root . withdraw ( )

ПРОГРАММИРОВАНИЕ НА PYTHON®

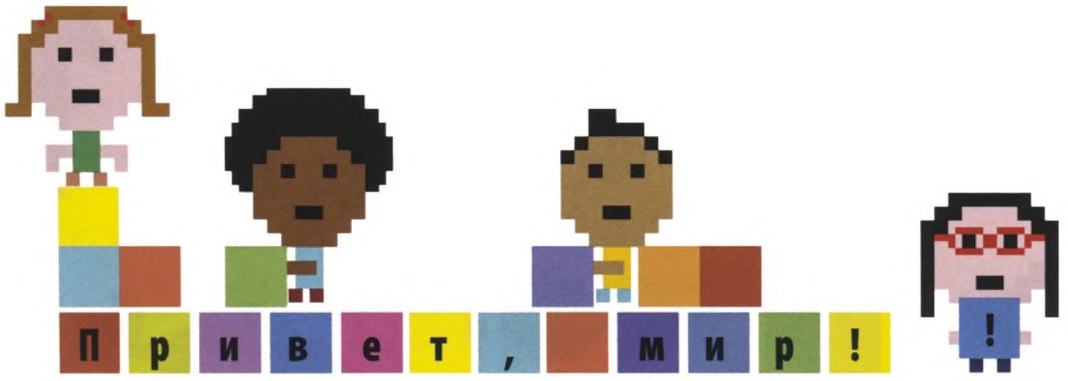
print ()

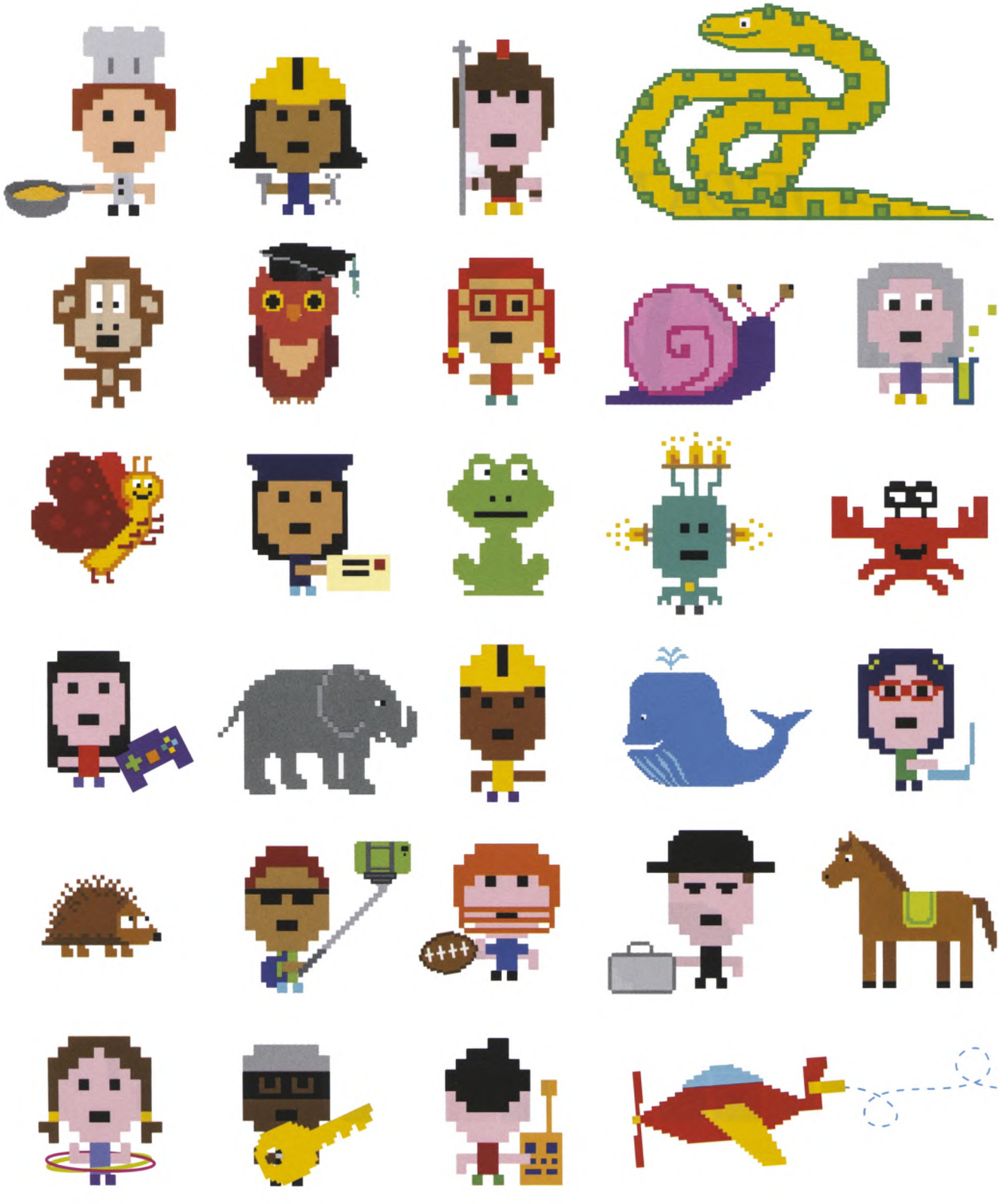
score = score + 1

message = 'Привет!'

* ИЛЛЮСТРИРОВАННОЕ РУКОВОДСТВО ДЛЯ ДЕТЕЙ *

Программирование на Python®

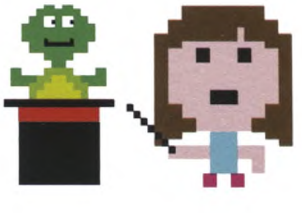




Программирование на Python®

ИЛЛЮСТРИРОВАННОЕ РУКОВОДСТВО ДЛЯ ДЕТЕЙ

Перевод с английского
Станислава Ломакина



Москва
«Манн, Иванов и Фербер»
2018

УДК 087.5:004.43
ББК 39.973.26-018.1
П78

Издано с разрешения Dorling Kindersley Limited

На русском языке публикуется впервые

Возрастная маркировка в соответствии с Федеральным законом № 436-ФЗ: 0+

Авторы: Кэрол Вордерман, Крэйг Стили, Клэр Квигли, Мартин Гудфеллоу, Дэниел Маккафферти, Джон Вудкок

Программирование на Python : иллюстрированное руководство для детей / К. Вордерман, К. Стили, К. Квигли, М. Гудфеллоу, Д. Маккафферти, Дж. Вудкок ; пер. с англ. С. Ломакина. — М. : Манн, Иванов и Фербер, 2018. — 224 с. : ил.

П78

ISBN 978-5-00117-399-1

В этой книге собраны 16 проектов с пошаговыми инструкциями, примерами кодов, блок-схемами и забавными сюжетными иллюстрациями. Следуя подробным указаниям, ребенок освоит базовые принципы программирования на одном из самых популярных компьютерных языков — Python, создаст программы и игры на внимание и быстроту реакции, в которые сам же потом и поиграет. Программы можно дорабатывать и изменять. Эта тренировка подарит уверенность в своих силах и стимул создавать собственные уникальные проекты.

Для детей от 10 лет и взрослых, которые делают первые шаги в программировании.

УДК 087.5:004.43
ББК 39.973.26-018.1

Издание для досуга
Для широкого круга читателей

Кэрол **Вордерман** и др.

ПРОГРАММИРОВАНИЕ НА PYTHON
Иллюстрированное руководство для детей

Главный редактор *Артем Степанов*
Руководитель направления *Анастасия Троян*
Ответственный редактор *Анна Шахова*
Научный редактор *Георгий Гаджиев*
Верстка обложки *Елизавета Краснова*
Верстка *Дмитрий Колесников*
Корректоры *Елена Бреге, Олег Пономарев, Дарья Балтрушайтис*

ООО «Манн, Иванов и Фербер»
www.mann-ivanov-ferber.ru
www.facebook.com/mifdetstvo
www.vk.com/mifdetstvo
www. instagram.com/mifdetstvo

ISBN 978-5-00117-399-1

Все права защищены.
Никакая часть данной книги
не может быть воспроизведена
в какой бы то ни было форме
без письменного разрешения
владельцев авторских прав.

Оригинальное название:
Computer Coding Python Projects for Kids
© 2017 Dorling Kindersley Limited
A Penguin Random House Company
© Издание на русском языке, перевод.
«Манн, Иванов и Фербер», 2018

Подписано в печать 08.02.2018.
Формат 84 × 108/16. Гарнитуры Myriad, Andale Mono. Бумага мелованная.
Печать офсетная. Усл. печ. л. 23,52.
Тираж 5000 экз.

Отпечатано в ТВВ, a. s., Словакия.

EAC

КЭРОЛ ВОРДЕРМАН — одна из самых популярных британских телеведущих, известная своими познаниями в математике, кавалер ордена Британской империи. Вела телешоу, посвященные науке и технике, такие как Tomorrow's World и How 2. На протяжении 26 лет была соведущей интеллектуальной математической телеигры Countdown. Кэрол — выпускница Кембриджа, она страстно увлекается популяризацией науки и программирования.

КРЭЙГ СТИЛИ — учитель информатики и проект-менеджер в CoderDojo Scotland — сети бесплатных секций по программированию для молодежи. Ранее Крэйг работал в Raspberry Pi Foundation, в Научном центре Глазго, а также участвовал в проекте телеканала BBC micro:bit. Первым компьютером Крэйга был ZX Spectrum.

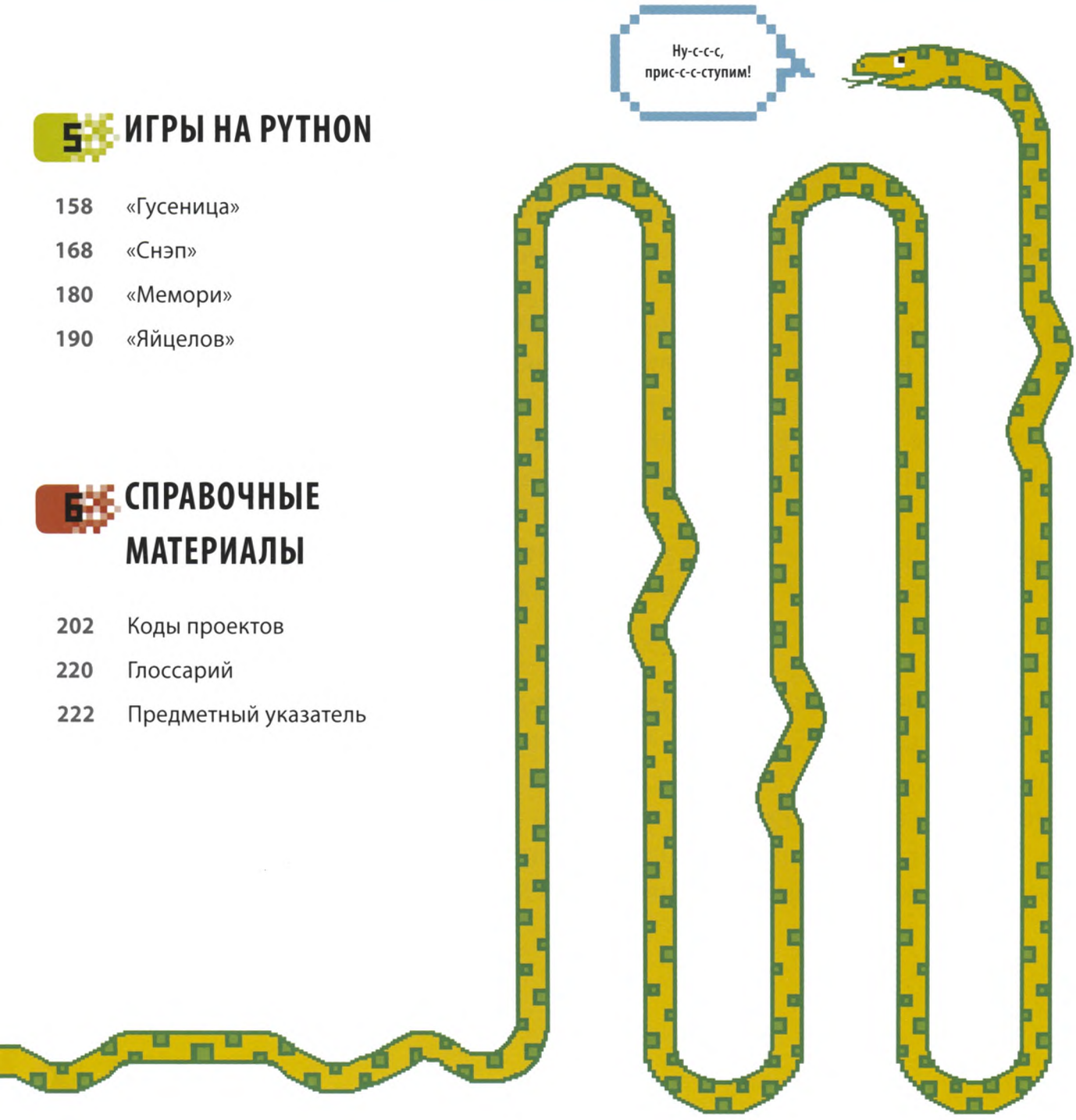
КЛЭР КВИГЛИ — изучала информатику в Университете Глазго, получила степени бакалавра и доктора наук. Работала в компьютерной лаборатории Кембриджского университета и в Научном центре Глазго, сейчас занята в проекте по созданию музыкального и технологического порталов для начальных классов школ Эдинбурга. Наставник в CoderDojo Scotland.

МАРТИН ГУДФЕЛЛОУ — имеет докторскую степень по информатике и опыт обучения программированию вплоть до университетского уровня. Разрабатывал обучающее ПО для CoderDojo Scotland, Skills Development Scotland, Glasgow Life и Highlands and Islands Enterprises, консультировал телеканал BBC по вопросам работы с цифровыми данными.

ДЭНИЕЛ МАККАФФЕРТИ — окончил Университет Стратклайда по специальности «Информатика». Разрабатывал ПО для больших и малых компаний из разных сфер, от банковского дела до телевидения. Дэниел живет в Глазго, обучает детей программированию, а свободное время посвящает семье и велопрогулкам.

ДЖОН ВУДКОК — изучал физику в Оксфордском университете и вычислительную астрофизику в Лондонском университете. Начал программировать в восемь лет, работал на самых разных компьютерах: от одночиповых микроконтроллеров до суперкомпьютеров мирового класса. Джон — соавтор бестселлера «Программирование для детей» и других книг издательства DK.

Содержание



5 ИГРЫ НА PYTHON

6 СПРАВОЧНЫЕ МАТЕРИАЛЫ

Введение

Мы живем в цифровом мире, в котором от компьютеров никуда не скрыться. Не так давно эти машины были громоздкими и шумными, теперь же это тихие компактные устройства, которые работают в автомобилях, телевизорах, телефонах и даже часах. Они помогают нам трудиться, читать, играть, смотреть фильмы, совершать покупки, общаться с родными и друзьями.

Современные компьютеры настолько просты в управлении, что пользоваться ими может каждый. Однако написать компьютерную программу способны далеко не все. Начав программировать, ты узнаешь принцип работы компьютера, а немного попрактиковавшись, сможешь создавать собственные приложения и игры, а также дорабатывать чужие программы.

Программирование — это не только увлекательное хобби, но и навык, востребованный сейчас во многих областях жизни: науке, искусстве, спорте и бизнесе.

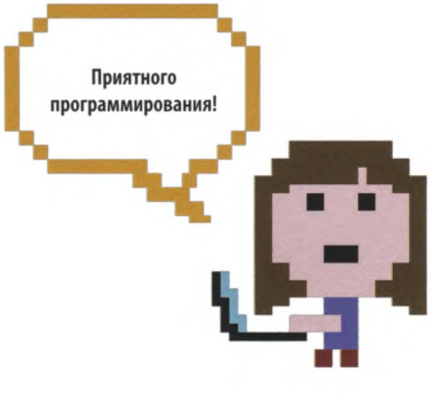
В мире существуют сотни разных языков программирования — от простых графических вроде Scratch до веб-языков, таких как JavaScript. Эта книга — про Python, один из самых популярных компьютерных языков. При всей его мощности и гибкости Python легко изучить, поэтому его любят и профессионалы, и новички. Если ты только начинаешь программировать или уже знаком с другим языком, в любом случае решение освоить Python — отличная идея!

Лучший способ научиться программировать — сразу приступить к делу. Следуя пошаговым инструкциям из этой книги, ты сможешь создать приложения и игры. Учиться проще, когда тебе нравится результат, вот почему мы постарались сделать проекты как можно более интересными.

Если ты новичок в программировании, начни с первого проекта и постепенно продвигайся вперед. Не волнуйся, если что-то не понимаешь до конца: чем больше ты напишешь кодов, тем лучше все усвоишь. И не переживай, если программа не заработает сразу, — ошибки делают даже профессионалы.

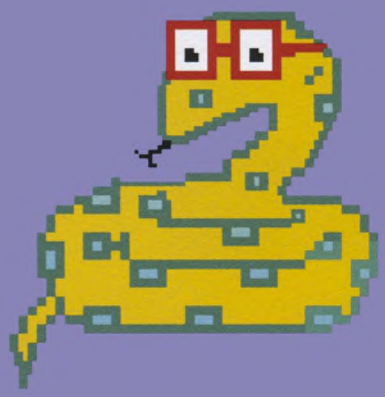
В конце описания каждого проекта есть список советов по доработке программы. Не бойся воплощать в жизнь и собственные идеи. Возможности программиста безграничны — нужно лишь немного знаний и воображения!

КЭРОЛ ВОРДЕРМАН



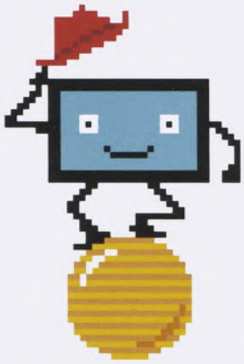
1

Знакомство с Python



Программирование

Программист — это человек, составляющий инструкции для компьютеров. Он может придумать такую программу, которая будет сама писать музыку, или управлять домашним роботом, или отправлять на Марс космические корабли.



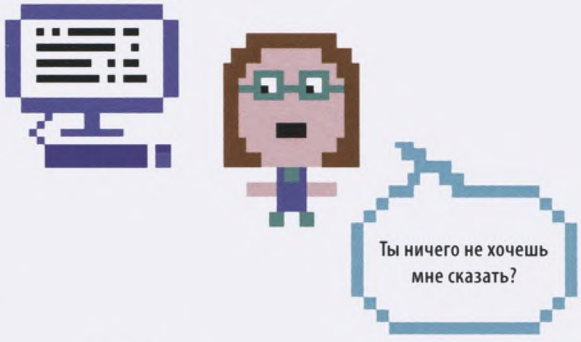
▽ **Дрессированный зверек**
Освоив язык программирования, ты сможешь создавать программы для компьютера, а он будет их выполнять — словно электронный зверек, которого ты обучил разным трюкам!

Глупые ящики

Компьютер — это всего лишь железный ящик. Он не знает, что делать, если ему не объяснить. Поскольку компьютеры не умеют думать, программистам приходится делать это за них, а затем писать подробные инструкции — программы.

Языки программирования

Чтобы посылать компьютеру команды, нужно освоить какой-нибудь язык программирования. Визуальные языки хороши для новичков, а профессионалы используют текстовые языки. В этой книге мы расскажем о Python — популярном текстовом языке программирования.



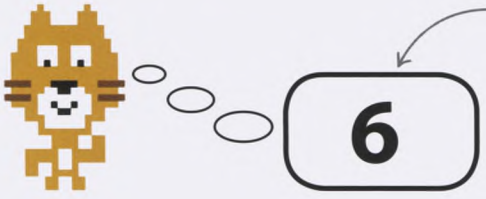
▽ **Scratch**
Scratch — это визуальный язык программирования. Он подходит для создания игр, мультфильмов и интерактивных мультимедийных проектов. Код на Scratch составляют из разноцветных командных блоков.

Оба этих кода делают одно и то же.

▽ **Python**
Python — это текстовый язык программирования. Программы на Python состоят из слов, букв, чисел и других символов. Чтобы ввести код, программисты набирают его на клавиатуре.

```
>>> 3 + 3
6
```

Чтобы увидеть результат, нужно, находясь в окне консоли, нажать ENTER.



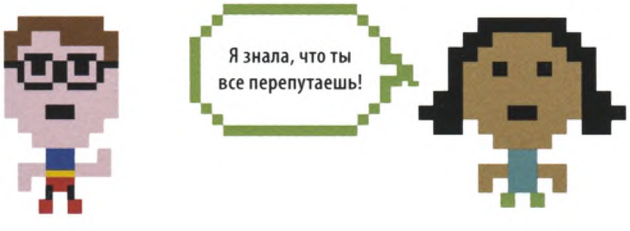
Результат работы программы — это появившееся на экране «облачко» с цифрой 6.

Программировать может каждый

Чтобы стать программистом, изучи основные правила и команды и начинай создавать программы, которые интересны лично тебе. Если увлекаешься наукой, напиши программу, рисующую графики по итогам твоих экспериментов. А если у тебя хороший вкус, создай волшебный мир для собственной видеоигры.

▽ Больше логики

Чтобы писать хорошие программы, программист должен мыслить ясно и рассуждать логически. Если инструкции расплывчаты, программа будет работать неверно. Обдумывай каждое действие и следи за верной последовательностью команд: ты ведь не надеваешь трусы на брюки!



▽ Внимание к деталям

Если ты легко находишь отличия на похожих картинках, у тебя есть все шансы стать отличным программистом. Умение находить в коде баги очень важно, ведь даже малюсенькой ошибки достаточно, чтобы нарушить работу всей программы. Зоркие программисты неустанно высматривают в кодах опечатки и логические ошибки. Порой это нелегко, но учиться на своих ошибках — верный способ прокачать профессиональные навыки.

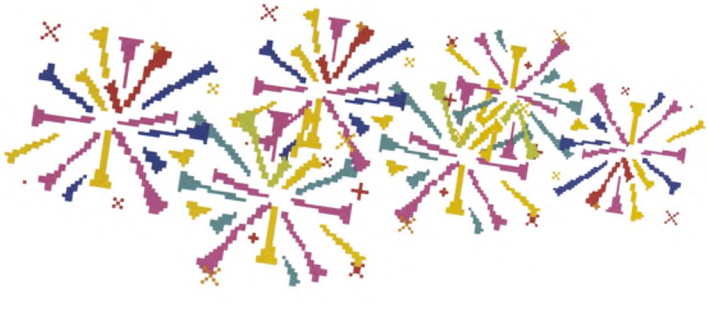
СЛЕНГ

Баги

Багами называют ошибки в коде, из-за которых программы работают неверно. Слово bug переводится как «жук». Дело в том, что первые компьютеры порой сбоили из-за насекомых, застрявших между контактами.

Начни программировать

Программирование может показаться непростым делом, но учиться программировать легко — надо лишь начать. Эта книга предлагает тебе попробовать свои силы в несложных проектах. Приступай к созданию игр и приложений прямо сейчас!



Знакомься: Python

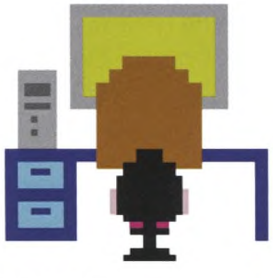
Python — один из самых популярных языков программирования. Появился он в 1990-х годах, а сейчас на нем работают миллионы приложений, игр и веб-сайтов.

Почему Python?

Python отлично подходит для знакомства с программированием. Его изучают во многих школах и университетах. Вот несколько причин его популярности.

△ **Легко читать, легко писать**

Python — текстовый язык программирования. Его инструкции состоят из английских и русских слов, знаков препинания, символов и чисел. Поэтому программы на Python легко читать, писать и понимать.



△ **Работает везде**

Код на Python является переносимым: его можно писать и запускать на самых разных компьютерах. Один и тот же код подходит для операционных систем Windows, macOS, Linux и даже Raspberry Pi: он везде работает одинаково.

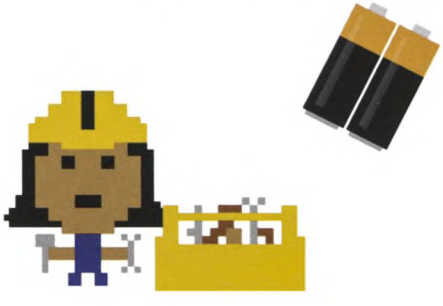
СЛЕНГ

Откуда название?

Python был назван вовсе не в честь питона, а в честь британской комик-группы «Монти Пайтон». Создатель языка Гвидо ван Россум — ее большой поклонник. Python-программисты часто отдают этому должное и включают в код шутки и цитаты из телепередачи «Летающий цирк Монти Пайтона».

▽ **Батарейки в комплекте**

«Батарейки в комплекте» — говорят программисты про Python, потому что вместе с языком в установочных файлах имеется все необходимое, чтобы сразу приступить к программированию.



△ **Удобные инструменты**

Python укомплектован множеством полезных инструментов и готовых фрагментов кода, вместе составляющих стандартную библиотеку. С ее помощью писать программы гораздо проще.

▷ **Отличная поддержка**

Python отлично документирован. Документация включает руководство для начинающих, справочный раздел и множество примеров кода.

Python в действии

Python применяют для решения множества интересных задач в области бизнеса, науки и техники. Например, с помощью программ, написанных на Python, можно управлять освещением и температурой в доме.

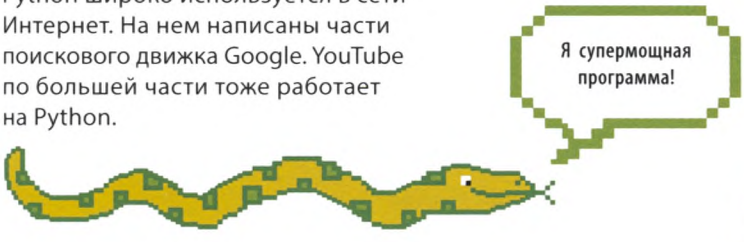
СОВЕТЫ ЭКСПЕРТА

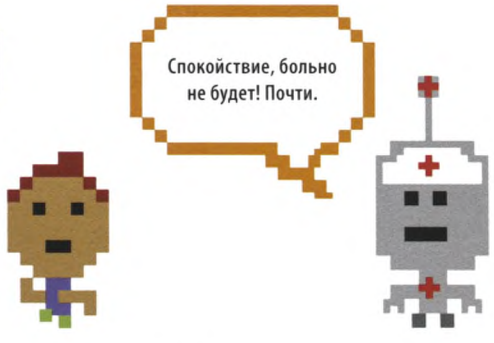
Интерпретатор

Многие языки программирования используют интерпретатор — специальную программу, переводящую команды одного языка на другой. При каждом запуске Python-программы интерпретатор строка за строкой превращает ее в особый, понятный компьютеру машинный код.

▽ **Python и интернет**

Python широко используется в сети Интернет. На нем написаны части поискового движка Google. YouTube по большей части тоже работает на Python.





△ **Чудеса медицины**

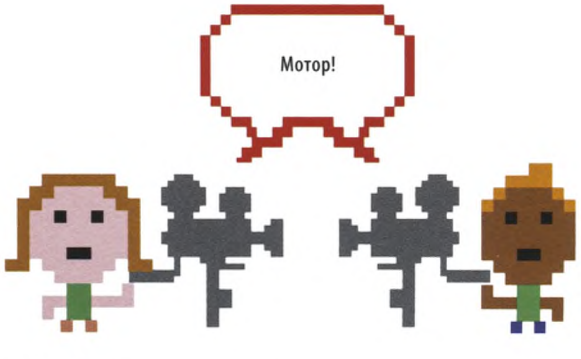
На Python пишут программы для роботов, выполняющих сложные хирургические операции. Робот-хирург может работать быстрее и точнее человека, при этом вероятность, что он допустит ошибку, меньше.

△ **Серьезный бизнес**

Python помогает банкам следить за средствами на счетах вкладчиков, а сетевым магазинам — устанавливать цены на товары.

△ **Другие миры**

В Центре управления полетами NASA Python-программы помогают космонавтам готовиться к полетам и отслеживать ход экспедиций.



△ **На большом экране**

На студии Disney Python помогает автоматизировать шаблонные задачи. Вместо того чтобы снова и снова рисовать однотипные кадры, аниматоры поручают такую работу программе. Это значительно сокращает время работы над мультфильмом.

Установка Python

Для всех проектов в этой книге требуется Python 3. Скачай эту версию программы и следуй инструкциям, подходящим для операционной системы твоего компьютера.

СЛЕНГ

IDLE

IDLE — это бесплатная среда разработки, предназначенная для начинающих программистов. Она устанавливается вместе с Python и включает в себя простой текстовый редактор для ввода и редактирования команд.

Python для Windows

Прежде чем устанавливать Python 3 для Windows, узнай, 32-битная или 64-битная у тебя система. Кликни «Пуск», правой кнопкой мыши кликни «Компьютер» и выбери «Свойства». Выбери «Система», если такая опция появится.

1 Перейди на сайт Python

Перейди на сайт Python по ссылке, указанной ниже. Кликни Downloads («Скачать»), чтобы перейти на страницу скачивания.

https://www.python.org/

2 Скачай Python

Кликни по самой свежей версии Python для Windows, номер которой начинается с цифры 3. Запустится скачивание. Из разных форматов инсталлятора выбери executable installer.

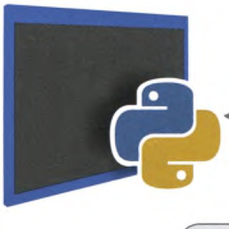
- Python 3.6.4 — 2017-12-19
  - Windows x86 executable installer
  - Windows x86-64 executable installer

Если у тебя 32-битная версия Windows, выбери этот инсталлятор.

Если у тебя 64-битная версия Windows, выбери этот инсталлятор.

3 Запусти инсталлятор

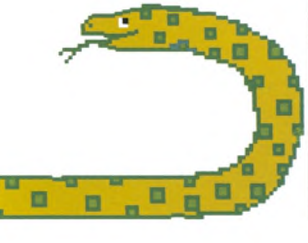
Сделай двойной клик по значку инсталлятора. Выбери «Установить для всех пользователей», кликни «Далее», не меняя настроек по умолчанию.



Кликни дважды по файлу инсталлятора.

4 Открой IDLE

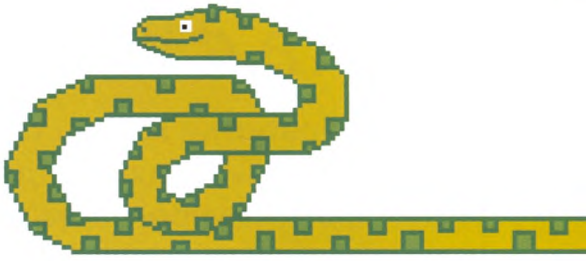
Когда установка завершится, запусти программу IDLE. Открой меню «Пуск», выбери «Все программы» и кликни IDLE. Должно открыться диалоговое окно, похожее на это.



```
Python 3.6.4 Shell
IDLE  File  Edit  Shell  Debug  Window  Help
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28) [MSC v.1900 32
bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

Python для Mac OS

Прежде чем устанавливать Python 3 на «Мак», узнай версию своей операционной системы. Кликни по иконке с яблоком в левом верхнем углу экрана и выбери пункт меню About this Mac.



1 **Перейди на сайт Python**

Перейди на сайт Python по ссылке, указанной ниже. Кликни Downloads («Скачать»), чтобы попасть на страницу скачивания.

https://www.python.org/

2 **Скачай Python**

Кликни по самой свежей версии Python 3, подходящей для операционной системы macOS. Скачивание файла Python.pkg начнется автоматически.

- Python 3.6.4 — 2017-12-19
  - Download macOS X 64 bit/32 bit installer

Номер версии может отличаться от этого. Главное — убедиться, что он начинается с цифры 3.

3 **Установи Python**

В папке Downloads ты найдешь файл с расширением .pkg и иконкой в виде открытой коробки, — сделай по ней двойной клик. На все запросы жми «Продолжить», затем кликни «Установить».

Кликни дважды, чтобы начать установку Python 3.

ВАЖНО!

Спроси разрешение

Никогда не устанавливай Python или любые другие программы без разрешения хозяина компьютера. Возможно, в ходе установки тебе потребуется также узнать у него пароль администратора.

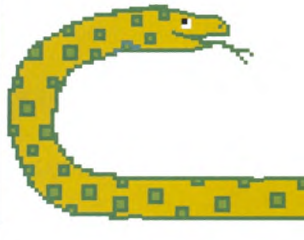
4 **Открой IDLE**

Когда установка завершится, запусти программу IDLE. Открой папку Applications («Приложения»), а в ней — папку Python. Двойным кликом запусти IDLE. Должно открыться диалоговое окно, как на картинке.

Python 3.6.4 Shell

IDLE File Edit Shell Debug Window Help

```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
```



Работа с IDLE

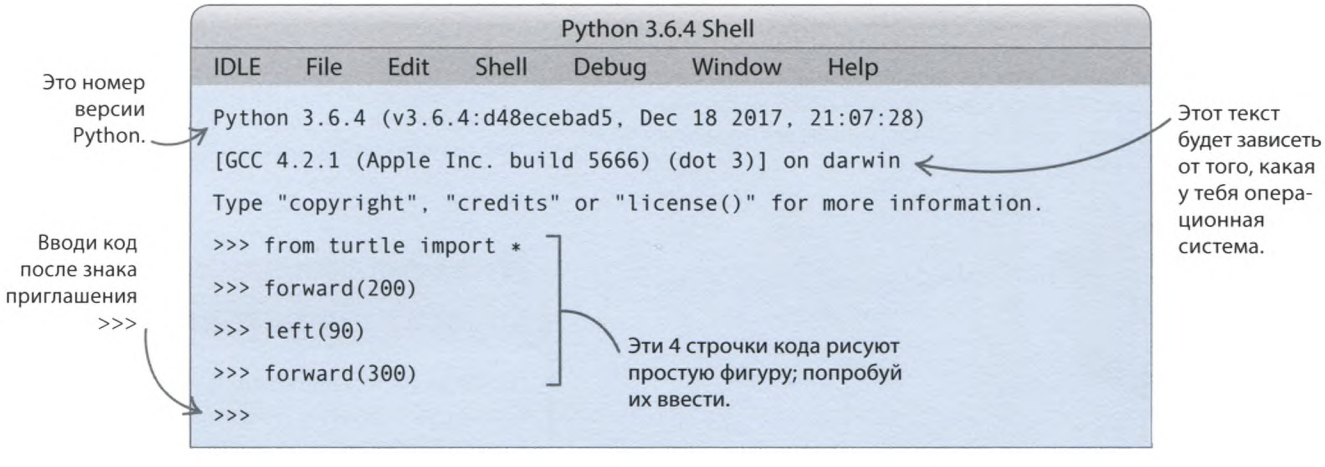
В среде разработки IDLE есть два окна. Окно программы подходит для создания и редактирования программ, а окно консоли — для мгновенной проверки работы кода.

Окно консоли

После запуска IDLE откроется окно консоли. Начинать знакомство с Python лучше с него: так тебе не придется создавать новый файл, в окне консоли код можно вводить сразу.

▽ **Работа в окне консоли**

Код, который ты введешь, запустится сразу, а любые сообщения, в том числе об ошибках, тут же появятся на экране. В окне консоли удобно тестировать части кода перед тем, как добавить их в большую программу.



СОВЕТЫ ЭКСПЕРТА

Разные окна

Чтобы тебе было легче отличать окно консоли от окна программы, в книге они показаны разными цветами: голубым и фиолетовым соответственно.

Окно консоли

Окно программы

▽ **Попробуй поработать в окне консоли**

Введи эти строки кода в окне консоли, нажимая после каждой клавишу ENTER. Результатом выполнения первой команды будет сообщение, вторая выведет на экран число. А что сделает третья строчка кода?

```
>>> print('Мне 10 лет')
```

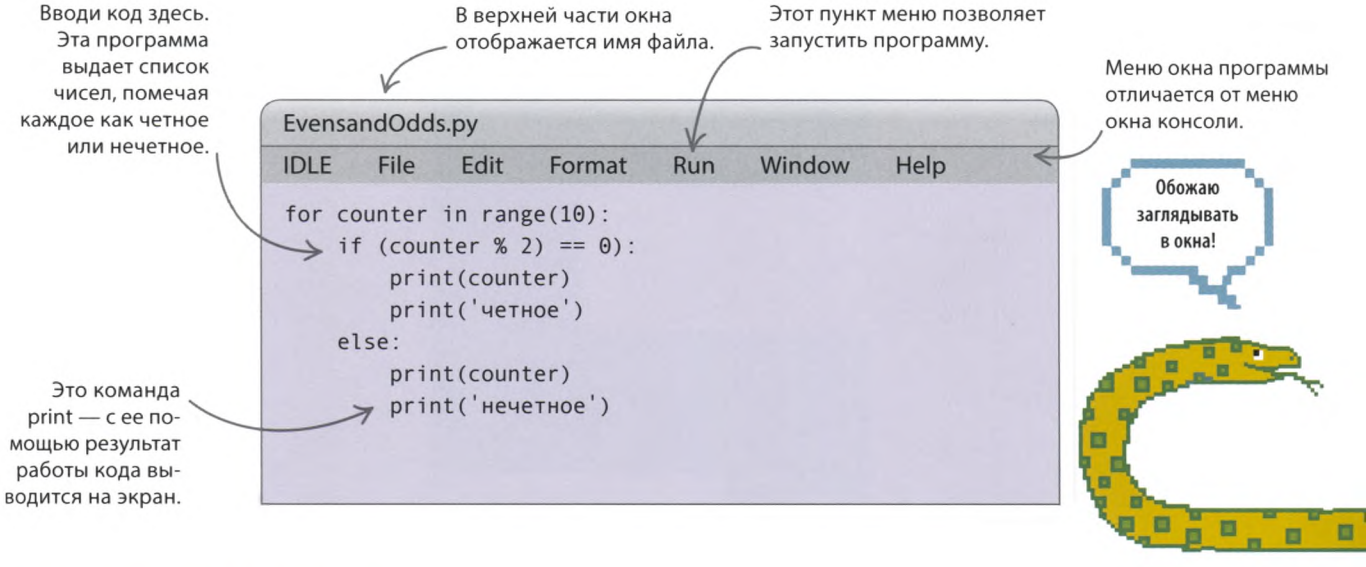
```
>>> 123 + 456 * 7 / 8
```

```
>>> ''.join(reversed('Время программировать'))
```

Окно программы

Код не сохраняется в окне консоли. Стоит его закрыть, и все, что ты ввел, пропадет навсегда. Поэтому создавать программу следует в окне программы. Оно позволяет сохранить код, а еще имеет встроенные инструменты, облегчающие написание программ и поиск ошибок.

▽ **Окно программы**
Чтобы открыть окно программы IDLE, кликни по меню File и выбери New File. Появится пустое окно программы, которое ты будешь использовать для написания и запуска проектов из этой книги.



СОВЕТЫ ЭКСПЕРТА

Цвета в коде

IDLE автоматически выделяет разные части кода определенными цветами. В тексте программы, который так размечен, легче разобраться и найти возможные опечатки.

◁ **Символы и имена**
Большая часть кода черного цвета.

◁ **Ключевые слова**
Некоторые слова, такие как **if** и **else**, являются ключевыми словами языка Python. Они отображаются оранжевым.

◁ **Вывод**
Текст, который программа выводит на экран, выделяется синим.

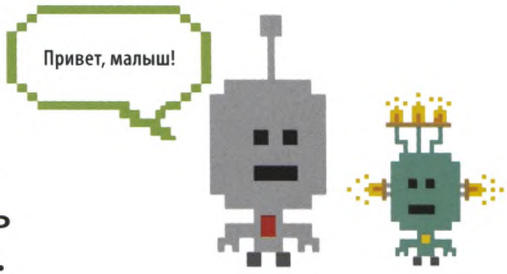
◁ **Встроенные команды**
Встроенные команды Python, такие как **print**, выделяются фиолетовым.

◁ **Ошибки**
Красный цвет предупреждает об ошибках в коде.

◁ **Текст в кавычках**
Текст в кавычках выделяется зеленым. Зеленая скобка рядом с текстом означает, что ты пропустил кавычку.

Первые шаги

Первая программа



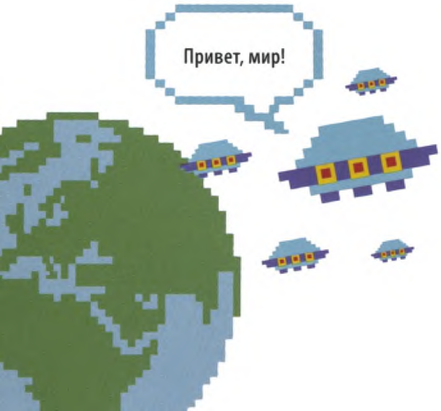
Ну вот, у тебя установлены Python и IDLE, а значит, можно начинать программировать. Выполни пошагово эти действия, чтобы создать программу, выводящую на экран приветствие.

Как это работает

Программа выведет на экран фразу «Привет, мир!», а затем спросит, как тебя зовут. После ввода имени она снова поздоровается, но уже обратится к тебе лично. Программа запоминает имя с помощью переменной. В программировании переменные служат для хранения информации.

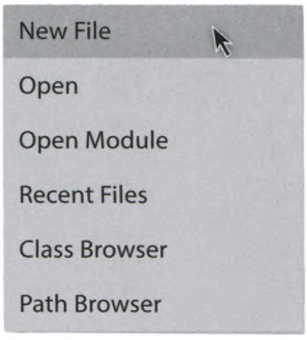
▷ **Блок-схема программы «Привет, мир!»**

При написании кода программисты используют особую диаграмму — блок-схему. Каждый шаг программы показан на ней в виде блока со стрелкой, ведущей к следующему блоку. Если шаг соответствует вопросу, от него может идти несколько стрелок, означающих разные ответы.



1 Запусти IDLE

После запуска IDLE появится окно консоли. Зайди в меню File и выбери New File — откроется пустое окно программы, в котором можно вводить код.



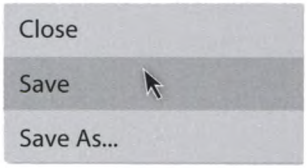
2 Введи первую строку

Введи строку кода, показанную ниже. Слово **print** — это команда, которая выводит на экран данные. В нашем случае — фразу «Привет, мир!».

```
print('Привет, мир!')
```

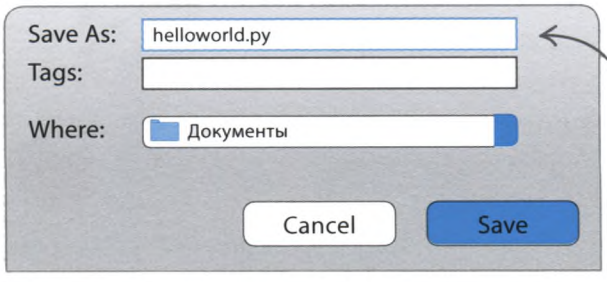
3 Сохрани файл

Открой меню File и выбери Save («Сохранить»).



4 **Заверши сохранение**

Появится диалоговое окно. Введи название файла на английском helloworld.py («привет, мир») и кликни Save.



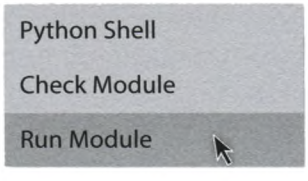
СЛЕНГ

файлы .py

Имена Python-программ обычно оканчиваются на «.py», чтобы их легче было отличать от других. При сохранении кода IDLE автоматически добавляет к имени «.py», так что вводить это расширение необязательно.

5 **Проверь код**

Запусти первую строку, чтобы проверить, работает ли программа. Для этого открой меню Run и выбери Run Module («Запустить модуль»). В окне консоли должно появиться сообщение «Привет, мир!».




```
>>>
Привет, мир!
>>>
```

Это сообщение появится в окне консоли.

6 **Исправь ошибки**

Если код не работает, сохраняй спокойствие! Все программисты допускают ошибки. Чтобы стать настоящим профессионалом, нужно уметь находить баги. Проверь свой код: не пропустил ли ты скобки? Правильно ли введена команда **print**? Исправь ошибки и снова запусти программу.

СОВЕТЫ ЭКСПЕРТА

Горячие клавиши

Запустить код в окне программы можно, нажав клавишу F5. Так получится гораздо быстрее, чем открывать меню Run и выбирать Run Module.

7 **Добавь еще строки**

Вернись в окно программы и добавь еще две строчки кода. Вторая строка должна запрашивать имя пользователя и сохранять его в переменной. Третья строка — показывать это имя при выводе на экран нового приветствия. Вводить можно полное имя, краткое или даже прозвище!

```
print('Привет, мир!')
person = input('Как тебя зовут? ')
print('Привет, ', person)
```

Эта строка запрашивает имя пользователя и сохраняет его в переменной person («пользователь»).

8 **Последний шаг**

Запусти программу в окне консоли. Введи свое имя и нажми ENTER — должно появиться приветствие с обращением. Ура, ты написал свою первую Python-программу! Продолжай в том же духе!

```
Привет, мир!
Как тебя зовут? Дима
Привет, Дима
```

Имя пользователя.

Переменные

Чтобы программа могла воспользоваться данными, они должны где-то храниться. Для этого и существуют переменные. Они нужны для самых разных действий: от подсчета очков в игре до создания списков.

△ **Ящик с данными**
Переменная похожа на ящик с табличкой. Ты можешь положить в него данные и, когда они понадобятся, найти их по названию ящика.

Как создать переменную

У переменной должно быть имя. Подумай, что в ней будет храниться и какое слово или словосочетание лучше всего напомнит тебе об этом. Напиши имя латиницей, поставь знак «равно» и укажи данные, которые ты хочешь сохранить. Это называется «присвоить значение переменной».

1 **Присвой значение**
Чтобы создать переменную **age** («возраст») и присвоить ей значение, введи в окне консоли эту строку кода.

Это значение будет храниться в переменной.

