllib.parse.quote(query)
    # Создаём список, в который будут помещаться все ссылки
    images = []
    # Запускаем цикл, для парсинга по страницам
    for i in range(pages):
        # Так как цикл начинается с 0, определяем номер страницы
        num_of_page = i + 1
        # Получаем все изображения со страницы
        img = get_imgs_from_page(query, num_of_page)

        # Если в переменной False, значит больше страниц с
        # фотографиями нет прерываем цикл
        if not img:
            break
        else:
            images += img

    return images
```

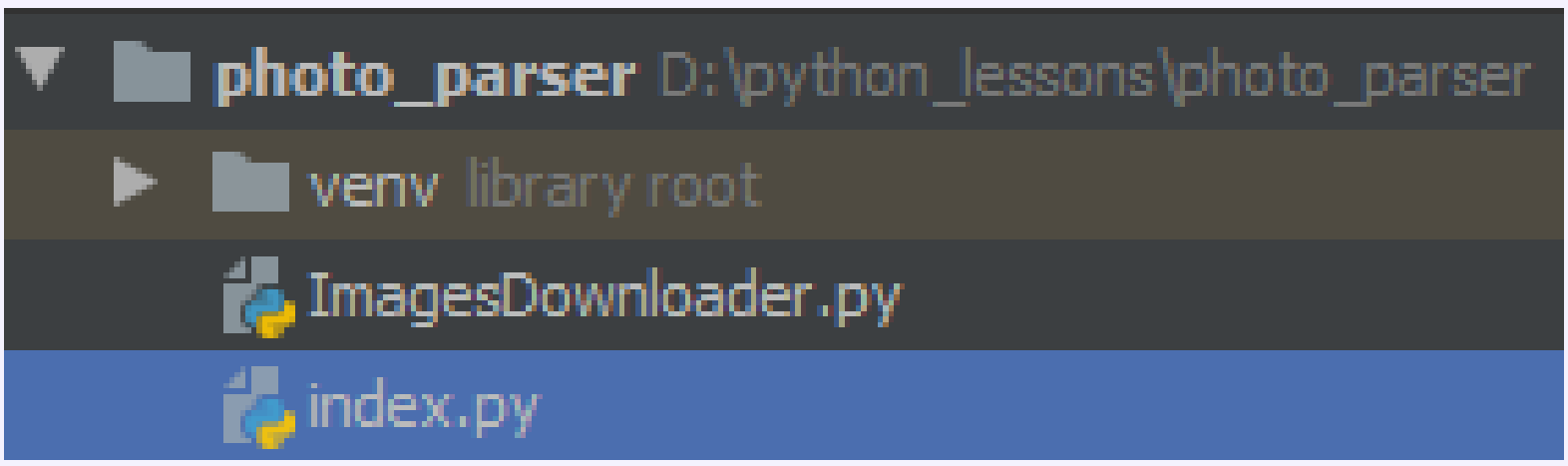
Сперва подготавливаем запрос. Когда мы формируем ссылку – текст, передаваемый в неё необходимо обработать, чтобы избежать шибок.

Запускаем цикл на заданное количество страниц, если мы задали спарсить 5 первых страниц из выдачи, значит задаём циклу отработать 5 итераций, и на каждой итерации скачиваем страницу, от страницы номер 1 до заданной. Так как цикл начинается с нуля, а нумерация страниц с единицы, на каждой итерации прибавляем к номеру текущей итерации одну единицу.

Проверяем, если на текущей итерации запрос вернул False, значит больше нет страниц с изображениями и необходимо прервать цикл.

Возвращаем все изображения, которые мы спарсили из выдачи.

Создадим ещё один модуль, назовём его index.py и вызовем в нём эту функцию, посмотрим, как она работает:



Импортируем наш модуль и вызовем метод `get_images`:

```
import ImagesDownloader
result = ImagesDownloader.get_images("money", 2)
print(f"Получили ссылки на: {len(result)} фотографий")
```

Выводит:

> … Получили ссылку на фото: https://media.istockphoto.com/vectors/vintage-money-font-with-shadow-vector-id500173082?k=6&m=500173082&s=612x612&w=0&h=bhIZiKhrXnu2snShd7cInAXYG7sfoxlZte6mSglKd2Q= Получили ссылку на фото: https://media.istockphoto.com/vectors/money-stack-of-us-dollars-vector-id909164260?k=6&m=909164260&s=612x612&w=0&h=yGgLOmvzzxf1rec71a2gg4breZNtN7ssnGUDdFOUuP8= Получили ссылку на фото: https://media.istockphoto.com/photos/business-audits-using-a-calculator-financial-data-investment-fund-at-picture-id1129810557?k=6&m=1129810557&s=612x612&w=0&h=uEl1W2RLjhjNrlRYNMBSGd5vxOVCdmP_VJVGPPiyQPA= Получили ссылки на: 120 фотографий

Вывел список всех фотографий в консоль, а также посчитал, сколько всего ссылок было получено.

Отлично, мы научили наш код получать ссылки на изображения. Теперь осталось, написать модуль для сохранения картинок.

Метод save_image

Сохраняет изображение в заданную папку. Мы указываем папку, в которую нужно сохранить изображение и ссылку на него:

```python
# Сохраняет отдельное изображение
def save_image(folder, link):
    # Получаем исходный код изображения
    image = requests.get(link, stream=True)

    # Извлекаем из ссылки, например:
    # https://media.istockphoto.com/vectors/finance-money-icons-vector-id1030878592?k=6&m=1030878592&s=612x612&w=0&h=HXSWx0FW-Re64cLJWL-LrbMx8Zx4rfDAC
    # Название изображения, например: finance-money-icons-vector-id1030878592
    filename = link.split("/")[4]
    filename = filename.split("?")[0]
    # Формируем путь до файла вида: папка/имя_файла.jpg
    path_to_file = f"{folder}/{filename}.jpg"
    print(f"Сохранили фото: {path_to_file}")
    # Сохраняем фото в файл
    with open(path_to_file, "wb") as file_obj:
        shutil.copyfileobj(image.raw, file_obj)
```

Передаём ссылку на изображение, скачиваем его с сервера в переменную. Извлекаем название изображения, далее сохраняем содержимое изображения в новый файл.

Теперь давайте напишем финальный метод, который будет скачивать все изображения по конкретному запросу в заданную папку.

Метод download_images

Получает список изображений по запросу, далее в цикле скачивает каждое изображение:

```python
# Скачивает все изображения по запросу в заданную папку
def download_images(folder, phrase, page):
    # Получаем все ссылки из выдачи по запросу
    images = get_images(phrase, page)
    # Сохраняем все ссылки
    for image in images:
        # Запускаем функцию сохранения в
        try:
            save_image(folder, image)
        except Exception as error:
            print(f"Проблема с записью файла: {error}")
```

Передаём папку в которую необходимо скачать изображение, далее запрос по которому эти изображения нужно спарсить и количество страниц по запросу, которые нужно спарсить.

Получаем все ссылки на изображения при помощи метода `get_images`. Далее проходим все ссылки в цикле и сохраняем каждую при помощи метода `save_image`.

Теперь, давайте запустим данный метод в нашем модуле `index.py`:

```python
import ImagesDownloader
ImagesDownloader.download_images("C:/Users/Moneymaker/Desktop/img", "money", 2)
```

После запуска в лог будет выведена информация о ходе выполнения скрипта. В папке, которую мы указали, начнут появляться изображения. Вот, собственно и все, наш парсер успешно работает. И теперь при помощи двух строчек кода можно скачивать большое количество стоковых изображений для ваших маркетинговых нужд.

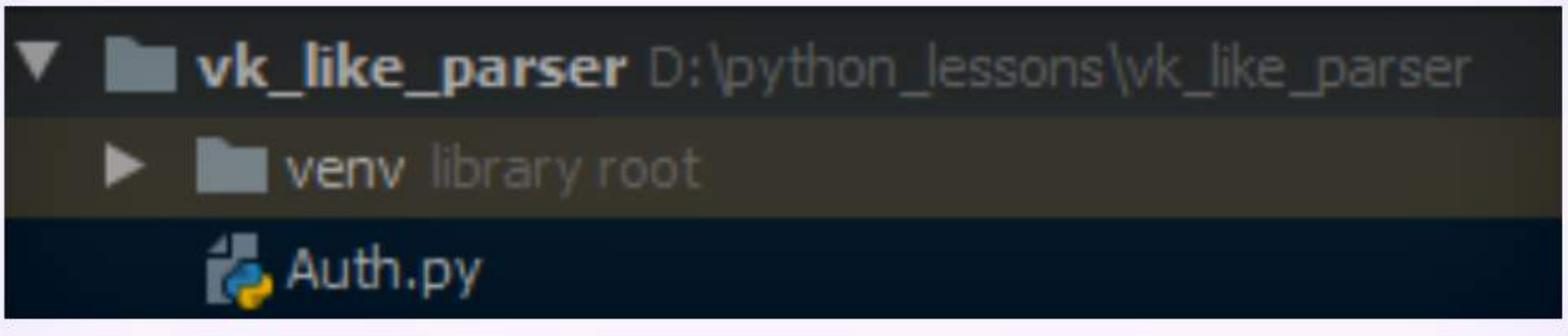
VK API. Пишем модуль парсинга лайков к постам из групп ВК

Содержание урока

Давайте автоматизируем сбор лайков у постов в сообществах VK при помощи их API. Этот функционал нами уже разрабатывался в 7 уроке 6 главы, но тогда мы реализовали его в виде функций, теперь же напишем его объектную версию.

Модуль для авторизации

Создадим проект, назовём его, `vk_like_parser`. Установим в него библиотеку `requests`. В корневой директории проекта создадим модуль `Auth`, в нём будет размещена логика класса для авторизации пользователя:



Импортируем в данный модуль модули `requests`, `math` и `json`:

```
import json
import math

import requests
```

Объявим класс и определим в нём конструктор:

```
class Auth:

    def __init__(self, login, password):
        # Инициализируем значение логина
        self.__login = login
        # Инициализируем значение парооля
        self.__password = password
        # Авторизуемся и сохраняем токен
        self.token = self.__direct_auth()
```

Таким образом, мы создаём объект класса сразу передавая ему логин и пароль и уже при создании класса происходит авторизация. Теперь давайте реализуем сам метод, при помощи которого мы произведём авторизацию и получение токена.

Напишем метод, который реализует прямую авторизацию:

```
def __direct_auth(self):
    # Формируем массив параметров для авторизации
    params = {
                "grant_type": "password",
                "client_id": "2274003",
                "client_secret": "hHbZxrka2uZ6jB1inYsH",
                "username": self.__login,
                "password": self.__password,
                "2fa_supported": "1",
                "scope":
        "notify,friends,photos,audio,video,stories,pages,status,notes,messages,wall,ads,offline,docs,groups,notifications,stats,email,market,
        stories,photos,app_widget,messages,docs,manage",
                "v": "5.92",
    }
    # Отправляем запрос на сервер
    result = requests.get("https://oauth.vk.com/token", params=params)
    # Декодируем результат в словарь
    response = json.loads(result.text)
    # Если сервер ответил и в ответе есть токен - возвращаем его
    if result.status_code == 200 and "access_token" in response.keys():
        return response["access_token"]
    # Иначе возвращаем False
    return False
```

Функционал реализован. Его логика следующая: при создании объекта мы передаём в него логин и пароль, после этого в конструкторе автоматически вызывается функция для авторизации и записывает токен в публичную переменную (`self.token`), значение которой можно получить сразу после создания объекта.

Полный листинг класса:

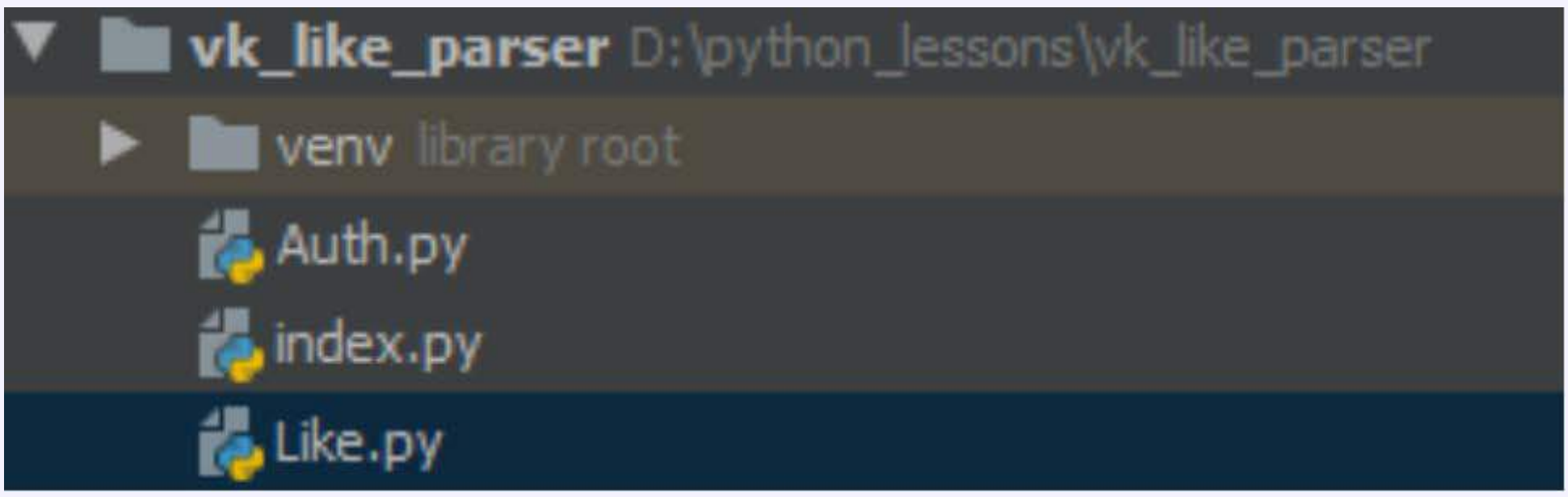
```
import json
import requests
```