```
>>> age = 12
```

Имя переменной.

2 **Выведи значение на экран**
Добавь эту строку кода. Нажми ENTER и посмотри, что получится.

```
>>> print(age)
12
```

Значение переменной age.

Команда print() выводит на экран значение переменной, указанной в скобках.

СОВЕТЫ ЭКСПЕРТА

Имена переменных

Переменным нужно давать понятные имена, чтобы программу было легче читать. Например, переменную для учета жизней игрока лучше назвать **lives_remaining** («оставшиеся жизни»), а не просто **lives** («жизни») или сокращенно **lr**. Имена могут содержать цифры, латинские буквы и знак подчеркивания, однако в начале всегда должна стоять буква. Следуй этим правилам, и все будет в порядке.

Как можно и как нельзя

- Имя переменной должно начинаться с латинской буквы.
- Имя может состоять из цифр и латинских букв.
- Не используй символы вроде -, /, # или @.
- Не ставь пробелы.
- Пробел можно заменить подчеркиванием (_).
- Регистр букв имеет значение. Python будет считать Score и score двумя разными переменными.
- Не называй переменные так же, как ключевые слова Python, например print.

СЛЕНГ

Целые и вещественные числа

В программировании используются два типа чисел: целые и вещественные. Вещественные числа — это десятичные дроби, которые записываются через точку. Целые числа хороши для подсчета чего-либо, а вещественные — для измерений.

Операции с числами

В переменных можно не только хранить числа, но и производить с ними математические операции: складывать, вычитать, умножать, делить и т. д. Обрати внимание: в Python математические символы (называемые операторами) «плюс» и «минус» имеют привычный вид, а «умножить» и «разделить» отличаются от используемых в школе.

Символ	Значение
+	сложить
-	вычесть
*	умножить
/	разделить

Некоторые операторы языка Python.

1 Реши пример

Введи этот код в окне консоли. В нем используются две переменные: **x** хранит число, а **y** — произведение двух чисел. Нажми ENTER, чтобы получить ответ.

```
>>> x = 6
>>> y = x * 7
>>> print(y)
42
```

Создай переменную x и присвой ей значение 6.

Умножь значение x на 7 и сохрани результат в y.

Выведи на экран значение переменной y.

Результат вычисления.

2 Измени значение переменной

Чтобы изменить значение переменной, нужно ввести после знака «равно» другое число. Присвой переменной **x** значение 10 и снова выведи на экран значение **y** с помощью команды **print()**. Как ты думаешь, что получится?

```
>>> x = 10
>>> print(y)
42
```

Измени значение x.

Результат остался прежним. Сейчас разберемся почему.

3 Обнови значение

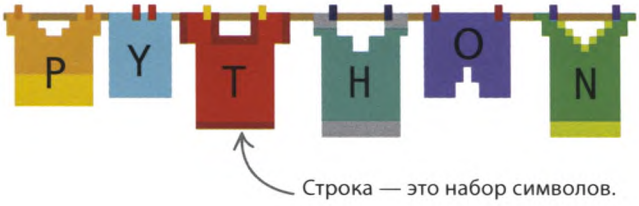
Чтобы получить правильный ответ, значение **y** нужно обновить. Для этого добавь в код эту строчку. Теперь после изменения **x** переменная **y** примет новое значение.

```
>>> x = 10
>>> y = x * 7
>>> print(y)
70
```

Обновляет значение y.

Работа со строками

Строками программисты называют данные, состоящие из букв, пробелов и других знаков. Строки используются почти в каждой программе. Это могут быть слова, предложения или просто набор символов — тех, которые можно ввести с клавиатуры, и даже тех, которые ввести нельзя.



1 **Строковые переменные**

Строки можно хранить в переменных. Введи этот код в окне консоли: он помещает строку 'инопланетянин Ино' в переменную **name** («имя»), а затем выводит ее на экран. Строки всегда нужно брать в кавычки — двойные или одинарные.

Кавычки означают, что в переменной хранится строка.

```
>>> name = 'инопланетянин Ино'
>>> print(name)
инопланетянин Ино
```

Нажми ENTER, чтобы вывести строку на экран.

2 **Склеивание строк**

Значения переменных часто объединяют для получения новых значений. Результат объединения (склейки) двух строк можно поместить в новую переменную. Попробуй ввести в окне консоли этот код.

Не забывай про кавычки.

```
>>> name = 'инопланетянин Ино'
>>> greeting = 'Привет, '
>>> message = greeting + name
>>> print(message)
Привет, инопланетянин Ино
```

Знак + склеивает строки.

При выводе строки на экран кавычки не отображаются.

СОВЕТЫ ЭКСПЕРТА

Длина строки

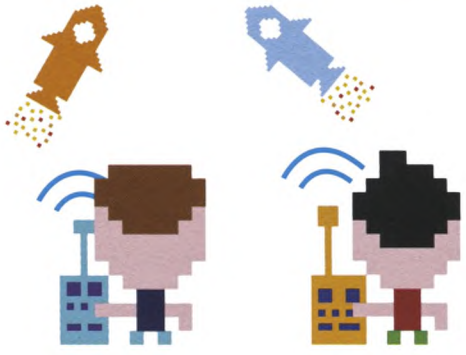
Для подсчета количества символов в строке (включая пробелы) есть удобная команда **len()** (*сокр. от* length — «длина»). По сути, **len()** — это так называемая функция (ты будешь часто пользоваться функциями, работая над проектами из этой книги). Чтобы узнать длину строки 'Привет, инопланетянин Ино', введи этот код после объявления переменной **message** («сообщение») и нажми ENTER.

```
>>> len(message)
25
```

Количество символов в строке.

Списки

Сохранить сразу несколько данных (особенно если важен их порядок) поможет список. Каждому его элементу Python присваивает номер, означающий позицию в списке. Содержимое списка можно изменять как угодно и когда угодно.



1 Не много ли переменных?

Представь, что ты пишешь программу для многопользовательской игры и хочешь сохранить в ней имена игроков двух команд. Можно создать для каждого имени по переменной, примерно так...

```
>>> rockets_player_1 = 'Рома'
>>> rockets_player_2 = 'Игорь'
>>> rockets_player_3 = 'Алена'
>>> planets_player_1 = 'Миша'
>>> planets_player_2 = 'Павел'
>>> planets_player_3 = 'Полина'
```

Если в каждой команде по три игрока, тебе понадобится шесть переменных.

2 Создай список

...но что если в каждой команде окажется шесть игроков? Тебе будет нелегко уследить за таким количеством переменных. Гораздо проще создать список. Для этого укажи элементы, которые в нем будут храниться, внутри квадратных скобок.

```
>>> rockets_players = ['Рома', 'Игорь',
'Алена', 'Рената', 'Саша', 'Лена']
>>> planets_players = ['Миша', 'Павел',
'Полина', 'Настя', 'Аня', 'Леша']
```

Этот список хранится в переменной planets_players («игроки команды “Планеты”»).

Элементы списка нужно разделять запятыми.

3 Получи элемент из списка

Когда данные хранятся в виде списка, работать с ними просто. Чтобы получить значение элемента, введи имя списка и укажи в квадратных скобках позицию элемента. Внимание: Python считает элементы с 0, а не с 1. Попробуй получить из списка имена разных игроков. Имя первого стоит в позиции 0, имя последнего — в позиции 5.

Так можно получить значение первого элемента списка (в позиции 0).

```
>>> rockets_players[0]
'Рома'
>>> planets_players[5]
'Леша'
```

Так можно получить значение последнего элемента списка (в позиции 5).

Чтобы получить значение элемента, нажми ENTER.

Принятие решений

Каждый день ты принимаешь решения, исходя из ответов на вопросы. Например, таких: «На улице дождь?», «Я сделал уроки?», «Я похож на лошадь?». Компьютер тоже решает, что делать дальше, на основе ответов на подобные вопросы.

Вопросы и сравнения

Вопросы, которые задает себе компьютер, обычно подразумевают сравнение чего-то с чем-то. Например, больше ли одно число другого. Если это так, программа выполняет один блок кода, если нет — другой.

▷ Булевы значения

На вопросы, которые задает компьютер, есть лишь два ответа: True («истина») и False («ложь»). Эти ответы называются булевыми значениями и всегда пишутся с прописной буквы. Булевы значения тоже можно хранить в переменных.

Переменная.

```
>>> answer_one = True
>>> answer_two = False
```

Булево значение.

СОВЕТЫ ЭКСПЕРТА

Знаки «равно»

В Python используется и одинарный знак «равно» (=), и двойной (==). Смысл у них разный. Одинарный знак позволяет присвоить переменной значение. Если ввести age = 10, то в переменной **age** сохранится значение 10. Двойной знак «равно» сравнивает два значения, как в этом примере.

Так присваивают значение переменной.

```
>>> age = 10
>>> if age == 10:
        print('Тебе десять лет.')
```

Так сравнивают значение переменной с числом.

Эта команда выведет на экран сообщение, если числа совпадут.

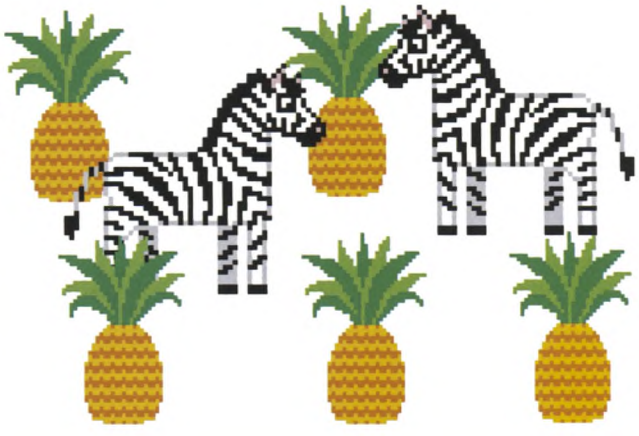
▽ Операторы сравнения

Символы в табличке ниже используются для сравнения значений, — возможно, ты встречался с ними на уроках математики. Программисты называют их операторами сравнения. Также в коде можно использовать логические операторы **and** («и») и **or** («или»).

Символ	Значение
==	равно
!=	не равно
<	меньше, чем
>	больше, чем

Зебры и ананасы

А теперь пример. Допустим, у тебя есть пять ананасов и две зебры. Выразим это с помощью переменных **pineapples** («ананасы») и **zebras** («зебры»). Введи этот код в окне консоли.



```
>>> pineapples = 5
>>> zebras = 2
```

В этой переменной хранится количество ананасов.

В этой переменной хранится количество зебр.

▽ ▷ Сравнения

Введи следующие строчки кода, чтобы сравнить значения двух переменных. В конце каждой строки нажимай ENTER, и Python сообщит, истинно это выражение (True) или ложно (False).

```
>>> zebras < pineapples
True
```

Зебр меньше, чем ананасов.

Ананасов больше, чем зебр.

```
>>> pineapples > zebras
True
```

Количество ананасов не равно количеству зебр.

```
>>> pineapples == zebras
False
```

СЛЕНГ

Логические операции

Выражения, состоящие из переменных, их значений и логических операторов, всегда возвращают булево значение — True или False — и называются логическими операциями. Все наши операции с ананасами и зебрами — логические.

Переменная.

Логический оператор.

```
>>> pineapples != zebras
True
```

Булево значение.

Переменная.

▽ Множественные сравнения

Операторы **and** и **or** позволяют объединять сравнения. При этом логическая операция **and** возвращает True, только если оба сравнения возвращают True, а **or** — если True возвращает хотя бы одно из двух сравнений.

```
>>> (pineapples == 3) and (zebras == 2)
False
```

Одно из двух сравнений (pineapples == 3) возвращает False, поэтому выражение в целом возвращает False.

```
>>> (pineapples == 3) or (zebras == 2)
True
```

Одно из двух сравнений (zebras == 2) возвращает True, поэтому выражение в целом возвращает True.

Американские горки

Объявление в парке аттракционов гласит, что кататься на американских горках может ребенок старше 8 лет, рост которого больше 1,4 метра. Маше 10 лет, а ее рост 1,5 метра. Проверим в окне консоли, пропустят ли Машу на аттракцион. В первых двух строчках кода объяви переменные и сохрани в них возраст и рост Маши. Затем добавь условия посещения аттракциона (логическую операцию) и нажми ENTER.

Присваивают значения переменным.

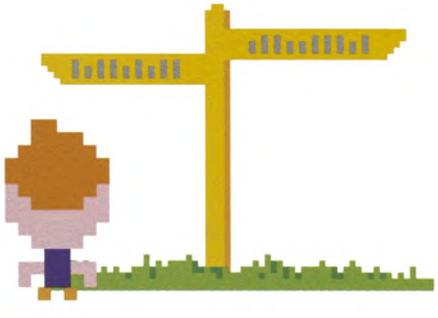
```
>>> age = 10
>>> height = 1.5
>>> (age > 8) and (height > 1.4)
True
```

Логическая операция, означающая «старше 8 лет и выше 1,4 метра».

Ура! Маша может прокатиться на американских горках!

Ветвление

Компьютеру часто приходится решать, какую часть кода выполнять дальше: ведь большинство программ при разных условиях должны работать по-разному. Работа программы похожа на путешествие по дороге с развилками, когда от выбора направления зависит, в какое место ты попадешь.



СЛЕНГ

Условие

Условие — это логическая операция (сравнение, возвращающее True или False), которая помогает компьютеру выбрать дорогу, если он достиг развилки на пути выполнения кода.

▷ **В школу или в парк?**

Представь, что ты каждый день решаешь, по какой дороге пойти, отвечая на вопрос «Сегодня выходной?». Если выходной, ты идешь в парк, а если нет, то в школу. В Python разные пути ведут к разным блокам кода. Блок может состоять из одной или нескольких команд, отделенных от остальной части кода отступами в 4 пробела. С помощью вопросов (условий) компьютер решает, какие блоки кода выполнять дальше.

▷ Один путь

Простейшее ветвление — это конструкция **if** («если»). У нее лишь один путь, и компьютер выберет его, если условие вернет True. Этот код спрашивает, темно ли на улице. Если пользователь отвечает «да» (д), программа делает вид, что компьютер уснул. В любом другом случае (если условие возвращает False) сообщение «Спокойной ночи!» не появится на экране.

```
is_dark = input('На улице темно? д/н)')
if is_dark == 'д':
    print('Спокойной ночи! Хр-р-р-р-р-р...')
```

Эта строка кода просит пользователя ответить «д» (да) или «н» (нет).

Условие.

Этот блок кода выполняется, если условие возвращает True.

Сообщение, которое появится на экране.

▷ Два пути

Хочешь, чтобы программа выбрала один путь, если условие возвращает True, и другой, если False? Тогда нужна команда с двумя ветвлениями — конструкция **if-else** («если-иначе»). Этот код спрашивает, есть ли у тебя щупальца. При ответе «да» (д) компьютер думает, что ты осьминог, иначе (н) считает тебя человеком. В каждом случае выводится свое сообщение.

```
tentacles = input('У тебя есть щупальца? (д/н)')
if tentacles == 'д':
    print('Ну и ну, осьминог за компьютером!')
else:
    print('Приветствую тебя, человек!')
```

Эта строка кода просит ответить на вопрос.

Условие.

Этот блок кода выполняется, если условие возвращает False.

Этот блок кода выполняется, если условие возвращает True.

▷ Множество путей

Когда нужно больше двух ветвлений, на помощь приходит конструкция **elif** (*сокр. от* else-if — «еще-если»). Этот код спрашивает, какая сегодня погода: дождь, снег или солнце. Затем он выбирает один из трех путей и выводит на экран подходящий совет.

```
weather = input ('Какая сегодня погода? (дождь/снег/солнце)')
if weather == 'дождь':
    print('Не забудьте зонтик!')
elif weather == 'снег':
    print('Не забудьте варежки!')
else:
    print('Не забудьте солнечные очки!')
```

Первое условие.

Этот блок кода выполнится, если первое условие вернет True.

Второе условие.

Этот блок кода выполнится, если оба условия вернут False.

Этот блок кода выполнится, если второе условие вернет True.

△ Как это работает

Конструкция **elif** обязательно должна стоять после **if** и перед **else**. Здесь **elif** проверяет, идет ли снег, только если условие **if** возвращает False. В код можно добавить еще больше конструкций **elif**, проверяющих другие варианты.

Циклы

Компьютеры умеют, не жалуясь, выполнять утомительную работу. О программистах такого не скажешь, зато они могут поручать однообразные задачи компьютеру, создавая для этого циклы. Цикл выполняет один и тот же блок кода снова и снова. В Python есть несколько типов цикла. Рассмотрим два из них.

Цикл for

Если ты знаешь, сколько раз программе нужно выполнить блок кода, тебе подойдет цикл **for** («для»). Эмма написала код, который десять раз выводит на экран «Тайный штаб Эммы О ' Брайен!», чтобы потом распечатать этот текст и повесить его на дверь. Попробуй запустить ее программу в окне консоли. (Введи строки кода и нажми ENTER, сотри пробелы, которые добавятся автоматически, и снова нажми ENTER.)

Это — переменная цикла.

Цикл повторяется 10 раз.

```
>>> for counter in range(1, 11):
        print('Тайный штаб Эммы О\'Брайен!')
```

Сделай отступ в 4 пробела.

Код, который повторяется в цикле, называется телом цикла.

▽ Переменная цикла

Переменная цикла отслеживает, сколько проходов уже сделано. Сначала она равна первому числу из списка, заданного командой **range(1, 11)** («диапазон»). На втором проходе в переменную цикла попадает второе число из списка и т. д. Когда программа переберет все числа в диапазоне от 1 до 10, цикл завершится.

СОВЕТЫ ЭКСПЕРТА

Функция range()

В Python функция **range()** хранит в себе список чисел, который начинается с первого указанного в скобках числа и заканчивается числом, на единицу меньшим второго. Поэтому **range(1, 4)** — это 1, 2 и 3, но не 4. В программе Эммы команда **range(1, 11)** соответствует такому списку: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

СОВЕТЫ ЭКСПЕРТА

Знак экранирования (\)

Фамилия О\' Брайен записана в коде через знак «обратный слеш». Он указывает на то, что идущий следом апостроф не нужно считать закрывающей кавычкой. Обратный слеш называют знаком экранирования. Сам он не выводится на экран, а лишь поясняет, как программе воспринимать следующий символ.

Цикл while

А что если ты не знаешь, сколько раз программе нужно повторить блок кода? Может, стоит заглянуть в хрустальный шар и увидеть будущее? Нет, просто воспользуйся циклом **while** («пока»).

▷ Условие цикла

У цикла **while** нет переменной, которая перебирает все значения, зато у него есть условие — логическая операция, возвращающая True или False. Работа цикла **while** похожа на работу билетного контролера. Если у тебя есть билет (True), он тебя пропустит в зал, а если нет (False), то ты не сможешь войти. В нашем случае если условие цикла не вернет True, программа не сможет войти в цикл!

▽ Башня из бегемотов

Ахмед написал программу, которая считает количество бегемотов в акробатической пирамиде. Изучи код и постарайся разобраться, как он работает.

Условие цикла.

Переменная, в которой хранится количество бегемотов.

Переменная, в которой хранится ответ на вопрос «Добавить бегемота?».

```
>>> hippos = 0
>>> answer = 'д'
>>> while answer == 'д':
        hippos = hippos + 1
        print('Бегемотов в пирамиде: ' + str(hippos))
        answer = input('Добавить бегемота? (д/н) ')
```

Эта строка кода выводит на экран сообщение о количестве бегемотов в акробатической пирамиде.

Добавляет еще одного бегемота.

Ответ пользователя становится новым значением переменной answer («ответ»).

▷ Как это работает

В коде Ахмеда условием цикла служит логическая операция answer == 'д'. Если она возвращает True, значит, пользователь хочет добавить бегемота. Тогда в теле цикла количество бегемотов увеличивается на 1 и на экран выводится вопрос, нужно ли добавить еще одного бегемота. Если пользователь отвечает «да» (д), условие цикла возвращает True, и цикл повторяется. А если «нет» (н), условие возвращает False, и программа выходит из цикла.

Бесконечный цикл

Иногда бывает необходимо, чтобы цикл повторялся все время, пока работает программа. Такой цикл называется бесконечным. Во многих видеоиграх используется бесконечный основной цикл.

Это условие никогда не вернет False.

```
>>> while True:
        print('Это бесконечный цикл!')
```

△ Вход в бесконечность

Чтобы цикл стал бесконечным, сделай его условием постоянное булево значение True. Поскольку оно никогда не изменится, цикл никогда не завершится. Запусти этот код. Условие всегда будет возвращать True, а значит, цикл будет выводить одно и то же сообщение, пока ты не выйдешь из программы.

▽ Выход из бесконечности

Бесконечный цикл можно создать специально. Например, эта программа только и делает, что спрашивает пользователя, надоела ли она ему. Если тот, устав читать одно и то же, введет «д», программа скажет, что это невежливо, и выйдет из цикла с помощью команды **break** («отмена»).

Условием цикла является булево значение True, а это значит, что цикл будет повторяться бесконечно, пока пользователь вводит 'н'.

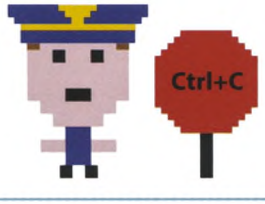
```
>>> while True:
        answer = input('Я тебе надоела? (д/н) ')
        if answer == 'д':
            print('Как невежливо!')
            break
```

Если пользователь введет 'д', запустится блок кода с командой break.

СОВЕТЫ ЭКСПЕРТА

Выход из цикла

Если ты не хочешь, чтобы цикл **while** получился бесконечным, убедись, что в его теле есть что-то, благодаря чему условие цикла может вернуть False. Но не волнуйся: если ты случайно создал бесконечный цикл, его можно остановить, нажав одновременно клавиши CTRL и C. Возможно, это сочетание клавиш понадобится нажать несколько раз.



Цикл внутри цикла

Может ли один цикл находиться в теле другого цикла? Да! Такой вложенный цикл похож на матрешку, куклы которой вставляются одна в другую. Получается, что внутренний цикл запускается внутри внешнего цикла.

СОВЕТЫ ЭКСПЕРТА

Отступы в теле цикла

Строки кода, составляющие тело цикла, нужно вводить с отступом в 4 пробела, иначе Python выдаст ошибку и программа не запустится. А если цикл вложен в другой, тело внутреннего цикла должно идти с отступом еще в 4 пробела. Обычно IDLE ставит отступы автоматически, но ты все равно не забывай проверять количество пробелов в начале каждой строки.

▷ Цикл в цикле

Эмма переделала свой код про тайный штаб так, чтобы программа трижды выводила на экран «ура, ура, победа!». Поскольку в этой фразе два слова «ура», Эмма решила использовать вложенный цикл.

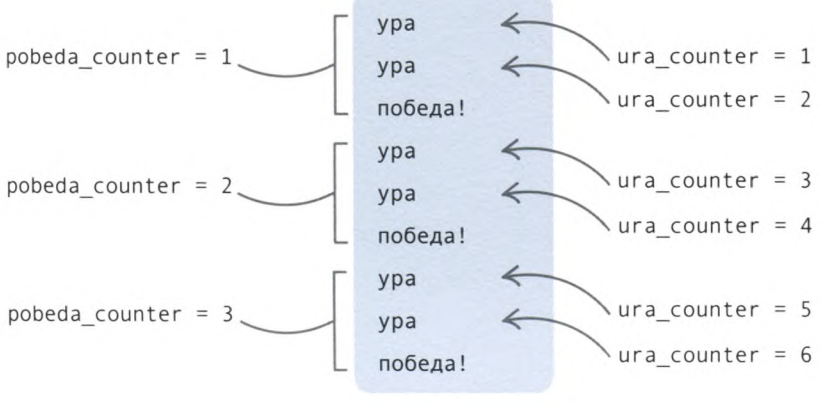
Переменная внешнего цикла — pobeda_counter.

```
>>> for pobeda_counter in range(1, 4):
        for ura_counter in range(1, 3):
            print('ура')
        print('победа!')
```

Тело внешнего цикла идет с отступом в 4 пробела.

Переменная внутреннего цикла — ura_counter.

Тело внутреннего цикла идет с отступом в 8 пробелов.



◁ Как это работает

Внутренний цикл **for** находится в теле внешнего цикла **for**, и на каждый проход внешнего цикла приходится два прохода внутреннего. Это значит, что код внешнего цикла запустится 3 раза, а внутреннего — 6 раз!

Тест «Животные»

Любишь ли ты игровые тесты? А собственный создать хочешь? Цель этого проекта — написать программу-тест «Животные», однако тему впоследствии будет несложно заменить.

Что происходит

Программа задает игроку вопросы про животных. Для ответа на каждый из них отводится три попытки — тест не стоит делать чересчур сложным! За каждый правильный ответ начисляется 1 очко. В конце теста программа показывает счет игрока.

Так выглядит игровой тест в окне консоли.

Python 3.6.4 Shell

```
Тест «Животные»
Какой медведь живет за полярным кругом? белый медведь
Ответ верный
Какое сухопутное животное самое быстрое? гепард
Ответ верный
Какое животное самое большое? жираф
Ответ неверный. Попробуйте еще раз. слон
Ответ неверный. Попробуйте еще раз. носорог
Правильный ответ: синий кит
Вы набрали очков: 2
```

Ответ вводим здесь.

Если ответ окажется неправильным, нужно ввести другой.

После трех неудачных попыток программа выводит на экран правильный ответ.

Итоговый счет игрока (максимум 3 очка).

Как это работает

Для создания этой программы тебе потребуется функция — именованный блок кода, выполняющий определенную задачу. Функция позволяет использовать код повторно, не вводя его каждый раз заново. В Python есть много встроенных функций, а также возможность создавать собственные.

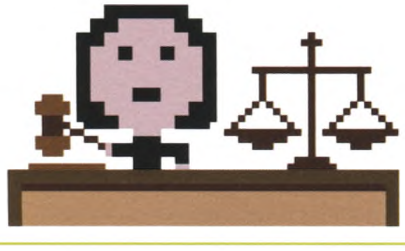
▷ Вызов функции

Чтобы воспользоваться функцией, ее надо вызвать, написав имя и поставив скобки. Для нашего теста тебе предстоит создать функцию, которая будет сравнивать ответ игрока с правильным, и вызвать ее несколько раз — после каждого вопроса.

СЛЕНГ

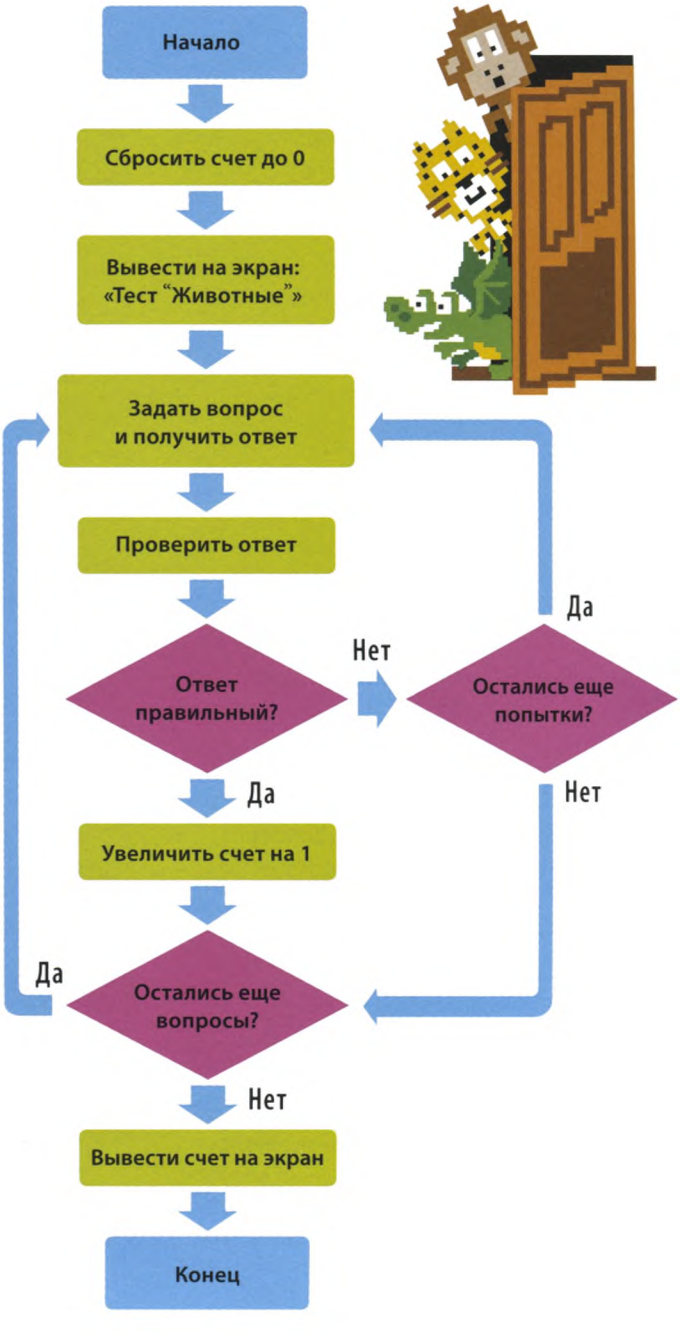
Регистр букв

Сравнивать ответ игрока с правильным нужно без учета регистра букв: не важно, прописными он будет написан или строчными, лишь бы слова совпадали. Это уточнение подходит не для всех случаев: если программа, проверяющая пароль, не учитывает регистр букв, подобрать пароль злоумышленнику станет легче. Но в нашем тесте слова «Гепард» и «гепард» лучше считать одинаковыми.



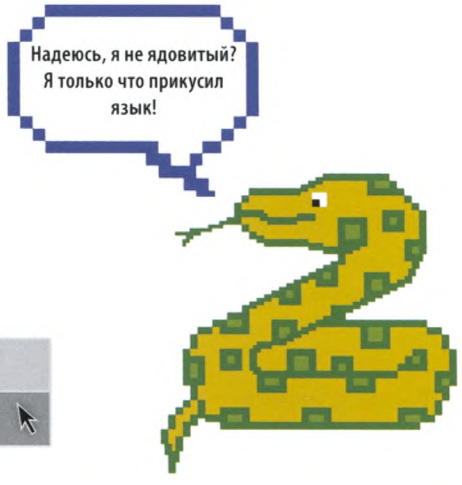
▽ Блок-схема теста «Животные»

Программа раз за разом проверяет, остались ли еще вопросы и использовал ли игрок все попытки. Количество набранных очков хранится в переменной. После того как игрок ответит на все вопросы, игра завершается.

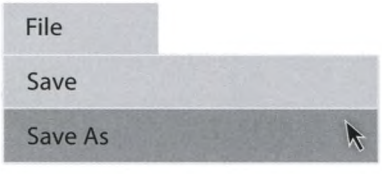


Приступим к написанию

Пришло время написать код для игрового теста! Сперва нужно ввести вопросы и создать механизм проверки ответов. Затем — добавить код, позволяющий игроку отвечать на каждый вопрос три раза.



1 **Создай новый файл**
Открой IDLE. В меню File выбери New File. Сохрани файл как animal_quiz.py («тест "Животные"»).



2 **Создай переменную score**
Введи этот код, чтобы создать переменную **score** («счет») с начальным значением 0.

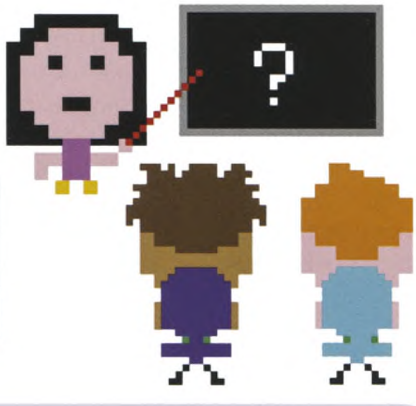
```
score = 0
```

С помощью этой переменной ты будешь следить за количеством очков, набранных игроком.

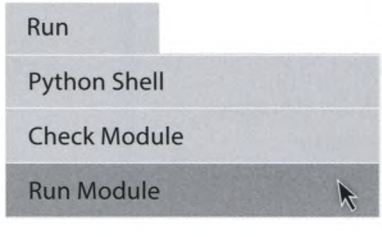
3 **Покажи название игры**
Добавь функцию **print()**, выводящую на экран название теста. Это будет первое, что игрок увидит после запуска программы.

```
score = 0
print('Тест «Животные»')
```

Эта фраза появится на экране.



4 **Запусти программу**
Запусти программу. Для этого открой меню Run и выбери Run Module. В окне консоли должно появиться название игры.



5 **Задай вопрос и введи ответ**
Эта строка кода задает вопрос и ожидает ответ пользователя. Ответ (пользовательский ввод) попадает в переменную **guess1** («вариант 1»). Запусти код и убедись, что вопрос появился на экране.

```
print('Тест «Животные»')
guess1 = input('Какой медведь живет за полярным кругом? ')
```

Все, что введет игрок, попадет в переменную guess1.

6 Создай функцию проверки

Теперь нужно выяснить, правильно ли ответил игрок. Добавь этот код в самое начало программы, перед строкой score = 0. Так ты создашь функцию **check_guess()** («проверить ответ»), которая проверяет, совпадает ли ответ игрока с правильным ответом. Два слова внутри скобок называются аргументами; вызывая функцию, ты присваиваешь ее аргументам значения.

```
def check_guess(guess, answer):
    global score
    if guess == answer:
        print('Ответ верный')
        score = score + 1
score = 0
```

В первой строке кода создаем функцию и в скобках указываем ее аргументы.

Эта строка кода означает, что переменная score — глобальная, то есть если ты изменишь ее значение, это отразится на работе всей программы.

Увеличивает счет на 1 очко.

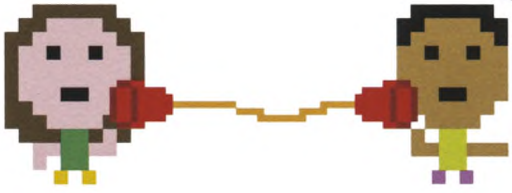
Не забудь про скобки.

7 Проверь ответ

Добавь в конец программы вызов функции **check_guess()**. Ее первым аргументом должен быть ответ игрока, а вторым — правильный ответ «белый медведь».

```
guess1 = input('Какой медведь живет за полярным кругом? ')
check_guess(guess1, 'белый медведь')
```

Правильный ответ.



8 Проверь работу программы

В окне консоли запусти код еще раз и введи правильный ответ. Вот что появится на экране.

```
Тест «Животные»
Какой медведь живет за полярным кругом? белый медведь
Ответ верный
```

9 Добавь еще вопросы

Одного вопроса для теста недостаточно! Добавь в код еще два вопроса, выполнив те же шаги. Пусть ответы пользователя попадают в переменные **guess2** и **guess3**.


```
score = 0
print('Тест «Животные»')
guess1 = input('Какой медведь живет за полярным кругом? ')
check_guess(guess1, 'белый медведь')
guess2 = input('Какое сухопутное животное самое быстрое? ')
check_guess(guess2, 'гепард')
guess3 = input('Какое животное самое большое? ')
check_guess(guess3, 'синий кит')
```

Первый вопрос.

Проверка ответа, попавшего в переменную guess1.

Проверка ответа, попавшего в переменную guess3.

10 Объяви счет

Следующая команда выведет на экран количество очков, набранных игроком после ответа на все вопросы. Добавь ее в конец кода.

```
guess3 = input('Какое животное самое большое? ')
check_guess(guess3, 'синий кит')

print('Вы набрали очков: ' + str(score))
```

Выводит на экран сообщение с количеством набранных очков.

△ Как это работает

В этой команде используется функция **str()** (*сокр. от* string — «строка»), которая преобразует тип данных «целое число» в «строку». Это необходимо делать, поскольку при попытке склеить строку с числом Python выдает ошибку.

11 Убери учет регистра

А если вместо «гепард» игрок введет «Гепард», получит ли он очко? Нет, программа скажет, что ответ неверный! Чтобы такого не произошло, код нужно сделать умнее. В Python есть функция **lower()** («понизить»), превращающая прописные буквы в строчные. Замени команду if guess == answer: на ту, что показана справа.

```
def check_guess(guess, answer):
    global score
    if guess.lower() == answer.lower():
        print('Ответ верный')
        score = score + 1
```

Измени на эту строку.

△ Как это работает

Перед тем как начать сравнивать два аргумента, программа с помощью функции **lower()** превратит все прописные буквы в строчные. Тогда проверка выдаст верный результат, как бы ни набрал игрок название животного.

12 Снова проверь работу программы

Запусти программу в окне консоли еще раз. Попробуй вводить правильные ответы, чередуя строчные и прописные буквы, и посмотри, что выйдет.

```
Тест «Животные»
Какой медведь живет за полярным кругом? белый медведь
Ответ верный
Какое сухопутное животное самое быстрое? ГЕПАРД
Ответ верный
Какое животное самое большое? СиНиЙ кИт
Ответ верный
Вы набрали очков: 3
```

При проверке ответов регистр букв не учитывается.

13 Дай игроку больше попыток

Сейчас у игрока есть только один шанс угадать животное. Можно облегчить ему жизнь и подарить не одну, а три попытки. Измени функцию **check_guess()** так, как показано ниже.

Не забудь сохранить свою работу.

```
def check_guess(guess, answer):
    global score
    still_guessing = True
    attempt = 0
    while still_guessing and attempt < 3:
        if guess.lower() == answer.lower():
            print('Ответ верный')
            score = score + 1
            still_guessing = False
        else:
            if attempt < 2:
                guess = input('Ответ неверный. Попробуйте еще раз. ')
            attempt = attempt + 1

    if attempt == 3:
        print('Правильный ответ: ' + answer)

score = 0
```

В этой переменной хранится лишь одно из двух булевых значений: True или False.

Цикл while запускает код проверки три раза (если игрок не введет правильный ответ раньше).

Убедись, что все строчки даны с нужным количеством отступов.

Конструкция else позволяет игроку дать другой ответ на тот же самый вопрос, если он не смог ответить правильно с первого раза.

Прибавляет 1 к количеству использованных попыток.

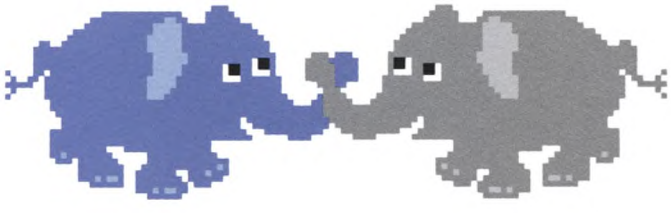
Выводит на экран правильный ответ после трех неудачных попыток.

△ Как это работает

Чтобы программа могла понять, верный ли ответ ввел игрок, используется переменная **still_guessing** («еще угадывает»). При неверном ответе в ней остается значение True, а если игрок угадает слово, оно меняется на False.

Что бы изменить?

Измени свой тест! Сделай его длиннее или короче, добавь другие типы вопросов или поменяй тему. Внося эти изменения, сохраняй код в новых файлах, чтобы не потерять первоначальный вариант игры.



◁ **Сделай длиннее**

Добавь в свой тест еще несколько вопросов. Например: «У какого животного есть длинный хобот?» (слон), «Какое млекопитающее умеет летать?» (летучая мышь). Или посложнее: «Сколько сердец у осьминога?» (три).

С помощью обратного слеша можно разбить длинную строку кода на две.

```
guess = input('Какое из этих животных рыба? \
1) Кит 2) Дельфин 3) Акула 4) Кальмар. Введите 1, 2, 3 или 4. ')
check_guess(guess, '3')
```

◁ **Предложи варианты ответа**

Вот пример кода для вопроса с несколькими вариантами ответа.

ЗАПОМНИ

Перевод строки

Вопросы с вариантами ответов лучше смотрятся, если ответы стоят в столбик, как в примере справа. Набрать текст с новой строки позволяет символ **\n**.

```
guess = input('Какое из этих животных рыба?\n \
1) Кит\n 2) Дельфин\n 3) Акула\n 4) Кальмар\n \
Введите 1, 2, 3 или 4. ')
check_guess(guess, '3')
```

```
Какое из этих животных рыба?
1) Кит
2) Дельфин
3) Акула
4) Кальмар
Введите 1, 2, 3 или 4.
```

Так этот вопрос будет выглядеть в окне консоли.

```
while still_guessing and attempt < 3:
    if guess.lower() == answer.lower():
        print('Ответ верный')
        score = score + 3 - attempt
        still_guessing = False
    else:
        if attempt < 2:
```

Код, который нужно ввести вместо score + 1.

◁ Меньше попыток — больше очков

Поощряй игрока за угадывание с меньшего количества попыток. Пусть за верный ответ с первого раза программа начисляет ему 3 очка, со второго — 2, а с третьего — 1. Для этого измени команду, в которой обновляется счет. Теперь он будет увеличиваться на 3 очка минус число неудачных попыток. Если игрок сразу ответит правильно, он получит 3 – 0 = 3 очка, если со второй попытки, то 3 – 1 = 2 очка, а если с третьей, то 3 – 2 = 1 очко.

▷ Ответы типа «да — нет»

Вот пример кода для вопросов с двумя вариантами ответа.

```
guess = input('Мыши — это млекопитающие. Да или нет? ')
check_guess(guess, 'Да')
```

▷ Измени сложность

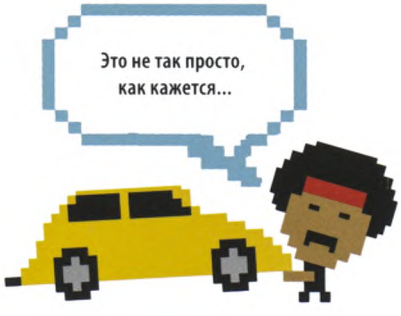
Чтобы усложнить тест, уменьши количество попыток ввода. Если это ответ типа «да — нет», не стоит давать игроку больше одной попытки, а для ответа на вопрос с несколькими вариантами ответа хватит, пожалуй, и двух попыток. Попробуй сам догадаться, как нужно изменить выделенные числа для одного и другого варианта.

```
def check_guess(guess, answer):
    global score
    still_guessing = True
    attempt = 0
    while still_guessing and attempt < 3:
        if guess.lower() == answer.lower():
            print('Ответ верный')
            score = score + 1
            still_guessing = False
        else:
            if attempt < 2:
                guess = input('Ответ неверный. Попробуйте еще раз. ')
            attempt = attempt + 1

    if attempt == 3:
        print('Правильный ответ: ' + answer)
```

Измени это число.

Измени это число.

Измени это число.



▷ Измени тему

Составь тест на другую тему, например про спорт, кино, музыку или на общую эрудицию. Можешь придумать вопросы про своих родных или друзей, в том числе провокационные, например: «У кого самый противный смех?»

Функции

Программисты любят хитрости, которые облегчают написание программ. Один из их самых известных трюков — дать особо полезному блоку кода имя. Тогда можно не вводить этот код каждый раз заново, а просто вызывать его по имени по мере необходимости. Такие именованные блоки кода называются функциями.

Как пользоваться функцией

Использование функции в коде называется вызовом. Чтобы вызвать функцию, надо ввести ее имя и поставить скобки, внутри которых поместить аргументы, необходимые для работы функции. Аргументы — это что-то вроде переменных, принадлежащих функции. Они позволяют передавать данные между разными частями кода. Если у функции нет аргументов, скобки оставляют пустыми.

СЛЕНГ

У функции есть...

Аргумент — значение, которое передается внутрь функции.

Возвращаемое значение — данные, которые можно вернуть из функции в основной код. Для этого служит ключевое слово **return** («вернуть»).

А еще функцию можно...

Вызвать — то есть использовать в коде.

Создать — когда ты после ключевого слова **def** пишешь код функции, это значит, что ты определяешь, или создаешь, функцию. Переменные тоже определяют — когда впервые присваивают им значения.

Встроенные функции

В Python есть встроенные функции для выполнения разных полезных задач: от ввода информации и вывода сообщений на экран до преобразования данных из одного типа в другой. Некоторыми встроенными функциями ты уже пользовался, например **print()** и **str()**, другими еще нет. Посмотри на эти примеры и попробуй ввести их в окне консоли.

Запрашивает имя пользователя.

```
>>> name = input('Как тебя зовут? ')
Как тебя зовут? Сара
>>> greeting = 'Привет, ' + name
>>> print(greeting)
Привет, Сара
```

Выводит на экран содержимое переменной greeting («приветствие»).

△ **Противоположные функции input() и print()**
Функция **input()** позволяет вводить данные с клавиатуры, а функция **print()** выводит результат работы кода на экран, например сообщение или результат математической операции.

▽ **Функция max()**

Функция **max()** находит наибольшее число среди всех аргументов, которые ей передают, то есть указывают внутри скобок через запятую.

```
>>> max(10, 16, 30, 21, 25, 28)
30
```

Самое большое число среди всех аргументов.

Всегда разделяй аргументы функции запятыми.

▽ **Функция min()**

Функция **min()** противоположна **max()**. Она выбирает из аргументов в скобках наименьшее число. Поэкспериментируй с этими функциями самостоятельно.

```
>>> min(10, 16, 30, 21, 25, 28)
10
```

Когда ты нажмешь ENTER, то увидишь на экране наименьшее число среди всех аргументов.

Другой способ вызова

У всех известных тебе типов данных — целых чисел, строк и списков — есть собственные функции. Чтобы их вызвать, нужно ввести сами данные или имя переменной, в которой они хранятся, через точку написать имя функции и поставить скобки. Попробуй набрать эти примеры в окне консоли.

Не забудь про точку.

Пустые скобки означают, что аргументов у функции нет.