```
class Auth:

    def __init__(self, login, password):
        # Инициализируем значение логина
        self.__login = login
        # Инициализируем значение парооля
        self.__password = password
        # Авторизуемся и сохраняем токен
        self.token = self.__direct_auth()

    def __direct_auth(self):
        # Формируем массив параметров для авторизации
        params = {
                        "grant_type": "password",
                        "client_id": "2274003",
                        "client_secret": "hHbZxrka2uZ6jB1inYsH",
                        "username": self.__login,
                        "password": self.__password,
                        "2fa_supported": "1",
                        "scope":
            "notify,friends,photos,audio,video,stories,pages,status,notes,messages,wall,ads,offline,docs,groups,notifications,stats,email,market,st
            ories,photos,app_widget,messages,docs,manage",
                        "v": "5.92",
        }
        # Отправляем запрос на сервер
        result = requests.get("https://oauth.vk.com/token", params=params)
        # Декодируем результат в словарь
        response = json.loads(result.text)
        # Если сервер ответил и в ответе есть токен - возвращаем его
        if result.status_code == 200 and "access_token" in response.keys():
            return response["access_token"]
        # Иначе возвращаем False
        return False
```

Теперь у нас есть класс для авторизации, и мы можем работать с АПИ. А именно, написать метод по получению списка пользователей лайкнувших тот или иной пост. Мы уже работали с этим методом в главе 6, но теперь давайте перепишем его в виде класса.

Модуль для лайкинга

Создадим модуль Like, и в нём класс Like:



Импортируем в данный модуль модули requests и json:

```
import json
import requests
```

Объявляем класс и конструктор класса:

```python
class Like:

    def __init__(self, token):
        self.__token = token
```

В данном случае, при создании объекта мы обязаны поместить в него токен, при помощи которого будут отправляться запросы.

Для получения списка пользователей, лайкнувших некоторый пост необходимо воспользоваться методом: `likes.getList`

Документация метода: https://vk.com/dev/likes.getList

Из 7 урока 6 главы мы знаем, что данные для лайка, а именно `id` паблика и `id` поста берутся из ссылки на сам пост.

Поэтому напишем метод, который извлекает из ссылки эти данные для запроса:

```python
def __get_link(self, link):
    # Разбиваем ссылку вида: https://vk.com/feed?w=wall-87651722_52363
    # на 2 части (1) https://vk.com/feed?w= и (2) -87651722_52363
    # заносим вторую часть (-87651722_52363) в переменную
    id = link.split("wall")[1]
    # Разбиваем id поста вида: -87651722_52363
    # на 2 части (1) -87651722 и (2) 52363
    # и возвращаем их в виде списка
    return id.split("_")
```

Теперь напишем метод, который будет отправлять запрос на сервер и возвращать список пользователей лайкнувших пост:

```python
def get(self, link, count):

    # Извлекаем данные из ссылки
    owner_id, item_id = self.__get_link(link)

    # Формируем словарь с параметрами для запроса
    params = {
        "type": "post",
        "owner_id": owner_id,
        "item_id": item_id,
        "count": 1000,
        "offset": 0,
        "access_token": self.__token,
        "v": "5.92",
    }
    # Определяем сколько запросов отправим
    count_of_iterations = math.ceil(count/1000)
    # Определяем список пользователей
    likes = []
    # Запускаем цикл по количеству итераций
    for i in range(count_of_iterations):
        # Отправляем запрос к API
        result = requests.post("https://api.vk.com/method/likes.getList", params=params)
        # Десериализуем ответ из JSON в словарь
        response = json.loads(result.text)
        # На каждой итерации, смещаем запрос на 1000 новых пользователей
        params["offset"] += 1000
        # Если сервер успешно ответил и в ответе нет ошибок
        if result.status_code == 200 and "response" in response.keys():
            # Если запрос вернул нам не пустой список
```

```
            if response["response"]["items"]:
                # Добавляем его в общий список
                likes.extend(response["response"]["items"])
            # Иначе лайков больше нет и нужно прервать цикл
            else:
                # Возвращаем список пользователей лайкнувших пост
                return likes
        else:
            # В любом случае возвращаем список пользователей лайкнувших пост
            return likes
    # Возвращаем полный список пользователей лайкнувших пост
    return likes
```

Логика данного метода следующая:

1. Передаём ссылку на пост и количество лайков, которое хотим спарсить.
2. Из ссылки извлекаем данные для запроса.
3. Определяем, сколько запросов нам совершить, чтобы извлечь заданное количество лайков (каждый запрос кратен 1000)
4. Запускаем цикл по количеству запросов. В рамках каждого прохода цикла получаем список лайков.
5. Проверяем, если список лайков пустой – значит мы получили все лайки у поста.
6. Выходим из цикла и возвращаем список с лайками.

Приведу полный листинг класса:

```
import json
import math

import requests


class Like:

    def __init__(self, token):
        self.__token = token

    def get(self, link, count):