```
>>> 'бабах'.upper()
'БАБАХ'
```

Новая строка, в которой все буквы прописные.

△ **Функция upper()**

Функция **upper()** («повысить») превращает все строчные буквы в имеющейся строке в прописные.

Эта функция принимает два аргумента.

```
>>> message = 'Python — это весело'
>>> message.replace('весело', ':D')
'Python — это :D'
```

Заменяет слово «весело» на смайлик.

△ **Функция replace()**

Этой функции нужны два аргумента: текст, который ты хочешь заменить, и строка, которую надо поставить на его место. Функция возвращает новую строку.

В переменной хранится список целых чисел.

```
>>> countdown = [1, 2, 3]
>>> countdown.reverse()
>>> print(countdown)
[3, 2, 1]
```

Теперь числа стоят в обратном порядке.

△ **Функция reverse()**

Используй эту функцию, чтобы изменить порядок элементов в списке на обратный. В этом примере функция «разворачивает» элементы списка **countdown** («обратный счет»), так что вместо [1, 2, 3] получается [3, 2, 1].

Создание функций

При создании собственных функций старайся называть их говорящими именами, поясняющими, что именно делают функции. Чтобы создать функцию, вычисляющую количество секунд в сутках, выполни следующие шаги.

1 Создай функцию

Создай в IDLE новый файл и сохрани его как functions.py («функции»). Введи в окне программы следующий код. Каждая строчка в теле функции должна начинаться с отступа в 4 пробела. Снова сохрани файл, запусти программу и посмотри, что произойдет.

```
def print_seconds_per_day():
    hours = 24
    minutes = hours * 60
    seconds = minutes * 60
    print(seconds)

print_seconds_per_day()
```

```
86400
```

Имя функции.

У функции пока нет аргументов.

Переменные.

Ключевое слово def означает, что следующий блок кода создает функцию.

Строчки, которые относятся к функции и идут после ее имени, должны начинаться с отступа в 4 пробела, чтобы Python принял их за тело функции.

Эта команда выводит на экран значение переменной seconds.

Вызов функции.

В окне консоли отобразится количество секунд в сутках.

2 Добавь аргументы

Если для работы функции нужны какие-нибудь значения, укажи их внутри скобок в качестве аргументов. Чтобы функция **print_seconds_per_day()** сообщала, сколько секунд в заданном количестве суток, измени ее код, как показано ниже. Теперь функция принимает аргумент **days** («дни»), так что при вызове ей нужно передавать количество суток.

```
def print_seconds_per_day(days):
    hours = days * 24
    minutes = hours * 60
    seconds = minutes * 60
    print(seconds)

print_seconds_per_day(7)
```

```
604800
```

Аргумент функции.

В этой переменной используется аргумент days.

Присваивает аргументу days значение 7.

Прежняя версия кода показана серым, а изменения — черным.

Количество секунд в 7 сутках.

СОВЕТЫ ЭКСПЕРТА

Главное правило

Важно, чтобы функции были созданы до их вызова в основном коде. Пока ты учишься программировать на Python, старайся помещать функции в самом начале программы. Тогда ты не сможешь вызвать по ошибке функцию, которая еще не создана.

3 Верни значение

У тебя есть функция, выполняющая полезную работу. Чтобы использовать результат ее работы в основном коде, нужно получить (вернуть) значение из функции. Для этого измени код так, как показано справа. Кроме того, имя функции стоит поменять, чтобы оно лучше отражало ее назначение. Пока не пытайся запустить этот код.

```
def convert_days_to_seconds(days):
    hours = days * 24
    minutes = hours * 60
    seconds = minutes * 60
    return seconds
```

Новое имя функции означает «перевести дни в секунды».

Команда return возвращает из функции значение переменной seconds («секунды»).

Строку с вызовом функции удаляем, поскольку имя и назначение функции поменялись.

4 Используй возвращаемое значение

Возвращаемое значение можно сохранить в переменной, чтобы использовать его позже. Добавь этот код после функции. Теперь значение функции попадает в переменную и используется для вычисления количества миллисекунд (тысячных долей секунды). Проверь работу этого кода, вводя разное количество дней.

Вызывает функцию, присваивая аргументу days значение 7.

```
def convert_days_to_seconds(days):
    hours = days * 24
    minutes = hours * 60
    seconds = minutes * 60
    return seconds

total_seconds = convert_days_to_seconds(7)
milliseconds = total_seconds * 1000
print(milliseconds)
```

```
604800000
```

Возвращаемое значение функции попадает в переменную total_seconds («всего секунд»).

Выводит значение переменной milliseconds («миллисекунд») на экран.

Это количество миллисекунд в 7 сутках.

Преобразует общее количество секунд в миллисекунды, сохраняя результат в переменной milliseconds.

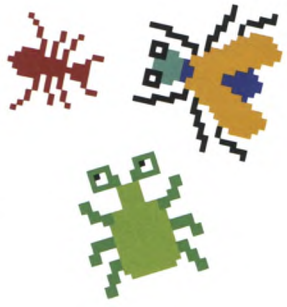
СОВЕТЫ ЭКСПЕРТА

Имя функции

На шаге 3 ты поменял название функции **print_seconds_per_day()** («напечатать количество секунд в сутках») на **convert_days_to_seconds()** («перевести дни в секунды»). Так же как в случае с переменными, важно, чтобы имя функции точно описывало ее назначение.

Тогда твой код будет гораздо проще понять. Правила именования функций такие же, как для переменных: имя может содержать цифры и латинские буквы, но начинаться должно с буквы. Если в имени несколько слов, разделяй их знаком подчеркивания.

Исправление ошибок



Если с кодом что-то не так, Python постарается выдать сообщение об ошибке. Поначалу такие сообщения могут сбивать с толку, однако они полезны, поскольку помогают понять, почему код не работает и как это исправить.

Сообщение об ошибке

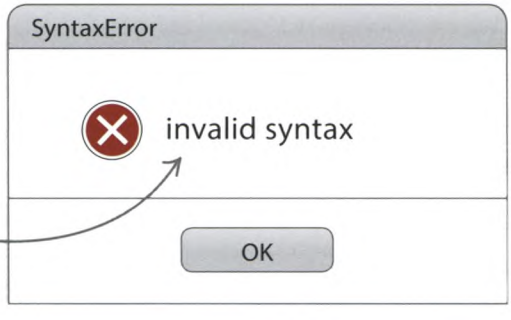
Сообщение об ошибке может выдать и окно программы, и окно консоли. В тексте сообщения по-английски указывается тип ошибки и место, в котором ее нужно искать.

▽ **Сообщение об ошибке в окне консоли**

В окне консоли сообщение об ошибке выделяется красным, а программа при этом завершается. Вот пример сообщения, в котором указана строка кода с ошибкой.

▽ **Сообщение об ошибке в окне программы**

Сообщение появится во всплывающем окне. Кликни OK, чтобы вернуться к коду; в месте ошибки ты увидишь красную пометку.

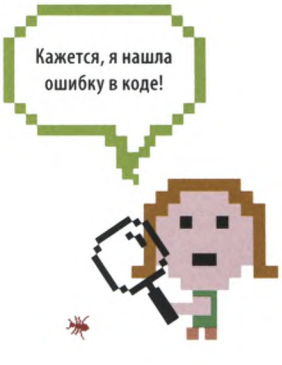


Ошибка синтаксиса означает, что какая-то команда введена неверно.

```
>>>
Traceback (most recent call last):
    File "Users/Craig/Developments/top-secret-python-book/age.py", line 21, in module>
        print('Мне '+ age + ' лет')
TypeError: Can't convert 'int' object to str implicitly
```

Обнаружена ошибка типизации (*см.* с. 50).

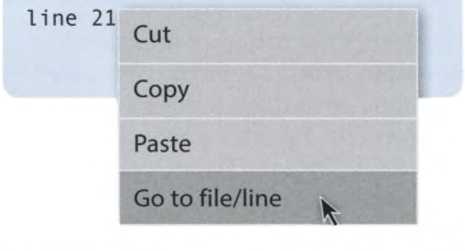
Ошибка в строке 21.



СОВЕТЫ ЭКСПЕРТА

Поиск багов

Когда в окне консоли появится сообщение об ошибке, кликни по нему правой кнопкой мыши и выбери Go to file/line. Окно программы откроется на проблемной строке кода.



Ошибка синтаксиса

Если у тебя высветилась ошибка синтаксиса (SyntaxError), значит, ты что-то неверно ввел — например, нажал не на ту клавишу. Не волнуйся, такой баг исправить легче всего. Тщательно изучи весь код и найди опечатку.

Здесь не хватает закрывающей круглой скобки.

```
input('Как тебя зовут?'
```

Пропущена первая одинарная кавычка.

```
print(Теперь твой ход')
```

▷ **На что обращать внимание**

Убедись, что у каждой скобки и кавычки есть пара. А может, ты не так написал ключевое слово? Все это может привести к ошибке синтаксиса.

Здесь опечатка: должно быть short_shots.

```
total_score = (long_shots * 3) + (shoort_shots * 2)
```

Ошибка отступа

Чтобы показать начало и конец блока кода, в Python используются отступы. Ошибка отступа (IndentationError) означает, что в структуре программы что-то не так. Запомни: если в конце строки стоит двоеточие, следующая строка должна начинаться с отступа. Чтобы добавить отступ, нажми клавишу ПРОБЕЛ 4 раза.

```
if weekday == True:
print('Пора в школу')
```

Такая запись приведет к ошибке отступа.

```
if weekday == True:
    print('Пора в школу')
```

4 пробела.

Чтобы исправить ошибку, нужно добавить отступ в начале второй строки.

▽ **Блоки и отступы**

В Python один блок кода нередко находится внутри другого блока: например, частью функции может быть цикл. Все строки определенного блока должны идти с одинаковым отступом. Хоть IDLE и старается ставить отступы автоматически, все равно нелишним будет проверить количество пробелов в каждом блоке.

Отступы позволяют программе понять, какая строка какому блоку принадлежит.

Ошибка типизации

Ошибка типизации (TypeError) означает, что в коде что-то не так с типами данных, например числа перепутаны со строками. Это похоже на то, что ты пытаешься испечь пирог в холодильнике: ничего не выйдет, ведь холодильник не печка! Не стоит удивляться такому сообщению об ошибке, если ты просишь Python о невозможном!


```
budget = 'пятьдесят' * 'пять'
```

В Python можно перемножать числа, но не строки.

```
hot_day = '20 градусов' > 15
```

Python не может сравнить строку с числом: это разные типы данных.

```
list = ['а', 'б', 'в']
find_biggest_number(list)
```

Эта функция ожидает получить список чисел, а ей передали список букв!

◁ **Примеры ошибок типизации**

Такие ошибки возникают, если попросить Python о чем-то бессмысленном: например, перемножить строки, или сравнить данные разных типов, или найти число в списке, состоящем из букв.

Ошибка именования

Ошибка именования (NameError) возникнет, если ввести в код имя еще не созданной переменной или функции. Чтобы такого не случилось, всегда объявляй переменные и создавай функции до того, как используешь их. Код функции лучше вводить в самом начале программы.

▷ **Ошибка именования**

Ошибка именования в этом коде не позволяет вывести сообщение «Мой город — Москва». Переменную **hometown** («родной город») нужно объявить до ее помещения в функцию **print()**.

Команда print() должна стоять после объявления переменной.

```
print('Мой город — ' + hometown)
hometown = 'Москва'
```

Логическая ошибка

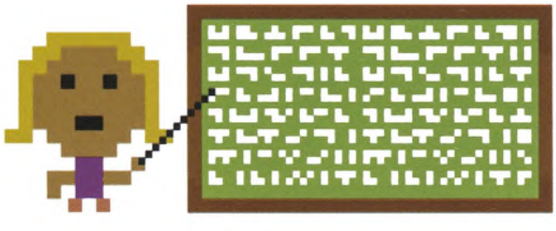
Иногда Python не выдает сообщений об ошибках, но ты понимаешь: с программой творится что-то неладное — она работает не так, как должна. Скорее всего, это значит, что в код закралась логическая ошибка. Возможно, ты все команды ввел правильно, но какой-то одной не хватает или строки перепутаны местами.


```
print('Караул! Вы потеряли жизнь!')
print(lives)
lives = lives - 1
```

Все команды введены правильно, но стоят не в том порядке.

◁ **Видишь ошибку?**

Этот код успешно запускается, и все же в нем есть логическая ошибка: значение переменной **lives** выводится на экран прежде, чем количество жизней уменьшается на 1. Игрок увидит неверное число! Чтобы это исправить, команду print(lives) нужно перенести в конец кода.



◁ **Строка за строкой**

Находить логические ошибки непросто, но со временем ты научишься их вылавливать. Для этого нужно тщательно, строка за строкой, проверять, что делает код. Прояви терпение, и в конце концов ты найдешь ошибку.

СОВЕТЫ ЭКСПЕРТА

Памятка по ловле багов

Порой тебе будет казаться, что ты никогда не заставишь программу работать как надо, однако не сдавайся! Следуя этим подсказкам, ты сможешь выловить большинство ошибок.

Проверь...

- Если ты ввел код из этой книги, но программа не работает, убедись, что твой код в точности совпадает с напечатанным.
- Нет ли в коде опечаток?
- Нет ли в начале строк лишних пробелов?
- Может, ты перепутал число и букву, например 0 (ноль) и O?
- Верно ли указаны прописные и строчные буквы?
- Есть ли для каждой открывающей скобки парная закрывающая? ( ) [ ] { }
- Есть ли для каждой одинарной или двойной кавычки парная закрывающая? ' ' " "
- Сохранил ли ты файл после того, как изменил код?
- Попроси кого-нибудь еще сравнить твой код с кодом из книги.

«Генератор паролей»

Пароли позволяют защитить наши компьютеры, почтовые аккаунты и учетные записи от посягательств посторонних лиц. Цель этого проекта — создать инструмент для генерации надежных и легко запоминающихся паролей.

▷ **Подсказка**

Пароль хорош, если его легко запомнить, но сложно подобрать или угадать.

Что происходит

Генератор паролей соединяет случайные слова, числа и символы в надежные комбинации. После запуска программа создаст пароль и выведет его на экран. Ты можешь запрашивать новый пароль снова и снова, пока не получишь тот, который тебе понравится.

СЛЕНГ

Взломщик паролей

Взломщик — это программа, с помощью которой хакеры подбирают пароли. Некоторые взломщики выдают миллионы вариантов в секунду, составляя их из распространенных слов и имен. Необычный пароль — лучшая защита от таких программ.

▽ **Блок-схема программы «Генератор паролей»**

Программа случайным образом выбирает каждую из четырех частей пароля, соединяет их и выводит результат на экран. Если ты запросишь другой пароль, программа повторит эти шаги, а если нет, завершит работу.

Как это работает

В ходе этого проекта ты научишься пользоваться модулем **random** («случайный»). С его помощью программа случайным образом выбирает прилагательные, существительные, числа и символы. Скоро к твоим услугам будут сложные, но запоминающиеся пароли вроде «oldapple14(» или «hotgoat&»!

Хитро, но просто!

Программа составляет хитрые пароли, но строчек кода в ней всего ничего, так что ее написание не займет много времени.

1 Создай новый файл

Открой IDLE, зайди в меню File, выбери New File и сохрани файл как password_picker.py («генератор паролей»).

2 Добавь модули

Загрузи модули **string** («строка») и **random** из стандартной библиотеки Python, добавив этот код в самое начало программы.

Модуль random нужен для выбора случайных значений.

```
import random
import string
```

Модуль string позволяет работать со строками — например, делить их на части или менять их вид.

3 Поздоровайся

Для начала создай приветственное сообщение.

Выводит на экран приветствие.

```
import random
import string
print('Добро пожаловать!')
```

4 Проверь работу программы

Запусти программу. В окне консоли должно появиться приветствие.

```
Добро пожаловать!
```

5 Создай список прилагательных

Для составления паролей понадобятся английские прилагательные и существительные. Удобнее всего их хранить в двух списках. Сначала создай список **adjectives** («прилагательные»), поставив его между командой **import** и вызовом функции **print()**. Затем присвой ему значение. Для этого внутри квадратных скобок введи элементы списка, разделив их запятыми.

Список хранится в переменной adjectives.

Каждый элемент списка является строкой.

Разделяй элементы запятыми.

```
import string

adjectives = ['sleepy', 'slow', 'hot',
              'cold', 'big', 'red',
              'orange', 'yellow', 'green',
              'blue', 'good', 'old',
              'white', 'free', 'brave']

print('Добро пожаловать!')
```

Список заключают в квадратные скобки.

6 Создай список существительных

Теперь создай список **nouns** («существительные»), хранящий существительные, и введи следующий код сразу после списка прилагательных. Не забывай про запятые и квадратные скобки.

```
              'white', 'free', 'brave']

nouns = ['apple', 'dinosaur', 'ball',
         'cat', 'goat', 'dragon',
         'car', 'duck', 'panda']

print('Добро пожаловать!')
```

Элементы списка нужно брать в кавычки.

7 Выбери слова

Чтобы создать пароль, программе нужно выбрать случайное прилагательное и случайное существительное. Получить их можно с помощью функции **choice()** («выбрать») из модуля **random**. Введи этот код после команды **print()**. (Функция **choice()** возвращает случайный элемент из переданного ей списка.)

```
print('Добро пожаловать!')

adjective = random.choice(adjectives)
noun = random.choice(nouns)
```

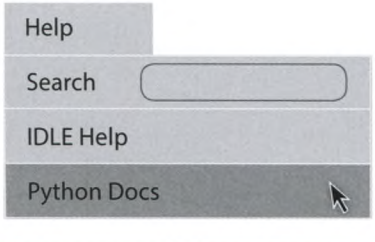
Прилагательное случайным образом выбирается из списка adjectives.

В эту переменную попадет случайное слово из списка существительных.

СОВЕТЫ ЭКСПЕРТА

Случайные числа

Случайные числа позволяют имитировать вытаскивание карты из колоды, подбрасывание монетки и т. д. Больше о случайных числах и модуле **random** можно узнать из справки (раздел Python Docs в меню Help).



8 Сгенерируй число

Получи случайное число от 0 до 99, вызвав функцию **randrange()** («диапазон случайного») из модуля **random**. Добавь этот код в конец программы.

```
noun = random.choice(nouns)
number = random.randrange(0, 100)
```

9 Получи символ

Снова воспользуйся функцией **random.choice()**, добавив в код команду для выбора случайного символа. Так пароль будет еще сложнее взломать!

```
number = random.randrange(0, 100)
special_char = random.choice(string.punctuation)
```

Это константа.

СОВЕТЫ ЭКСПЕРТА

Константа

Константа — особый вид переменных, содержимое которых нельзя изменять. Константа **string.punctuation** («строка. пунктуация») содержит строку с символами. Чтобы ее увидеть, введи в окне консоли import string, а затем print(string.punctuation).

```
>>> import string
>>> print(string.punctuation)
!"#$%&'()*+,-./:;<=>?@[\]^_'{|}~
```

Символы, хранящиеся в этой константе.

10 Создай пароль

Пришла пора соединить все имеющиеся части в один надежный пароль. Добавь эти строчки кода в конец программы.

Пароль будет храниться в этой переменной.

Преобразует случайное число в строку.

```
password = adjective + noun + str(number) + special_char
print('Новый пароль: %s' % password)
```

Выводит на экран новый пароль.

СОВЕТЫ ЭКСПЕРТА

Строки и целые числа

Функция **str()** преобразует целое число в строку. Если попытаться сложить число и строку без этой функции, Python выдаст ошибку. Попробуй сам: введи в окне консоли команду print ('трасса ' + 95).

```
>>> print('трасса ' + 95)
Traceback (most recent call last):
    File '<pyshell#0>', line 1, in <module>
        print('трасса ' + 95)
TypeError: Can't convert 'int' object to str implicitly
```

Сообщение об ошибке.

Чтобы исправить эту ошибку, преобразуй число в строку при помощи функции **str()**.

```
>>> print('трасса ' + str(95))
трасса 95
```

Число нужно написать внутри скобок функции str().

11 **Запусти программу**

Самое время запустить программу и посмотреть, что появится в окне консоли. Если возникнут ошибки, не волнуйся, а внимательно проверь весь код.

```
Добро пожаловать!
Новый пароль: bluegoat92=
```

У тебя должен отобразиться другой случайный пароль.

Не забудь сохранить свою работу.

12 **Еще один?**

Чтобы программа могла генерировать новые пароли по запросу пользователя, нам понадобится цикл **while**. Добавь в программу этот код. Он спрашивает, нужно ли подобрать другой пароль, и сохраняет ответ пользователя в переменной **response** («ответ»).

```
print('Добро пожаловать!')

while True:
    adjective = random.choice(adjectives)
    noun = random.choice(nouns)
    number = random.randrange(0, 100)
    special_char = random.choice(string.punctuation)

    password = adjective + noun + str(number) + special_char
    print('Новый пароль: %s' % password)

    response = input('Сгенерировать другой пароль? Введите д или н: ')
    if response == 'н':
        break
```

Начало цикла while.

Эти строчки нужно вводить с отступом, чтобы они попали в цикл while.

Завершает цикл while.

Функция input() запрашивает у пользователя ответ.

При ответе «да» (д) цикл запустится снова, а при ответе «нет» (н) он завершится.

13 **Выбери лучший пароль**

Вот и всё! Теперь ты можешь создавать надежные пароли, которые легко запоминаются!

```
Добро пожаловать!
Новый пароль: yellowapple42}
Сгенерировать другой пароль? Введите д или н: д
Новый пароль: greenpanda13*
Сгенерировать другой пароль? Введите д или н: н
```

Чтобы получить новый пароль, введи здесь «д».

Введи «н», чтобы выйти из программы.

Что бы изменить?

А теперь давай усовершенствуем программу, добавив ей новые возможности. Подумай: как можно сделать пароль еще надежнее?

▷ Больше слов

Чтобы увеличить количество паролей, которые может выдать программа, добавь в списки прилагательных и существительных новые слова.

```
nouns = ['apple', 'dinosaur', 'ball',
         'cat', 'goat', 'dragon',
         'car', 'duck', 'panda',
         'telephone', 'banana', 'teacher']
```

```
while True:

    for num in range(3):
        adjective = random.choice(adjectives)
        noun = random.choice(nouns)
        number = random.randrange(0, 100)
        special_char = random.choice(string.punctuation)

        password = adjective + noun + str(number) + special_char
        print('Новый пароль: %s' % password)

    response = input('Сгенерировать другой пароль? Введите д или н: ')
```

Цикл for сработает трижды, выдав три разных пароля.

Эти строки должны быть с отступом.

△ Несколько паролей

Измени код так, чтобы программа выдавала сразу три пароля. В этом тебе поможет цикл **for**. Помести его внутрь цикла **while**.

▷ Длинный пароль

Чтобы пароль получился длиннее (и надежнее), добавь в него еще одно слово. Например, случайное забавное прилагательное.

Добавь случайное забавное прилагательное.

```
Новый пароль: hairybluepotato33%
```

Модули

Модуль — это блок готового кода, который решает типовую задачу. Включив в программу модули, ты сможешь сосредоточиться на более интересных задачах. К тому же модулями пользуется много людей, и потому баги в них встречаются редко.

Встроенные модули

В комплекте с Python идет множество модулей. Все вместе они называются стандартной библиотекой. Вот несколько любопытных модулей, которые могут тебе пригодиться.

△ statistics

Модуль **statistics** («статистика») поможет найти среднее значение или выяснить, какое число чаще всего встречается в списке. С его помощью можно, например, узнать средний балл среди игроков.

▷ datetime

Модуль **datetime** («дата-время») предназначен для работы с датами. С его помощью ты сможешь узнать сегодняшнюю дату и выяснить, сколько дней осталось ждать какого-то события.

▷ random

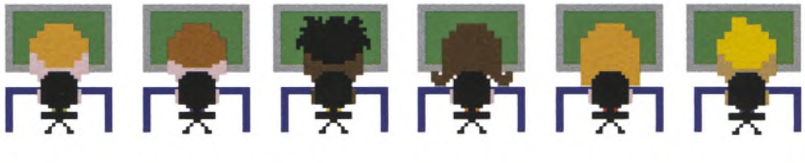
Ты уже использовал этот модуль для генерации случайных паролей. Он хорош, если в код нужно добавить элемент случайности.

▷ webbrowser

Этот модуль позволяет управлять веб-браузером, открывая интернет-страницы прямо из твоего кода.

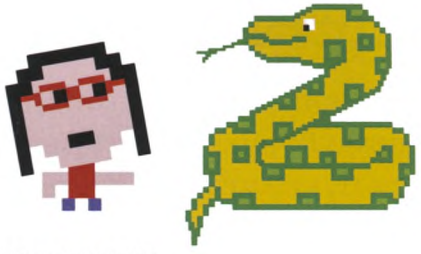
▷ socket

Модуль **socket** («разъем») позволяет программам общаться по сети или через интернет. Его можно использовать при создании онлайн-игры.



Использование модуля

Для начала модуль нужно загрузить в программу с помощью команды **import**. Сделать это можно разными способами, в зависимости от того, как именно ты хочешь его использовать.



▷ import...

Ключевое слово **import** открывает доступ ко всему содержимому модуля, однако название модуля придется писать перед вызовом каждой принадлежащей ему функции. Этот код загружает все функции модуля **webbrowser** и открывает сайт Python с помощью функции **open()** («открыть»).

```
>>> import webbrowser
>>> webbrowser.open('https://docs.python.org/3/library')
```

Загружает модуль webbrowser.

Название модуля следует писать перед названием каждой принадлежащей ему функции.

▷ from... import...

Если тебе нужна лишь часть модуля, ключевое слово **from** («из») позволит загрузить только ее. При этом названия функций можно вводить как обычно. Вот пример загрузки из модуля **random** функции **choice()**, которая выбирает случайный элемент из списка.

```
>>> from random import choice
>>> direction = choice(['С', 'Ю', 'З', 'В'])
>>> print(direction)
```

Из модуля random загрузится только функция choice().

Название модуля указывать не нужно.

Выводит на экран случайно выбранную сторону света.

▷ from... import... as...

Иногда нужно изменить имя загружаемого модуля или функции — например, потому что такое имя уже существует или оно плохо запоминается. Для этого служит ключевое слово **as** («как»). После него нужно написать новое имя. Вот пример, в котором функция **time()** («время»), возвращающая текущее время, загружается под именем **time_now()** («текущее время»). Результат работы программы — количество секунд, прошедших с 1 января 1970 года (даты, от которой ведет отсчет времени большинство компьютеров).

```
>>> from time import time as time_now
>>> now = time_now()
>>> print(now)
1527847928.510616
```

Загружает и переименовывает функцию time().

В этой переменной сохраняется переименованная функция.

Количество секунд, прошедших с 00:00 1 января 1970 года.

«Девять жизней»

В этой остросюжетной игре нужно называть буквы и угадывать слова. Если букву угадать не получается, игрок теряет жизнь. Думай над каждым ответом, ведь когда жизни закончатся, игра завершится!

Что происходит

Программа выводит на экран загаданное слово, в котором вместо букв — знаки вопроса. Если ты отгадаешь букву, программа ее «откроет». Если же ты знаешь слово, введи его целиком. Игра закончится, если слово угадано либо израсходованы все жизни.

Знаки вопроса, заменяющие буквы, служат подсказкой для игрока.

Оставшиеся жизни показаны в виде сердечек.

```
['?', '?', '?', '?', '?']
Осталось жизней:♥♥♥♥♥♥♥♥♥
Угадайте букву или слово целиком: а
['?', '?', '?', '?', 'а']
Осталось жизней:♥♥♥♥♥♥♥♥♥
Угадайте букву или слово целиком: и
['?', 'и', '?', '?', 'а']
Осталось жизней:♥♥♥♥♥♥♥♥♥
Угадайте букву или слово целиком: у
Неправильно. Вы теряете жизнь
['?', 'и', '?', '?', 'а']
Осталось жизней:♥♥♥♥♥♥♥♥
Угадайте букву или слово целиком: ц
['?', 'и', 'ц', 'ц', 'а']
Осталось жизней:♥♥♥♥♥♥♥♥
Угадайте букву или слово целиком: н
Неправильно. Вы теряете жизнь
['?', 'и', 'ц', 'ц', 'а']
Осталось жизней:♥♥♥♥♥♥♥
Угадайте букву или слово целиком: пицца
Победа! Было загадано слово пицца
```

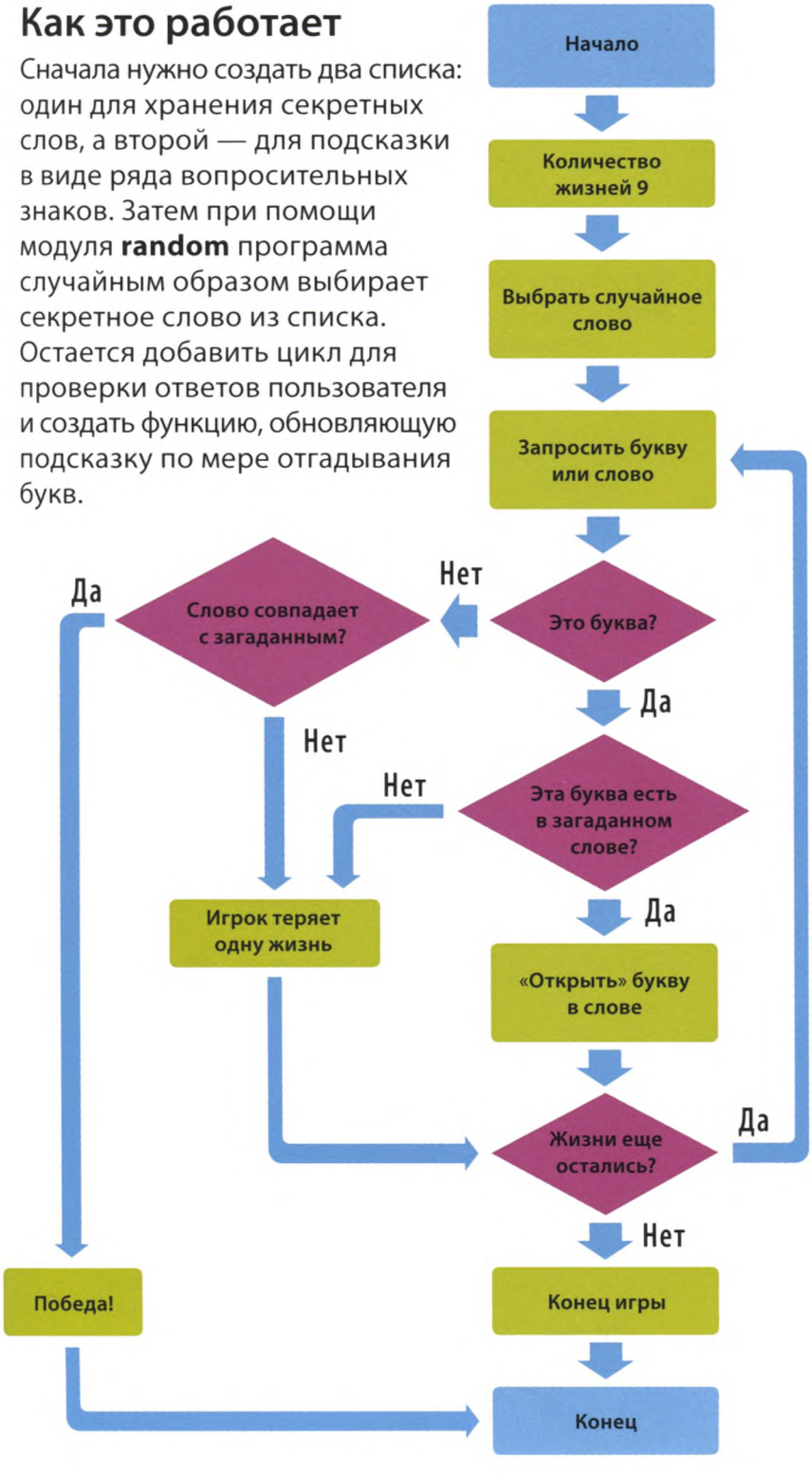
При каждом правильном ответе в слове «открывается» угаданная буква.

Если ответ неправильный, одно сердечко исчезает (жизнь сгорает).

Если ты знаешь слово, введи его целиком.

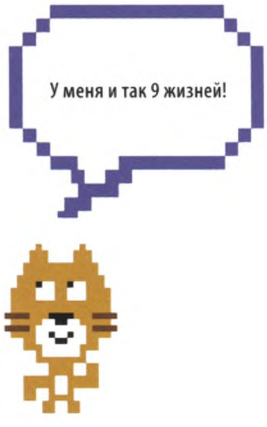
Как это работает

Сначала нужно создать два списка: один для хранения секретных слов, а второй — для подсказки в виде ряда вопросительных знаков. Затем при помощи модуля **random** программа случайным образом выбирает секретное слово из списка. Остается добавить цикл для проверки ответов пользователя и создать функцию, обновляющую подсказку по мере отгадывания букв.



▽ **Блок-схема программы «Девять жизней»**

Эта блок-схема сложна на вид, однако код игры сравнительно невелик. Основная часть программы — это цикл, проверяющий, есть ли указанные буквы в загаданном слове и остались ли еще жизни у игрока.



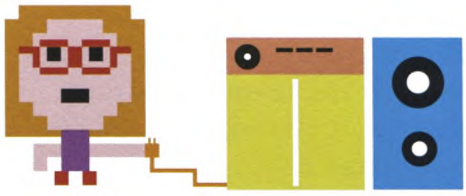
СОВЕТЫ ЭКСПЕРТА

Символы Юникода

В письменных языках мира буквы, цифры, знаки препинания, смайлики и другие элементы записываются при помощи символов. Чтобы компьютер мог отобразить их на экране, символы нужно закодировать. Для кодирования используются различные системы. Так, для знаков английского языка применяется набор символов ASCII. Сердечки для нашего проекта можно взять из набора Юникод, в котором есть множество значков, включая эти.

Подготовка

Создавать код игры «Девять жизней» лучше в два этапа. Сначала загрузим необходимые модули и объявим несколько переменных, а затем напишем основной код программы.



1 Создай новый файл

Открой IDLE, создай новый файл и сохрани его как nine_lives.py («девять жизней»).

2 Загрузи модуль

В этом проекте используется модуль **random**. Введи следующую строчку кода, чтобы его загрузить.

```
import random
```

3 Объяви переменную

После строки с командой **import** объяви переменную **lives**, в которой будет храниться количество оставшихся жизней (попыток ввода).

```
import random

lives = 9
```

В начале игры у игрока 9 жизней.

4 Создай список со словами

Программа будет загадывать только те слова, которые ты поместишь в список **words** («слова»). Добавь этот код после объявления переменной **lives.**

```
lives = 9
words = ['пицца', 'ангел', 'мираж', 'носки',
         'выдра', 'петух']
```

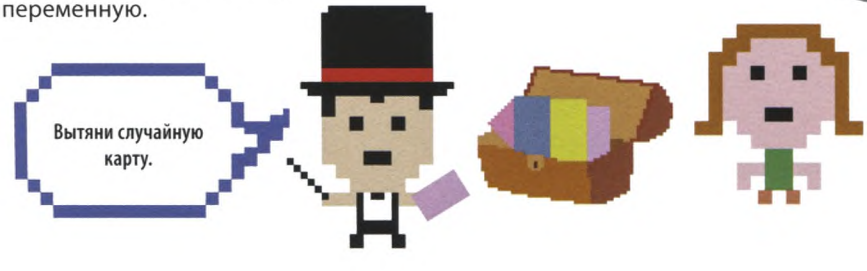
Каждый элемент списка — это строка, состоящая из 5 символов.

5 Добавь слова

В начале каждой игры программа будет случайным образом выбирать слово и сохранять его в переменной **secret_word** («секретное слово»). Добавь команду, объявляющую эту переменную.

```
words = ['пицца', 'ангел', 'мираж', 'носки',
         'выдра', 'петух']
secret_word = random.choice(words)
```

Здесь используется функция choice() из модуля random.



6 Создай список с подсказкой

Создай еще один список с подсказкой. Поначалу в нем будут только знаки вопроса, но по мере отгадывания слова они будут заменяться буквами. Список можно записать так: clue = ['?', '?', '?', '?', '?'], однако удобнее ввести следующий код, который делает то же самое. Добавь эту строку после объявления переменной **secret_word.**

```
secret_word = random.choice(words)
clue = list('?????')
```

Список из 5 знаков вопроса хранится в переменной clue.

7 Покажи количество жизней

Для отображения жизней в этом проекте используется символ Юникода «сердечко». Добавь код, сохраняющий его в переменной **heart_symbol** («символ “сердечко”»).

```
clue = list('?????')
heart_symbol = u'\u2764'
```

8 Состояние игры

Объяви переменную, которая будет хранить данные о том, угадал игрок слово или нет. Присвой ей значение False, поскольку в начале игры слово, разумеется, еще не отгадано. Введи этот код после объявления переменной **heart_symbol**.

```
heart_symbol = u'\u2764'
guessed_word_correctly = False
```

Одно из двух булевых значений (True или False).

СОВЕТЫ ЭКСПЕРТА

Длина слова

Будь внимателен: добавляй в игру только слова из 5 букв, ведь в списке с подсказкой есть место только для 5 символов. Если слово будет длиннее, при попытке обратиться к несуществующим элементам списка **clue** («подсказка») Python выдаст ошибку.

```
Index error: list assignment index
out of range
```

Если же ты добавишь слово с меньшим количеством букв, ошибок не возникнет, однако игрок все равно увидит 5 знаков вопроса и подумает, что загаданное слово длиннее, чем оно есть. Так программа будет работать со словом «жук».

```
['?', '?', '?', '?', '?']
Осталось жизней:♥♥♥♥♥♥♥♥♥
Угадайте букву или слово целиком: ж
['ж', '?', '?', '?', '?']
Осталось жизней:♥♥♥♥♥♥♥♥♥
Угадайте букву или слово целиком: у
['ж', 'у', '?', '?', '?']
Осталось жизней:♥♥♥♥♥♥♥♥♥
Угадайте букву или слово целиком: к
['ж', 'у', 'к', '?', '?']
Осталось жизней:♥♥♥♥♥♥♥♥♥
Угадайте букву или слово целиком:
```

Последние два знака вопроса лишние: таких букв в слове нет.

Игрок никогда не выиграет, поскольку не сможет «открыть» последние два знака вопроса!

Основной код

Основная часть кода — это цикл, который запрашивает у игрока букву и проверяет, есть ли она в слове. Если буква угадана, подсказка обновляется с помощью функции **update_clue()**. Сначала создадим эту функцию, а потом напишем сам цикл.

9 Есть ли буква в слове?

Если введенная пользователем буква есть в загаданном слове, подсказку нужно обновить. Для этого создай функцию **update_clue()** («обновить подсказку»), принимающую три аргумента: букву, загаданное слово и подсказку. Добавь этот код после объявления переменной **guessed_word_correctly** («слово отгадано верно»).

▷ **Как это работает**

В теле функции есть цикл **while**, перебирающий загаданное слово по буквам и проверяющий, совпадает ли хоть одна из них с введенной пользователем, которая хранится в переменной **guessed_letter** («отгаданная буква»). Переменная **index** («индекс») содержит позицию буквы, которая обрабатывается кодом цикла в текущий момент.

```
guessed_word_correctly = False


def update_clue(guessed_letter, secret_word, clue):
    index = 0
    while index < len(secret_word):
        if guessed_letter == secret_word[index]:
            clue[index] = guessed_letter
        index = index + 1
```

Функция len() возвращает количество букв в строке; в данном случае их 5.

Увеличивает значение переменной index на 1.

При совпадении введенной пользователем буквы с буквой загаданного слова программа подставляет ее в подсказку, используя значение index для поиска позиции в списке clue.

10 Угадай букву или слово

Программа должна раз за разом просить игрока ввести букву или слово целиком, пока он его не отгадает или пока не закончатся его жизни. Добавь этот код после функции **update_clue()**.

```
        index = index + 1

while lives > 0:
    print(clue)
    print('Осталось жизней: ' + heart_symbol * lives)
    guess = input('Угадайте букву или слово целиком: ')

    if guess == secret_word:
        guessed_word_correctly = True
        break

    if guess in secret_word:
        update_clue(guess, secret_word, clue)
    else:
        print('Неправильно. Вы теряете жизнь')
        lives = lives - 1
```

Цикл будет запускаться, пока жизни не закончатся.

Просит игрока ввести букву или слово.

Когда слово будет угадано, эта команда завершит цикл.

Выводят на экран подсказку и количество оставшихся жизней.

Обновляют подсказку, если введенная буква есть в слове.

Если буквы в слове нет (else), количество жизней уменьшается на 1.

СОВЕТЫ ЭКСПЕРТА

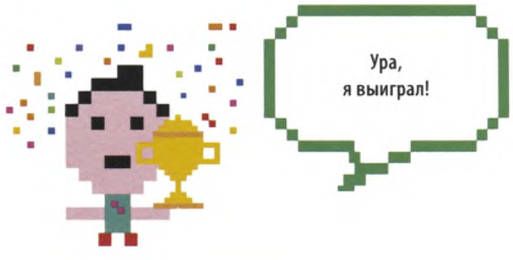
Повторение строк

В строке print('Осталось жизней:' + heart_symbol * lives), которая выводит на экран сердечки, использован хитрый трюк: чтобы повторить строку указанное количество раз, ее можно умножить на число. Например, код print(heart_symbol * 10) выведет на экран 10 сердечек. Попробуй ввести его в окне консоли.

```
>>> heart_symbol = u'\u2764'
>>> print(heart_symbol * 10)
♥♥♥♥♥♥♥♥♥♥
```

11 Игрок победил?

Когда игрок введет слово целиком, нужно проверить, угадал ли он его. Если значение переменной **guessed_word_correctly** равно True, значит, цикл завершился прежде, чем закончились жизни, и игрок победил. Иначе (**else**) он проиграл. Добавь этот код в конец программы.




```
lives = lives - 1

if guessed_word_correctly:
    print('Победа! Было загадано слово ' + secret_word)
else:
    print('Проигрыш! Было загадано слово ' + secret_word)
```

Сокращенная запись команды if guessed_word_correctly = True

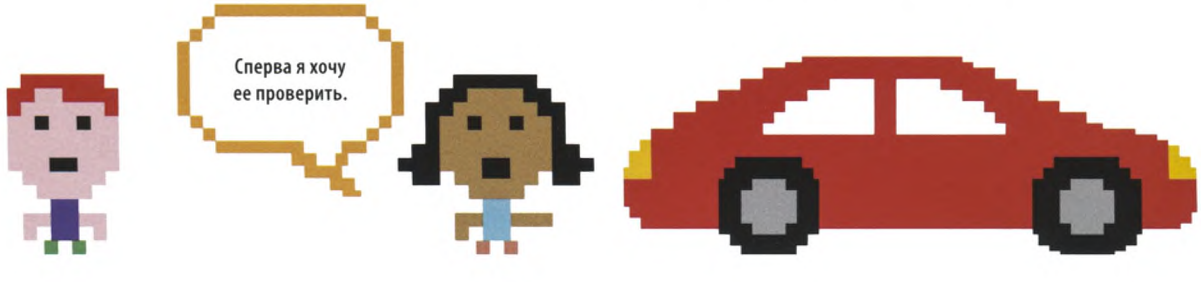
Не забудь сохранить свою работу.

12 Проверь работу программы

Запусти игру и убедись, что она работает. Если что-то пошло не так, тщательно проверь код на наличие ошибок и исправь их. А потом позови друзей сразиться в чемпионате под названием «Девять жизней»!

```
['?', '?', '?', '?', '?']
Осталось жизней:♥♥♥♥♥♥♥♥♥♥
Угадайте букву или слово целиком:
```

Введи букву, чтобы начать игру!



Что бы изменить?

Эту игру можно дорабатывать как душе угодно: усложнять или упрощать, добавлять слова или изменять их длину.

▽ Добавь слова

Увеличь список слов для отгадывания: их может быть сколько угодно много, однако не забывай, что все слова должны состоять из 5 букв.

```
words = ['пицца', 'ангел', 'мираж', 'носки', 'выдра', 'петух', 'банан', 'кокос', 'олень']
```

▽ Измени количество жизней

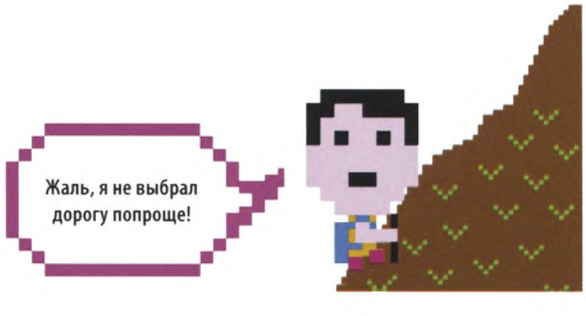
Увеличь или уменьши количество жизней, чтобы сделать игру проще или сложнее. Для этого обнови значение переменной **lives**, созданной на шаге 3.

◁ Длинные слова

Если ты считаешь, что 5 букв — это слишком просто, возьми слова подлиннее, лишь бы количество букв в них было одинаковым. Пролистай толковый словарь и найди самые длинные и необычные слова — пусть твоя игра станет чертовски сложной!

Добавь уровни сложности

Чтобы играть стало интереснее, предложи игроку выбрать уровень сложности: чем выше сложность, тем меньше жизней.



1 Запроси уровень

Перед циклом **while** добавь код, предлагающий игроку выбрать уровень сложности.

```
difficulty = input('Выберите уровень сложности (1, 2 или 3):\n 1 Легкий\n 2 Средний\n 3 Трудный\n')
difficulty = int(difficulty)

while lives > 0:
```

Превращает строковое значение переменной difficulty в целое число.

2 **Проверь работу программы**
Запусти программу и убедись, что она работает. В окне консоли должно появиться такое сообщение.

```
Выберите уровень сложности (1, 2 или 3):
1 Легкий
2 Средний
3 Трудный
```

3 **Задай сложность**
С помощью команд **if**, **elif** и **else** задай количество жизней для каждого уровня сложности: например, 12 для легкого, 9 для среднего и 6 для трудного. Добавь следующий код после команд, запрашивающих у пользователя уровень сложности.


```
difficulty = input('Выберите уровень сложности (1, 2 или 3):\n 1 Легкий\n 2 Средний\n 3 Трудный\n')
difficulty = int(difficulty)

if difficulty == 1:
    lives = 12
elif difficulty == 2:
    lives = 9
else:
    lives = 6
```

Слова разной длины

А если добавить в игру слова с разным количеством букв? Сколько знаков вопроса должно быть в списке, если длина секретного слова неизвестна заранее? Вот хитрый способ, который поможет решить эту проблему.

1 **Создай пустой список**
Не заполняй список с подсказкой знаками вопроса, а оставь его пустым.

```
clue = []
```

Внутри квадратных скобок должно быть пусто.

2 Добавь новый цикл

Чтобы после выбора слова список **clue** оказался нужной длины, используй этот код. Узнав количество букв в слове, он добавит в **clue** по одному знаку вопроса для каждой буквы.

```
clue = []
index = 0
while index < len(secret_word):
    clue.append('?')
    index = index + 1
```

Функция append() («добавить») добавляет новый элемент в конец списка.

Умное завершение игры

Сейчас, чтобы закончить игру, надо ввести слово целиком. Давай изменим код так, чтобы игра прекращалась, как только игрок откроет последнюю неотгаданную букву.

1 Создай еще одну переменную

Создай переменную для хранения количества неотгаданных букв. Введи этот код перед функцией **update_clue**.

В начале игры все буквы неизвестны.

```
unknown_letters = len(secret_word)
```

2 Измени функцию

Измени код функции **update_clue()**, как показано ниже. Когда игрок отгадает букву, программа вычтет из значения переменной **unknown_letters** («неотгаданные буквы») то количество букв, которое откроется в слове.


```
def update_clue(guessed_letter, secret_word, clue, unknown_letters):
    index = 0
    while index < len(secret_word):
        if guessed_letter == secret_word[index]:
            clue[index] = guessed_letter
            unknown_letters = unknown_letters - 1
        index = index + 1

    return unknown_letters
```

Добавь этот аргумент в функцию update_clue.

Уменьшает значение переменной unknown_letters на 1, когда находит каждую отгаданную букву в слове.

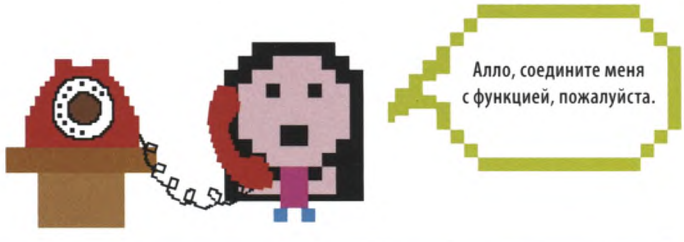
Возвращает количество неотгаданных букв.

◁ **Как это работает**

Зачем изменять значение переменной **unknown_letters** именно внутри функции **update_clue()**? Почему бы просто не вычесть единицу, выяснив, что игрок угадал букву? Потому что этот вариант сработает, лишь если ни одна буква в слове не повторяется, — иначе значение **unknown_letters** будет неверным. Обновляя переменную внутри функции, программа уменьшает ее значение на 1 для каждой буквы, повторяющейся в слове, и делает это одновременно с проверкой букв на совпадение с ответом игрока.

3 Передай значение переменной

Кроме того, нужно изменить функцию **update_clue()**, передав в нее значение переменной **unknown_letters** и сохранив новое значение функции обратно в переменную.




```
if guess in secret_word:
    unknown_letters = update_clue(guess, secret_word, clue, unknown_letters)
else:
    print('Неправильно. Вы теряете жизнь')
    lives = lives - 1
```

Передача значения переменной unknown_letters в функцию update_clue().

Присваивает переменной unknown_letters новое значение.

4 Выигрыш

Если значение переменной **unknown_letters** уменьшилось до 0, значит, игрок отгадал все буквы. Добавь этот код в конец основного цикла. Теперь программа автоматически распознает выигрыш, если все буквы в слове открыты.

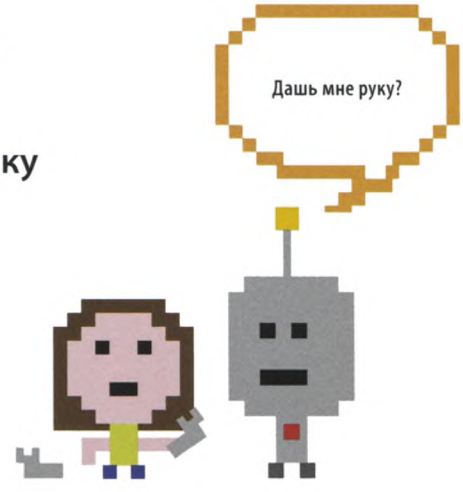
```
        lives = lives - 1

    if unknown_letters == 0:
        guessed_word_correctly = True
        break
```

Команда break завершает цикл, если все буквы в слове отгаданы.

Черепашья графика

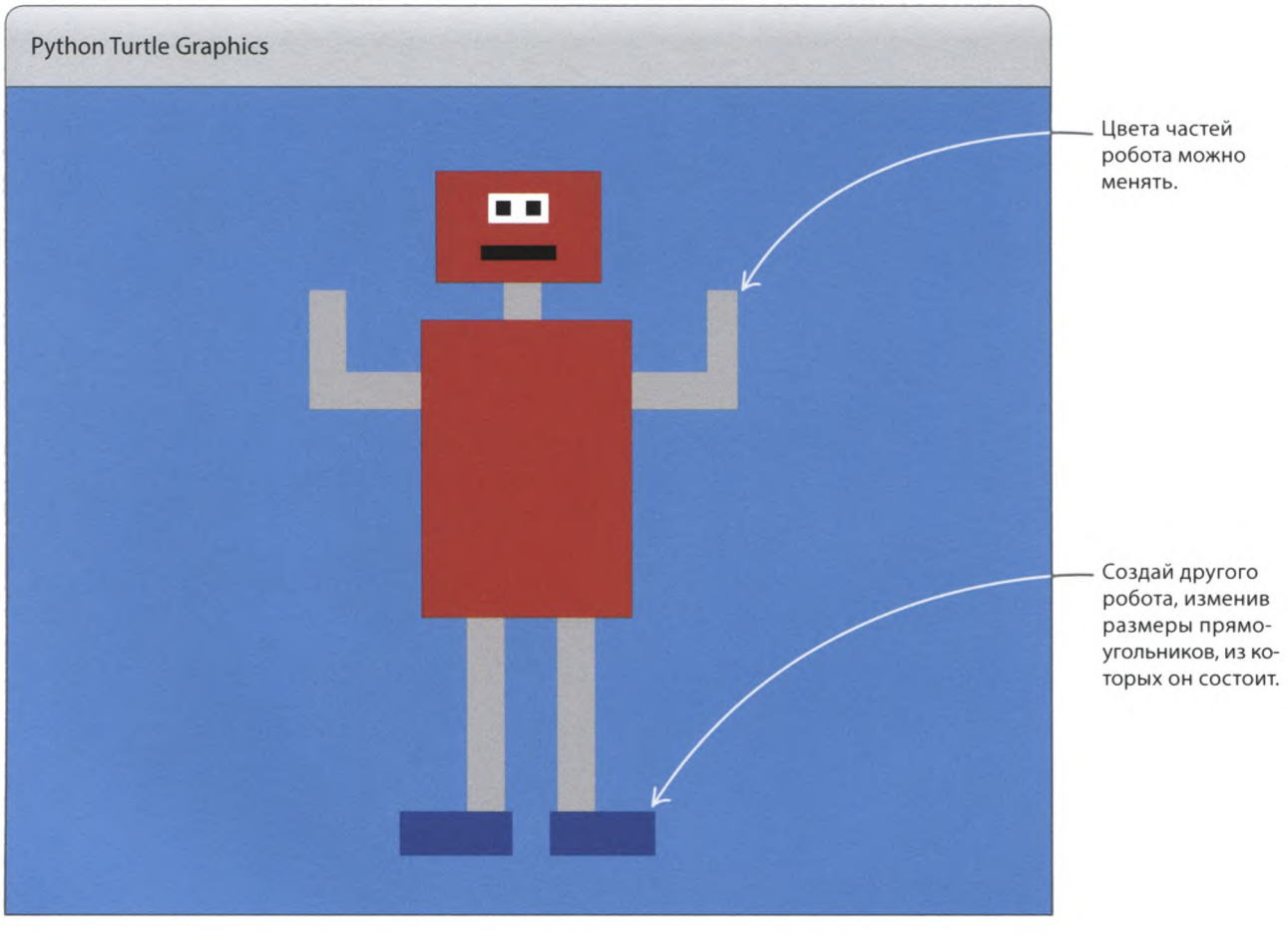
«Сборщик роботов»

Создавать графические изображения на Python легко. Модуль turtle («черепашка») позволяет перемещать по экрану черепашку, рисующую картинки. Цель этого проекта — научить черепашку рисовать робота-андроида!



Что происходит

После запуска программы появляется графическое окно, по которому движется черепашка, рисуя дружелюбного робота. Можно наблюдать, как она создает его часть за частью.



Как это работает

Сначала создадим функцию, которая рисует прямоугольники, а затем научим программу составлять из них робота. Размер и цвет прямоугольников можно выбирать, меняя аргументы функции: ноги могут быть длинными и узкими, глаза квадратными и т. д.

▽ **Не называй меня черепахой!**

Никогда не сохраняй файлы программ, использующих черепашку, под именем turtle.py: Python запутается и выдаст кучу ошибок.

▽ **Рисование черепашкой**

Модуль **turtle** позволяет управлять инструментом «черепашка», который оснащен пером. С помощью функций модуля черепашку можно заставить двигаться в нужном направлении и рисовать самые разные картинки. Также можно указать, когда опустить перо и начать рисовать, а когда поднять и переместиться в другую часть экрана, не оставляя за собой следа.

Черепашка движется вперед на 100 пикселей, поворачивает налево на 90 градусов и проходит еще 50 пикселей.

```
t.forward(100)
t.left(90)
t.forward(50)
```

▽ **Блок-схема программы «Сборщик роботов»**

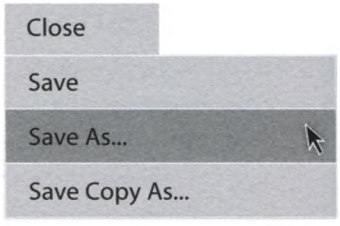
Блок-схема показывает, какие команды выполняет программа. Сначала она задает цвет фона и скорость черепашки, затем черепашка по частям рисует робота, начиная с ног и заканчивая головой.

Рисуем прямоугольники

Загрузим модуль **turtle** и нарисуем с его помощью прямоугольники.

1 **Создай новый файл**

Открой IDLE и создай новый файл, сохранив его как robot_builder.py («сборщик роботов»).



2 **Загрузи модуль turtle**

Добавь в начало программы эту строчку. Команда import turtle as t позволяет вызывать функции модуля **turtle**, указывая вместо полного названия модуля просто «t». Это все равно что называть Ярослава сокращенным именем Яр.

```
import turtle as t
```

Создает для модуля turtle псевдоним «t».

3 **Создай функцию rectangle()**

Создай функцию для рисования прямоугольников, из которых состоит робот. Функция **rectangle()** («прямоугольник») принимает три параметра: ширину, высоту и цвет прямоугольника. Воспользуемся циклом, в котором во время каждого из двух проходов будет появляться одна горизонтальная и одна вертикальная сторона прямоугольника. Добавь эту функцию после кода, введенного на шаге 2.

Как и все языки программирования, Python использует американское написание color («цвет»), а не английское colour.

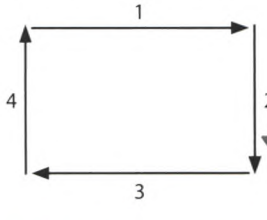
```
def rectangle(horizontal, vertical, color):
    t.pendown()
    t.pensize(1)
    t.color(color)
    t.begin_fill()
    for counter in range(1, 3):
        t.forward(horizontal)
        t.right(90)
        t.forward(vertical)
        t.right(90)
    t.end_fill()
    t.penup()
```

Опускает перо и начинает рисовать.

Числа внутри списка range (1, 3) означают, что цикл сделает 2 прохода.

Этот блок кода рисует прямоугольник.

Поднимает перо, чтобы прекратить рисование.

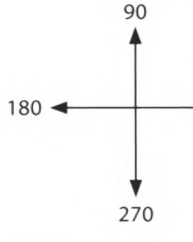


В таком порядке черепашка рисует стороны прямоугольника.

СОВЕТЫ ЭКСПЕРТА

Режим черепашки

Твоя черепашка работает в стандартном режиме. Это значит, что после запуска программы она будет смотреть вправо. С помощью функции **setheading()** («задать направление») ее можно развернуть на определенный угол: значение 90 направит ее вверх, 180 — влево, а 270 — вниз.



Обычно черепашка выглядит как стрелочка. Эта строчка превращает ее в черепашку.

```
t.shape('turtle')
t.setheading(0)
t.forward(80)
```

СОВЕТЫ ЭКСПЕРТА

Скорость черепашки

Скорость черепашки можно менять от самой медленной до самой быстрой, вызывая функцию **t.speed()** («скорость») с одним из этих значений в качестве аргумента: slowest, slow, normal, fast и fastest.

4 Добавь фон

Теперь нужно подготовить черепашку к рисованию и задать цвет фона графического окна. Вначале перо черепашки должно быть поднято — до тех пор, пока она не начнет рисовать ступню робота (шаг 5). Введи следующие строки после кода, добавленного на шаге 3.

Поднимает перо черепашки.

Устанавливает скорость «медленно».

```
t.penup()
t.speed('slow')
t.bgcolor('Dodger blue')
```

Делает фон светло-васильковым.

Создаем робота

Теперь можно приступать к созданию робота. Давай нарисуем его по частям, начиная со ступней. Робот будет состоять из прямоугольников разного размера и цвета, которые нужно расположить в разных частях окна.

5 Нарисуй ступни

Помести черепашку в позицию первой ступни и нарисуй прямоугольник, вызвав функцию **rectangle()**. То же касается и второй ступни. Введи эти строчки кода после тех, что были добавлены на шаге 4, запусти программу и проверь, появились ли ступни робота.

Этот комментарий поясняет, какую часть робота мы рисуем.

```
# ступни
t.goto(-100, -150)
rectangle(50, 20, 'blue')
t.goto(-30, -150)
rectangle(50, 20, 'blue')
```

Перемещает черепашку в точку с координатами x = –100, y = –150.

Рисует синий прямоугольник шириной 50 и высотой 20 пикселей.

СОВЕТЫ ЭКСПЕРТА

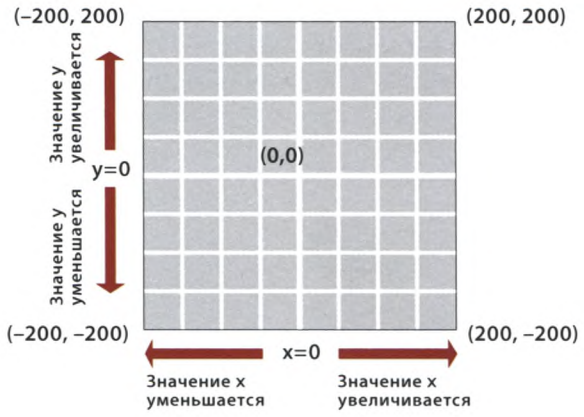
Комментарий (#)

Некоторые строчки кода в этой программе начинаются со знака «решетка» (#). То, что стоит после него, — это комментарий. Комментарии нужны, чтобы пояснять работу кода. Python знает об этом и игнорирует такие строчки.

СОВЕТЫ ЭКСПЕРТА

Координаты черепашки

Python выставляет размеры графического окна так, чтобы оно вписывалось в экран. Предположим, что размер окна 400 × 400 пикселей. Для определения местоположения черепашки Python использует координаты. Это значит, что любой точке окна соответствуют 2 числа. Первое — координата *x* — показывает, насколько черепашка смещена влево или вправо от центра окна. Второе число — координата *y* — обозначает смещение черепашки от центра вверх или вниз. Координаты обычно пишут в скобках и первой ставят *x*: (x, y).



6 Нарисуй ноги

Следующие команды заставляют черепашку переместиться в то место, откуда начнется рисование ног. Введи эти строчки после кода, добавленного на шаге 5, и снова запусти программу.

```
# ноги
t.goto(-25, -50)
rectangle(15, 100, 'grey')
t.goto(-55, -50)
rectangle(-15, 100, 'grey')
```

Перемещает черепашку в точку с координатами x = –25, y = –50.

Рисует левую ногу.

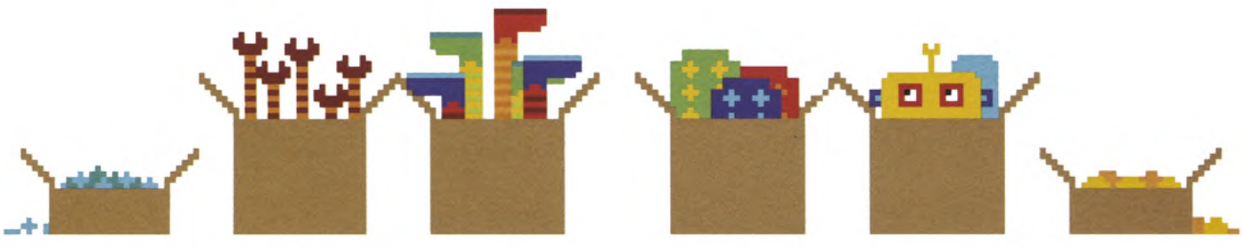
Рисует правую ногу.

7 Нарисуй туловище

Введи эти строчки после кода, добавленного на шаге 6. Запусти программу: должно появиться туловище робота.

```
# туловище
t.goto(-90, 100)
rectangle(100, 150, 'red')
```

Рисует красный прямоугольник размером 100 на 150 пикселей.



8 Нарисуй руки

Каждая рука состоит из двух частей: верхней — плеча — и нижней — предплечья. Введи эти строчки после кода, добавленного на шаге 7, запусти программу и убедись, что у робота появились руки.

```
# руки
t.goto(-150, 70)
rectangle(60, 15, 'grey')
t.goto(-150, 110)
rectangle(15, 40, 'grey')

t.goto(10, 70)
rectangle(60, 15, 'grey')
t.goto(55, 110)
rectangle(15, 40, 'grey')
```

Плечо левой руки.

Предплечье левой руки.

Предплечье правой руки.

Плечо правой руки.

9 Нарисуй шею

Теперь создадим шею. Введи команды рисования шеи после кода, добавленного на шаге 8.

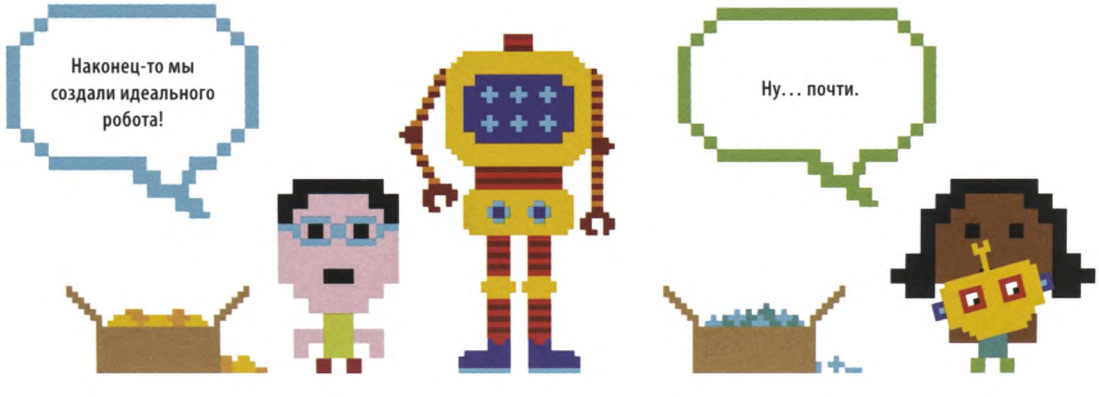
```
# шея
t.goto(-50, 120)
rectangle(15, 20, 'grey')
```

10 Нарисуй голову

Упс, ты нарисовал безголового робота! Чтобы снабдить беднягу головой, введи эти команды после кода, добавленного на шаге 9.

```
# голова
t.goto(-85, 170)
rectangle(80, 50, 'red')
```

Не забудь сохранить свою работу.



11 Нарисуй глаза

Давай добавим роботу глаза, чтобы он видел, куда направляется. Для этого нарисуем в районе головы белый прямоугольник с двумя черными квадратиками (это белки и зрачки глаз). Создавать функцию, рисующую квадраты, не нужно, ведь квадрат — это равносторонний прямоугольник. Введи эти команды после кода, добавленного на шаге 10.

```
# глаза
t.goto(-60, 160)
rectangle(30, 10, 'white')
t.goto(-55, 155)
rectangle(5, 5, 'black')
t.goto(-40, 155)
rectangle(5, 5, 'black')
```

Рисует белки глаз.

Рисует левый зрачок.

Рисует правый зрачок.

12 Нарисуй рот

Теперь давай нарисуем роботу рот. Введи эти строчки после кода, добавленного на шаге 11.

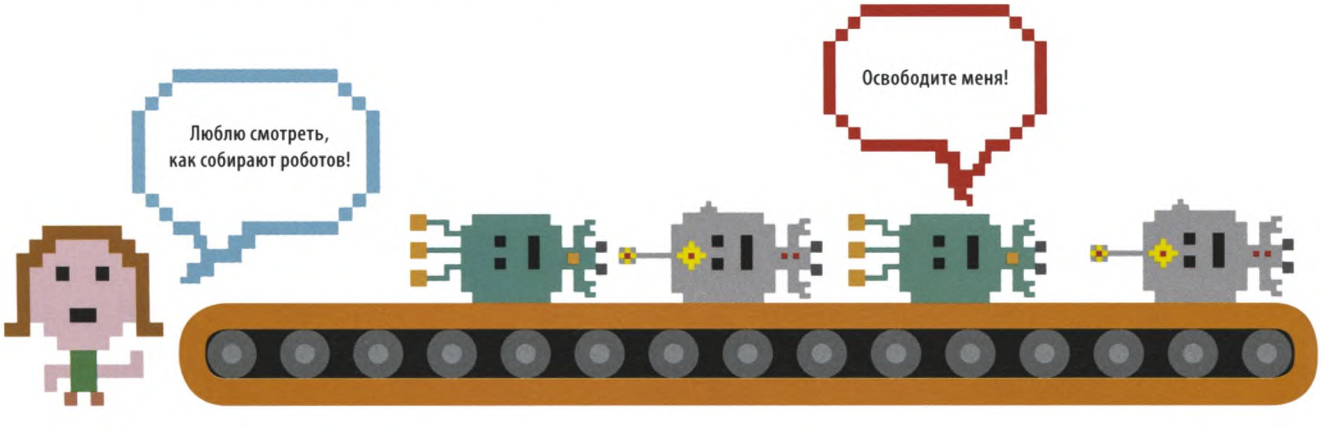
```
# рот
t.goto(-65, 135)
rectangle(40, 5, 'black')
```

13 Спрячь черепашку

И наконец, спрячь черепашку, иначе она останется у робота на лице. Введи эту строчку после кода, добавленного на шаге 12. Запусти программу и наблюдай, как создается робот.

```
t.hideturtle()
```

Делает черепашку невидимой.



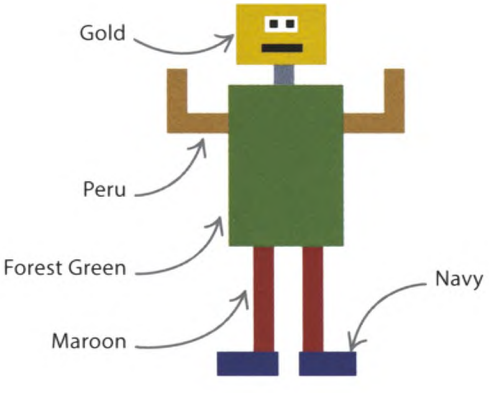
Что бы изменить?

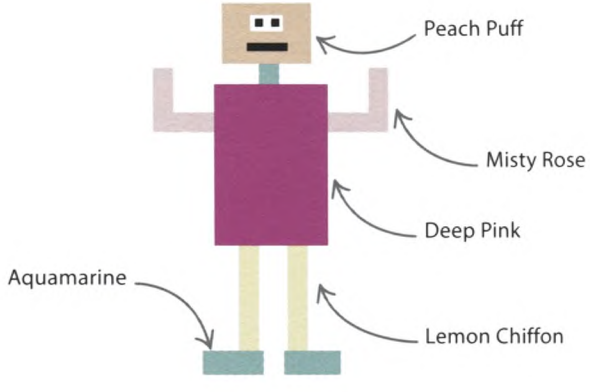
Итак, программа запускается и работает. А не хочешь ли ты изменить внешний вид робота? Вот несколько идей, как это можно сделать.

Поменяй цвета

Робот и так разноцветный, но ты можешь сделать его еще ярче! Например, подобрать цвета, подходящие к обоям в твоей комнате, или напоминающие цвета любимого футбольного клуба. Справа и ниже перечислены некоторые оттенки, известные черепашке.

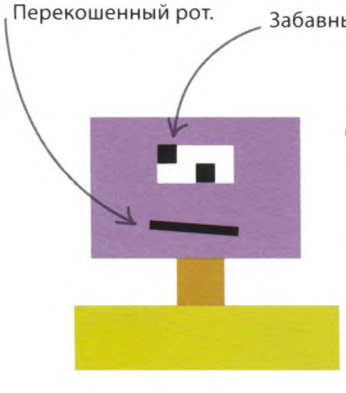
Lawn Green | Seashell | Blue
Purple | Light Blue | Yellow
Goldenrod | Hot Pink | Thistle





▷ Измени лицо

А еще можно изменить выражение лица робота. Например, этот код сделает его глаза и рот перекошенными.




```
# глаза
t.goto(-60, 160)
rectangle(30, 10, 'white')
t.goto(-60, 160)
rectangle(5, 5, 'black')
t.goto(-45, 155)
rectangle(5, 5, 'black')

# рот
t.goto(-65, 135)
t.right(5)
rectangle(40, 5, 'black')
```

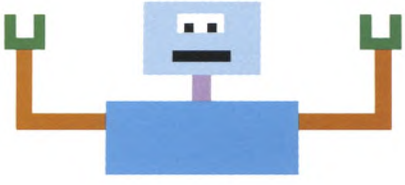
Сдвигает правый зрачок вниз.

Сдвигает левый зрачок в левый верхний угол белого прямоугольника, чтобы казалось, будто робот вращает глазами.

Отклоняет черепашку вправо, чтобы рот перекосился.

▷ **Рука помощи**

Добавь этот код, если хочешь приделать к рукам робота кисти-захваты. Можешь дать волю воображению и поменять их форму так, чтобы получились крюки, клешни или что-то еще!



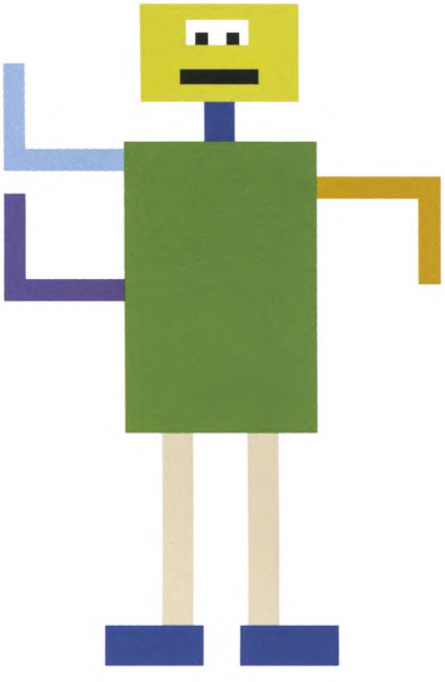
```
# руки
t.goto(-155, 130)
rectangle(25, 25, 'green')
t.goto(-147, 130)
rectangle(10, 15, t.bgcolor())
t.goto(50, 130)
rectangle(25, 25, 'green')
t.goto(58, 130)
rectangle(10, 15, t.bgcolor())
```

Рисует зеленый квадрат — основу кисти-захвата.

Рисует маленький прямоугольник цвета фона, чтобы придать кисти форму захвата.

Цельные руки

Если руки нарисованы в два этапа, то менять их положение или добавлять дополнительные руки не очень удобно. Поэтому предлагаю создать функцию, которая рисует руку целиком.



1 Создай функцию arm()

Добавь в код эту новую функцию, которая рисует руку заданным цветом.

```
    t.end_fill()
    t.penup()

def arm(color):
    t.pendown()
    t.begin_fill()
    t.color(color)
    t.forward(60)
    t.right(90)
    t.forward(50)
    t.right(90)
    t.forward(10)
    t.right(90)
    t.forward(40)
    t.left(90)
    t.forward(50)
    t.right(90)
    t.forward(10)
    t.end_fill()
    t.penup()
    t.setheading(0)
```

Заливает нарисованный контур цветом.

Задает цвет.

Выполняя эти команды, черепашка рисует руку.

Прекращает заливку цветом.

Снова разворачивает черепашку вправо.

2 Добавь руки

Замени все команды, которые стоят между комментарием # руки и комментарием # шея, на этот код, трижды вызывающий функцию **arm()** («рука»).

```
# руки
t.goto(-90, 85)
t.setheading(180)
arm('light blue')

t.goto(-90, 20)
t.setheading(180)
arm('purple')

t.goto(10, 85)
t.setheading(0)
arm('goldenrod')
```

Разворачивает черепашку, которая находится в левой части окна, влево.

Функция arm() рисует руку голубого цвета.

Разворачивает черепашку, которая находится в правой части окна, вправо.

▽ Руки в движении

Теперь, когда руки рисуются цельными, несложно поменять их положение, чтобы создать впечатление, будто робот чешет затылок или танцует! Для этого, перед тем как приступить к рисованию руки, разверни черепашку с помощью функции **setheading()**.

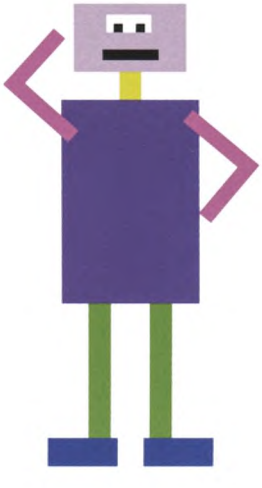
```
# руки
t.goto(-90, 80)
t.setheading(135)
arm('hot pink')

t.goto(10, 80)
t.setheading(315)
arm('hot pink')
```

Разворачивает черепашку в сторону левого верхнего угла экрана.

Рисует левую руку.

Разворачивает черепашку в сторону правого нижнего угла экрана.

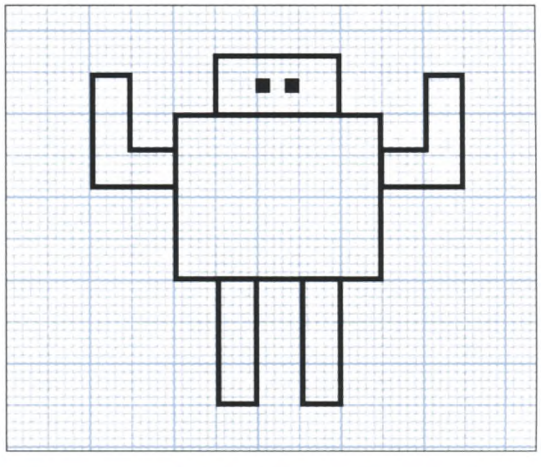
Рисует правую руку.



СОВЕТЫ ЭКСПЕРТА

Пробы и ошибки

Не всегда то, что ты задумал, будет получаться с первого раза: без утомительных проб и ошибок не обойтись. Если добавить после строчки t.speed('slowest') команды print(t.window_width()) («ширина окна») и print(t.window_height()) («высота окна»), Python отобразит в окне консоли ширину и высоту графического окна. Зная их, можно нарисовать на листе в клетку или миллиметровке робота и определить координаты его частей.



«Радуга-пружинка»

Так же как из простых строк кода создаются сложные программы, из простых фигур можно составлять замысловатые изображения. Написав код «Радуги-пружинки», ты сможешь комбинировать цвета и фигуры и создавать шедевры цифрового искусства!

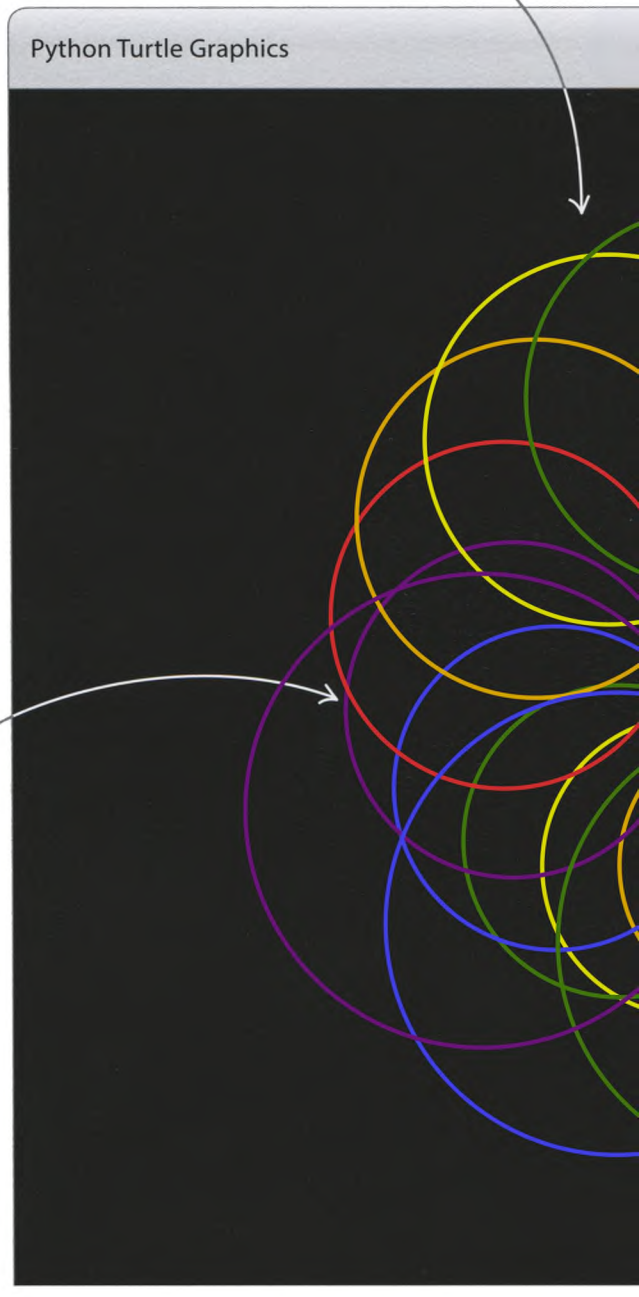
Что происходит

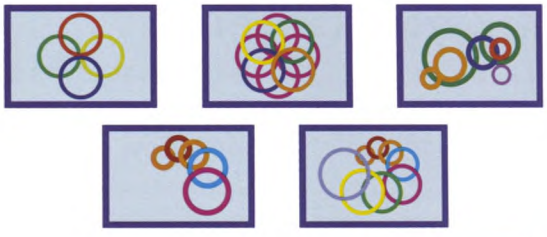
Черепашка рисует окружности разных цветов и размеров в разных частях экрана, создавая яркий замысловатый узор.

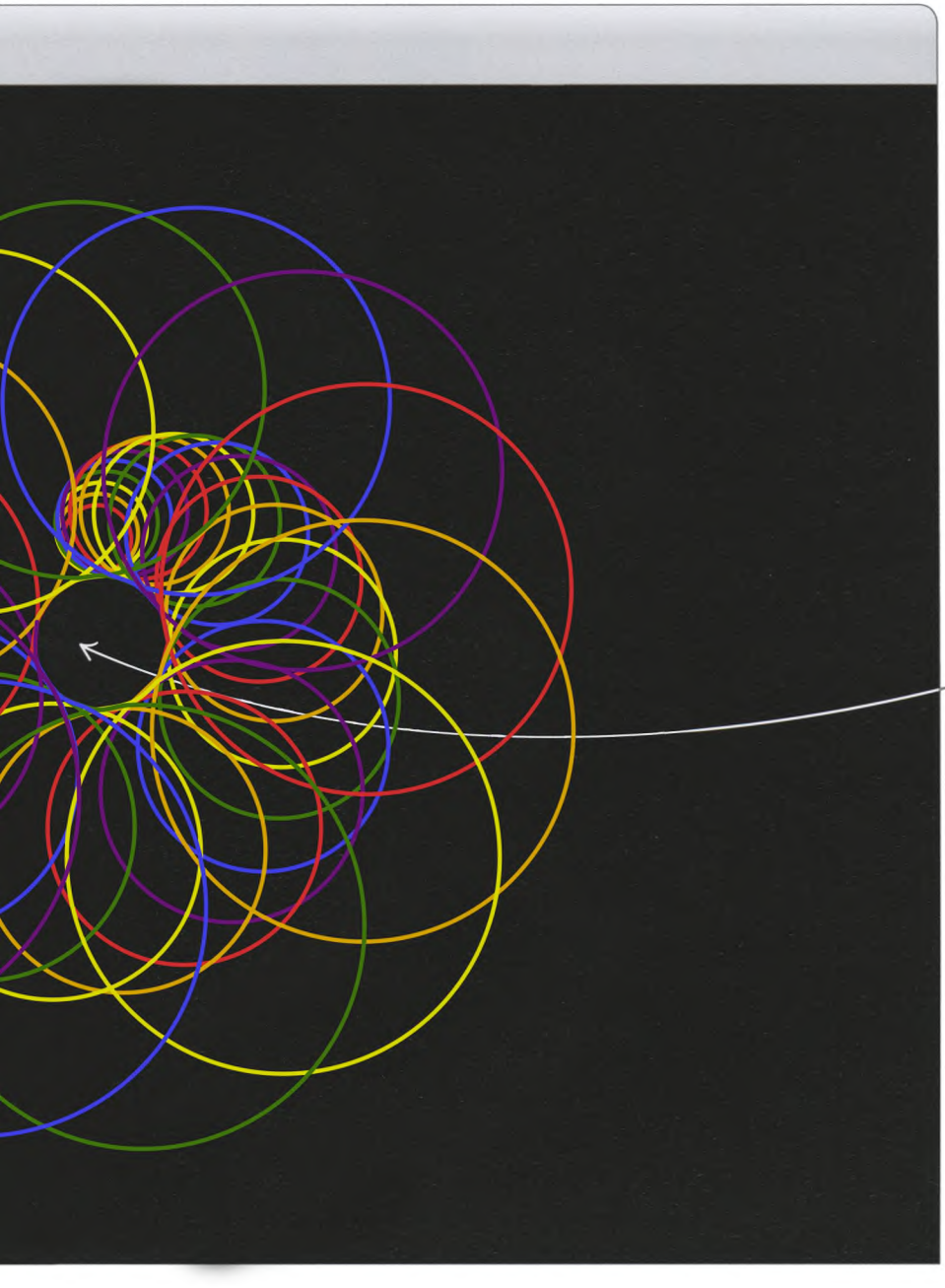
△ **Растущая спираль**
Окружности будут накладываться одна на другую со сдвигом, образуя выходящую из центра спираль.

Каждая окружность отличается от предыдущей цветом и размером.

Программа скрывает черепашку, чтобы она не заслоняла рисунок.







Черепашка начинает рисовать окружности из центра экрана.

◁ **Гибкая программа**

Чем дольше работает программа, тем сложнее получается узор на экране. Меняя аргументы функции, рисующей окружности, ты сможешь создать еще более психоделические картины.

Как это работает

В этом проекте используется модуль **turtle** и хитрая техника наложения окружностей друг на друга по спирали. При отображении каждой окружности программа немного меняет аргументы функции, чтобы следующая окружность отличалась от предыдущей и узор получался интереснее.

▽ **Блок-схема программы «Радуга-пружинка»**
Программа задает параметры, которые не должны меняться (вроде скорости черепашки), а потом входит в цикл. Код цикла выбирает цвет, рисует окружность, поворачивает и смещает черепашку, а затем повторяется снова, завершаясь лишь вместе с программой.

СОВЕТЫ ЭКСПЕРТА

Перебор цветов

Функция **cycle()** («окружность») из модуля **itertools** (*сокр. от* iterator tools — «инструменты итераторов») делает узоры разноцветными. Она перебирает элементы в списке цветов по кругу, снова и снова.

Давай рисовать!

Сначала нарисуем обычную окружность, а затем будем повторять это действие с небольшими изменениями. Позднее код можно будет доработать, чтобы сделать узор интереснее.

1 **Создай новый файл**
Открой IDLE, создай новый файл и сохрани его как rainbow-spiral.py.

2 **Загрузи модуль turtle**
Загрузи **turtle** — главный из необходимых тебе модулей. Добавь эту строчку кода в начало программы.

```
import turtle
```

Загружает модуль turtle.

3 Настрой черепашку

Этот код вызывает функции модуля **turtle**, которые задают цвет фона, скорость черепашки и толщину пера.

```
import turtle

turtle.bgcolor('black')
turtle.speed('fast')
turtle.pensize(4)
```

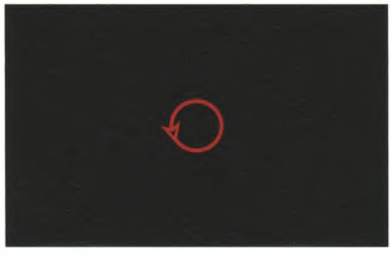
Цвет фона.

Скорость черепашки.

Толщина пера.

4 Выбери цвет, нарисуй окружность

Задай цвет окружности и проверь работу кода. Добавь эти две строчки в конец программы и запусти ее.



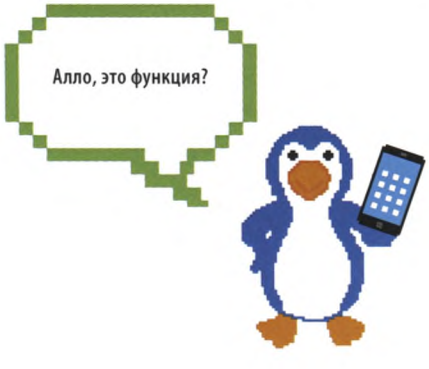
```
import turtle

turtle.bgcolor('black')
turtle.speed('fast')
turtle.pensize(4)

turtle.pencolor('red')
turtle.circle(30)
```

Цвет окружности.

Команда для рисования окружности.

5 Больше окружностей

На экране появится окружность. Но нам мало одной, поэтому поместим команды для рисования окружности внутрь функции, добавив еще одну строчку, чтобы функция сама себя вызывала. Этот трюк с бесконечным самозапуском называется рекурсией. Поскольку функцию нужно создать до ее вызова, помести этот код перед основным кодом программы.




```
import turtle

def draw_circle(size):
    turtle.pencolor('red')
    turtle.circle(size)
    draw_circle(size)

turtle.bgcolor('black')
turtle.speed('fast')
turtle.pensize(4)
draw_circle(30)
```

Здесь используется аргумент size («размер»).

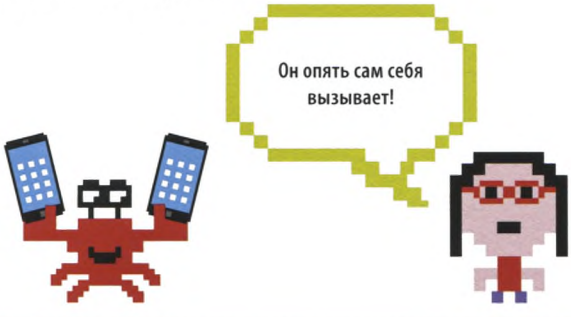
Функция draw_circle() («нарисовать окружность») сама себя вызывает бесконечное количество раз.

Первый вызов функции.

СОВЕТЫ ЭКСПЕРТА

Рекурсия

Ситуация, при которой функция вызывает саму себя, называется рекурсией. Это еще один способ зациклить код. Обычно при рекурсии значения аргументов все время меняются. В программе «Радуга-пружинка» при каждом вызове функции **draw_circle()** разным становится положение окружности.



6 Проверь код

Запусти программу. Ты увидишь, как черепашка упорно рисует одинаковые окружности. Сейчас мы это исправим.

7 Измени цвет и увеличь размер

Чтобы получить интересный узор, внеси в код изменения. Теперь цвет окружности будет меняться, а размер увеличиваться. В этом коде функция **cycle()** («окружность») принимает список значений и возвращает цикличный список, который можно бесконечно перебирать функцией **next()** («следующий»).

```
import turtle
from itertools import cycle

colors = cycle(['red', 'orange', 'yellow', 'green', 'blue', 'purple'])

def draw_circle(size):
    turtle.pencolor(next(colors))
    turtle.circle(size)
    draw_circle(size + 5)

turtle.bgcolor('black')
turtle.speed('fast')
turtle.pensize(4)
draw_circle(30)
```

Загружает функцию cycle().

Создает цикличный список цветов.

Берет следующий цвет из списка.

Добавляет 5 пикселей к размеру предыдущей окружности.

8 **Улучши узор**

Запусти программу. Теперь окружности разные, но узор получается пока еще не очень интересным. Добавим ему затейливости, заставив изменяться угол поворота черепашки и местоположение каждой окружности. Внеси в код правку, выделенную черным, запусти программу и посмотри, что будет.

Не забудь сохранить свою работу.