        # Извлекаем данные из ссылки
        owner_id, item_id = self.__get_link(link)

        # Формируем словарь с параметрами для запроса
        params = {
            "type": "post",
            "owner_id": owner_id,
            "item_id": item_id,
            "count": 1000,
            "offset": 0,
            "access_token": self.__token,
            "v": "5.92",
        }
        # Определяем сколько запросов отправим
        count_of_iterations = math.ceil(count/1000)
        # Определяем список пользователей
        likes = []
        # Запускаем цикл по количеству итераций
        for i in range(count_of_iterations):
            # Отправляем запрос к API
            result = requests.post("https://api.vk.com/method/likes.getList", params=params)
            # Десериализуем ответ из JSON в словарь
            response = json.loads(result.text)
            # На каждой итерации, смещаем запрос на 1000 новых пользователей
            params["offset"] += 1000
            # Если сервер успешно ответил и в ответе нет ошибок
            if result.status_code == 200 and "response" in response.keys():
```

```
                # Если запрос вернул нам не пустой список
                if response["response"]["items"]:
                    # Добавляем его в общий список
                    likes.extend(response["response"]["items"])
                # Иначе лайков больше нет и нужно прервать цикл
                else:
                    # Возвращаем список пользователей лайкнувших пост
                    return likes
            else:
                # В любом случае возвращаем список пользователей лайкнувших пост
                return likes
        # Возвращаем полный список пользователей лайкнувших пост
        return likes

    def __get_link(self, link):
        # Разбиваем ссылку вида: https://vk.com/feed?w=wall-87651722_52363
        # на 2 части (1) https://vk.com/feed?w= и (2) -87651722_52363
        # заносим вторую часть (-87651722_52363) в переменную
        id = link.split("wall")[1]
        # Разбиваем id поста вида: -87651722_52363
        # на 2 части (1) -87651722 и (2) 52363
        # и возвращаем их в виде списка
        return id.split("_")
```

Пишем контроллер

Напишем контроллер:

```
from Auth import Auth
from Like import Like

# Создаём объект
likes = Like(Auth("login", "password").token)
# Получаем список пользователей лайкнувших определённый пост
users = likes.get("https://vk.com/apostils?w=wall-152496414_5315", 5000)

if users:
    # Выводим поличество полученных пользователей
    print(len(users))
    # Выводим список полученных пользователей
    print(users)
else:
    print("При парсинге произощла проблема")
```

Устанавливаем веб-сервер под Windows

Содержание урока

Принцип работы программы

Для работы с API ВК, как мы знаем, требуется специальный токен. Получить его нужно один раз для каждого аккаунта, а затем пользоваться неограниченное количество времени, при том, менять его нужно только если вы по какой-то причине сгенерировали новый токен для этого аккаунта.

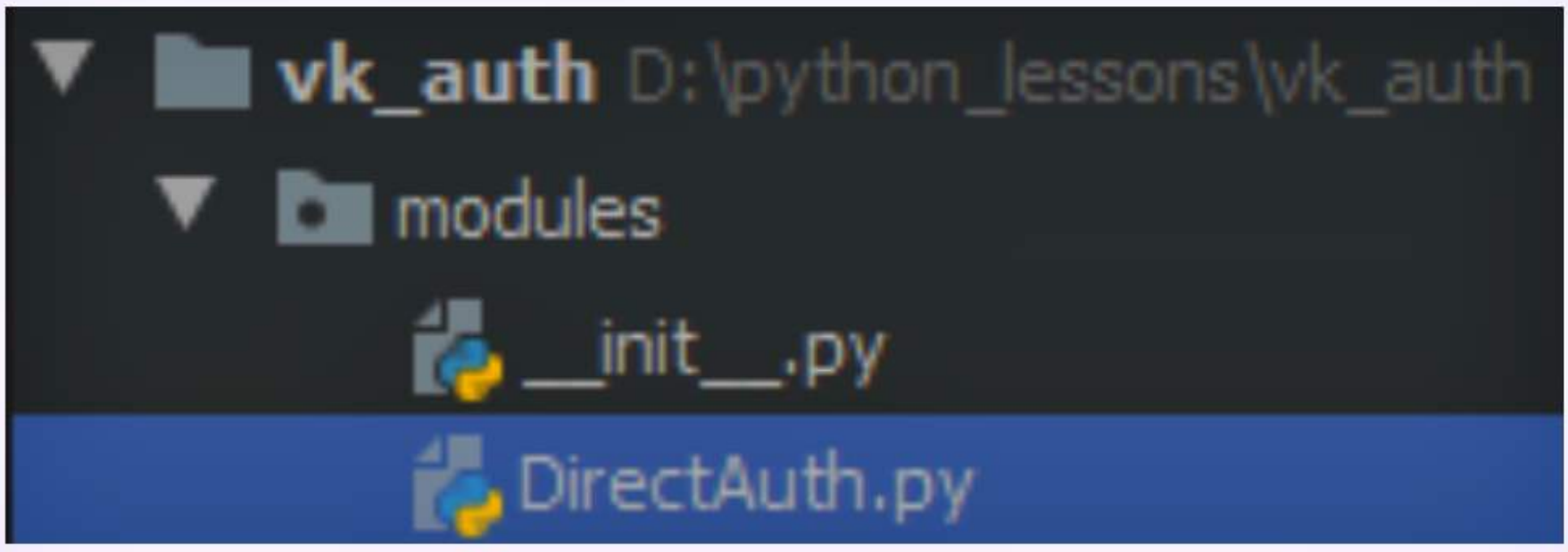
В данном уроке мы разберём чуть более продвинутый класс для авторизации с возможностью вводить каптчу, в том случае если ВК выбрасывает проверку. Дело в том, что иногда при прямой авторизации ВК не принимает логин и пароль сразу, а возвращает ошибку, при которой нужно ввести каптчу.

Вместе с этой ошибкой приходит ссылка на изображение с каптчей, а также её ID. Соответственно, необходимо открыть изображение, разгадать каптчу и послать повторный запрос на авторизацию, но вместе с ним передать 2 дополнительных параметра: ID каптчи и её текст.

Это практически полная копия класса `Auth` из предыдущего урока, но с немного изменённой логикой обработки возвращаемого ответа от сервера. Теперь, если сервер возвращает статус `need_captcha` мы запустим написанную нами функцию для разгадывания каптчи.

Пишем модуль авторизации

Создадим проект, назовём его `vk_auth`. В нём создадим пакет modules, внутри которого создадим `DirectAuth`:



Установим библиотеку `requests`. Далее, в модуль `DirectAuth` импортируем:

```
import json
import requests
```

Создаём класс:

```
class DirectAuth:
```

Главные условия для авторизации – это логин и пароль, поэтому их обязательно установить в качестве параметров конструктора:

```
def __init__(self, login, password):
    # Инициализируем значение логина
    self.__login = login
    # Инициализируем значение парооля
    self.__password = password
    # Авторизуемся и сохраняем токен
    auth_obj = self.__direct_auth()
    if "access_token" in auth_obj.keys():
        self.token = auth_obj["access_token"]
    elif "error" in auth_obj.keys() and auth_obj["error"] == "need_captcha":
        self.token = self.__resolve_captcha(auth_obj)
    else:
        # В остальных случаях возвращаем False
        self.token = False
        # Выводим сообщение о непредвиденной ошибке
        print(f"Непредвиденная ошибка: {auth_obj}")
```

При этом, процесс авторизации инициируется непосредственно в кнструкторе. Мы запускаем функцию `direct_auth`, которая производит авторизацию по логину и паролю. Если в ответе нет авторизационного токена – посылаем ответ в функцию `resolve_captcha`, которая и будет с нашей помощью разгадывать каптчу. Если токен пришёл, то возвращаем его в переменную `token`.

В конструкторе используется две функции:

1. `direct_auth` – производит авторизацию
2. `resolve_captcha` – разгадывает куптчу

Давайте реализуем их. Начнём с функции авторизации:

```
def __direct_auth(self, captcha_sid=None, captcha_key=None):
    # Формируем массив параметров для авторизации
    params = {
        "grant_type": "password",
        "client_id": "2274003",
        "client_secret": "hHbZxrka2uZ6jB1inYsH",
        "username": self.__login,
        "password": self.__password,
        "2fa_supported": "1",
        "captcha_sid": captcha_sid,
        "captcha_key": captcha_key,
        "scope": "notify,friends,photos,audio,video,stories,pages,status,notes,messages,wall,ads,offline,docs,"
                 "groups,notifications,stats,email,market,stories,photos,app_widget,messages,docs,manage",
        "v": "5.92",
    }
    # Отправляем запрос на сервер
    result = requests.get("https://oauth.vk.com/token", params=params)
```

```
    # Декодируем результат в словарь
    response = json.loads(result.text)
    # Если сервер ответил и в ответе есть токен - возвращаем его
    return response
```

Она практически не изменилась с прошлого урока, здесь я лишь убрал проверки и просто возвращаю ответ сервера, который анализируется в дальнейшем. А также добавил два параметра, которые передаются на сервер, это `id` каптчи и текст каптчи.

Эти параметры не обязательные, так что они срабатывают только в том случае, если вы явно передаёте данные для разгадывания каптчи.

Самая интересная часть. Функция разгадывания каптчи:

```
def __resolve_captcha(self, auth_obj):
    # Выводим на экран сообщение о том, что нужно разгадать каптчу
    print(f"Каптча! Введите текст с изображения: {auth_obj['captcha_img']}")
    # Вводим каптчу из консоли
    captcha = input("Код: ")
    # ID каптчи получаем из объекта с ответом
    captcha_sid = auth_obj["captcha_sid"]
    # Авторизуемся передавая параметры каптчи
    auth_obj = self.__direct_auth(captcha_sid, captcha)
    # Если после авторизации с каптчей вернулся токен
    if "access_token" in auth_obj.keys():
        # Возвращаем токен
        return auth_obj["access_token"]
    # Если каптча была введена неверно
    elif "error" in auth_obj.keys() and auth_obj["error"] == "need_captcha":
        # То запускаем эту же функцию передавая ей новые данные ответа
        self.__resolve_captcha(auth_obj)
    # В остальных случаях, возвращаем False
    print(f"Непредвиденная обишка: {auth_obj}")
    return False
```

Если мы запускаем функцию (передавая ей объект с ошибкой `need_captcha`), то в терминал выводится сообщение:

> Каптча! Введите текст с изображения: ссылка_на_изображение

Далее, терминал ожидает ввода текста с изображения:

```
captcha = input("Код: ")
```

Который вы должны ввести.

После этого, когда вы ввели текст с изображением, отправляется повторный запрос на авторизацию:

```
auth_obj = self.__direct_auth(captcha_sid, captcha)
```

Далее, если после этого автори зация прошла успешно – возвращаем токен. Если снова возвращается ошибка `need_captcha`, значит вы неправильно ввели каптчу, поэтому данная функция запускает саму себя и снова требует введения каптчи:

```
# Если каптча была введена неверно
elif "error" in auth_obj.keys() and auth_obj["error"] == "need_captcha":
    # То запускаем эту же функцию передавая ей новые данные ответа
    self.__resolve_captcha(auth_obj)
```

И так будет до тех пор, пока вы не введёте правильный ответ. Если же вернулась какая-то другая ошибка – возвращаем `False` и прописываем в консоль сообщение об этом.

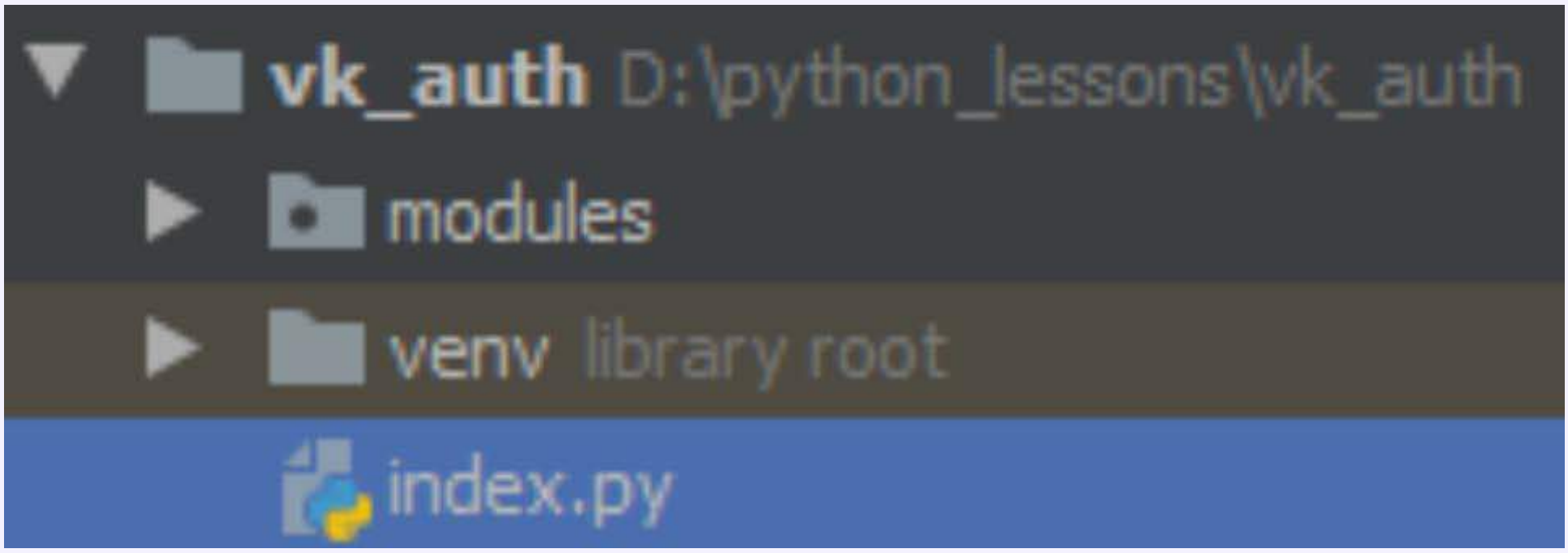
Полный листинг класса:

```
import json
import requests


class DirectAuth:

    def __init__(self, login, password):
        # Инициализируем значение логина
        self.__login = login
        # Инициализируем значение парооля
        self.__password = password
        # Авторизуемся и сохраняем токен
        # auth_obj = self.__direct_auth()
        auth_obj = json.loads('{"error":"need_captcha","captcha_sid":"854844498568","captcha_img":"http:\/\/api.vk.com\/
    captcha.php?sid=854844498568&s=1"}')
        if "access_token" in auth_obj.keys():
            self.token = auth_obj["access_token"]
        elif "error" in auth_obj.keys() and auth_obj["error"] == "need_captcha":
            self.token = self.__resolve_captcha(auth_obj)
        # В остальных случаях возвращаем False
        self.token = False
        # Выводим сообщение о непредвиденной ошибке
        print(f"Непредвиденная ошибка: {auth_obj}")

    def __resolve_captcha(self, auth_obj):
        # Выводим на экран сообщение о том, что нужно разгадать каптчу
        print(f"Каптча! Введите текст с изображения: {auth_obj['captcha_img']}")
        # Вводим каптчу из консоли
        captcha = input("Код: ")
        # ID каптчи получаем из объекта с ответом
        captcha_sid = auth_obj["captcha_sid"]
        # Авторизуемся передавая параметры каптчи
        auth_obj = self.__direct_auth(captcha_sid, captcha)
        # Если после авторизации с каптчей вернулся токен
        if "access_token" in auth_obj.keys():
            # Возвращаем токен
            return auth_obj["access_token"]
        # Если каптча была введена неверно
        elif "error" in auth_obj.keys() and auth_obj["error"] == "need_captcha":
            # То запускаем эту же функцию передавая ей новые данные ответа
            self.__resolve_captcha(auth_obj)
        # В остальных случаях, возвращаем False
        print(f"Непредвиденная обишка: {auth_obj}")
        return False

    def __direct_auth(self, captcha_sid=None, captcha_key=None):
        # Формируем массив параметров для авторизации
        params = {
            "grant_type": "password",
            "client_id": "2274003",
            "client_secret": "hHbZxrka2uZ6jB1inYsH",
            "username": self.__login,
            "password": self.__password,
            "2fa_supported": "1",
```

```
            "captcha_sid": captcha_sid,
            "captcha_key": captcha_key,
            "scope": "notify,friends,photos,audio,video,stories,pages,status,notes,messages,wall,ads,offline,docs,"
                     "groups,notifications,stats,email,market,stories,photos,app_widget,messages,docs,manage",
            "v": "5.92",
        }
            # Отправляем запрос на сервер
            result = requests.get("https://oauth.vk.com/token", params=params)
            # Декодируем результат в словарь
            response = json.loads(result.text)
            # Если сервер ответил и в ответе есть токен - возвращаем его
            # return response
            return json.loads('{"error":"need_captcha","captcha_sid":"854844498568","captcha_img":"http:\/\/
    api.vk.com\/captcha.php?sid=854844498568&s=1"}')
```

Пишем контроллер

В корневой директории создадим модуль `index.py`:



Пропишем в нём следующий код:

```
from modules.DirectAuth import DirectAuth

direct_auth = DirectAuth("login", "password")
print(direct_auth.token)
```

Запустив этот код, мы можем получить либо токен, если всё прошло успешно:

```
D:\python_lessons\vk_auth\venv\Scripts\python.exe D:/python_lessons/vk_auth/index
90dba75759131a26b5b1f3dd35fb98d8f30e9e427025cdc979eb7ee1480b64cd591594bb6194085db
```

Либо собщение о том, что нужно ввести каптчу:

```
D:\python_lessons\vk_auth\venv\Scripts\python.exe D:/python_lessons/vk_auth/index.py
Каптча! Введите текст с изображения: http://api.vk.com/captcha.php?sid=8548444985688s=1
Код:
```

В таком случае, переходим по ссылке и вводим код с изображения в терминал:

```
D:\python_lessons\vk_auth\venv\Scripts\python.exe D:/python_lessons/vk_auth/index.py
Каптча! Введите текст с изображения: http://api.vk.com/captcha.php?sid=8548444985688s=1
Код: h5yzui87
```

Нажимаем «Enter». Если каптча введена правильно, то скорее всего придёт токен. Если неправильно, то придёт новая ссылка на каптчу, и вы должны ввести ответ повторно:

```
D:\python_lessons\vk_auth\venv\Scripts\python.exe D:/python_lessons/vk_auth/index.py
Каптча! Введите текст с изображения: http://api.vk.com/captcha.php?sid=8548444985688s=1
Код: h5yzui87dd
Каптча! Введите текст с изображения: http://api.vk.com/captcha.php?sid=8548444985688s=1
Код: ddddd
Каптча! Введите текст с изображения: http://api.vk.com/captcha.php?sid=8548444985688s=1
Код: ddfrerfg
Каптча! Введите текст с изображения: http://api.vk.com/captcha.php?sid=8548444985688s=1
Код:
```