```
def draw_circle(size, angle, shift):
    turtle.pencolor(next(colors))
    turtle.circle(size)
    turtle.right(angle)
    turtle.forward(shift)
    draw_circle(size + 5, angle + 1, shift + 1)

turtle.bgcolor('black')
turtle.speed('fast')
turtle.pensize(4)
draw_circle(30, 0, 1)
```

Добавь новые аргументы.

Поворачивает черепашку по часовой стрелке.

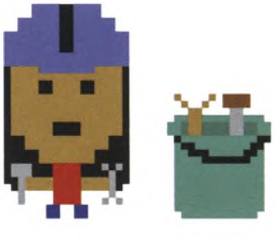
Продвигает черепашку вперед.

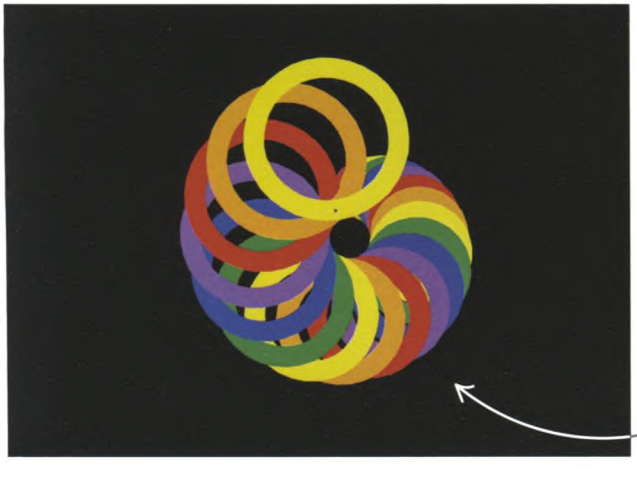
Величина угла и смещения черепашки для каждой новой окружности увеличиваются.

Присвой начальные значения новым аргументам.

Что бы изменить?

Итак, программа работает. А это значит, что с ее кодом можно поэкспериментировать и создать еще более фантастические узоры.





◁ Широкие линии

Увеличь толщину линий и посмотри, каким станет узор. Первоначальная толщина равнялась 4 пикселям. А что если указать 40?

```
turtle.pensize(40)
```

Теперь окружности стали шире.


```
def draw_circle(size, angle, shift):
    turtle.bgcolor(next(colors))
    turtle.pencolor(next(colors))
    turtle.circle(size)
    turtle.right(angle)
    turtle.forward(shift)
    draw_circle(size + 5, angle + 1, shift + 1)

turtle.speed('fast')
turtle.pensize(4)
draw_circle(30, 0, 1)
```

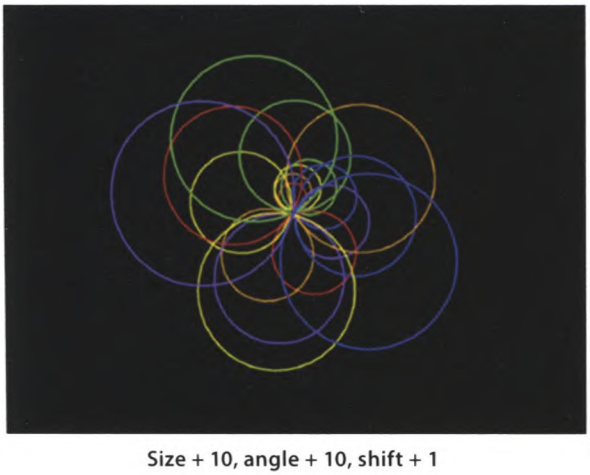
Теперь цвет фона задается в цикле.

◁ **Безумные цвета**

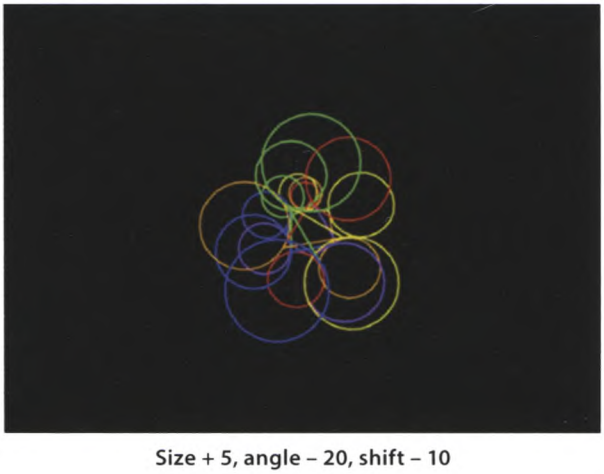
А что если при каждом повторе цикла менять не только цвет окружностей, но и цвет фона? Может выйти кое-что занятное! Для этого перенеси команду **turtle.bgcolor()** (*сокр. от* background color — «цвет фона»), задающую цвет фона, в функцию **draw_circle()**. А чтобы цвета менялись постоянно, используй цикличный список.

▽ **Придумай новые узоры**

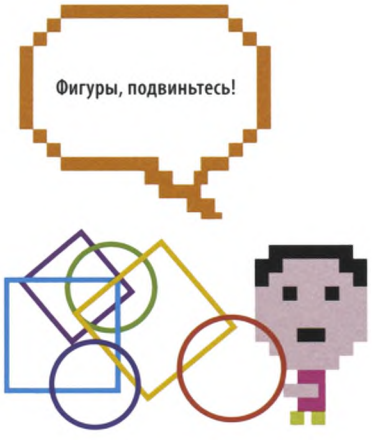
Вид узора зависит от чисел, которые прибавляются к аргументам функции, таким как **size** («размер»), **shift** («смещение») и **angle** («угол»), при каждом ее вызове. Попробуй увеличить или уменьшить эти числа. Как это повлияет на узор?



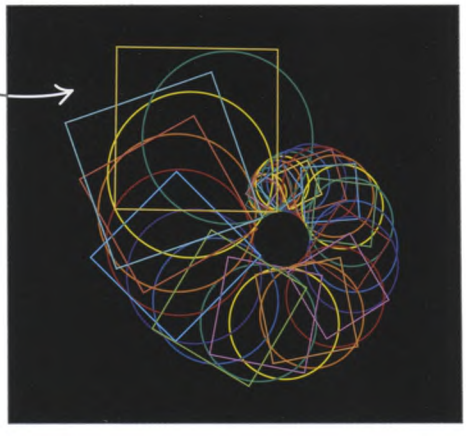
Size + 10, angle + 10, shift + 1



Size + 5, angle – 20, shift – 10



Можно добавлять разные фигуры.



▽ Добавь другие фигуры

Как поменяется узор, если кроме окружностей программа будет рисовать и другие фигуры? Скажем, через раз выдавать квадрат. Попробуй ввести этот код, но будь внимателен: название функции изменилось!

```
import turtle
from itertools import cycle

colors = cycle(['red', 'orange', 'yellow', 'green', 'blue', 'purple'])

def draw_shape(size, angle, shift, shape):
    turtle.pencolor(next(colors))
    next_shape = ''
    if shape == 'circle':
        turtle.circle(size)
        next_shape = 'square'
    elif shape == 'square':
        for i in range(4):
            turtle.forward(size * 2)
            turtle.left(90)
        next_shape = 'circle'
    turtle.right(angle)
    turtle.forward(shift)
    draw_shape(size + 5, angle + 1, shift + 1, next_shape)

turtle.bgcolor('black')
turtle.speed('fast')
turtle.pensize(4)
draw_shape(30, 0, 1, 'circle')
```

Добавь новый аргумент shape («фигура»).

Цикл запускается четырежды — по одному разу для каждой стороны квадрата.

Поворачивает черепашку.

Продвигает черепашку вперед.

Чередует окружности и квадраты.

Первая фигура — окружность.

«Звездное небо»

Заполни экран яркими звездами! Для их создания тебе понадобится модуль turtle. Положение каждой звезды, ее цвет, размер и форма будут выбираться случайным образом.

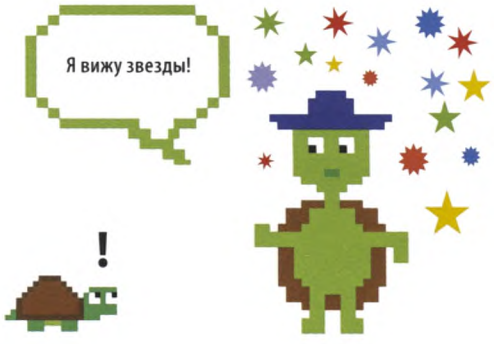
Что происходит

После запуска на экране появится ночное небо, в котором загорится одинокая звезда. По мере работы программы будут возникать все новые звезды разных форм и цветов. Чем дальше, тем более причудливым и красочным станет небо.

СОВЕТЫ ЭКСПЕРТА

Цвета на экране

Экранное изображение состоит из крохотных точек — пикселей, светящихся красным, зеленым и синим. В этом проекте цвет каждой звезды определяется тремя числами, которые обозначают насыщенность красного, зеленого и синего цветов, дающих при наложении нужный оттенок.



Можешь выбрать любой цвет фона на свой вкус, но лучше всего звезды смотрятся на темном фоне, таком как синий.

Черепашка (желтая стрелочка) рисует звезду. Когда звезда будет готова, Python зальет ее цветом.

Звезды появляются в случайных местах.

Цвет каждой звезды задается с помощью трех случайных чисел.

Меняя код, можно варьировать размеры звезд и количество их лучей.

◁ **Полный экран звезд**
Программа «Звездное небо» рисует звезды по одной, но, поскольку в ней используется бесконечный цикл **while**, она будет делать это бесконечно! Также в коде можно менять диапазон размеров звезд.

Как это работает

Код этого проекта рисует звезды в случайных точках графического окна. Сначала напишем функцию для создания одной звезды, а затем добавим бесконечный цикл, чтобы разбросать по экрану другие звезды.

▽ **Блок-схема программы «Звездное небо»**
Блок-схема программы проста и не содержит каких-либо условий и ветвлений. После того как черепашка изобразит первую звезду, цикл повторится, и звезды будут возникать без остановки, пока ты не завершишь программу.

◁ **Считая звезды**
Безоблачной ночью на небе можно увидеть около 2500 звезд. Чтобы программа нарисовала столько же, придется подождать больше полутора часов!

Рисуем звезду

Прежде чем писать код функции, нужно разобраться, как нарисовать одну звезду с помощью черепашки, и тогда ты сможешь создать множество звезд.

1 Создай новый файл
Открой IDLE. В меню File выбери New File. Сохрани файл как starry_night.py («звездное небо»).

2 Загрузи turtle
Введи эту строку кода в окне программы. Она загружает модуль **turtle**, который и будет рисовать звезды.

```
import turtle as t
```

Загружает модуль turtle.

3 Задай параметры звезды

После строки с загрузкой модуля добавь этот код. Он определяет размер и форму звезды, а также объясняет черепашке, как нужно двигаться.

```
import turtle as t

size = 300
points = 5
angle = 144

for i in range(points):
    t.forward(size)
    t.right(angle)
```

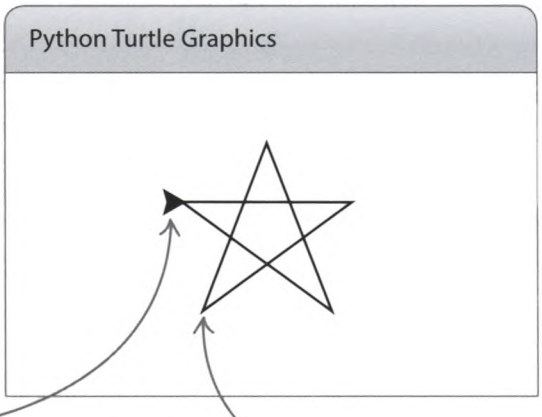
Эти переменные задают размер и форму звезды.

Величина углов, образованных двумя лучами, выходящими из каждой вершины звезды, записывается в градусах.

Цикл for нужен для того, чтобы черепашка повторяла одни и те же движения для каждого луча звезды.

4 Нарисуй звезду

Чтобы проверить работу программы, открой меню Run и выбери Run Module. Появится графическое окно (оно может оказаться позади других окон), на котором черепашка будет рисовать первую звезду.



Не забудь сохранить свою работу.

Черепашка-стрелочка двигается, оставляя за собой линию.

Звезда создается поэтапно, отрезок за отрезком.

5 Вычисли угол

Хорошо бы научить черепашку рисовать звезды с разным количеством лучей. Измени эту строчку кода. Пусть она вычисляет угол, на который нужно развернуть черепашку, чтобы получилась звезда с заданным количеством лучей.

```
import turtle as t

size = 300
points = 5
angle = 180 - (180 / points)

for i in range(points):
    t.forward(size)
    t.right(angle)
```

Величина угла зависит от количества лучей (вершин) звезды.

6 Раскрась звезду!

Мы нарисовали отличную звезду, но сейчас она выглядит скучновато. Давай добавим ей цвет. Внеси в код следующие изменения, чтобы звезда стала желтой.

```
import turtle as t

size = 300
points = 5
angle = 180 - (180 / points)

t.color('yellow')
t.begin_fill()
for i in range(points):
    t.forward(size)
    t.right(angle)

t.end_fill()
```

Задает желтый цвет для звезды.

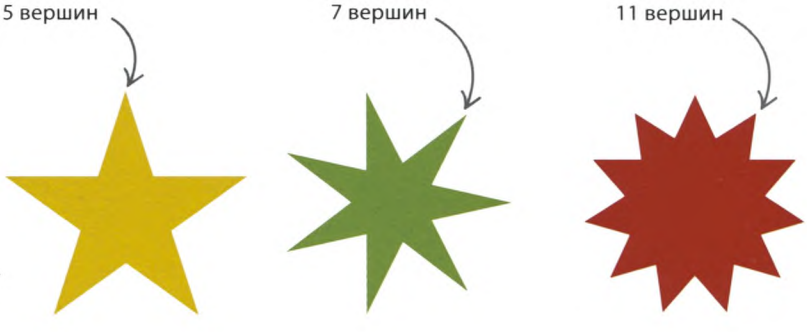
Заливает звезду цветом.

7 Запусти программу

Черепашка должна нарисовать желтую звезду. Попробуй изменить в коде ее цвет.

8 Нарисуй разные звезды

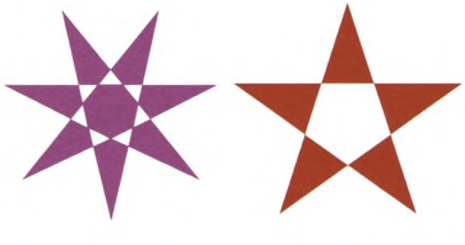
Попробуй поменять значение переменной **points** («вершины»), и ты увидишь, что программа может рисовать самые разные звезды. Однако обрати внимание: код работает лишь для звезд с нечетным количеством лучей.

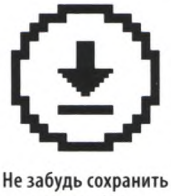


СОВЕТЫ ЭКСПЕРТА

Дырявые звезды

На разных компьютерах звезда может выглядеть по-разному. Если в середине ты увидишь дырку, не волнуйся: вид звезды зависит от типа компьютера, так что твой код в этом не виноват.

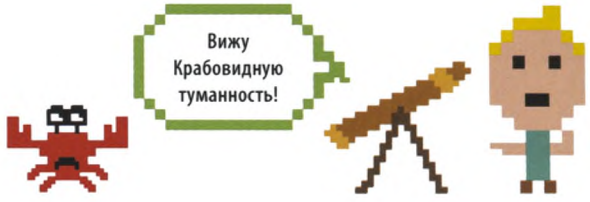




Не забудь сохранить свою работу.

Звезды в небе

А теперь давай поместим код, рисующий звезду, в функцию, с помощью которой можно будет усыпать звездами все небо.



Для создания звезды функция draw_star() использует 5 параметров: количество вершин, размер, цвет и координаты фигуры.

```
import turtle as t

def draw_star(points, size, col, x, y):
    t.penup()
    t.goto(x, y)
    t.pendown()
    angle = 180 - (180 / points)
    t.color(col)
    t.begin_fill()
    for i in range(points):
        t.forward(size)
        t.right(angle)
    t.end_fill()

# Основной код
t.Screen().bgcolor('dark blue')
draw_star(5, 50, 'yellow', 0, 0)
```

Координаты *x* и *y* задают положение звезды на экране.

Комментарий начинается со знака «решетка» (#), поэтому Python его пропускает. Это просто заметка, которая поясняет код.

Задает темно-синий цвет фона.

Вызов функции draw_star().

Черепашка рисует в центре окна желтую пятиконечную звезду размером 50 пикселей.

9 Создай функцию draw_star()

Измени код так, как показано справа, чтобы команды для рисования звезды попали внутрь функции. Теперь, чтобы изобразить звезду, тебе будет достаточно просто вызвать в основном коде функцию **draw_star()** («нарисовать звезду»).

10 Запусти программу

Черепашка должна нарисовать желтую звезду на темно-синем фоне.

ЗАПОМНИ

Комментарии

Программисты часто оставляют в коде комментарии, напоминающие, что делают те или иные части программы. Комментарий должен начинаться со знака «решетка» (#). Все, что идет после него, Python не считает командой. Добавляй в свой код комментарии (такие, как # Основной код), и дорабатывать программы после долгого перерыва станет гораздо проще.

11 Случайные числа

Давай разнообразим программу, добавив в нее случайные числа. Введи эту строчку после импорта модуля **turtle**. Она загрузит функции **randint()** (*сокр. от* random integer — «случайное целое число») и **random()** из модуля **random**.

```
import turtle as t
from random import randint, random

def draw_star(points, size, col, x, y):
```

12 Создай цикл

Внеси следующие изменения в основной код программы. Цикл **while** будет раз за разом обновлять случайные числа, задающие размер звезды, ее форму, цвет и положение на экране.

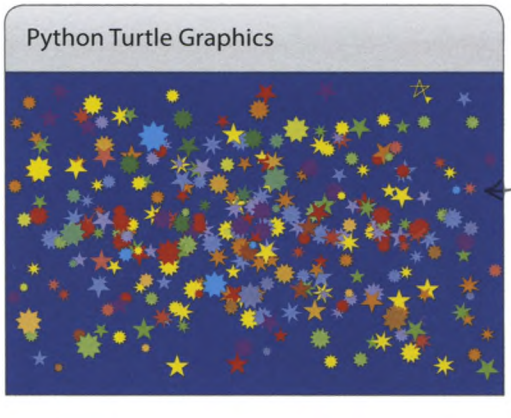
```
# Основной код
t.Screen().bgcolor('dark blue')

while True:
    ranPts = randint(2, 5) * 2 + 1
    ranSize = randint(10, 50)
    ranCol = (random(), random(), random())
    ranX = randint(-350, 300)
    ranY = randint(-250, 250)

    draw_star(ranPts, ranSize, ranCol, ranX, ranY)
```

Строчка с переменной ranPts (*сокр. от* random points — «случайное количество вершин») задает диапазон для количества вершин звезды; сейчас это нечетные числа от 5 до 11.

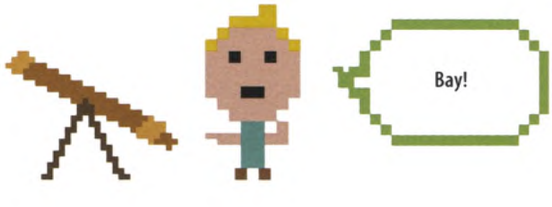
Эта строчка тоже изменилась. Теперь при вызове функции draw_star() используются случайные значения аргументов.

13 И вновь запусти программу

Черепашка начнет медленно заполнять окно звездами разных цветов, форм и размеров.



Черепашка рисует звезды со случайными параметрами в случайных местах.



ЗАПОМНИ

Невидимая черепашка

Если тебе не хочется смотреть на черепашку, запомни команду, которая делает ее невидимой. Добавь в код эту строчку, и звезды будут появляться на экране как по волшебству!

```
# Основной код
t.hideturtle()
```

Что бы изменить?

Теперь ты умеешь создавать звезды и можешь применить новые знания в собственных проектах. Вот тебе еще несколько идей на эту тему.

△ **Разнообразие форм**

Меняя числа в скобках функции **randint()**, которая присваивает значения переменным **ranPts** и **ranSize** («случайный размер»), ты можешь делать звезды более или менее непохожими друг на друга.

▷ **Звезды и мышка**

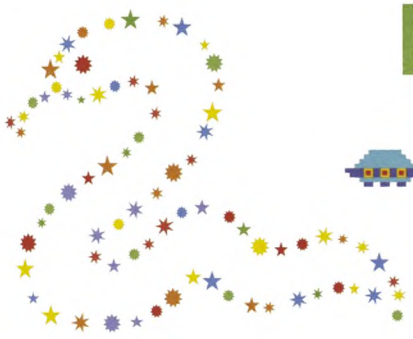
Попробуй с помощью функции **t.onScreenClick()** («кликнуть по экрану») сделать так, чтобы звезды возникали по щелчку мышки.

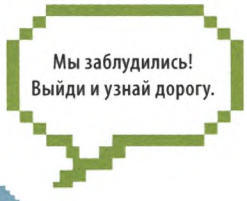
▽ **Ускорь черепашку**

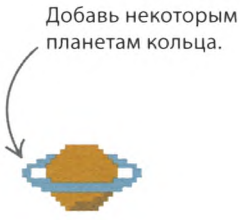
Функция **speed()** позволяет менять скорость, с которой черепашка рисует звезды. Добавь в начало основного кода строчку t.speed(0), чтобы скорость увеличилась. Все параметры черепашки можно узнать в меню Help.

▽ **Создай созвездие**

Созвездие — это скопление звезд на ночном небе. Создай список с координатами звезд (*x*, *y*) в придуманном тобой созвездии, а затем нарисуй их, перебирая элементы списка в цикле **for**.







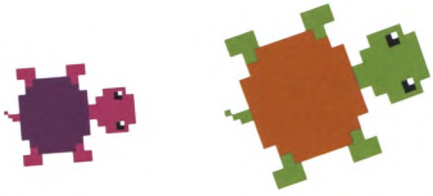

```
def draw_planet(col, x, y):
    t.penup()
    t.goto(x, y)
    t.pendown()
    t.color(col)
    t.begin_fill()
    t.circle(50)
    t.end_fill()
```

▷ **Нарисуй планеты**

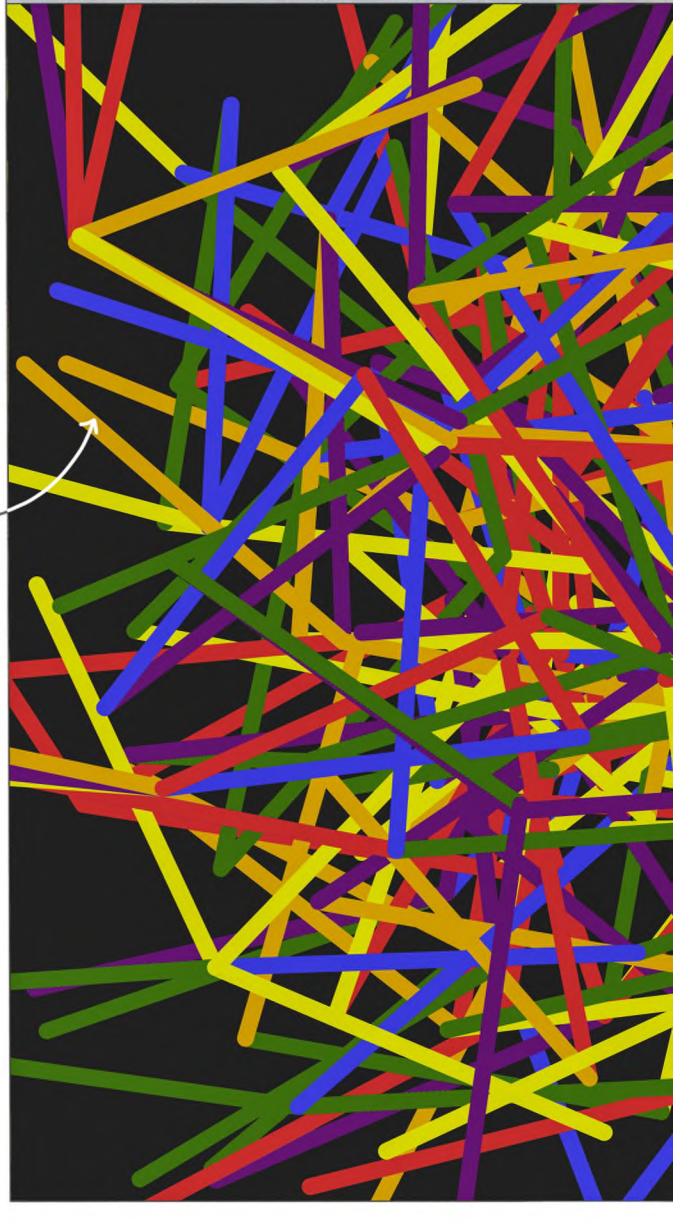
Изучи работу функции **t.circle()** и попробуй нарисовать с ее помощью планеты. Начать можно с этого кода.

«Безумная радуга»

С помощью черепашки можно рисовать самые разные картинки и узоры. Похоже, в этом проекте черепашка разошлась: такой радуги на небе не увидишь!



Python Turtle Graphics



Что происходит

Программа просит пользователя выбрать длину и толщину линий, которые будет рисовать черепашка. Затем черепашка начинает сновать по экрану, оставляя за собой линии разных цветов. Узор, который при этом получится, будет зависеть от выбранных значений.

У черепашки есть перо, которым она рисует линии, перемещаясь по экрану.

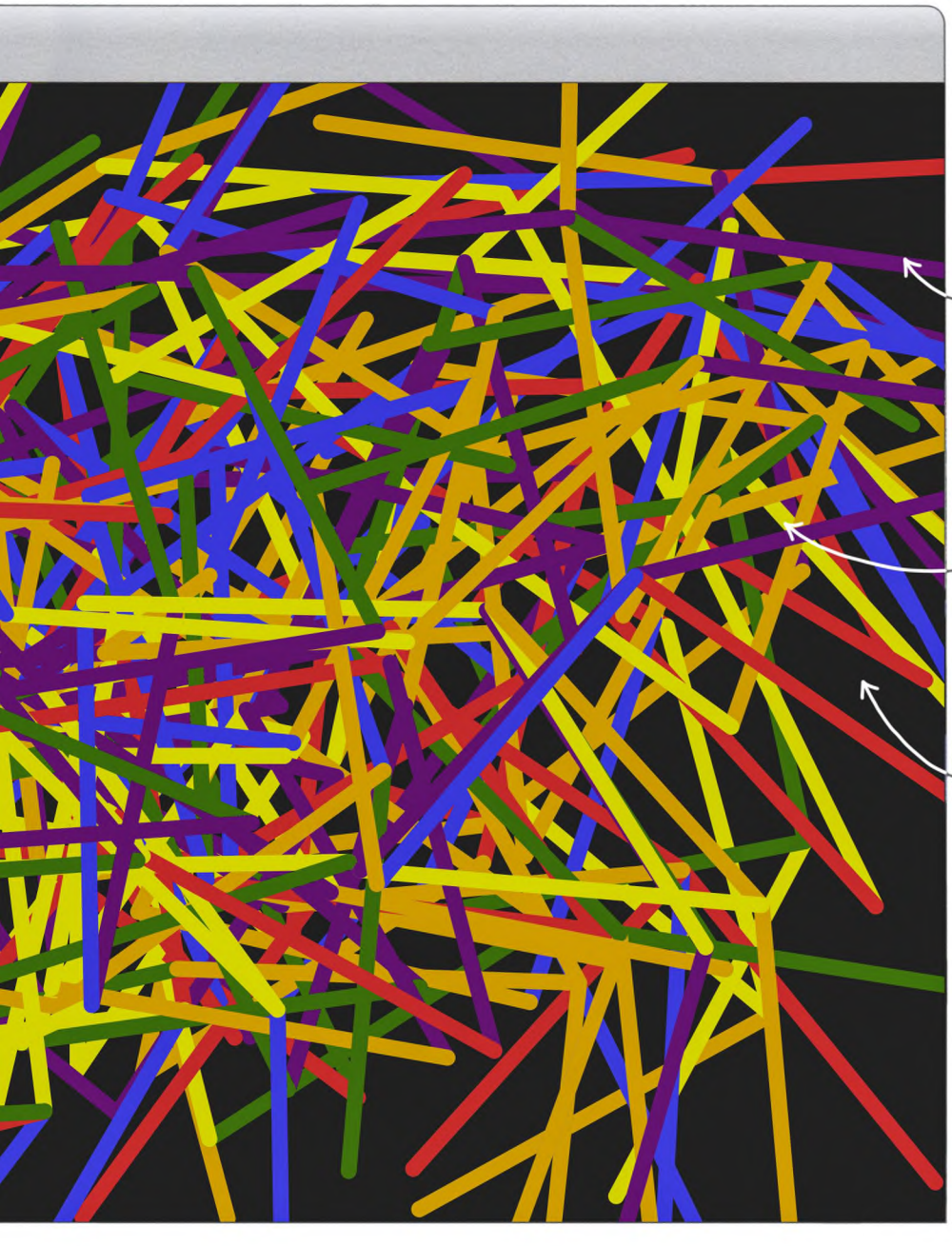
СОВЕТЫ ЭКСПЕРТА

Какой цвет следующий?

В программе «Безумная радуга» функция **choice()** из модуля **random** выбирает случайные цвета линий. Это значит, что предсказать цвет очередной линии невозможно.

```
t.pencolor(random.choice(pen_colors))
```

Черепашка выберет один из шести цветов, хранящихся в списке pen_colors («цвета пера»).



Черепашка рисует зеленые, красные, оранжевые, желтые, синие и фиолетовые линии.

Путь черепашки может отклоняться на угол от 0 до 180 градусов.

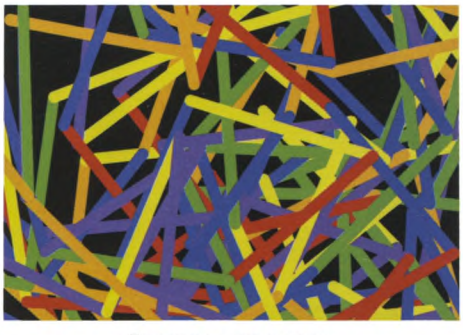
Функция get_line_length() позволяет пользователю задавать длинную, среднюю и короткую линии.

◁ **Буйство красок**

Поскольку в программе используется бесконечный цикл **while**, черепашка будет рисовать линии до тех пор, пока ты не закроешь окно. Кроме цвета, толщины и длины линий, можно поменять цвет, форму и скорость самой черепашки.

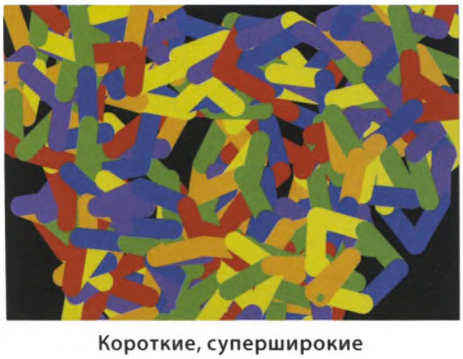
Как это работает

Каждый узор, который создает программа, не похож на предыдущий, поскольку перед началом рисования новой линии черепашка разворачивается в случайном направлении. Цвет линии также выбирается случайно из списка цветов. Результат предугадать невозможно!



Длинные, широкие

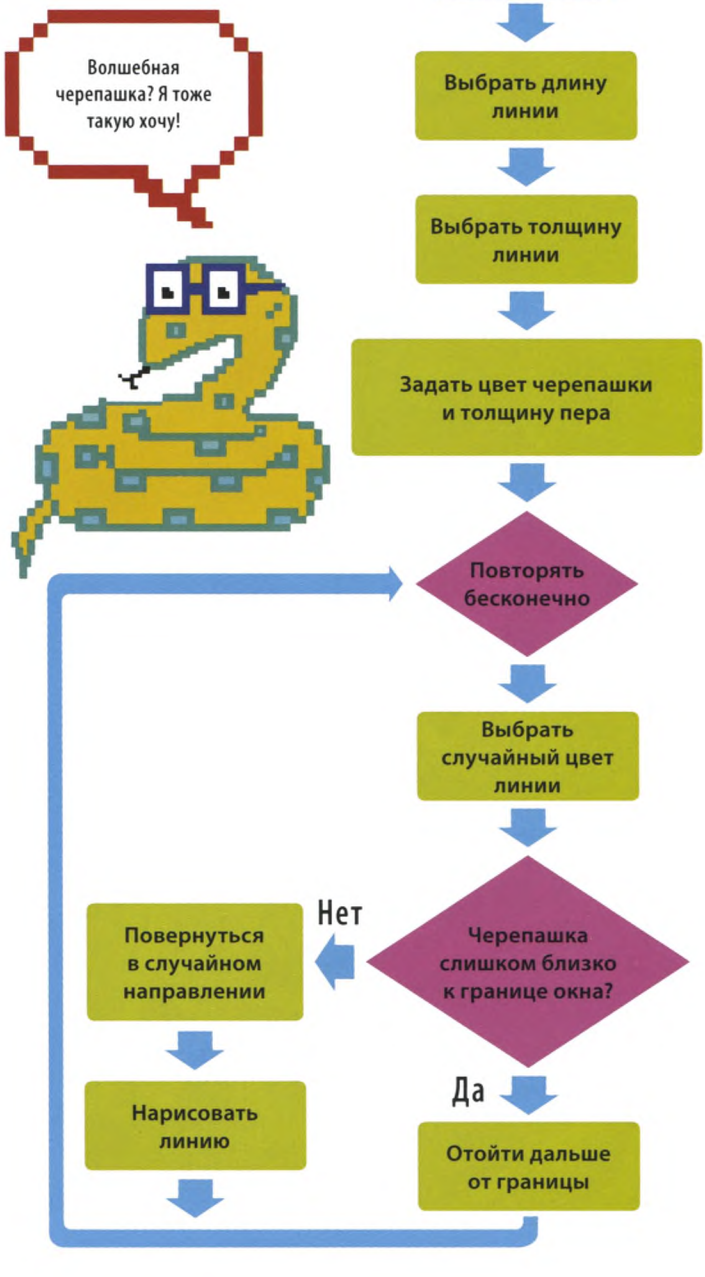
Средние, тонкие



Короткие, суперширокие

▽ **Блок-схема программы «Безумная радуга»**

В коде используется бесконечный цикл, рисующий линии все время, пока программа работает. Лишь когда ты закроешь окно, черепашка прекратит перемещаться по экрану.



◁ Черепашка-беглец
Черепашка, которой дают полную свободу, склонна убегать за границы окна. Чтобы она не уходила слишком далеко, добавь код, проверяющий позицию черепашки. Иначе проект «Безумная радуга» превратится в проект «Сбежавшая черепашка»!

Начнем

Работу над проектом начнем с создания нового файла, загрузки модулей и написания функций, которые запрашивают значения у пользователя.

1 Создай новый файл
Открой IDLE и создай новый файл. Сохрани его как rainbow.py («радуга»).

2 Добавь модули
Добавь в начало программы 2 строчки кода, импортирующие модули **turtle** и **random**. Команда import turtle as t нужна для того, чтобы при вызове функций из этого модуля указывать «t» вместо полного «turtle».

```
import random
import turtle as t
```

3 Запроси длину
Создай функцию **get_line_length()** («получить длину линии»), позволяющую пользователю выбрать, какие линии рисовать черепашке: длинные, средние или короткие. Эта функция понадобится лишь на шаге 5, однако лучше подготовить ее заранее. Введи этот код после команд, добавленных на шаге 2.

```
import turtle as t

def get_line_length():
    choice = input('Выберите длину линий (длинные, средние, короткие): ')
    if choice == 'длинные':
        line_length = 250
    elif choice == 'средние':
        line_length = 200
    else:
        line_length = 100
    return line_length
```

Предлагает пользователю выбрать длину линий.

Возвращает значение переменной line_length обратно в код.

Для коротких линий задает длину 100 пикселей.

4 Запроси толщину

Создай функцию **get_line_width()** («получить толщину линии»), предлагающую пользователю выбрать, какие линии рисовать черепашке: суперширокие, широкие или тонкие. Эта функция (как и функция **get_line_length()**) понадобится на шаге 5. Введи следующие строчки после кода, добавленного на шаге 3.

```
    return line_length


def get_line_width():
    choice = input('Выберите толщину линий (суперширокие, широкие, тонкие): ')
    if choice == 'суперширокие':
        line_width = 40
    elif choice == 'широкие':
        line_width = 25
    else:
        line_width = 10
    return line_width
```

Предлагает пользователю выбрать толщину линий.

Если пользователь выбрал тонкие линии, присваивает переменной значение 10.

Возвращает значение переменной line_width обратно в код.

5 Используй функции

Итак, у тебя есть две функции, и теперь их можно использовать для запроса длины и толщины линий. Добавь эти строчки кода в конец программы и сохрани файл.

```
    return line_width


line_length = get_line_length()
line_width = get_line_width()
```

6 Проверь работу программы

Запусти программу в окне консоли и проверь, правильно ли работают функции.

Ответ пользователя.

```
Выберите длину линий (длинные, средние, короткие): длинные
Выберите толщину линий (суперширокие, широкие, тонкие): тонкие
```

Позовем черепашку

Пришло время написать код, который создаст графическое окно и подготовит черепашку к работе.

7 Создай окно

Введи следующие строчки после кода, добавленного на шаге 5. В них задается цвет фона, а также параметры черепашки: ее форма, цвет, скорость и толщина пера, которым она будет рисовать линии.

```
line_width = get_line_width()

t.shape('turtle')
t.fillcolor('green')
t.bgcolor('black')
t.speed('fastest')
t.pensize(line_width)
```

Делает черепашку похожей на черепаху, а не на стрелочку.

Красит черепашку в зеленый цвет.

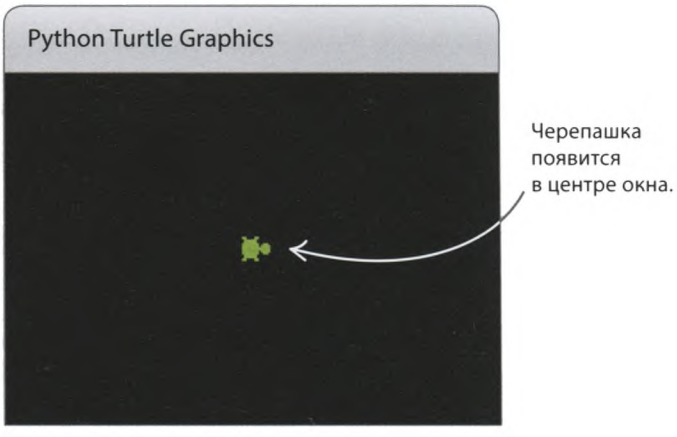
Задает черный цвет фона.

Задает скорость черепашки.

Устанавливает толщину линий в зависимости от выбора пользователя.

8 Запусти программу

Если снова запустить программу, после запроса параметров линий появится графическое окно, а в нем — черепашка. Рассмотри ее хорошенько: недолго ей стоять на месте!



9 Границы перемещений

Чтобы черепашка не убегала, обозначим границы, отступив по 100 пикселей от каждого края окна. Проверять, не выходит ли черепашка за границы, будем с помощью функции. Введи эти строчки после кода, добавленного на шаге 4, и перед кодом шага 5.

```
    return line_width

def inside_window():
    left_limit = (-t.window_width() / 2) + 100
    right_limit = (t.window_width() / 2) - 100
    top_limit = (t.window_height() / 2) - 100
    bottom_limit = (-t.window_height() / 2) + 100
    (x, y) = t.pos()
    inside = left_limit < x < right_limit and bottom_limit < y < top_limit
    return inside

line_length = get_line_length()
```

Отступим 100 пикселей вправо от левого края окна.

100 пикселей влево от правого края окна.

100 пикселей вниз от верхнего края окна.

100 пикселей вверх от нижнего края окна.

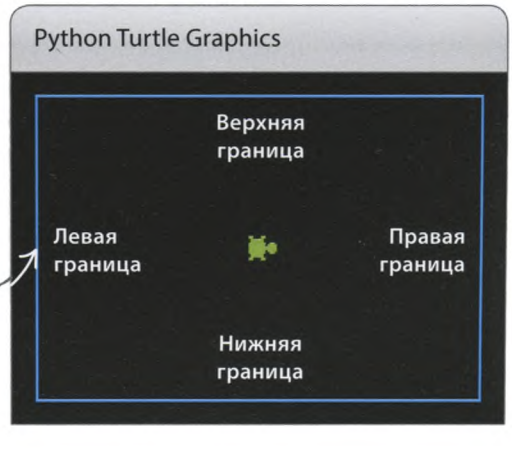
Если черепашка находится внутри ограниченной области, переменная inside принимает значение True, иначе — False.

Получает текущие координаты черепашки (x, y).

Возвращает значение переменной inside обратно в код.

▷ Как это работает

Код проверяет, находится ли *x*-координата черепашки между правой и левой границами окна, а *y*-координата — между верхней и нижней.

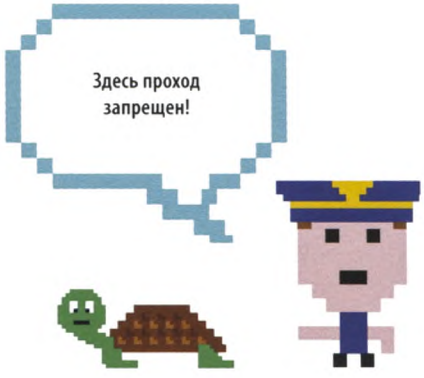


Голубой прямоугольник показывает границы; на экране он виден не будет.

Не забудь сохранить свою работу.

Черепашка, за работу!

Давай напишем функции, перемещающие черепашку, а затем создадим цикл **while**, который отправит ее рисовать безумную радугу!



10 Безумная линия

Добавь следующие строчки после кода, введенного на шаге 9, и перед кодом шага 5. Эта функция поворачивает черепашку в другую сторону. При этом за ней тянется линия случайно выбранного цвета. Программа будет вызывать эту функцию снова и снова, чтобы получилась безумная радуга. Если черепашка выйдет за границы, заданные на шаге 9, функция развернет ее назад.

Варианты цветов линий хранятся в списке.

Поставь обратный слеш, чтобы разбить длинную строку кода надвое.

```
    return inside


def move_turtle(line_length):
    pen_colors = ['red', 'orange', 'yellow', 'green', \
                  'blue', 'purple']
    t.pencolor(random.choice(pen_colors))
    if inside_window():
        angle = random.randint(0, 180)
        t.right(angle)
        t.forward(line_length)
    else:
        t.backward(line_length)


line_length = get_line_length()
```

Выбирает случайный цвет линии из списка.

Выбирает случайный угол поворота от 0 до 180 градусов.

Черепашка движется вперед на line_length пикселей.

▷ Как это работает

Этот код вызывает функцию **inside_window()** («внутри окна»), которая проверяет, не вышла ли черепашка за границы окна. Если всё в порядке, черепашка поворачивает направо на случайный угол от 0 (нет поворота) до 180 градусов (обратное направление) и движется вперед. Если же она выходит за границы, то возвращается обратно.

Проверяет, находится ли черепашка внутри ограниченной области.

Отклоняет черепашку вправо на случайный угол.

Если черепашка вышла за границы окна, она возвращается назад.

11 Черепашка, вперед!

И наконец, введи код, запускающий процесс рисования. Добавь эти строчки в самый конец программы, после команд, введенных на шаге 7. Сохрани программу, запусти ее и полюбуйся на свою первую безумную радугу!

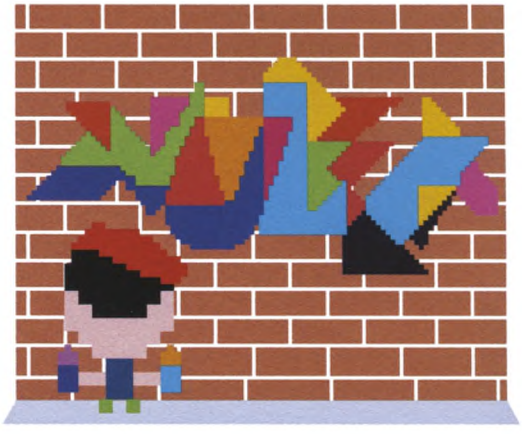
Запускает бесконечный цикл, чтобы черепашка рисовала линии без остановки.

```
t.speed('fastest')
t.pensize(line_width)

while True:
    move_turtle(line_length)
```

Рисует одну линию.

Что бы изменить?

Достаточно ли безумная получилась у тебя радуга? Не очень? Вот несколько способов сделать ее еще более фантастической!



▽ Неожиданные цвета

Цвет в Python можно задавать RGB-значением, то есть комбинацией красного (R), зеленого (G) и синего (B) цветов. Если выбрать насыщенность этих трех составляющих случайным образом, получится совершенно неожиданный результат. Попробуй изменить код функции **move_turtle()** («переместить черепашку») так, чтобы вместо названий цветов там использовались случайные RGB-значения. Запусти код и посмотри, что выйдет!

Замени эти 2 строчки кода...

```
pen_colors = ['red', 'orange', 'yellow', 'green', 'blue', 'purple']
t.pencolor(random.choice(pen_colors))
```

...на эти 5.

```
t.colormode(255)
red = random.randint(0, 255)
blue = random.randint(0, 255)
green = random.randint(0, 255)
t.pencolor(red, green, blue)
```

▽ Разные линии

Не стоит ограничиваться линиями одной толщины — этот трюк позволит изобразить еще более сумасшедшую радугу! Толщина линий будет меняться случайным образом. Добавь эту строчку в функцию **move_turtle()** после вызова **t.pencolor()** («цвет пера»).

```
t.pensize(random.randint(1,40))
```

СОВЕТЫ ЭКСПЕРТА

RGB-цвет

Синему цвету соответствует RGB-значение (0, 0, 255), поскольку в нем максимальное количество синего (B) и совсем нет красного (R) и зеленого (G). Чтобы использовать RGB-значения для пера черепашки, включи RGB-режим командой t.colormode(255) («цветовой режим»), иначе Python выдаст ошибку.

Это значение показывает насыщенность красной составляющей цвета (от 0 до 255).

```
t.pencolor(0, 0, 255)
```

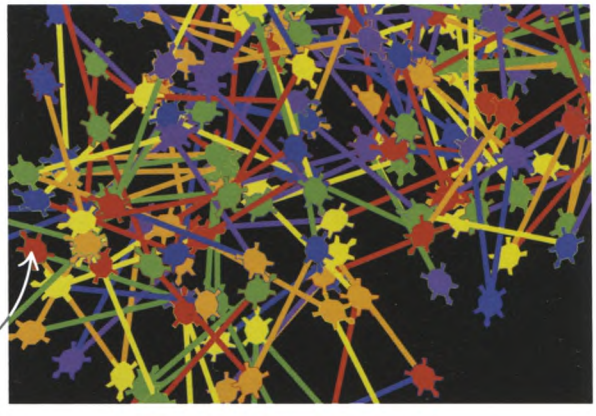
Насыщенность зеленой составляющей.

Насыщенность синей составляющей.

```
t.pencolor('blue')
```

▽ **Черепаховые штампы**

Скрепи линии своей радуги отпечатками в виде черепашки. Сделай это с помощью функции **stamp()** («штамповать») модуля **turtle**, которая добавит изображение черепашки в начало каждой линии (а еще можешь написать функцию, которая рисует линии, целиком состоящие из штампов, и использовать ее вместо **t.forward()** («вперед») и **t.backward()** («назад»)). Добавь эти строчки в функцию **move_turtle()** после вызова **t.pencolor()**.



Отпечатки в виде черепашки как будто скрепляют линии.

```
def move_turtle(line_length):
    pen_colors = ['red', 'orange', 'yellow', 'green', 'blue', 'purple']
    t.pencolor(random.choice(pen_colors))
    t.fillcolor(random.choice(pen_colors))
    t.shapesize(3,3,1)
    t.stamp()
    if inside_window():
```

Красит черепашку в случайный цвет.

Оставляет на экране отпечаток в виде черепашки.

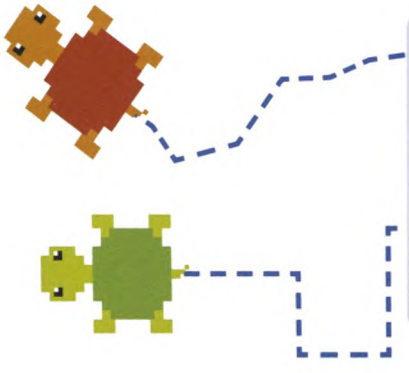
Увеличивает черепашку в 3 раза.

Крутые или плавные повороты?

Программа может запрашивать, на сколько градусов разворачивать черепашку, чтобы на стыках ее путей получались тупые, прямые или острые углы. Выполни эти шаги и посмотри, что будет.

1 Создай функцию

Создай функцию, позволяющую выбрать тип поворота. Введи эти строчки перед функцией **get_line_length()**, добавленной на шаге 3 при написании основного кода.



Запрашивает тип углов.

```
import turtle as t

def get_turn_size():
    turn_size = input('Выберите тип углов (тупые, прямые, острые): ')
    return turn_size

def get_line_length():
```

2 Другие движения

Заменим функцию **move_turtle()** новой версией, которая принимает дополнительный аргумент **turn_size** («величина поворота»). Кроме того, строчку angle = random.randint(0, 180) мы заменили на код, который возвращает разную величину углов поворота в зависимости от значения переменной **turn_size**.

```
def move_turtle(line_length, turn_size):
    pen_colors = ['red', 'orange', 'yellow', 'green', \
                  'blue', 'purple']
    t.pencolor(random.choice(pen_colors))
    if inside_window():
        if turn_size == 'тупые':
            angle = random.randint(120, 150)
        elif turn_size == 'прямые':
            angle = random.randint(80, 90)
        else:
            angle = random.randint(20, 40)
        t.right(angle)
        t.forward(line_length)
    else:
        t.backward(line_length)
```

Тупые углы: от 120 до 150 градусов.

Прямой угол и близкие к нему: от 80 до 90 градусов.

Острые углы: от 20 до 40 градусов.

3 Запроси остроту угла

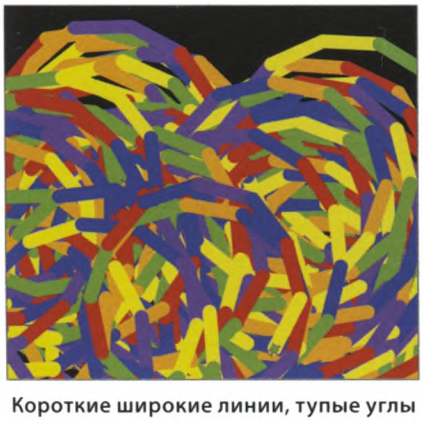
Теперь добавь в основной код вызов функции **get_turn_size()** («получить величину поворота»), которая запрашивает у пользователя остроту угла для поворота черепашки.

```
line_length = get_line length()
line_width = get_line_width()
turn_size = get_turn_size()
```

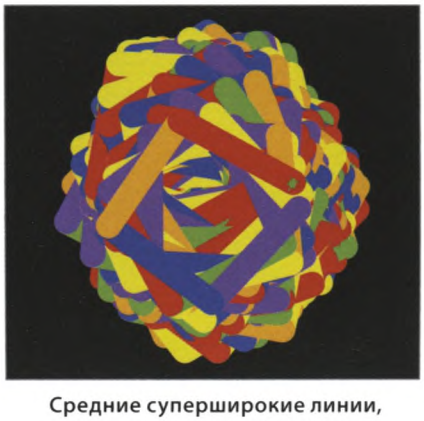
4 Основная часть

И наконец, измени строчку с вызовом функции **move_turtle()**, добавив в нее аргумент **turn_size**.

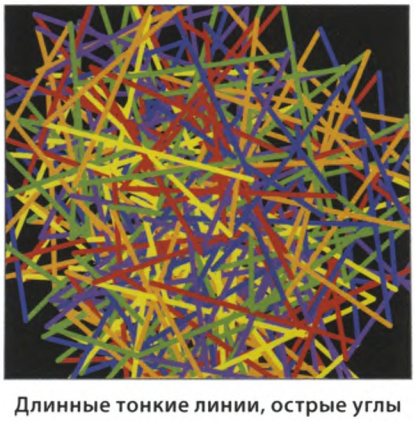
```
while True:
    move_turtle(line_length, turn_size)
```



Короткие широкие линии, тупые углы



Средние суперширокие линии, прямые углы



Длинные тонкие линии, острые углы

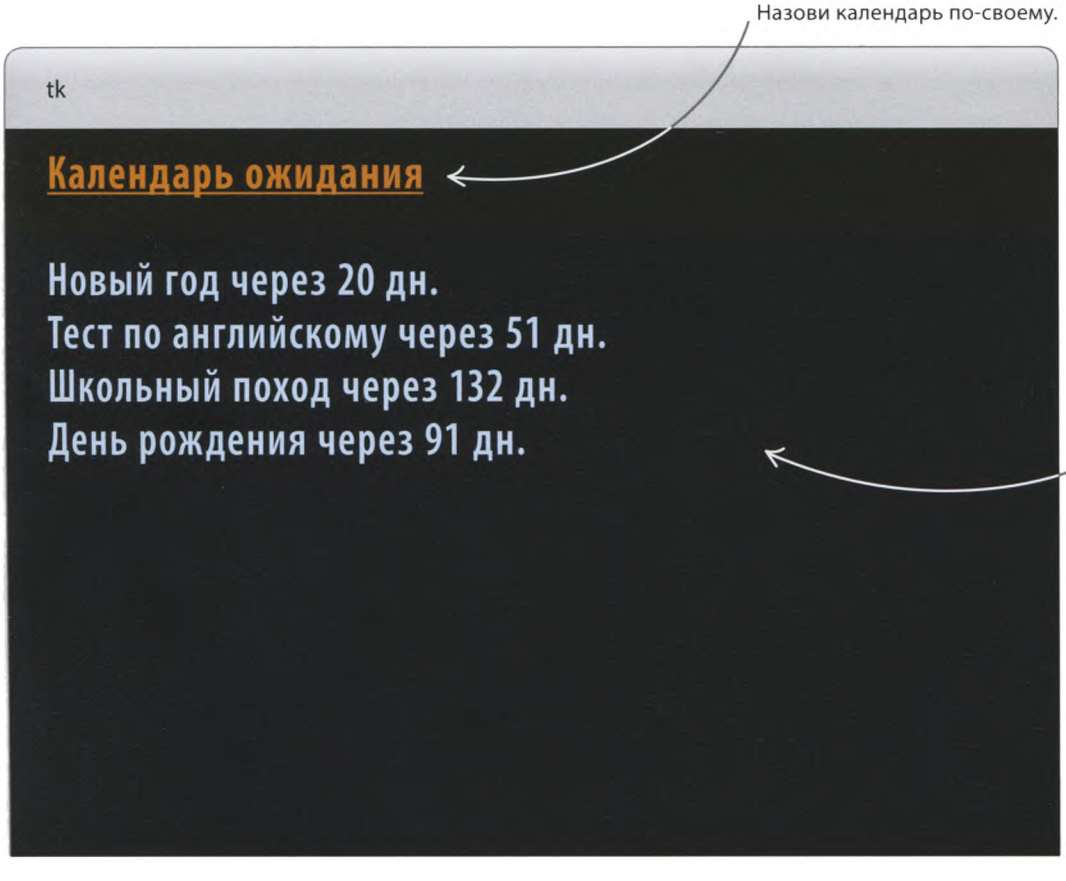
Забавные программы

«Календарь ожидания»

Предвкушая какое-то приятное событие, всегда хочется знать, сколько дней до него осталось. В этом проекте ты познакомишься с модулем Tkinter* и создашь программу, считающую дни до важной даты.

Что происходит

Программа выводит список событий и показывает, сколько дней осталось до каждого из них. Запустив программу на следующий день, ты обнаружишь, что числа уменьшились на 1. Введи даты — и ты больше не пропустишь важное событие или... срок сдачи домашней работы!



* Tkinter (*сокр. от* toolkit interface) — набор инструментов для интерфейса.

Как это работает

Программа считывает информацию из текстового файла, в котором содержатся названия событий и их даты. Это называется «ввод из файла». С помощью модуля **datetime** («дата-время») программа вычисляет разницу в днях между текущей датой и датами событий, а результаты выводит в окне, созданном модулем **Tkinter**.

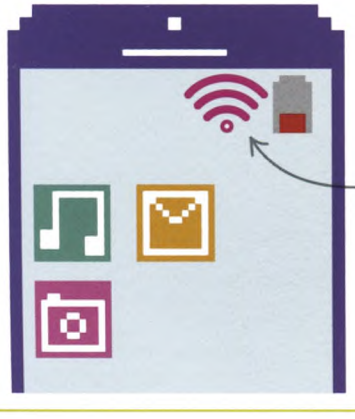
▽ **Модуль Tkinter**

Модуль **Tkinter** — это набор инструментов, которые помогают работать с графикой и вести диалог с пользователем. **Tkinter** позволяет выводить данные в отдельном окне, вид которого ты можешь запрограммировать сам.

СЛЕНГ

GUI

Графический интерфейс пользователя, или GUI (graphical user interface), — это визуальная часть программы, предназначенная для взаимодействия с пользователем. К GUI относятся, например, иконки и выпадающее меню на экране смартфона. Модуль **Tkinter** позволяет конструировать элементы GUI с помощью виджетов — готовых частей кода, создающих кнопки, ползунки, меню и т. д.



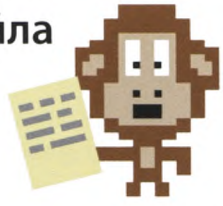
GUI смартфона включает в себя иконки, которые показывают пользователю мощность Wi-Fi-сигнала и заряд батареи.

▽ **Блок-схема программы «Календарь ожидания»**

В этом проекте список событий создается отдельно от кода — в обычном текстовом файле. Сперва программа считывает из него события и даты, а затем вычисляет и выводит на экран количество оставшихся до них дней.

Создание и чтение текстового файла

Все данные для программы «Календарь ожидания» должны храниться в текстовом файле. Создать его можно в IDLE.



Вводи даты в формате «день/месяц/год».

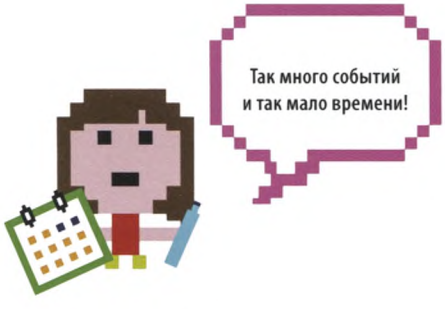
1 Создай текстовый файл

Создай в IDLE текстовый файл и впиши в него несколько важных событий: сначала название, потом, через запятую, дату. Каждое событие должно стоять отдельной строкой. Убедись, что между запятой и датой нет пробела.

events.txt

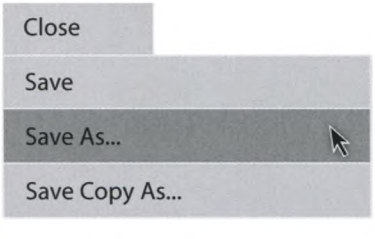
```
Новый год,01/01/2019
Тест по английскому,01/02/2019
Школьный поход,24/04/2019
День рождения,13/03/2019
```

Сначала идет название события.



2 Сохрани текстовый файл

Сохрани текстовый файл: открой меню File, выбери Save As и назови файл events.txt («события»). Теперь можно писать код программы.



3 Создай файл программы

Создай файл для Python-программы, сохранив его как countdown_calendar.py («календарь ожидания») — обязательно в той же папке, что и events.txt.

4 Загрузи модули

Для этого проекта нужны два модуля: **Tkinter** и **datetime**. **Tkinter** поможет создать несложный графический интерфейс, а **datetime** будет выполнять подсчет дней. Добавь в программу эти две строчки кода.

```
from tkinter import Tk, Canvas
from datetime import date, datetime
```

Загрузка модулей Tkinter и datetime.

5 Создай холст

Введи эти строчки после кода, добавленного на шаге 4. В первой строке с помощью базового виджета **Tkinter** создается окно. Во второй благодаря виджету **Canvas** («Холст») в окно добавляется холст, который позволяет выводить текст и изображения на экран.

Помещает холст в окно Tkinter.

Создает окно Tkinter.

Создает холст размером 800 × 800 пикселей.

```
root = Tk()
c = Canvas(root, width=800, height=800, bg='black')
c.pack()
c.create_text(100, 50, anchor='w', fill='orange',\
font='Arial 28 bold underline', text='Календарь ожидания')
```

Добавляет текст на холст c. Текст выводится начиная с позиции x = 100, y = 50, слева направо.

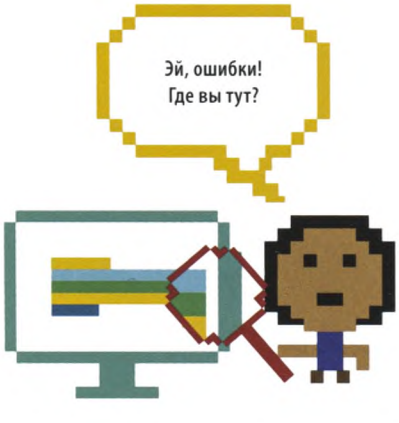
СЛЕНГ

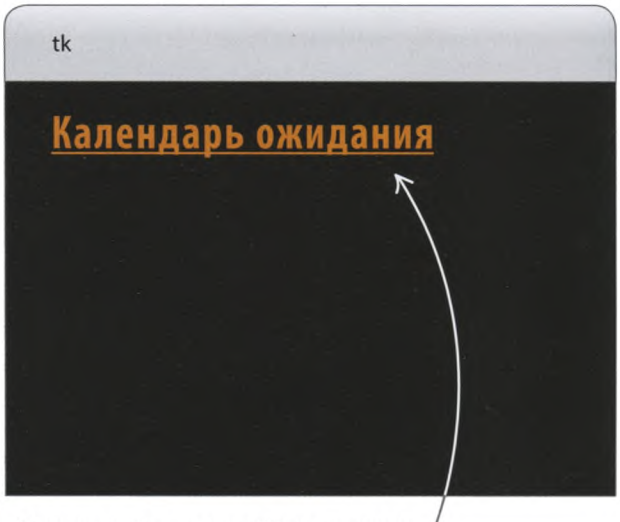
Холст

Холст **Tkinter** — это прямоугольная область для отображения фигур, картинок и текста, с которыми пользователь может взаимодействовать. Эта область похожа на холст художника, только рисуют на ней не кистями и красками, а кодом!

6 Запусти программу

Запусти код. Должно открыться окно с названием программы. Если этого не случилось, посмотри, нет ли в окне консоли сообщений об ошибках, и тщательно, строчка за строчкой, проверь код.





Можешь выбрать другой цвет надписи, изменив строчку с вызовом функции c.create_text() («создать текст»).

7 Считай данные из текстового файла

После загрузки модулей создай функцию **get_events()** («получить события») для считывания данных из текстового файла и помести в нее пустой список, в который будут записываться считанные события.

```
from datetime import date, datetime
def get_events():
    list_events = []
root = Tk()
```

Создает пустой список list_events.

8 Открой текстовый файл

Следующая команда открывает файл events.txt, чтобы программа могла считать данные из него. Введи эту строчку после кода, добавленного на шаге 7.

```
def get_events():
    list_events = []
    with open('events.txt', encoding='utf-8') as file:
```

Открывает текстовый файл.

Добавь это выражение для корректного считывания русского текста из файла .txt.

9 Создай цикл

Для записи данных из файла в код программы создай цикл. Тело цикла будет выполняться для каждой строки файла events.txt.

```
def get_events():
    list_events = []
    with open('events.txt', encoding='utf-8') as file:
        for line in file:
```

Выполняет код цикла для каждой строки текстового файла.

10 Удали невидимый символ

Вводя в текстовый файл данные на шаге 1, ты нажимал ENTER в конце каждой строки. При этом в файл добавлялись невидимые символы «перевод строки». В отличие от тебя, Python их прекрасно видит. Добавь в цикл следующий код: он будет убирать эти знаки из считанных строк.

```
    with open('events.txt', encoding='utf-8') as file:
        for line in file:
            line = line.rstrip('\n')
```

Удаляет символ «перевод строки».

В Python символ «перевод строки» выглядит так: '\n'.

11 Сохрани детали события

Информация о событии попадает в переменную **line** («строка») как строка — например, 'Новый год,01/01/2019'. С помощью команды **split()** («разделить») раздели эту строку надвое. Символы до запятой и после станут отдельными элементами, которые нужно поместить в список **current_event** («текущее событие»).

```
        for line in file:
            line = line.rstrip('\n')
            current_event = line.split(',')
```

Разделяет строку на 2 части по запятой.

СОВЕТЫ ЭКСПЕРТА

Модуль datetime

Модуль **datetime** сильно упрощает вычисления, связанные с датами и временем. Знаешь ли ты, например, в какой день недели родился? Введи в окне консоли эти строки и получи ответ.

Введи день своего рождения в формате «год, месяц, день».

Импортирует все элементы модуля.

```
>>> from datetime import *
>>> print(date(2007, 12, 4).weekday())
1
```

Это число обозначает день недели, от 0 (понедельник) до 6 (воскресенье). Значит, 4 декабря 2007 года был вторник.

ЗАПОМНИ

Позиции в списке

Python нумерует элементы в списке начиная с 0. Так, первый элемент списка **current_event**, 'Новый год', находится в позиции 0, а второй, '01/01/2019', — в позиции 1. Поэтому твой код преобразует в дату именно элемент **current_event[1].**

12 Используй модуль datetime

Событие 'Новый год' попадает в список **current_event** в виде двух элементов: 'Новый год' и '01/01/2019'. С помощью модуля **datetime** преобразуй второй элемент (в позиции 1) в понятную для Python дату. Добавь эти строчки кода в конец тела функции **get_events()**.

Преобразует второй элемент списка из строки в дату.

```
    current_event = line.split(',')
    event_date = datetime.strptime(current_event[1], '%d/%m/%Y').date()
    current_event[1] = event_date
```

Теперь во втором элементе будет храниться дата.

13 Добавь событие в список

Теперь список **current_event** содержит название события (в виде строки) и его дату. Добавь значение **current_event** в список событий **list_events**. Вот окончательный код функции **get_events()**.

```
def get_events():
    list_events = []
    with open('events.txt', encoding='utf-8') as file:
        for line in file:
            line = line.rstrip('\n')
            current_event = line.split(',')
            event_date = datetime.strptime(current_event[1], '%d/%m/%Y').date()
            current_event[1] = event_date
            list_events.append(current_event)
    return list_events
```

После выполнения этой строчки кода цикл повторится для следующей строки текстового файла.

Когда все строки будут считаны, функция get_events() вернет заполненный список событий в основной код программы.

Обратный отсчет

В следующей части программы «Календарь ожидания» ты создашь функцию, вычисляющую разницу в днях между текущей датой и датой грядущего события, а затем напишешь код, выводящий события на холст **Tkinter**.

14 Найди разницу

Создай **days_between_dates()** («дней между датами») — функцию, которая будет возвращать разницу в днях между двумя датами. Модуль **datetime** позволяет легко вычитать одну дату из другой. Введи после функции **get_events()** следующий код, сохраняющий строку с количеством дней в переменной **time_between** («времени между»).

```
def days_between_dates(date1, date2):
    time_between = str(date1 - date2)
```

Функция принимает 2 даты.

Результат в виде строки попадает в эту переменную.

Вычитает одну дату из другой, чтобы получить разницу в днях.

15 Раздели строку

Если до Нового года осталось 27 дней, в **time_between** появится такая строка: `'27 дн., 0:00:00'` (нули обозначают часы, минуты и секунды). Из этой информации тебе нужно лишь число в начале строки, поэтому снова воспользуйся командой **split()**. Введи выделенный черным код после команд, добавленных на шаге 14: он превратит строку в список **number_of_days** («количество дней»), состоящий из трех элементов: `'27'`, `'дн.'` и `'0:00:00'`.

```
def days_between_dates(date1, date2):
    time_between = str(date1 - date2)
    number_of_days = time_between.split(' ')
```

На этот раз строка разделится по каждому из пробелов.

16 Верни количество дней

Чтобы завершить создание функции, нужно лишь получить из нее элемент в нулевой позиции списка (в нашем случае это 27). Добавь такую строчку кода в конец тела функции.

```
def days_between_dates(date1, date2):
    time_between = str(date1 - date2)
    number_of_days = time_between.split(' ')
    return number_of_days[0]
```

Разница в днях хранится в позиции 0 списка number_of_days.

17 Получи список событий

Теперь, когда все функции написаны, их можно использовать в основном коде. Добавь эти строки в самый конец программы. Первая строчка вызывает функцию **get_events()**, сохраняя список событий в переменной **events**. Вторая строчка с помощью модуля **datetime** получает текущую дату и сохраняет ее в переменной **today** («сегодня»).

Раздели длинную строку кода надвое с помощью обратного слеша.

```
c.create_text(100, 50, anchor='w', fill='orange', \
font='Arial 28 bold underline', text='Календарь ожидания')

events = get_events()
today = date.today()
```

Не забудь сохранить свою работу.

18 Отобрази результаты

Рассчитай количество дней, оставшихся до каждого события из списка, и выведи результат на экран, — для этого тебе понадобится цикл **for**. Вызывай функцию **days_between_dates()** для каждого события, сохраняя результат в переменной **days_until** («дней до»), а затем выведи эти данные на экран с помощью функции **create_text()** из модуля **Tkinter**. Введи эти строчки после кода, добавленного на шаге 17.

Тело цикла повторяется для каждого события в списке.

Получает название события.

С помощью функции days_between_dates() считает количество дней между текущей датой и датой события.

```
for event in events:
    event_name = event[0]
    days_until = days_between_dates(event[1], today)
    display = '%s через %s дн.' % (event_name, days_until)
    c.create_text(100, 100, anchor='w', fill='lightblue', \
                  font='Arial 28 bold', text=display)
```

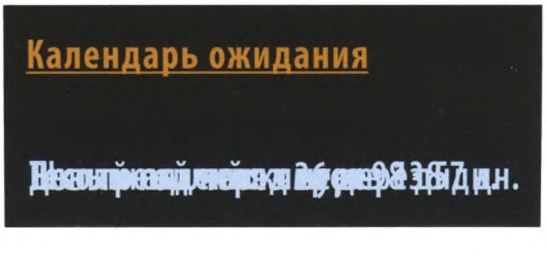
Создает строку с данными, которые нужно отобразить на экране.

Обратный слеш позволяет перенести код на следующую строку.

Отображает текст на экране в позиции (100, 100).

19 Проверь работу программы

Запусти код. Похоже, что все строки текста отобразились друг поверх друга. Как думаешь, что не так в коде и как это исправить?



20 **Раздвинь строки**

Дело в том, что весь текст выводится на экран в одной и той же позиции (100, 100). Если мы создадим переменную **vertical_space** («вертикальный пробел») и будем прибавлять ее значение к *y*-координате каждой строки, увеличивая ее при каждом проходе цикла, это раздвинет строки и решит нашу проблему!

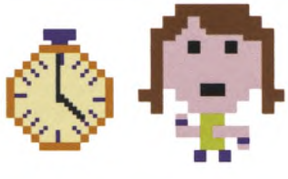
Календарь ожидания

Новый год через 26 дн.
Тест по английскому через 57 дн.
Школьный поход через 138 дн.
День рождения через 98 дн.

```
vertical_space = 100


for event in events:
    event_name = event[0]
    days_until = days_between_dates(event[1], today)
    display = '%s через %s дн.' % (event_name, days_until)
    c.create_text(100, vertical_space, anchor='w', fill='lightblue', \
                  font='Arial 28 bold', text=display)

    vertical_space = vertical_space + 30
```

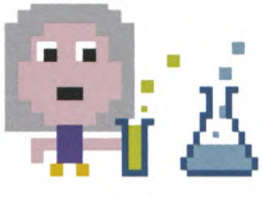
21 **Начинаем отсчет!**

Готово! Ты написал весь код программы «Календарь ожидания». Теперь запусти ее и испытай в деле.



Что бы изменить?

Попробуй внести в программу несколько изменений. Некоторые из них простые, другие посложнее, но подсказки тебе помогут.



▷ **Перекрась холст**

Чтобы оживить внешний вид программы, задай холсту другой фоновый цвет. Для этого замени строчку c = Canvas на эту.

```
c = Canvas(root, width=800, height=800, bg='green')
```

Цвет фона можно поменять на любой другой.

▷ Отсортируй!

Почему бы не сделать так, чтобы события выводились по порядку? Добавь прямо перед циклом **for** следующий код. Для сортировки по возрастанию — от наименьшего количества оставшихся дней до наибольшего — в нем используется функция **sort()** («сортировать»).

```
vertical_space = 100
events.sort(key=lambda x: x[1])
for event in events:
```

Ставит события списка по порядку в зависимости от количества оставшихся дней.

▽ Апгрейд текста

Освежи вид графического окна, поменяв размер, цвет и стиль заголовка.

Укажи свой любимый цвет.

```
c.create_text(100, 50, anchor='w', fill='pink', font='Courier 36 bold underline', \
              text='Долгожданные события')
```

Если хочешь, поменяй заголовок.

Попробуй другой шрифт, например Courier.

▽ Напоминания

Можно сделать так, чтобы события, до которых осталось совсем мало времени, подсвечивались. Пусть события следующей недели отображаются красным.

```
for event in events:
    event_name = event[0]
    days_until = days_between_dates(event[1], today)
    display = '%s через %s дн.' % (days_until, event_name)
    if (int(days_until) <= 7):
        text_col = 'red'
    else:
        text_col = 'lightblue'
    c.create_text(100, vertical_space, anchor='w', fill=text_col, \
                  font='Arial 28 bold', text=display)
```

Символ <= означает «меньше или равно».

Красит текст в соответствующий дате цвет.

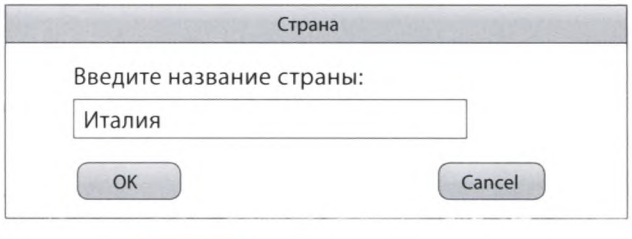
Функция int() преобразует строку в целое число. Например, строка '5' станет числом 5.

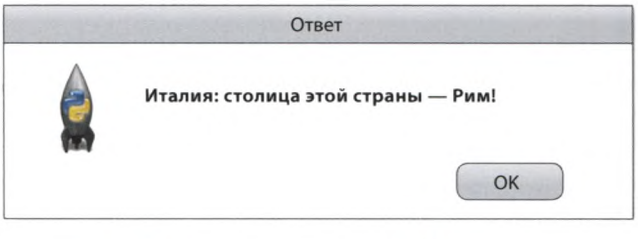
«Знаток»

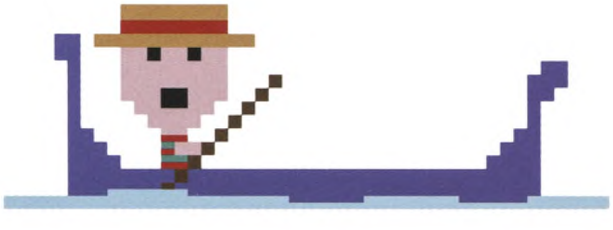
Ты знаешь все столицы мира? Или можешь перечислить игроков любимой футбольной команды? Каждый является знатоком в своей области. Задача этого проекта — создать программу, которая не только отвечает на вопросы, но и учится новому, превращаясь в знатока.

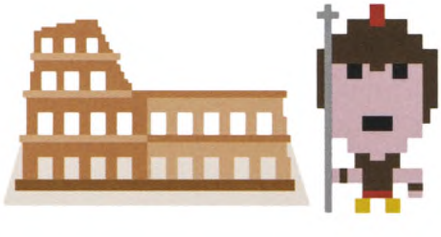
Что происходит

Перед пользователем появляется диалоговое окно с просьбой ввести название страны. После ввода программа сообщает название ее столицы. Если же компьютер не знает ответ, то просит пользователя ввести название столицы самому. Чем дольше люди будут взаимодействовать с программой, тем умнее она станет!









Здесь пользователь пишет название страны.

Если программа не знает столицу, она попросит пользователя ее ввести.

Как это работает

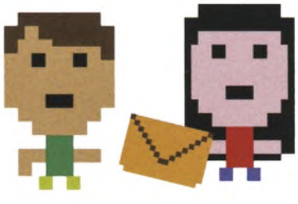
Программа считывает информацию о странах и их столицах из текстового файла. Для создания диалоговых окон, позволяющих компьютеру общаться с пользователем, воспользуйся модулем **Tkinter**. Когда пользователь введет новую столицу, эти данные тоже запишутся в файл.

△ **Словари**

Названия стран и их столиц мы будем хранить в словаре. Словарь в Python похож на список, каждый элемент которого состоит из двух частей: ключа и значения. Как правило, найти данные в словаре проще, чем в длинном списке.

▷ **Диалог с пользователем**

Программа использует два виджета модуля **Tkinter**. Первый, **simpledialog** («простой диалог»), отображает всплывающее окно с запросом страны. Второй, **messagebox** («окно сообщения»), — окно с названием столицы.



▽ **Блок-схема программы «Знаток»**

После запуска программа считывает данные из текстового файла. Затем входит в бесконечный цикл, чтобы задавать вопросы до тех пор, пока пользователь ее не закроет.

СЛЕНГ

Экспертная система

Экспертная система — это программа, которая является специалистом в конкретной области. Она знает ответы на многие вопросы, умеет принимать решения и давать советы. Все это возможно благодаря программисту, который ввел в программу нужные данные, а также правила их использования.

△ **Автомобильный эксперт**

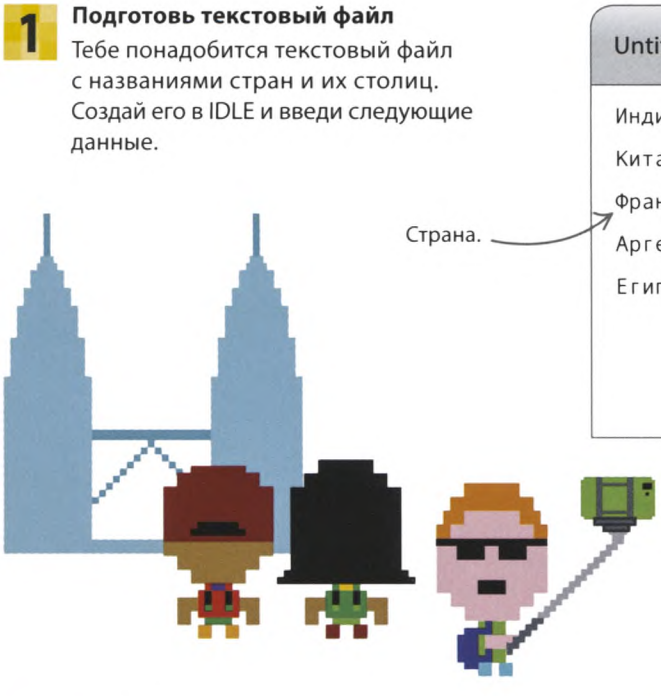
Автомобильные компании создают экспертные системы, которые знают всё о выпускаемых машинах. Если автомобиль сломается, мастер сервис-центра воспользуется одной-единственной экспертной системой вместо того, чтобы обращаться к тысяче механиков!

Первые шаги

Выполни эти шаги, чтобы создать свою экспертную систему. Тебе нужно внести названия стран и их столиц в текстовый файл, а также создать окно **Tkinter** и словарь для хранения данных.

1 Подготовь текстовый файл

Тебе понадобится текстовый файл с названиями стран и их столиц. Создай его в IDLE и введи следующие данные.



Untitled.txt

```
Индия/Нью-Дели
Китай/Пекин
Франция/Париж
Аргентина/Буэнос-Айрес
Египет/Каир
```

Страна.

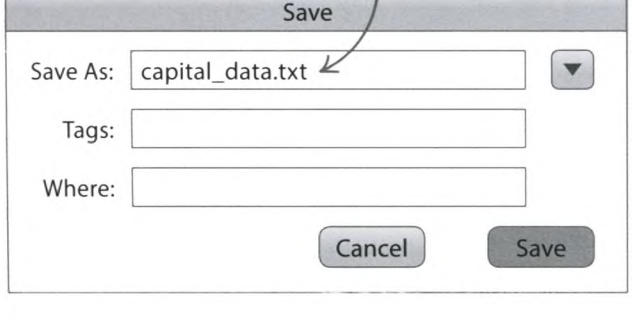
Столица.

Слеш (/) отделяет название столицы от названия страны.

2 Сохрани текстовый файл

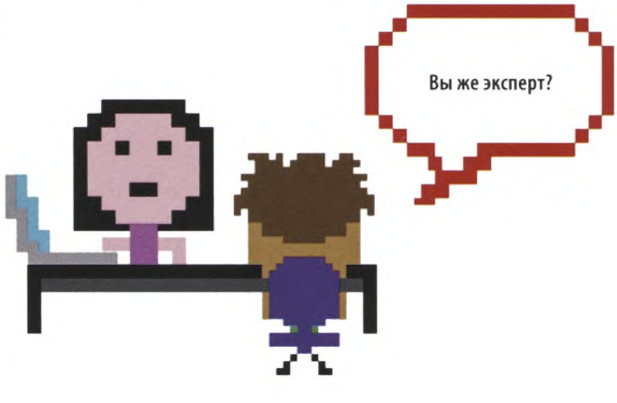
Сохрани файл, из которого программа будет брать данные, под именем capital_data.txt («данные о столицах»).

В конце имени файла вместо «ру» напиши «txt».



3 Создай Python-файл

Для создания программы тоже нужен файл; сохрани его как expert.py («знаток») в той же папке, что и текстовый файл.



4 Загрузи модуль Tkinter

Для программы «Знаток» нужны некоторые виджеты модуля **Tkinter**. Добавь эту строчку в начало программы.

Загрузи эти два виджета модуля Tkinter.

```
from tkinter import Tk, simpledialog, messagebox
```

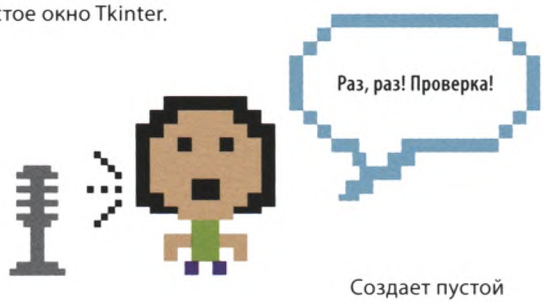
5 Запусти Tkinter

Введи следующий код в окне программы. Он отобразит в окне консоли название проекта, а также запустит модуль **Tkinter**, который автоматически создаст ненужное нам окно (третьей строчкой кода мы его закроем).

```
print('Знаток — Столицы мира')
root = Tk()
root.withdraw()
```

Прячет окно Tkinter.

Создает пустое окно Tkinter.



6 Проверь работу программы

Запусти код. В окне консоли должно появиться название проекта.

Создает пустой словарь.

Элементы словаря помещай внутрь фигурных скобок.

7 Подготовь словарь

Введи после кода, добавленного на шаге 5, эту строчку: она создаст словарь **the_world** («мир»), в котором будут храниться названия стран и их столиц.

```
the_world = {}
```

СОВЕТЫ ЭКСПЕРТА

Словарь

Словарь — это еще один способ хранить данные в Python. Словарь похож на список, элементы которого состоят из двух частей — ключа и значения. Попробуй ввести этот код в окне консоли.

```
favourite_foods = {'Саша': 'пицца', 'Мила': 'оладьи', 'Рома': 'пудинг'}
```

Это ключ. Это значение.

После ключа ставится двоеточие.

Элементы словаря разделяются запятыми.

В словарях используются фигурные скобки.

▽ **1.** Чтобы увидеть содержимое словаря **favourite_foods** («любимая еда»), выведи его на экран.

```
print(favourite_foods)
```

Напиши эту команду в окне консоли и нажми ENTER.

▽ **2.** Добавь в словарь Юлю и ее любимую еду — печенье.

```
favourite_foods['Юля'] = 'печенье'
```

Ключ. Значение.

▽ **3.** Мила подумала и решила, что суши она любит больше, чем оладьи. Эти данные в словаре можно обновить.

```
favourite_foods['Мила'] = 'суши'
```

Новое значение.

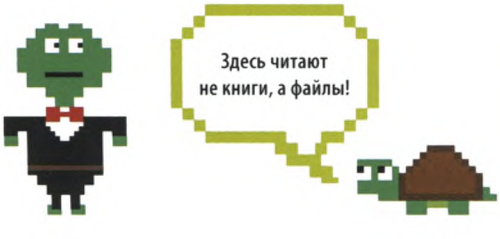
▽ **4.** И, наконец, узнать, что любит Рома, можно, просто указав его имя в качестве ключа.

```
print(favourite_foods['Рома'])
```

Напиши ключ, чтобы узнать его значение.

Время функций!

Следующий этап работы над проектом — создание необходимых программе функций.



8 **Считай данные из текстового файла**

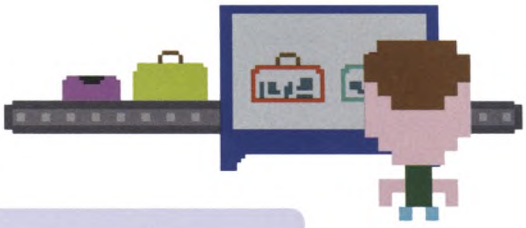
Тебе потребуется функция для считывания данных из текстового файла. Она будет примерно такой же, как в программе «Календарь ожидания». Добавь эти строки кода после загрузки модуля **Tkinter**.

```
from tkinter import Tk, simpledialog, messagebox

def read_from_file():
    with open('capital_data.txt', encoding='utf-8') as file:
```

Эта команда открывает текстовый файл.

9 Строка за строкой

Воспользуйся циклом **for**, чтобы перебрать текстовый файл строка за строкой. Как и в программе «Календарь ожидания», нам необходимо избавиться от невидимых символов «перевод строки». Затем следует получить названия страны и ее столицы, сохранив их в двух переменных. Это можно сделать одной строчкой кода с помощью команды **split()**.



```
def read_from_file():
    with open('capital_data.txt', encoding='utf-8') as file:
        for line in file:
            line = line.rstrip('\n')
            country, city = line.split('/')
```

Удаляет символ «перевод строки».

Слово, стоящее перед слешем, сохраняется в переменной country («страна»).

Слово после слеша сохраняется в переменной city («город»).

Символ '/' разделяет строку.

10 Добавь данные в словарь

После обработки первой строки текстового файла в переменную **country** попадет 'Индия', а в **city** — 'Нью-Дели'. Введи этот код, чтобы добавить данные в словарь.

```
def read_from_file():
    with open('capital_data.txt', encoding='utf-8') as file:
        for line in file:
            line = line.rstrip('\n')
            country, city = line.split('/')
            the_world[country] = city
```

Это значение.

Это ключ.

11 Вывод в файл

Название столицы, которое введет пользователь, нужно сохранить в текстовом файле. Это называется «вывод в файл». Работает он аналогично вводу, но данные должны не считываться, а записываться. Для вывода в файл создай новую функцию **write_to_file()** («записать в файл») после кода, добавленного на шаге 10.

```
def write_to_file(country_name, city_name):
    with open('capital_data.txt', 'a', encoding='utf-8') as file:
```

Эта функция будет добавлять новые названия стран и их столиц в текстовый файл.

Буква «a» включает режим вывода в файл.

12 Запиши данные в текстовый файл

Добавь код для записи новых данных. Сначала программа запишет в текстовый файл символ «перевод строки» (чтобы перейти на следующую строку), затем название страны и через слеш (/) название столицы. Например, так: Египет/Каир. Python автоматически закроет текстовый файл после того, как эти данные будут записаны.


```
def write_to_file(country_name, city_name):
    with open('capital_data.txt', 'a', encoding='utf-8') as file:
        file.write('\n' + country_name + '/' + city_name)
```

Основной код

Все необходимые функции написаны, пора переходить к основному коду программы.

13 Считай данные из текстового файла

Первым делом программа должна загрузить данные из текстового файла. Введи эту строчку после кода, добавленного на шаге 7.

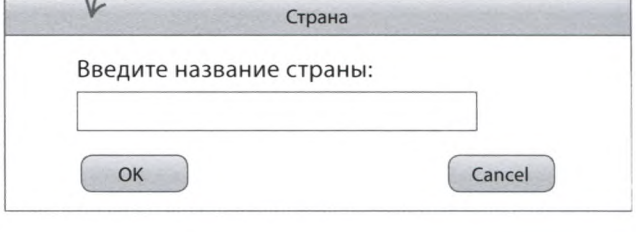
Вызов функции read_from_file() («считать из файла»).

```
read_from_file()
```

14 Создай бесконечный цикл

Добавь следующий код, чтобы создать бесконечный цикл. В теле цикла находится функция **simpledialog.askstring()** («простой диалог. запросить строку») из модуля **Tkinter**. Она создает диалоговое окно с сообщением и полем для ввода текста. Снова запусти программу. Должно появиться окошко с запросом названия страны (оно может открыться позади других окон).

Окно, созданное функцией simpledialog.askstring().



Сообщение, которое отобразится в окне.

```
read_from_file()


while True:
    query_country = simpledialog.askstring('Страна', 'Введите название страны:')
```

Введенная пользователем строка запишется в эту переменную.

Заголовок диалогового окна.

15 Ответ известен?

Теперь добавь конструкцию **if**, чтобы проверить, знает ли программа столицу введенной страны. Это так, если в словаре есть соответствующий элемент.


```
while True:
    query_country = simpledialog.askstring('Страна', 'Введите название страны:')

    if query_country in the_world:
```

Вернет True, если введенная страна есть в словаре the_world.

16 Покажи ответ

Если в словаре **the_world** есть такая страна, нужно, чтобы программа получила ее столицу и вывела ответ на экран. Для этого воспользуйся функцией **messagebox.showinfo()** («окно сообщения.показать информацию») из модуля **Tkinter**. Она покажет ответ в окошке с кнопкой OK. Введи этот код после конструкции **if**.

Не забудь сохранить свою работу.

Ищет ответ в словаре, используя значение переменной query_country («запрос страны») в качестве ключа.

```
    if query_country in the_world:
        result = the_world[query_country]
        messagebox.showinfo('Ответ',
                            query_country + ': столица этой страны — ' + result + '!')
```

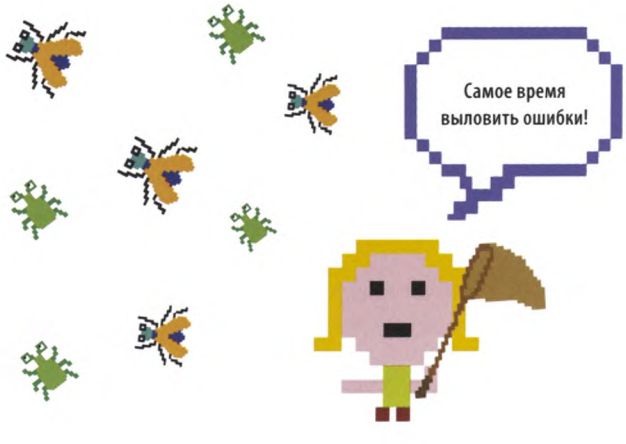
Название окошка.

В эту переменную попадет ответ — значение из словаря.

Сообщение, которое появится в диалоговом окне.

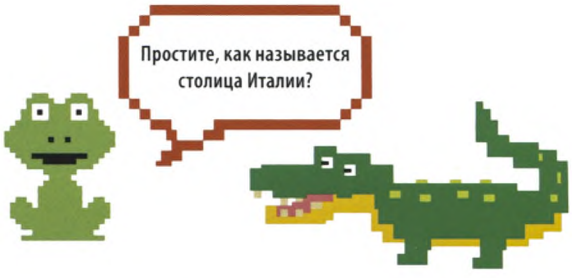
17 Проверь работу программы

Если в твоем коде есть баг, самое время его найти и устранить. Когда программа запросит название страны, введи «Франция». Появился ли на экране правильный ответ? Если нет, внимательно изучи код и постарайся найти ошибку. Также посмотри, что сделает программа, если ты введешь страну, которой нет в текстовом файле.



18 **Научи программу!**

И наконец, добавь еще несколько строк кода после конструкции **if**. Если страны в словаре нет, программа запросит название столицы у пользователя. Его ответ добавится в словарь, чтобы программа его запомнила. Кроме того, функция **write_to_file()** сохранит его в текстовом файле.




```
    if query_country in the_world:
        result = the_world[query_country]
        messagebox.showinfo('Ответ',
                            query_country + ': столица этой страны — ' + result + '!')
    else:
        new_city = simpledialog.askstring('Научите меня',
                                          'Я не знаю, ' +
                                          'как называется столица страны ' + query_country + '!')
        the_world[query_country] = new_city
        write_to_file(query_country, new_city)

root.mainloop()
```

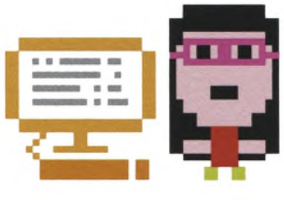
Просит пользователя ввести столицу и помещает его ответ в переменную new_city («новый город»).

Добавляет значение переменной new_city в словарь, используя значение query_country как ключ.

Сохраняет введенный пользователем ответ в текстовом файле, чтобы он попал в базу знаний программы.

19 **Запусти программу**

Вот ты и создал компьютерного знатока! Запусти программу и узнай, на что она способна!



Что бы изменить?

Попробуй внести эти изменения, чтобы вывести программу на новый уровень и сделать ее еще умнее.

▷ **Вокруг света**

Сделай из программы географического гения. Для этого введи в текстовый файл названия всех стран мира и их столиц. Не забудь, что каждая запись должна идти отдельной строкой и выглядеть так: Страна/Столица.

▽ Регистр букв

Если пользователь забудет, что страну надо писать с прописной буквы, программа не найдет ответ. Как решить эту проблему? Вот один из способов.

```
query_country = simpledialog.askstring('Страна', 'Введите название страны:')
query_country = query_country.capitalize()
```

Эта функция превращает первую букву в строке в прописную.

sports_teams.txt

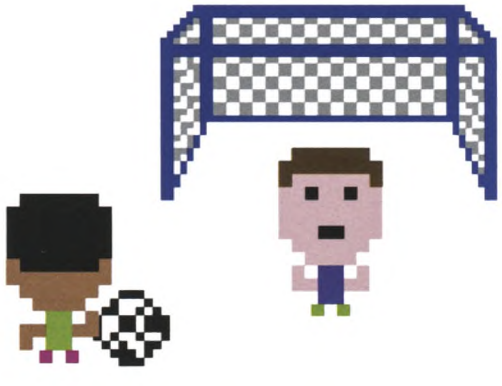
```
Манчестер Юнайтед/Жозе Моуринью
Реал Мадрид/Зинедин Зидан
Ривер Плейт/Марсело Гальярдо
```

Имя тренера.

Название команды.

◁ Другие данные

Сейчас программа знает только страны и их столицы. Это можно изменить, заполнив текстовый файл сведениями из той области, в которой ты разбираешься. Например, ввести в файл названия футбольных команд и имена их тренеров.

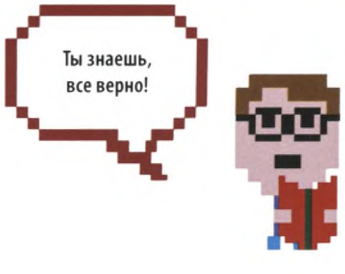


▷ Сверка фактов

Программа записывает в текстовый файл новые сведения, однако она не может знать, верны ли они. Измени код, чтобы новые ответы попадали в отдельный текстовый файл. Потом ты сможешь просмотреть его, исправить и добавить верные сведения в основной файл.

```
def write_to_file(country_name, city_name):
    with open('new_data.txt', 'a', encoding='utf-8') as file:
        file.write('\n' + country_name + '/' + city_name)
```

Сохраняет новые данные в текстовый файл new_data.txt («новые данные»).



«Тайная переписка»

Стань криптографом и отправь друзьям зашифрованные сообщения — так, чтобы не посвященные в вашу тайну люди не смогли ничего понять!

Что происходит

Программа спросит, хочешь ли ты зашифровать сообщение или расшифровать его, а затем предложит ввести сам текст. Если ты выбрал шифрование, сообщение примет вид полной тарабарщины. А если дешифровку, введенная тобой чепуха снова станет читаемым текстом!

СЛЕНГ

Криптография

Слово «криптография» образовано от греческих слов «скрытый» и «письмо». Около 4000 лет эта наука изучает шифры, которые люди придумывают для обмена секретными сообщениями.

Шифр — система условных знаков для передачи секретной информации.

Зашифровать — скрыть смысл сообщения.

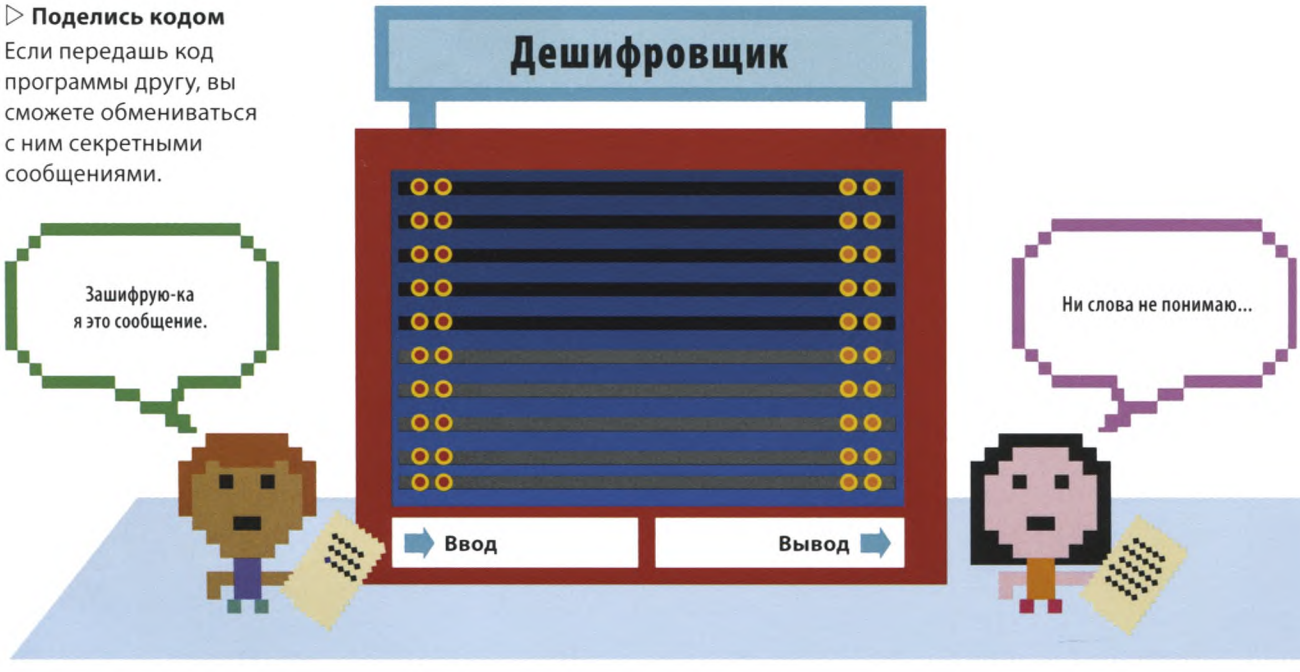
Дешифровать — рассекретить смысл зашифрованного сообщения.

Шифровка — сообщение, набранное тайными символами.

Открытый текст — сообщение перед шифрованием.

▷ **Поделись кодом**

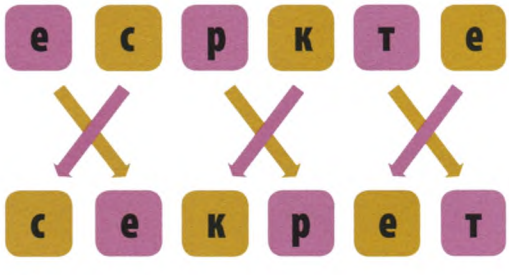
Если передашь код программы другу, вы сможете обмениваться с ним секретными сообщениями.



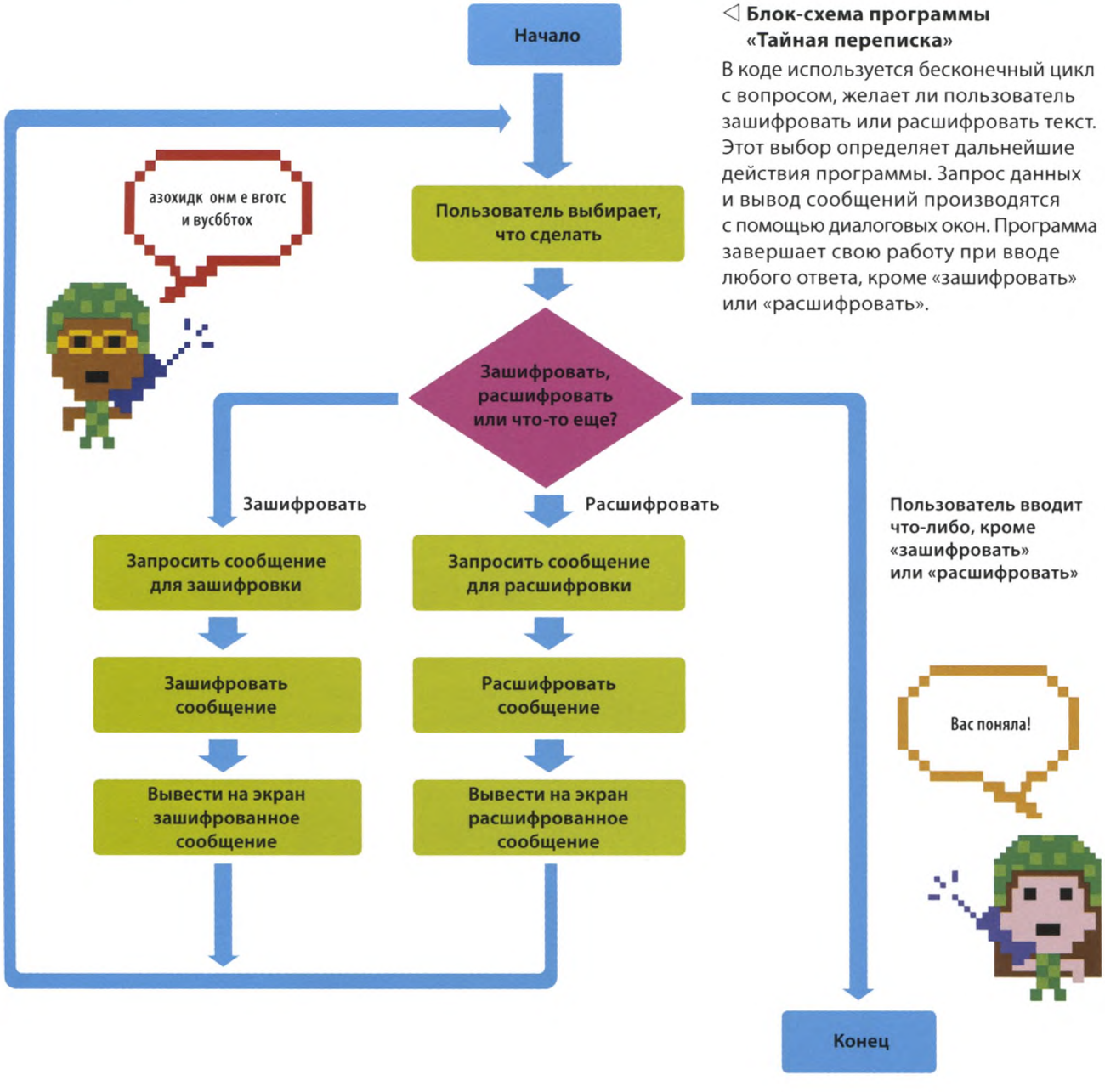
Как это работает

Программа меняет порядок букв в сообщении, делая его нечитаемым. Для этого она выясняет, какие буквы стоят в четных, а какие — в нечетных позициях, и меняет местами сначала буквы в первой паре, затем во второй и т. д. Кроме того, программа может снова сделать зашифрованное сообщение понятным, вернув буквы на прежние места.

△ **Шифрование**
После ввода сообщения в программу она меняет местами буквы внутри каждой пары, скрывая смысл текста.



△ **Дешифровка**
Когда ты или твой друг расшифровываете сообщение, программа ставит буквы на прежние места.

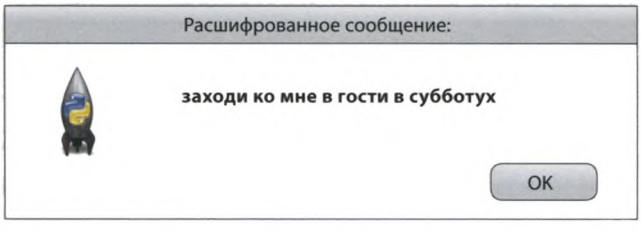


◁ Блок-схема программы «Тайная переписка»

В коде используется бесконечный цикл с вопросом, желает ли пользователь зашифровать или расшифровать текст. Этот выбор определяет дальнейшие действия программы. Запрос данных и вывод сообщений производятся с помощью диалоговых окон. Программа завершает свою работу при вводе любого ответа, кроме «зашифровать» или «расшифровать».

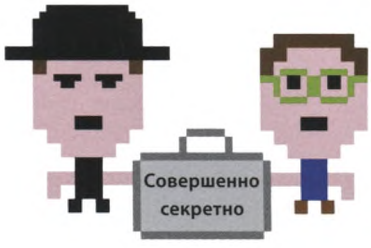
▷ Загадочный «икс»

Программе нужно, чтобы в сообщении было четное количество символов (включая пробелы). Если пользователь введет нечетное количество знаков, программа добавит в конец букву x («икс»). Тебе и твоим друзьям-агентам это не помешает, ведь вы знаете, что «икс» ничего не значит!



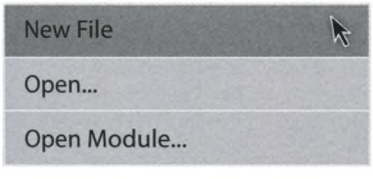
Создание GUI

Тебе предстоит писать код в 2 этапа: сначала создать функции, запрашивающие ввод пользователя, а затем — код шифрования и дешифровки. Приступай не мешкая: отправить кому-нибудь секретное сообщение может понадобиться в любой момент!



1 **Создай новый файл**

Создай в IDLE новый файл и сохрани его как secret_messages.py.



2 **Загрузи модули**

Тебе понадобятся некоторые виджеты модуля **Tkinter**. Виджет **messagebox** позволит отображать сообщения, а **simpledialog** — запрашивать ввод пользователя. Добавь эту строчку кода в начало программы.

```
from tkinter import messagebox, simpledialog, Tk
```

3 **Зашифровать или расшифровать?**

Создай функцию **get_task()** («получить задание»), которая будет выводить диалоговое окно с вопросом, что нужно сделать: зашифровать или расшифровать сообщение. Добавь ее после кода, введенного на шаге 2.

Эта строчка предлагает пользователю ввести «зашифровать» или «расшифровать» и сохраняет его ответ в переменной task («задание»).

```
def get_task():
    task = simpledialog.askstring('Задание', 'Что сделать: зашифровать или расшифровать?')
    return task
```

Возвращает значение переменной task обратно в код, вызвавший эту функцию.

Это заголовок диалогового окна.

4 **Введи сообщение**

Создай функцию **get_message()** («получить сообщение»), которая будет выводить на экран окошко с запросом текста для шифрования или дешифровки. Добавь ее после кода, введенного на шаге 3.

Эта строчка предлагает пользователю ввести сообщение и сохраняет его в переменной message.

```
def get_message():
    message = simpledialog.askstring('Сообщение', 'Введите секретное сообщение: ')
    return message
```

Возвращает значение переменной message обратно в код, вызвавший эту функцию.

5 Запусти Tkinter

Эта команда запускает модуль **Tkinter** и создает пустое окно. Введи ее после функции, созданной на шаге 4.

```
root = Tk()
```

Если окно Tkinter тебя раздражает, добавь команду root.withdraw(), как в предыдущем проекте.

6 Создай бесконечный цикл

Теперь, когда интерфейсные функции созданы, нам понадобится бесконечный цикл, вызывающий их в правильном порядке. Введи этот код после команды, добавленной на шаге 5.

```
while True:
    task = get_task()
    if task == 'зашифровать':
        message = get_message()
        messagebox.showinfo('Сообщение для зашифровки:', message)
    elif task == 'расшифровать':
        message = get_message()
        messagebox.showinfo('Сообщение для дешифровки:', message)
    else:
        break
```

Запрашивает задание.

Получает открытый текст.

Выводит сообщение в диалоговом окне.

Получает зашифрованный текст.

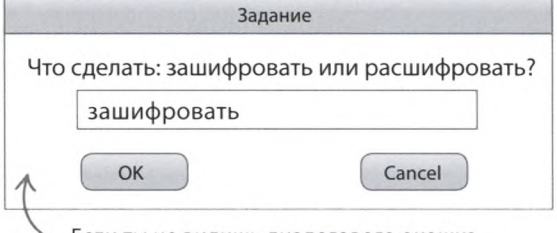
Выводит сообщение в диалоговом окне.

Завершает цикл при вводе чего-либо, кроме слов «зашифровать» или «расшифровать».

7 Проверь работу программы

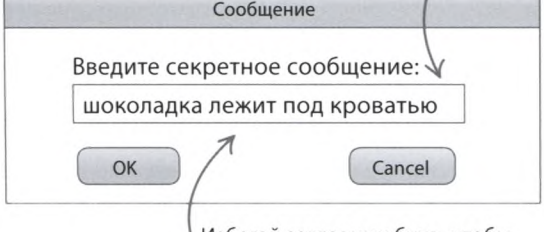
Запусти код; должно появиться окошко с вопросом, что ты хочешь сделать: зашифровать или расшифровать текст. Затем возникнет другое окно — для ввода сообщения. И наконец, должно появиться окно с текстом введенного сообщения. Если это не так, внимательно проверь код.

Введи нужное тебе действие.

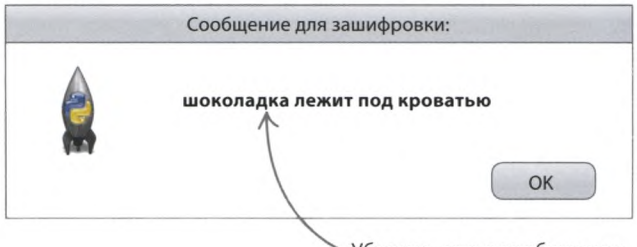


Если ты не видишь диалогового окошка, поищи его под окнами программы и консоли.

Введи открытый текст.



Избегай заглавных букв, чтобы шифровку было труднее разгадать.



Убедись, что с сообщением все в порядке, и нажми OK.

Засекреть сообщение!

Теперь, когда интерфейс работает, самое время написать код для шифрования и дешифровки тайных посланий.

8 Четное ли количество?

Нам понадобится функция, которая проверяет, четное ли количество символов во введенном пользователем сообщении. При помощи оператора % она узнаёт, делится ли это число на 2 без остатка. Если делится, значит, символов четное количество. Добавь функцию **is_even()** («является четным числом») после кода, введенного на шаге 2.

```
def is_even(number):
    return number % 2 == 0
```

Возвращает True или False в код, вызывающий функцию is_even().

Вернет True, если число четное.

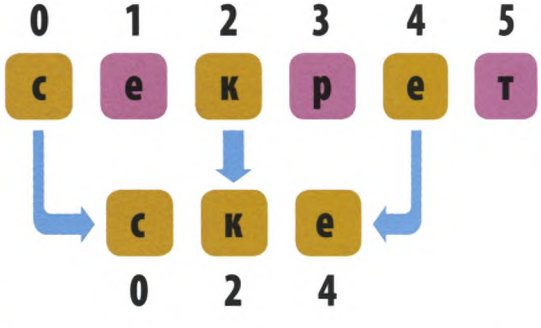
СОВЕТЫ ЭКСПЕРТА

Остаток от деления (%)

Если между двумя числами поставить оператор %, Python выдаст остаток от деления первого числа на второе. Таким образом, 4 % 2 будет 0, а 5 % 2 — 1, поскольку 5 не делится на 2 без остатка, а остаток от деления — число 1. Проверь работу этого оператора в окне консоли.

9 Получи буквы в четных позициях

Создай функцию, которая принимает сообщение и возвращает список, состоящий из букв в четных позициях. В коде используется цикл **for**, который перебирает все буквы в строке, хранящейся в переменной **message**. Добавь функцию **get_even_letters()** («получить четные буквы») после кода, введенного на шаге 8.



Список для хранения букв в четных позициях.

```
def get_even_letters(message):
    even_letters = []
    for counter in range(0, len(message)):
        if is_even(counter):
            even_letters.append(message[counter])
    return even_letters
```

Цикл, проверяющий все буквы сообщения.

Если буква стоит в четной позиции, Python добавляет ее в конец списка.

Возвращает список букв обратно в код, вызвавший эту функцию.

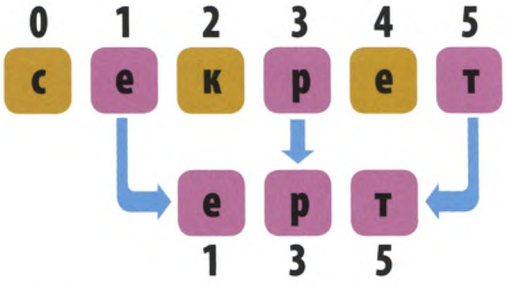
Не забудь сохранить свою работу.

10 Получи буквы в нечетных позициях

Теперь нужно создать похожую функцию, возвращающую список букв в нечетных позициях. Введи этот код.

Оператор отрицания not возвращает противоположное булево значение: True становится False, а False — True.

```
def get_odd_letters(message):
    odd_letters = []
    for counter in range(0, len(message)):
        if not is_even(counter):
            odd_letters.append(message[counter])
    return odd_letters
```



ЗАПОМНИ

Длина списка и строки

Для получения длины строки служит функция **len()**. Так, len('секрет') вернет число 6. Однако Python отсчитывает элементы списка (а строка — это список, элементами которого являются буквы) начиная с 0. Значит, первый символ этой строки стоит в позиции 0, а позиция последнего символа не 6, а 5.

11 Поменяй буквы местами

Ты поместил четные буквы в один список, а нечетные — в другой, и теперь можешь использовать эти списки для зашифровки сообщения. Следующая функция берет буквы поочередно из двух списков и кладет их в третий, но не в первоначальном порядке, а начиная с нечетной позиции. Введи код функции **swap_letters()** («переставленные буквы») после строк, добавленных на шаге 10.

```
def swap_letters(message):
    letter_list = []
    if not is_even(len(message)):
        message = message + 'x'
    even_letters = get_even_letters(message)
    odd_letters = get_odd_letters(message)
    for counter in range(0, int(len(message)/2)):
        letter_list.append(odd_letters[counter])
        letter_list.append(even_letters[counter])
    new_message = ''.join(letter_list)
    return new_message
```

Добавляет к сообщениям с нечетным количеством символов букву x («икс»).

Цикл, который будет перебирать элементы списков с буквами в четных и нечетных позициях.

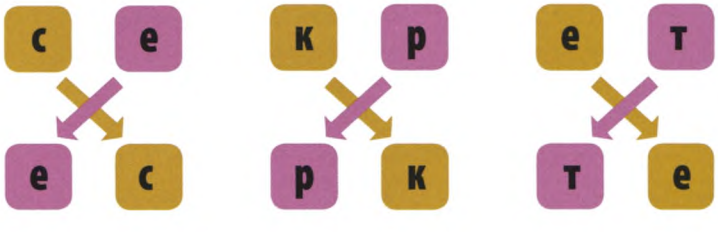
Добавляет в новое сообщение следующую нечетную букву.

Добавляет в новое сообщение следующую четную букву.

Функция join() («объединить») превращает список букв в строку.

▷ Как это работает

Функция **swap_letters()** поочередно помещает буквы в нечетных и четных позициях в новый список. При этом первой становится буква, которая в исходном сообщении была второй.



СОВЕТЫ ЭКСПЕРТА

Целочисленные позиции

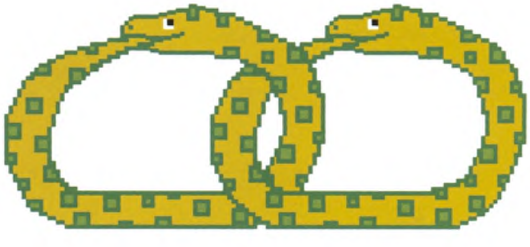
В цикле **for** используется значение len(message)/2, поскольку списки с буквами в четных и нечетных позициях вдвое короче, чем исходное сообщение. Тем не менее при делении получится вещественное число (такое, как 3.0 или 4.0), а не целое (3 или 4). Python выдает ошибку при попытке использовать вещественное число в качестве позиции в списке, поэтому нам понадобится функция **int()**, превращающая вещественное число в целое.

```
>>> mystring = 'секрет'
>>> mystring[3.0]
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    mystring[3.0]
TypeError: string indices must be integers
```

Указав позицию в списке в виде вещественного числа (3.0 вместо целого 3), ты увидишь такое сообщение об ошибке.

12 Измени цикл

Функция **swap_letters()** обладает ценным свойством: если применить ее к зашифрованному сообщению, она его расшифрует. Иначе говоря, функция годится как для шифрования, так и для дешифровки. Внеси эти исправления в цикл **while**, созданный на шаге 6.



```
while True:
    task = get_task()
    if task == 'зашифровать':
        message = get_message()
        encrypted = swap_letters(message)
        messagebox.showinfo('Зашифрованное сообщение:', encrypted)
    elif task == 'расшифровать':
        message = get_message()
        decrypted = swap_letters(message)
        messagebox.showinfo('Расшифрованное сообщение:', decrypted)
    else:
        break
```

Использует swap_letters() для зашифровки сообщения.

Показывает зашифрованное сообщение.

Использует swap_letters() для расшифровки сообщения.

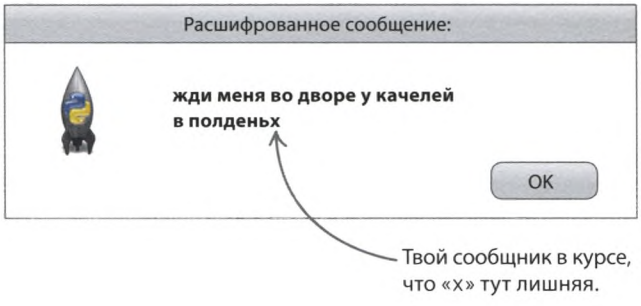
Показывает расшифрованное сообщение.

13 **Запусти шифрование**

Чтобы проверить работу программы, введи в окне «Задание»: «зашифровать». Когда появится окно с запросом сообщения, введи какую-нибудь шпионскую фразу, вроде «жди меня во дворе у качелей в полдень».

14 **Запусти дешифровку**

Скопируй зашифрованный текст из окошка сообщения. На следующем проходе цикла введи «расшифровать», вставь зашифрованный текст и кликни OK. Ты должен увидеть исходное сообщение.



15 **Расшифруй это!**

Твоя программа-дешифровщик готова. Проверь ее, расшифровав эти строки. Теперь можешь передать код своему другу и начать секретную переписку!

ыву псшеонр саишрфвола иесрктеон еосбоещинхе

омолокм жоони пслозьвота ьак кенивидым еечнрлиха

Что бы изменить?

Вот несколько способов усложнить жизнь вражескому агенту (например, любопытному брату или сестре), который может перехватить твои сообщения.

▷ **Убери пробелы**

Один из способов сделать шифровку более надежной — убрать из текста все пробелы и знаки препинания (точки, запятые и т. д.). Просто введи сообщение без них и предупреди друга, что так и задумано.

Задом наперед

Чтобы взломать шифр стало еще сложнее, поменяй порядок букв в тексте, зашифрованном функцией **swap_letters()**, на обратный. Для этого тебе пригодятся две отдельные функции для шифрования и дешифровки.

1 Функция шифрования

Пусть функция **encrypt()** («зашифровать») меняет местами буквы в парах, а затем меняет порядок букв в строке на обратный. Введи эти строчки после функции **swap_letters()**.

Меняет порядок букв на обратный после перестановки букв в парах.

```
def encrypt(message):
    swapped_message = swap_letters(message)
    encrypted_message = ''.join(reversed(swapped_message))
    return encrypted_message
```

2 Функция дешифровки

Добавь функцию **decrypt()** («дешифровать») после функции **encrypt()**. Она сначала «разворачивает» зашифрованный текст, а потом восстанавливает порядок букв с помощью **swap_letters()**.

Отменяет обратный порядок букв.

```
def decrypt(message):
    unreversed_message = ''.join(reversed(message))
    decrypted_message = swap_letters(unreversed_message)
    return decrypted_message
```

Меняет местами буквы в парах, возвращая их в исходное положение.

Не забудь сохранить свою работу.

3 Измени код цикла

Измени код бесконечного цикла так, чтобы вместо **swap_letters()** в нем вызывались новые функции.

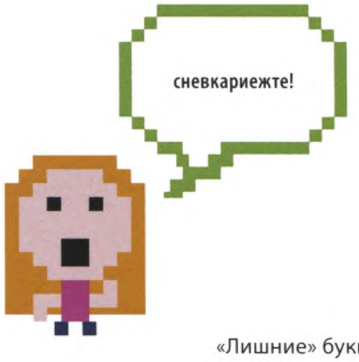
```
while True:
    task = get_task()
    if task == 'зашифровать':
        message = get_message()
        encrypted = encrypt(message)
        messagebox.showinfo('Зашифрованное сообщение:', encrypted)
    elif task == 'расшифровать':
        message = get_message()
        decrypted = decrypt(message)
        messagebox.showinfo('Расшифрованное сообщение:', decrypted)
    else:
        break
```

Замени функцию swap_letters() на encrypt().

Замени функцию swap_letters() на decrypt().

Добавь лишние буквы

Еще один способ усложнить шифровку — добавить после каждого символа по случайной букве. Так слово «секрет» может превратиться в «сбегкираеато» или «сневкариежте». Как и для трюка с разворачиванием строки, тут нужны две функции.



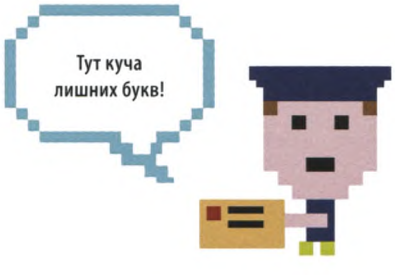
1 Подключи еще один модуль

Импортируй функцию **choice()** из модуля **random** — она нужна, чтобы выбирать случайные буквы из списка. Добавь эту строчку в начало программы, после загрузки модуля **Tkinter**.

```
from tkinter import messagebox, simpledialog, Tk
from random import choice
```

2 Зашифруй

Для шифрования нужен список с буквами, которые мы будем вставлять в сообщение. Следующий код перебирает символы сообщения в цикле, каждый раз добавляя в список **encrypted_list** («список для шифрования») по одной исходной и одной «лишней» букве.




```
def encrypt(message):
    encrypted_list = []
    fake_letters = ['а', 'б', 'в', 'г', 'д', 'е', 'ж', 'и', 'к', 'л', 'м', 'н', 'о']
    for counter in range(0, len(message)):
        encrypted_list.append(message[counter])
        encrypted_list.append(choice(fake_letters))
    new_message = ''.join(encrypted_list)
    return new_message
```

Список с «лишними» буквами.

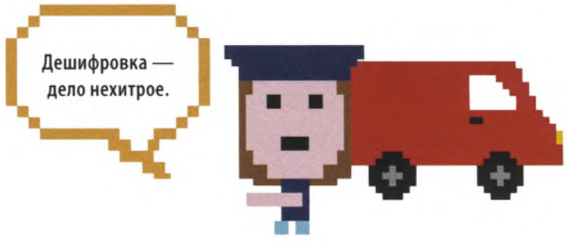
Добавляет в список encrypted_list символ из переменной message.

Добавляет в список encrypted_list «лишнюю» букву из списка fake_letters («ложные буквы»).

Склеивает символы из списка encrypted_list в строку.

3 Расшифруй

Дешифровка выполняется довольно просто. Все буквы исходного сообщения стоят в четных позициях, а значит, их можно получить, воспользовавшись функцией **get_even_letters()**.




```
def decrypt(message):
    even_letters = get_even_letters(message)
    new_message = ''.join(even_letters)
    return new_message
```

Получает буквы исходного сообщения.

Склеивает буквы в строке even_letters и помещает в переменную new_message.

4 Вызови новые функции

Теперь необходимо сделать так, чтобы в бесконечном цикле вместо **swap_letters()** вызывались функции **encrypt()** и **decrypt()**. Поэтому код должен выглядеть так.


```
while True:
    task = get_task()
    if task == 'зашифровать':
        message = get_message()
        encrypted = encrypt(message)
        messagebox.showinfo('Зашифрованное сообщение:', encrypted)
    elif task == 'расшифровать':
        message = get_message()
        decrypted = decrypt(message)
        messagebox.showinfo('Расшифрованное сообщение:', decrypted)
    else:
        break
```

▷ Мультишифрование

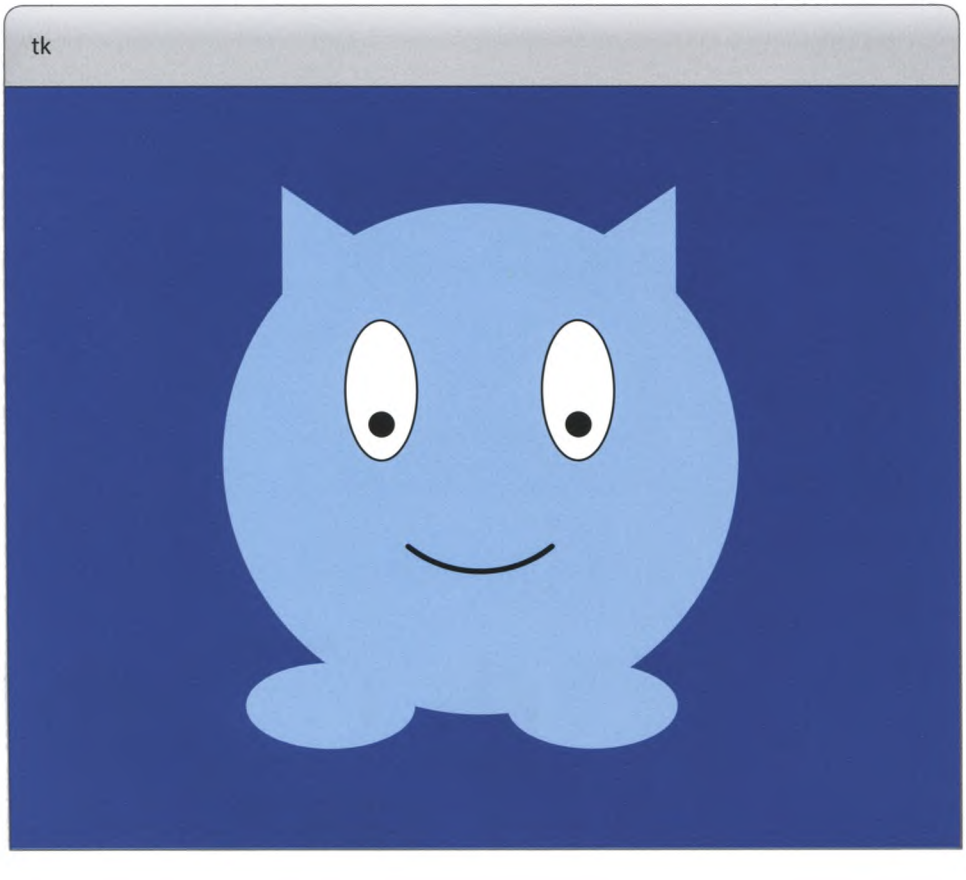
Чтобы еще больше усложнить шифр, можно использовать в программе сразу все хитрости из этого раздела: добавить лишние буквы, поменять местами буквы в парах, а затем изменить порядок букв на обратный!

«Экранный питомец»

Ты хотел бы, чтобы во время выполнения на компьютере домашней работы тебя развлекал милый зверек? Цель этого проекта — создать компьютерного питомца. С ним не заскучаешь, ведь для счастья ему нужны уход и внимание.

Что происходит

Сразу после запуска программы на экране появится зверек небесно-голубого цвета, который будет улыбаться и моргать. Выражение мордочки этого милого создания будет то нейтральным (как на картинке внизу), то счастливым, то нахальным, то грустным — в зависимости от того, как ты к нему относишься. Не бойся, кусаться он точно не станет!



Экранный питомец живет в окошке Tkinter.

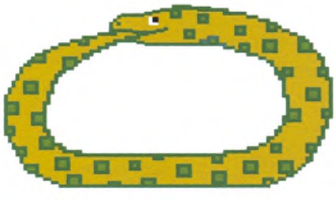
△ **Счастливая мордашка**
Щеки питомца покроются румянцем, если ты «погладишь» его мышкой.

△ **Нахальная мордашка**
Если «пощекочешь» зверька двойным кликом мышки, он покажет язык.

△ **Грустная мордашка**
Если забыть про зверька, он загрустит. «Погладь» его, чтобы питомец снова повеселел.

Как это работает

Функция **root.mainloop()** модуля **Tkinter** запускает цикл обработки пользовательского ввода. Этот цикл создает GUI, реагирующий на клики мышкой и другие команды, и работает до тех пор, пока ты не закроешь окно.



▷ **Анимация**
Кроме того, функция **root.mainloop()** подходит для создания анимации. Поручи ей вызов функций, меняющих картинку в заданные моменты времени, и зверек «оживет».

СЛЕНГ

Код, управляемый событиями

«Экранный питомец» — программа, управляемая событиями. Это значит, что ее работа зависит от пользовательского ввода. Программа ожидает событий, таких как нажатие клавиши или клик мышкой, вызывая разные функции для их обработки. Текстовые и графические редакторы, видеоигры — все это программы, управляемые событиями.

▽ **Блок-схема программы «Экранный питомец»**
Блок-схема показывает последовательность работы программы и логику выбора того или иного варианта в зависимости от действий пользователя. Код запускается внутри бесконечного цикла и отслеживает настроение питомца с помощью переменной.

Рисуем зверька

Итак, приступим. Сначала нужно создать окно, в котором будет жить экранный питомец, а затем — код, который его нарисует.

1 Создай новый файл

Открой IDLE. Зайди в меню File, выбери New File и сохрани файл как screen_pet.py («экранный питомец»).

2 Загрузи модуль Tkinter

В начале программы нужно загрузить части модуля **Tkinter** и создать окно для экранного питомца. Для этого введи следующий код.

Эта строчка загружает части модуля Tkinter, необходимые для этого проекта.

```
from tkinter import HIDDEN, NORMAL, Tk, Canvas
root = Tk()
```

Запускает Tkinter и создает окно.

3 Создай холст

Создай холст синего цвета с именем «c». На нем ты будешь рисовать зверька. Введи этот код после команды, создающей окно **Tkinter**. Эти 4 строчки — начало основного кода.

Создает холст размером 400 × 400 пикселей.

Задает синий цвет фона.

```
from tkinter import HIDDEN, NORMAL, Tk, Canvas
root = Tk()
c = Canvas(root, width=400, height=400)
c.configure(bg='dark blue', highlightthickness=0)
c.pack()
root.mainloop()
```

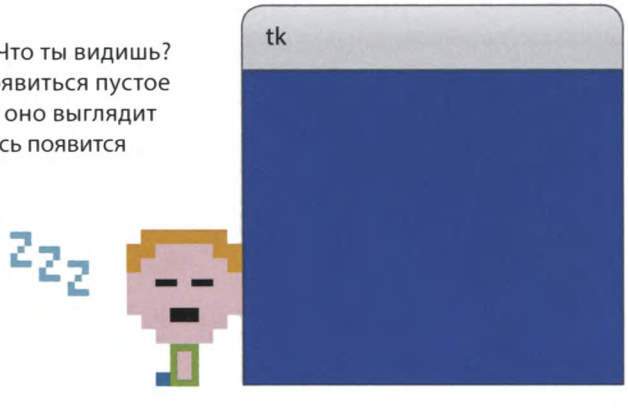
Упорядочивает содержимое окна Tkinter.

Все команды с «c.» в начале относятся к холсту.

Запускает функцию, которая отслеживает события, такие как клики мышкой.

4 Запусти программу

Запусти программу. Что ты видишь? На экране должно появиться пустое синее окно. Пока что оно выглядит уныло, но вскоре здесь появится зверек!



Не забудь сохранить свою работу.

5 Нарисуй экранного питомца

Чтобы изобразить зверька, введи этот код перед последними двумя строчками. Каждая часть питомца рисуется отдельной командой. Числа (координаты) указывают, в каком месте они должны располагаться.

Сохраняет цвет зверька в переменной c.body_color («цвет туловища»), чтобы не вводить каждый раз 'SkyBlue1'.

```
c.configure(bg='dark blue', highlightthickness=0)
c.body_color = 'SkyBlue1'
body = c.create_oval(35, 20, 365, 350, outline=c.body_color, fill=c.body_color)
ear_left = c.create_polygon(75, 80, 75, 10, 165, 70, outline=c.body_color, fill=c.body_color)
ear_right = c.create_polygon(255, 45, 325, 10, 320, 70, outline=c.body_color, \
                             fill=c.body_color)
foot_left = c.create_oval(65, 320, 145, 360, outline=c.body_color, fill= c.body_color)
foot_right = c.create_oval(250, 320, 330, 360, outline=c.body_color, fill= c.body_color)

eye_left = c.create_oval(130, 110, 160, 170, outline='black', fill='white')
pupil_left = c.create_oval(140, 145, 150, 155, outline='black', fill='black')
eye_right = c.create_oval(230, 110, 260, 170, outline='black', fill='white')
pupil_right = c.create_oval(240, 145, 250, 155, outline='black', fill='black')

mouth_normal = c.create_line(170, 250, 200, 272, 230, 250, smooth=1, width=2, state=NORMAL)

c.pack()
```

left и right обозначают здесь левую и правую стороны окна.

Эти пары координат определяют начало, середину и конец рта.

Рот — это дуга толщиной в 2 пикселя.

СОВЕТЫ ЭКСПЕРТА

Координаты Tkinter

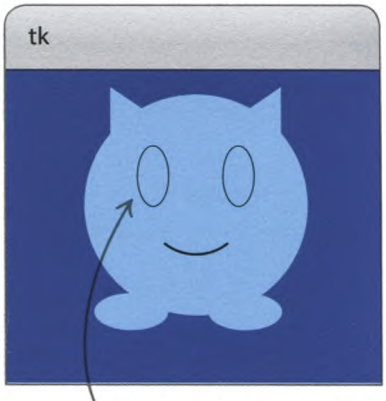
В командах рисования используются *x*- и *y*-координаты. Левому краю окна **Tkinter** соответствует *x*-координата 0. Значение *x* возрастает слева направо, достигая 400 у правой границы. Верхнему краю окна соответствует *y*-координата 0, а нижнему — 400.

6 Снова запусти программу

Запусти программу: в середине окна **Tkinter** должен появиться твой экранный питомец!

Моргающий зверек

Выглядит экранный питомец мило, однако он совершенно неподвижен! Научи его моргать с помощью двух функций: одна будет открывать и закрывать глаза, а другая — указывать, сколько им быть открытыми и закрытыми.



В момент моргания глаза закрашены голубым цветом, зрачков не видно.

7 Научи питомца открывать и закрывать глаза

Добавь в начало программы, после первой строчки кода, функцию **toggle_eyes()** («переключить глаза»). Она будет менять рисунок глаз, либо закрашивая их и пряча зрачки (глаза закрыты), либо возвращая им исходный вид (глаза открыты).

Получает текущий цвет глаз: белый, если глаза открыты, и голубой, если закрыты.

Помещает в переменную new_color новый цвет глаз, меняя его на противоположный.

```
from tkinter import HIDDEN, NORMAL, Tk, Canvas

def toggle_eyes():
    current_color = c.itemcget(eye_left, 'fill')
    new_color = c.body_color if current_color == 'white' else 'white'
    current_state = c.itemcget(pupil_left, 'state')
    new_state = NORMAL if current_state == HIDDEN else HIDDEN
    c.itemconfigure(pupil_left, state=new_state)
    c.itemconfigure(pupil_right, state=new_state)
    c.itemconfigure(eye_left, fill=new_color)
    c.itemconfigure(eye_right, fill=new_color)
```

Получает состояние зрачков: NORMAL (видны) или HIDDEN (скрыты).

Помещает в переменную new_state новое состояние зрачков, меняя его на противоположное.

Показывают или скрывают зрачки, меняя их состояние.

Меняют цвет глаз.

СЛЕНГ

Переключение

Смена состояния на противоположное называется переключением. Это подобно тому, как с помощью кнопки выключателя ты то включаешь, то выключаешь свет в комнате. Код моргания «переключает» состояние глаз: «закрывает» их, если они открыты, и «открывает», если закрыты.

8 **Научи питомца моргать**

Моргание можно показать, сделав так, чтобы глаза долго оставались открытыми, а закрывались лишь на мгновение. Добавь после кода, введенного на шаге 7, функцию **blink()** («моргать»). Она «закрывает» глаза зверька на четверть секунды (250 миллисекунд), а затем просит функцию **mainloop()** снова вызвать ее через 3 секунды (3000 миллисекунд).

```
    c.itemconfigure(eye_right, fill=new_color)

def blink():
    toggle_eyes()
    root.after(250, toggle_eyes)
    root.after(3000, blink)

root = Tk()
```

Закрывает глаза.

Открывает глаза через 250 миллисекунд.

Заставляет зверька снова моргнуть через 3000 миллисекунд.

9 **Создай анимацию!**

Добавь эту команду в основной код, прямо перед последней строкой, и запусти программу. Через 1 секунду (1000 миллисекунд) твой питомец моргнет и будет это делать, пока ты не закроешь окно.

```
root.after(1000, blink)
root.mainloop()
```

Ждет 1000 миллисекунд и включает моргание.

Смена настроения

Сейчас зверек выглядит довольным, но давай сделаем его еще более радостным — добавим счастливую улыбку и яркий румянец!

10 **Счастливая мордочка**

Добавь этот код к командам создания зверька после строки, рисующей «нейтральный» рот. Кроме счастливой улыбки и румянца этот код также создает «грустный» рот, но пока все эти рисунки невидимы.

Создает счастливую улыбку.

Создает «грустный» рот.

```
mouth_normal = c.create_line(170, 250,200, 272, 230, 250, smooth=1, width=2, state=NORMAL)
mouth_happy = c.create_line(170, 250, 200, 282, 230, 250, smooth=1, width=2, state=HIDDEN)
mouth_sad = c.create_line(170, 250, 200, 232, 230, 250, smooth=1, width=2, state=HIDDEN)

cheek_left = c.create_oval(70, 180, 120, 230, outline='pink', fill='pink', state=HIDDEN)
cheek_right = c.create_oval(280, 180, 330, 230, outline='pink', fill='pink', state=HIDDEN)

c.pack()
```

Создают розовые кружки румянца.

11 **Счастливое лицо**

Теперь создай функцию **show_happy()** («показать счастье») — она будет отображать счастливую улыбку и румянец, когда пользователь «гладит» зверька курсором мышки. Введи этот код после функции **blink()**.

СЛЕНГ

Обработчик события

Функция **show_happy()** — обработчик события. Это значит, что она будет вызываться, если произойдет некое событие (в нашем случае «поглаживание»), чтобы его обработать. В повседневной жизни функцию «вымыть пол» можно назвать обработчиком события «пролил чай».

Проверяет, находится ли курсор мышки над зверьком.

event.x и event.y — это координаты курсора мышки.

```
root.after(3000, blink)


def show_happy(event):
    if (20 <= event.x <= 350) and (20 <= event.y <= 350):
        c.itemconfigure(cheek_left, state=NORMAL)
        c.itemconfigure(cheek_right, state=NORMAL)
        c.itemconfigure(mouth_happy, state=NORMAL)
        c.itemconfigure(mouth_normal, state=HIDDEN)
        c.itemconfigure(mouth_sad, state=HIDDEN)
    return
```

Показывают кружки румянца.

Показывает счастливую улыбку.

Прячет «нейтральный» рот.

Прячет «грустный» рот.

СОВЕТЫ ЭКСПЕРТА

Фокус ввода

Tkinter не поймет, что ты двигаешь мышкой над питомцем, если окно не будет сфокусировано. Чтобы оно получило фокус ввода, то есть способность принимать ввод пользователя, достаточно кликнуть по окну мышкой.

12 **Счастливое поглаживание**

После запуска программы зверек будет моргать сам по себе. Однако вызвать у него счастливую улыбку может лишь событие. В **Tkinter** движение курсора мышки называется событием **<Motion>**. Его нужно связать с функцией-обработчиком при помощи команды **bind()** («связать»). Добавь эту строчку в основной код, запусти программу и «погладь» зверька.

```
c.pack()

c.bind('<Motion>', show_happy)

root.after(1000, blink)
root.mainloop()
```

Связывает движение курсора мышки с показом счастливого лица.

13 Спрячь счастливое лицо

Экранный питомец должен выглядеть счастливым, только когда его гладят. Добавь после функции **show_happy()** новую функцию **hide_happy()** («спрятать счастье»), которая возвращает зверьку нейтральное выражение лица.

Не забудь сохранить свою работу.

```
def hide_happy(event):
    c.itemconfigure(cheek_left, state=HIDDEN)
    c.itemconfigure(cheek_right, state=HIDDEN)
    c.itemconfigure(mouth_happy, state=HIDDEN)
    c.itemconfigure(mouth_normal, state=NORMAL)
    c.itemconfigure(mouth_sad, state=HIDDEN)
    return
```

Прячут кружки румянца.

Прячет счастливую улыбку.

Показывает «нейтральный» рот.

Прячет «грустный» рот.

14 Свяжи выход курсора со счастьем

Введи команду, которая связывает событие **<Leave>** (выход курсора мышки за пределы окна) с вызовом функции **hide_happy()**, и запусти программу.

```
c.bind('<Motion>', show_happy)
c.bind('<Leave>', hide_happy)


root.after(1000, blink)
```

Каков нахал!

До сих пор твой питомец вел себя примерно. Давай добавим ему нахальства! Напиши код, благодаря которому зверек высунет язык и скосит глаза, если ты «пощекочешь» его двойным кликом мышки.

15 Нарисуй язык

Добавь эти строчки в код рисования зверька после команды, изображающей «грустный» рот. Язык будет состоять из двух фигур: прямоугольника и овала.

```
mouth_sad = c.create_line(170, 250, 200, 232, 230, 250, smooth=1, width=2, state=HIDDEN)
tongue_main = c.create_rectangle(170, 250, 230, 270, outline='red', fill='red', state=HIDDEN)
tongue_tip = c.create_oval(170, 250, 230, 300, outline='red', fill='red', state=HIDDEN)


cheek_left = c.create_oval(70, 180, 120, 230, outline='pink', fill='pink', state=HIDDEN)
```

16 **Настрой флаговые переменные**

Добавь две флаговые переменные, чтобы отслеживать, скошены ли зрачки зверька и высунут ли его язык. Введи этот код перед строкой, запускающей моргание, которую ты добавил в основной код на шаге 9.

```
c.eyes_crossed = False
c.tongue_out = False


root.after(1000, blink)
```

Флаговые переменные для зрачков и языка.

17 **«Переключи» язык**

Функция **toggle_tongue()** («переключить язык») «переключает» положение языка, делая его или высунутым, или убранным. Введи этот код перед функцией **show_happy()**, созданной на шаге 11.

```
def toggle_tongue():
    if not c.tongue_out:
        c.itemconfigure(tongue_tip, state=NORMAL)
        c.itemconfigure(tongue_main, state=NORMAL)
        c.tongue_out = True
    else:
        c.itemconfigure(tongue_tip, state=HIDDEN)
        c.itemconfigure(tongue_main, state=HIDDEN)
        c.tongue_out = False


def show_happy(event):
```

Проверяет, высунут ли язык.

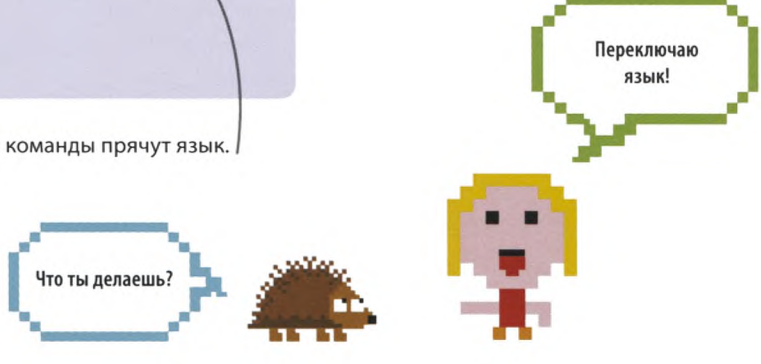
Если язык убран, эти команды делают его высунутым.

Присваивает флаговой переменной булево значение, означающее, что язык высунут.

Если язык уже высунут, запускается код else.

Присваивает флаговой переменной булево значение, означающее, что язык убран.

Эти команды прячут язык.



СОВЕТЫ ЭКСПЕРТА

Флаговые переменные

Флаговые переменные позволяют отслеживать что-либо, находящееся в одном из двух состояний. Изменив состояние, флаг нужно обновить. Табличка «занято/свободно» на дверях туалета — такой же флаг: закрывая дверь, ты включаешь режим «занято», а открывая — режим «свободно».

```
    root.after(3000, blink)

def toggle_pupils():
    if not c.eyes_crossed:
        c.move(pupil_left, 10, -5)
        c.move(pupil_right, -10, -5)
        c.eyes_crossed = True
    else:
        c.move(pupil_left, -10, 5)
        c.move(pupil_right, 10, 5)
        c.eyes_crossed = False
```

Проверяет, скошены ли у зверька глаза.

Если зрачки в нормальном положении, эти команды делают глаза скошенными.

Присваивает флаговой переменной булево значение, означающее, что глаза скошены.

Если глаза уже скошены, запускается код else.

Эти команды возвращают зрачки в нормальное положение.

Присваивает флаговой переменной булево значение, означающее, что глаза не скошены.

18 «Переключи» зрачки

Чтобы глаза стали скошенными, нужно изменить положение зрачков. Функция **toggle_pupils()** («переключить зрачки») меняет состояние зрачков зверька с обычного на скошенное и наоборот. Введи этот код после функции **blink()**, добавленной на шаге 8.

19 Управляй нахальством

Создай функцию, с помощью которой зверек будет одновременно скашивать глаза и высовывать язык. Введи этот код после функции **toggle_tongue()**, добавленной на шаге 17. Чтобы через секунду зверек принял нормальный вид, воспользуйся функцией **root.after()** («основа.после»), как ты это делал раньше с **blink()**.

Не забудь сохранить свою работу.

```
def cheeky(event):
    toggle_tongue()
    toggle_pupils()
    hide_happy(event)
    root.after(1000, toggle_tongue)
    root.after(1000, toggle_pupils)
    return
```

Высовывает язык.

Скашивает глаза.

Прячет счастливое лицо.

Убирает язык через 1 секунду (1000 миллисекунд).

Возвращает зрачки в нормальное состояние через 1 секунду (1000 миллисекунд).

20 Свяжи двойной клик с нахальством

Чтобы управлять нахальством зверька, свяжи событие «двойной клик» с функцией **cheeky()** («нахальный»). Введи эту строчку кода после команды, добавленной на шаге 14. Запусти программу, сделай двойной клик в окне и полюбуйся на нахала!

```
c.bind('<Motion>', show_happy)
c.bind('<Leave>', hide_happy)
c.bind('<Double-1>', cheeky)
```

В Tkinter событие «двойной клик» называется <Double-1>.

Грустный зверек

И наконец, научи экранного питомца реагировать на нехватку внимания: пусть бедняга грустит, если его не гладить около минуты!

21 Задай уровень настроения

Введи эту строчку кода перед флаговой переменной, которую ты добавил на шаге 16. Сначала уровень настроения будет равен 10. Чем выше это значение, тем довольнее твой питомец!

```
c.happy_level = 10
c.eyes_crossed = False
```

Вначале у зверька будет уровень настроения 10.

22 Вызови функцию

Введи следующую строчку кода после команды моргания, добавленной на шаге 9. Это указание через 5 секунд (5000 миллисекунд) вызвать функцию **sad()** («грустный»), которую тебе предстоит создать на шаге 23.

```
root.after(1000, blink)
root.after(5000, sad)
root.mainloop()
```

23 Создай функцию sad()

Добавь после функции **hide_happy()** функцию **sad()**. При нулевом значении переменной **c.happy_level** она показывает грустное выражение лица, а в любом другом случае уменьшает значение **c.happy_level** на 1. Подобно **blink()**, функция **sad()** вызывает саму себя через 5 секунд.

```
def sad():
    if c.happy_level == 0:
        c.itemconfigure(mouth_happy, state=HIDDEN)
        c.itemconfigure(mouth_normal, state=HIDDEN)
        c.itemconfigure(mouth_sad, state=NORMAL)
    else:
        c.happy_level -= 1
    root.after(5000, sad)
```

Проверяет значение c.happy_level на равенство 0.

Если значение переменной c.happy_level равно 0, прячет и счастливое, и нейтральное выражение лица.

Показывает грустное выражение лица.

Если значение c.happy_level больше 0...

...уменьшает значение c.happy_level на 1.

Снова вызывает sad() через 5 секунд (5000 миллисекунд).

24 **Взбодрись, зверек!**

Что нужно делать, чтобы экранный питомец не грустил? И как его развеселить, если он в печали? Разумеется, кликнуть по окну и «погладить» зверька! Добавь эту строчку кода в функцию **show_happy()**, созданную на шаге 11. Теперь функция вернет в переменную **c.happy_level** значение 10, чтобы зверек оставался веселым сразу после «поглаживания». Запусти программу и, когда питомец загрустит, «погладь» его.

Не забудь сохранить свою работу.

```
    c.itemconfigure(mouth_normal, state = HIDDEN)
    c.itemconfigure(mouth_sad, state = HIDDEN)
    c.happy_level = 10
return
```

Возвращает уровень настроения 10.

Что бы изменить?

Похож ли твой зверек на идеального питомца? Если нет, можешь изменить его повадки или добавить программе новые возможности. Вот несколько советов.

Будь милашкой, а не нахалом

Может, ты не любишь нахалов? Тогда измени код, чтобы при двойном клике питомец дружески тебе подмигивал, а не корчил рожи.

СОВЕТЫ ЭКСПЕРТА

Больше радости

Приятно то и дело гладить зверька, однако это отвлекает от выполнения домашней работы. Чтобы твой питомец расстраивался не так часто, увеличь максимальный уровень его настроения.

Увеличь это число.

```
c.happy_level = 10
c.eyes_crossed = False
```

1 Добавь функцию **toggle_left_eye()** («переключить левый глаз») после функции **blink()**. Работает она так же, но «переключает» только один глаз.

```
def toggle_left_eye():
    current_color = c.itemcget(eye_left, 'fill')
    new_color = c.body_color if current_color == 'white' else 'white'
    current_state = c.itemcget(pupil_left, 'state')
    new_state = NORMAL if current_state == HIDDEN else HIDDEN
    c.itemconfigure(pupil_left, state=new_state)
    c.itemconfigure(eye_left, fill=new_color)
```

2 Функция **wink()** («подмигнуть») однократно закрывает и открывает левый глаз зверька, создавая иллюзию, будто он подмигивает. Добавь ее после **toggle_left_eye()**.

```
def wink(event):
    toggle_left_eye()
    root.after(250, toggle_left_eye)
```

3 Не забудь изменить в основном коде команду, связывающую действие с событием **<Double-1>**, поменяв вызов **cheeky()** на **wink()**.

```
c.bind('<Double-1>', wink)
```

Поменяй cheeky на wink.

Радужный зверек

Изменив значение переменной **c.body_color**, можно покрасить экранного питомца в любой другой цвет. Не знаешь, какой выбрать? Тогда напиши функцию, постоянно меняющую цвет зверька!

1 Загрузи модуль **random**. Введи команду после строчки, импортирующей части **Tkinter**.

```
from tkinter import HIDDEN, NORMAL, Tk, Canvas
import random
```

2 Добавь перед основным кодом новую функцию **change_color()** («изменить цвет»): она присваивает переменной **c.body_color** случайный цвет из списка **pet_colors** и перекрашивает зверька. Благодаря функции **random.choice()** цвет питомца будет меняться совершенно непредсказуемо!

```
def change_color():
    pet_colors = ['SkyBlue1', 'tomato', 'yellow', 'purple', 'green', 'orange']
    c.body_color = random.choice(pet_colors)]
    c.itemconfigure(body, outline=c.body_color, fill=c.body_color)
    c.itemconfigure(ear_left, outline=c.body_color, fill=c.body_color)
    c.itemconfigure(ear_right, outline=c.body_color, fill=c.body_color)
    c.itemconfigure(foot_left, outline=c.body_color, fill=c.body_color)
    c.itemconfigure(foot_right, outline=c.body_color, fill=c.body_color)
    root.after(5000, change_color)
```

Список цветов, в которые можно покрасить зверька.

Случайным образом выбирает цвет из списка.

Эти команды обновляют цвет туловища, ног и ушей экранного питомца.

Программа снова вызывает change_color() через 5 секунд (5000 миллисекунд).

3 И наконец, добавь сразу перед последней строкой программы вызов функции **change_color()** через 5 секунд (5000 миллисекунд) после запуска программы.

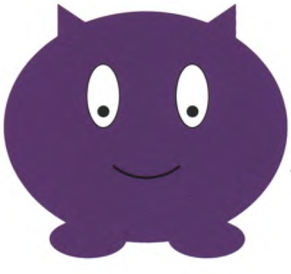
```
root.after(5000, change_color)
```

Зверек начнет менять цвет через 5 секунд после запуска программы.

4 Если хочешь, подставь в код другие значения, чтобы цвет экранного питомца менялся не так быстро. Кроме того, ты можешь изменить в списке цвета или добавить новые.

Покорми меня!

Помимо щекотки и поглаживаний, питомцы нуждаются в еде. Подумай, как можно накормить экранного жителя.



Растущему организму нужно много еды!

1 Попробуй добавить в окне питомца кнопку «Покорми меня!» и создать функцию кормления, которую можно вызывать нажатием этой кнопки.

2 Пусть после нескольких нажатий кнопки «Покорми меня!» зверек вырастает. Вот строчка кода, увеличивающая размер его туловища.

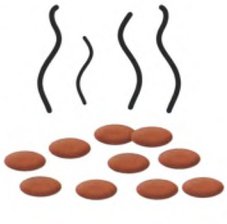
Меняет форму овала, который соответствует туловищу экранного питомца.

```
body = c.create_oval(15, 20, 395, 350, outline=c.body_color, fill=c.body_color)
```

3 Напиши код, уменьшающий тело зверька до прежних размеров, если питомец не получает вдоволь еды.

▷ Убери за мной!

Раз экранный питомец ест, значит, он должен и какать! Пускай после кормления зверька в окне возникает теплая кучка. Добавь кнопку «Убрать туалет», связав ее нажатие с функцией, убирающей кучку с экрана.



СОВЕТЫ ЭКСПЕРТА

Размер окна

Если добавить в окно питомца кнопки или другие элементы интерфейса, зверьку может стать тесно. В этом случае увеличь размер окна Tkinter, поменяв его ширину и высоту в строке, создающей холст в начале программы.

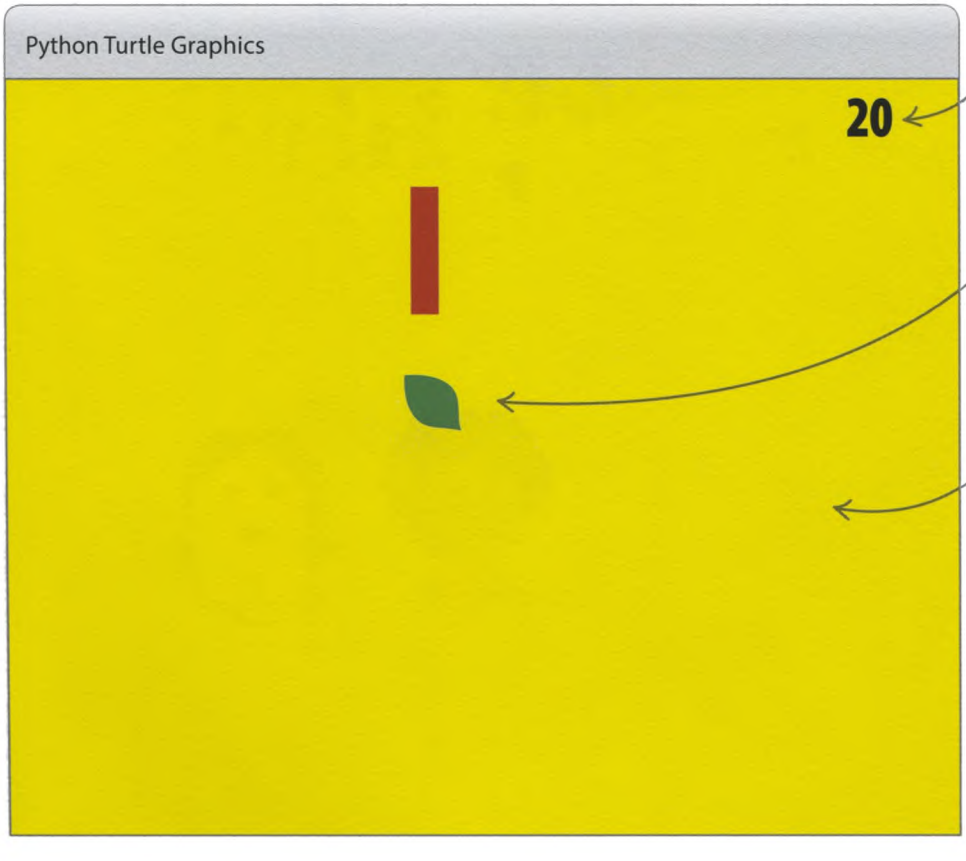
Игры на Python

«Гусеница»

Если ты так долго сидишь за компьютером, что успел проголодаться, знай: ты не одинок. Встречай звезду этого проекта — прожорливую гусеницу! С помощью модуля turtle ты научишься анимировать ее и других персонажей, а также управлять ими с клавиатуры.

Что происходит

С помощью клавиш-стрелок игрок управляет гусеницей. Каждый съеденный ею лист приносит очки, а также повышает уровень сложности игры, делая гусеницу длиннее и проворнее. Не выпускай ее за пределы графического окна, иначе игре конец!



Счет выводится в правом верхнем углу игрового окна.

Съеденный гусеницей лист исчезает, а затем в другом месте игрового поля появляется новый.

Чтобы начать игру, нужно кликнуть по окну и нажать ПРОБЕЛ.

◁ **Возрастание сложности**

Чем больше листьев съест гусеница, тем сложнее станет игра. По мере того как наша героиня растет и ускоряется, игроку тоже придется действовать энергичнее, иначе гусеница выйдет за пределы графического окна.

Как это работает

В этом проекте используются две основные черепашки: одна рисует гусеницу, другая — листья. Новые листья возникают в случайных местах графического окна. Обнаружив, что гусеница съела лист, программа увеличивает значения переменных, хранящих счет игры, скорость гусеницы и ее длину. Функция проверяет, вышла ли гусеница за пределы окна, и если это так, то игра прекращается.

◁ **Блок-схема программы «Гусеница»**
Гусеницу по экрану перемещает код бесконечного цикла. При каждом его проходе картинка немного смещается вперед. Быстрое многократное прохождение цикла создает ощущение, что гусеница движется.

Первые шаги

Код для этой веселой игры на удивление прост. Тебе предстоит задать параметры черепашек, написать код основного цикла и настроить интерфейс для управления гусеницей.

1 Приступим
Открой IDLE, создай новый файл и сохрани его как caterpillar.py («гусеница»).

2 Загрузи модули
Добавь в программу команды, импортирующие модули **turtle** и **random**. Третья строчка кода задает фоновый цвет игрового окна.

Задает желтый цвет фона.

```
import random
import turtle as t

t.bgcolor('yellow')
```

3 Создай черепашку для гусеницы

Нам нужна черепашка, которая нарисует гусеницу. Введи следующий код — он создаст гусеницу, задав ее цвет, форму и скорость. Функция **caterpillar.penup()** («гусеница.поднять перо») отключает перо, чтобы черепашка могла перемещаться по экрану, не оставляя следа.

```
caterpillar = t.Turtle()
caterpillar.shape('square')
caterpillar.color('red')
caterpillar.speed(0)
caterpillar.penup()
caterpillar.hideturtle()
```

Создает черепашку, рисующую гусеницу.

Нам не нужно, чтобы гусеница перемещалась по экрану до того, как игра начнется.

Эта команда прячет черепашку.

4 Создай черепашку для листьев

Введи эти строчки после кода, добавленного на шаге 3, — они создают вторую черепашку, рисующую листья. Список из 6 пар координат задает форму листа. Запомнив эту форму, черепашка сможет рисовать такие же листья снова и снова. Вызов функции **hideturtle()** («спрятать черепашку») делает черепашку невидимой.

Эта черепашка будет рисовать листья.

Координаты точек контура листа.

```
leaf = t.Turtle()
leaf_shape = ((0, 0), (14, 2), (18, 6), (20, 20), \
              (6, 18), (2, 14))
t.register_shape('leaf', leaf_shape)
leaf.shape('leaf')
leaf.color('green')
leaf.penup()
leaf.hideturtle()
leaf.speed(0)
```

С помощью обратного слеша можно разбить длинную строку кода надвое.

Сообщает черепашке форму листа.

5 Добавь текст

Создай еще двух черепашек — для вывода текста. Одна будет сообщать, что для начала игры следует нажать ПРОБЕЛ, а другая — показывать в углу экрана счет. Добавь эти строчки после кода, введенного на шаге 4.

Позже нужно будет проверить, началась ли игра.

```
game_started = False
text_turtle = t.Turtle()
text_turtle.write('Нажмите ПРОБЕЛ, чтобы начать игру',\
                  align='center', font=('Arial', 16, 'bold'))
text_turtle.hideturtle()

score_turtle = t.Turtle()
score_turtle.hideturtle()
score_turtle.speed(0)
```

Отображает на экране текст.

Прячет черепашку, оставляя текст.

Создает черепашку, выводящую на экран счет.

Чтобы счет обновлялся, черепашка должна стоять на месте.

Основной цикл

Итак, черепашки настроены и готовы к бою. Пришла пора написать код, который вдохнет в игру жизнь.

6 **Команда-заглушка**

Написание кода функции можно отложить на потом с помощью команды **pass** («пропустить»). Введи после настройки черепашек эти функции с заглушками. Позже ты добавишь в них код.

```
def outside_window():
    pass


def game_over():
    pass


def display_score(current_score):
    pass


def place_leaf():
    pass
```

Чтобы скорее получить работающую основу программы (скелет), можно использовать заглушки для функций, код которых еще не написан.

СОВЕТЫ ЭКСПЕРТА

Заглушка pass

Если пока неясно, что будет внутри функции, в ее теле (которое нельзя оставлять пустым) помещают команду-заглушку **pass**, чтобы вернуться к ней позже и добавить в функцию рабочий код.

7 **Подготовься к запуску**

После четырех функций с заглушками добавь функцию **start_game()** («начать игру»). Она создает ряд переменных, которые задают параметры экрана и анимации перед началом игры.

```
def start_game():
    global game_started
    if game_started:
        return
    game_started = True

    score = 0
    text_turtle.clear()

    caterpillar_speed = 2
    caterpillar_length = 3
    caterpillar.shapesize(1, caterpillar_length, 1)
    caterpillar.showturtle()
    display_score(score)
    place_leaf()
```

Если игра уже началась, команда return завершит вызов функции start_game(), когда та запустится повторно.

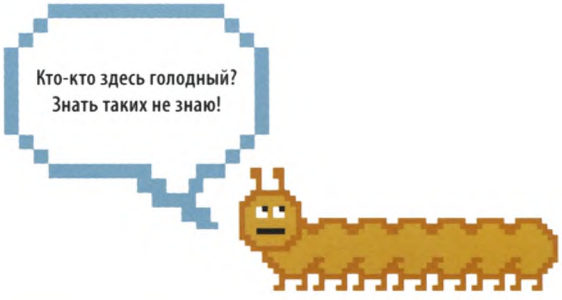
Очищает экран от сообщений.

Черепашка создает гусеницу нужной формы.

Создает на экране первый лист.

8 Движущий цикл

Код основного цикла сдвигает гусеницу вперед и проверяет 2 вещи. Во-первых, доползла ли она до листа. Если это так, счет увеличивается, на экране появляется новый лист, гусеница становится длиннее и проворнее. Во-вторых, не вышла ли гусеница за пределы окна. Если это так, игра заканчивается. Добавь основной цикл после кода, введенного на шаге 7.




```
place_leaf()

while True:
    caterpillar.forward(caterpillar_speed)
    if caterpillar.distance(leaf) < 20:
        place_leaf()
        caterpillar_length = caterpillar_length + 1
        caterpillar.shapesize(1, caterpillar_length, 1)
        caterpillar_speed = caterpillar_speed + 1
        score = score + 10
        display_score(score)
    if outside_window():
        game_over()
        break
```

Если между гусеницей и листом меньше 20 пикселей, она его съедает.

Лист съеден, значит, нужно создать новый.

Увеличивают длину гусеницы.

9 Свяжи клавишу с началом игры

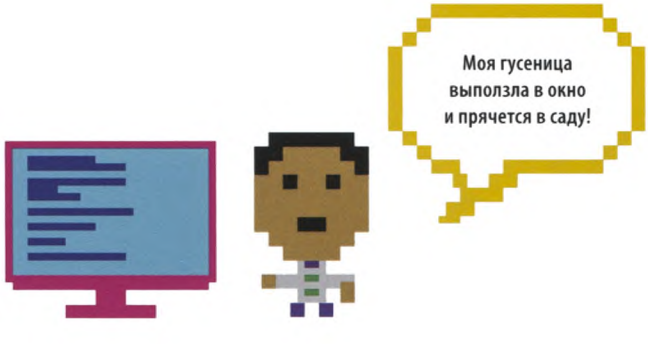
Введи эти строчки кода после только что созданной функции **start_game()**. Функция **onkey()** («на клавишу») связывает клавишу ПРОБЕЛ с вызовом **start_game()**, то есть с началом игры. Функция **listen()** («слушать») позволяет программе отслеживать нажатия клавиш.

```
t.onkey(start_game, 'space')
t.listen()
t.mainloop()
```

Игра начинается после нажатия клавиши ПРОБЕЛ.

10 Проверь работу программы

Запусти программу. После нажатия на ПРОБЕЛ должна появиться гусеница, которая тут же уползет за границы окна. Если программа не работает, тщательно проверь код на наличие ошибок.



Заполняем пробелы

Настало время добавить вместо заглушек рабочий код. Сделав это для всех функций, запусти игру и посмотри, что изменилось.

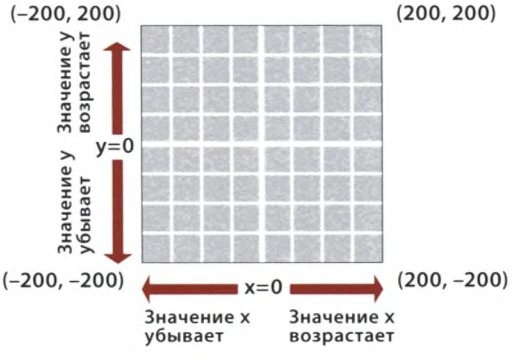
11 Охраняй границы

Добавь этот код в функцию **outside_window()** («за пределами окна») вместо команды **pass**. Он находит координаты каждой границы окна и узнаёт текущую позицию гусеницы. Сравнивая эти координаты, код выясняет, выползла ли гусеница за пределы экрана. Запусти программу: теперь гусеница, дойдя до края, должна остановиться.

```
def outside_window():
    left_wall = -t.window_width() / 2
    right_wall = t.window_width() / 2
    top_wall = t.window_height() / 2
    bottom_wall = -t.window_height() / 2
    (x, y) = caterpillar.pos()
    outside = \
            x < left_wall or \
            x > right_wall or \
            y < bottom_wall or \
            y > top_wall
    return outside
```

Эта функция возвращает 2 значения (так называемый кортеж).

Если любое из вышестоящих условий вернет True, в переменную outside попадет True.



◁ Как это работает

Центр окна имеет координаты (0, 0). Поскольку ширина окна равна 400 пикселям, правая граница отстоит от центра на половину ширины, то есть на 200 пикселей. Координаты левой границы код получает так же, только вычитая половину ширины из нуля (0 – 200 = –200). Координаты верхней и нижней границ он находит подобным образом.

12 Конец игры!

Пускай, если гусеница выползет за край, на экране появится сообщение о завершении игры. Добавь в функцию **game_over()** («конец игры») код, который спрячет гусеницу и лист, а затем выведет надпись «Конец игры!».


```
def game_over():
    caterpillar.color('yellow')
    leaf.color('yellow')
    t.penup()
    t.hideturtle()
    t.write('Конец игры!', align='center', font=('Arial', 30, 'normal'))
```

Выравнивает текст по центру.

13 Покажи счет

Функция **display_score()** («отобразить счет») дает черепашке **score_turtle** указание написать в углу экрана новый счет. Эта функция будет вызываться всякий раз, когда гусеница доползет до листа.

```
def display_score(current_score):
    score_turtle.clear()
    score_turtle.penup()
    x = (t.window_width() / 2) - 50
    y = (t.window_height() / 2) - 50
    score_turtle.setpos(x, y)
    score_turtle.write(str(current_score), align='right', \
                       font=('Arial', 40, 'bold'))
```

50 пикселей от правого края окна.

50 пикселей от верхнего края окна.

14 Создай новый лист

Когда гусеница съедает лист, вызывается функция **place_leaf()** («разместить лист»), перемещающая лист в новую позицию. Она выбирает два случайных числа от –200 до 200, которые становятся *x*- и *y*-координатами нового листа.

```
def place_leaf():
    leaf.ht()
    leaf.setx(random.randint(-200, 200))
    leaf.sety(random.randint(-200, 200))
    leaf.st()
```

ht — это сокращенное название функции hideturtle().

Выбирает случайные координаты листа.

st — это сокращенное название функции showturtle().

15 Разверни гусеницу

Для управления гусеницей с помощью клавиатуры понадобятся 4 новые функции — по одной для каждого направления движения. Создай их после функции **start_game()**. Чтобы игра не выглядела слишком простой, нужно научить гусеницу делать повороты только на 90°. Каждая функция сначала проверяет, куда ползет гусеница, и с помощью функции **setheading()** меняет ее курс, если это возможно.

```
            game_over()
            break

def move_up():
    if caterpillar.heading() == 0 or caterpillar.heading() == 180:
        caterpillar.setheading(90)

def move_down():
    if caterpillar.heading() == 0 or caterpillar.heading() == 180:
        caterpillar.setheading(270)

def move_left():
    if caterpillar.heading() == 90 or caterpillar.heading() == 270:
        caterpillar.setheading(180)

def move_right():
    if caterpillar.heading() == 90 or caterpillar.heading() == 270:
        caterpillar.setheading(0)
```

Проверяет, ползет ли гусеница вправо или влево.

Аргумент 270 разворачивает гусеницу вниз.

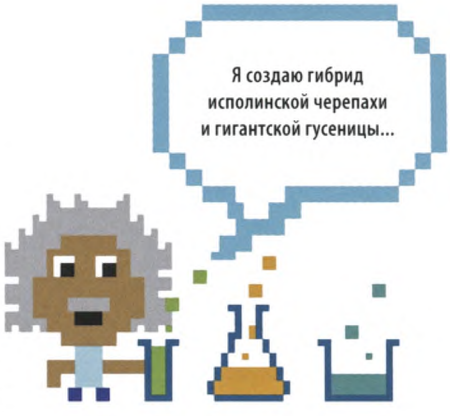
16 **Свяжи повороты с клавишами**
И наконец, с помощью **onkey()** свяжи функции поворотов с клавишами-стрелками. Введи эти строчки после вызова **onkey()**, добавленного на шаге 9. Ну все, игра готова. Интересно, сколько очков ты наберешь?

```
t.onkey(start_game, 'space')
t.onkey(move_up, 'Up')
t.onkey(move_right, 'Right')
t.onkey(move_down, 'Down')
t.onkey(move_left, 'Left')
t.listen()
```

Вызывает функцию move_up() («двигаться вверх») при нажатии клавиши ВВЕРХ.

Что бы изменить?

Теперь, когда игра готова, усовершенствовать ее будет несложно. Давай добавим вторую гусеницу — помощника или соперника!



Игра для двух игроков

Если создать вторую гусеницу, управляемую другими клавишами, ты сможешь позвать в игру друга, и тогда гусеницы съедят в 2 раза больше листьев!

1 **Создай новую гусеницу**
Для начала нужно создать вторую гусеницу. Введи эти строчки в начале программы, после кода, создающего первую гусеницу.

```
caterpillar2 = t.Turtle()
caterpillar2.color('blue')
caterpillar2.shape('square')
caterpillar2.penup()
caterpillar2.speed(0)
caterpillar2.hideturtle()
```

2 **Добавь аргумент**
Чтобы использовать функцию **outside_window()** для обеих гусениц, добавь в нее аргумент, указывающий, какую именно гусеницу нужно проверить.

```
def outside_window(caterpillar):
```

3 **Спрячь вторую гусеницу**
Сейчас функция **game_over()** прячет первую гусеницу. Добавь команду, которая спрячет и вторую гусеницу тоже.

```
def game_over():
    caterpillar.color('yellow')
    caterpillar2.color('yellow')
    leaf.color('yellow')
```

4 Измени код функции

В главную функцию игры, **start_game()**, нужно добавить код для создания второй гусеницы. Сначала задай ее контур и разверни в противоположную первой гусенице сторону. Затем добавь этот код в цикл **while**, чтобы гусеница поползла. А чтобы она могла есть листья, сделай в условии **if** еще одну проверку. Также нужна строчка кода, которая увеличивает длину гусеницы. И наконец, измени вызов функции **outside_window()**.


```
score = 0
text_turtle.clear()

caterpillar_speed = 2
caterpillar_length = 3
caterpillar.shapesize(1, caterpillar_length, 1)
caterpillar.showturtle()
caterpillar2.shapesize(1, caterpillar_length, 1)
caterpillar2.setheading(180)
caterpillar2.showturtle()
display_score(score)
place_leaf()

while True:
    caterpillar.forward(caterpillar_speed)
    caterpillar2.forward(caterpillar_speed)
    if caterpillar.distance(leaf) < 20 or leaf.distance(caterpillar2) < 20:
        place_leaf()
        caterpillar_length = caterpillar_length + 1
        caterpillar.shapesize(1, caterpillar_length, 1)
        caterpillar2.shapesize(1, caterpillar_length, 1)
        caterpillar_speed = caterpillar_speed + 1
        score = score + 10
        display_score(score)
    if outside_window(caterpillar) or outside_window(caterpillar2):
        game_over()
```

Задает начальную длину второй гусеницы.

Разворачивает вторую гусеницу влево.

При каждом проходе цикла вторая гусеница сдвигается вперед.

Проверяет, съела ли вторая гусеница лист.

Делает вторую гусеницу длиннее.

Вышла ли вторая гусеница за пределы окна?

5 Управляй!

Назначь клавиши управления второй гусеницей. В нашем случае это будут w (вверх), a (влево), s (вниз) и d (вправо), но ты можешь выбрать любой другой вариант. Также тебе понадобятся 4 новые функции и 4 вызова **onkey()**, которые свяжут эти функции с клавишами.

```
def caterpillar2_move_up():
    if caterpillar2.heading() == 0 or caterpillar2.heading() == 180:
        caterpillar2.setheading(90)

def caterpillar2_move_down():
    if caterpillar2.heading() == 0 or caterpillar2.heading() == 180:
        caterpillar2.setheading(270)

def caterpillar2_move_left():
    if caterpillar2.heading() == 90 or caterpillar2.heading() == 270:
        caterpillar2.setheading(180)

def caterpillar2_move_right():
    if caterpillar2.heading() == 90 or caterpillar2.heading() == 270:
        caterpillar2.setheading(0)

t.onkey(caterpillar2_move_up, 'w')
t.onkey(caterpillar2_move_right, 'd')
t.onkey(caterpillar2_move_down, 's')
t.onkey(caterpillar2_move_left, 'a')
```

△ Игра-состязание

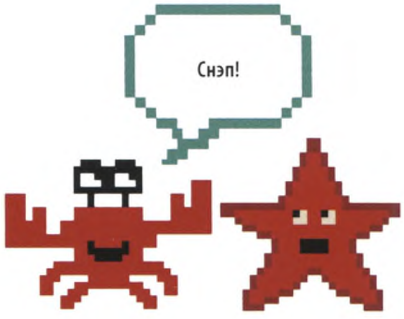
Подумай, как можно доработать программу, чтобы она сохраняла счет каждого игрока, а в конце объявляла победителя. Подсказка: для хранения счета второго игрока понадобится еще одна переменная. Когда чья-нибудь гусеница съест лист, нужно увеличить счет только этого игрока. И наконец, когда игра завершится, необходимо сравнить очки и узнать, кто победил.

▽ Легче или сложнее

Меняя в коде цикла значения, на которые увеличивается длина гусеницы (+1) и ее скорость (+2), можно настроить уровень сложности игры. Чем больше эти значения, тем сложнее играть, и наоборот.

«Снэп»

Сразись с друзьями в цифровую версию карточной игры «Снэп»! Эта динамичная игра для двоих требует внимательности и молниеносной реакции. Вместо карт на экране будут появляться цветные фигуры.

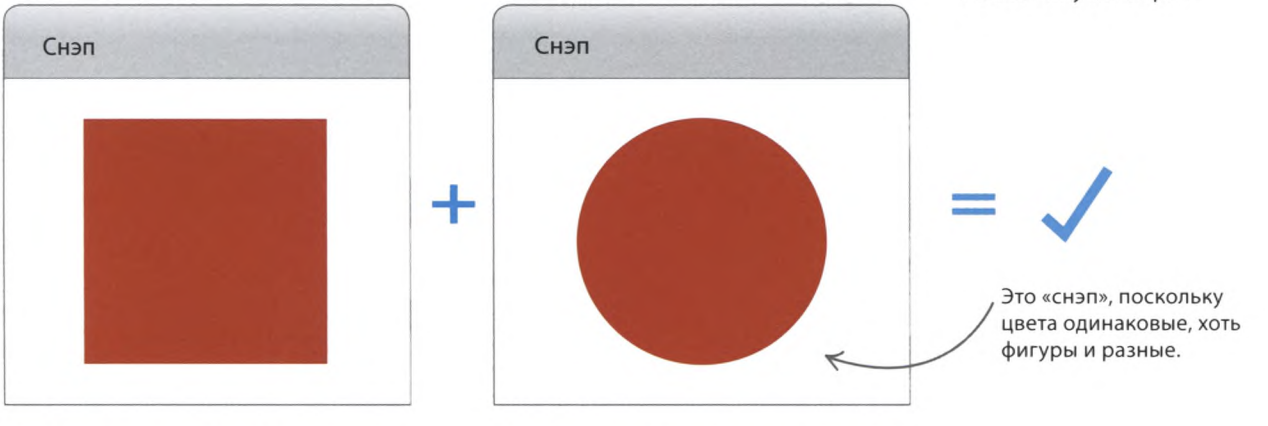


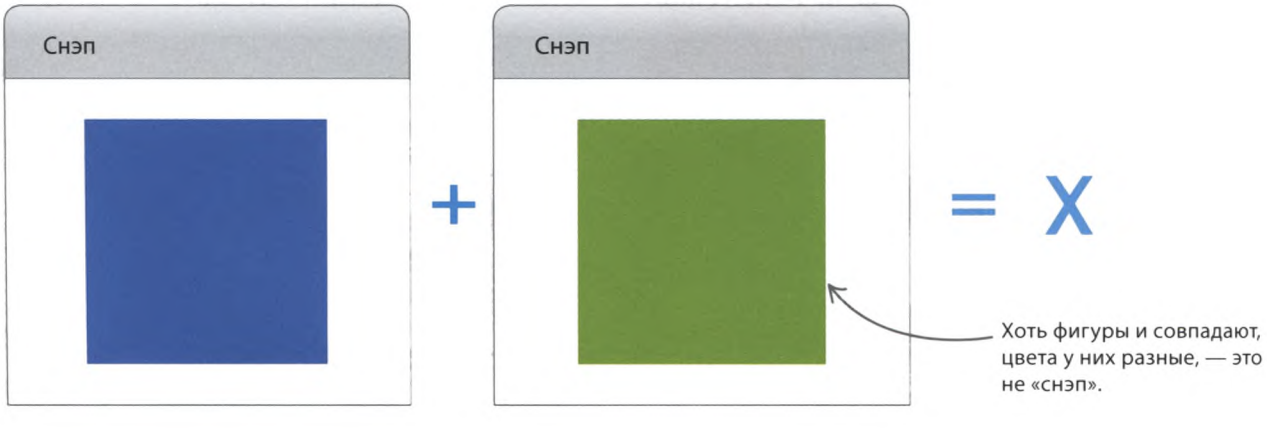
▽ **Запуск игры**

Игра работает в окне Tkinter. При запуске программы оно может открыться позади других окон, так что его не будет видно. В этом случае подвинь окна IDLE, однако не мешкай, ведь фигуры начинают появляться на экране через 3 секунды после запуска игры.

Что происходит

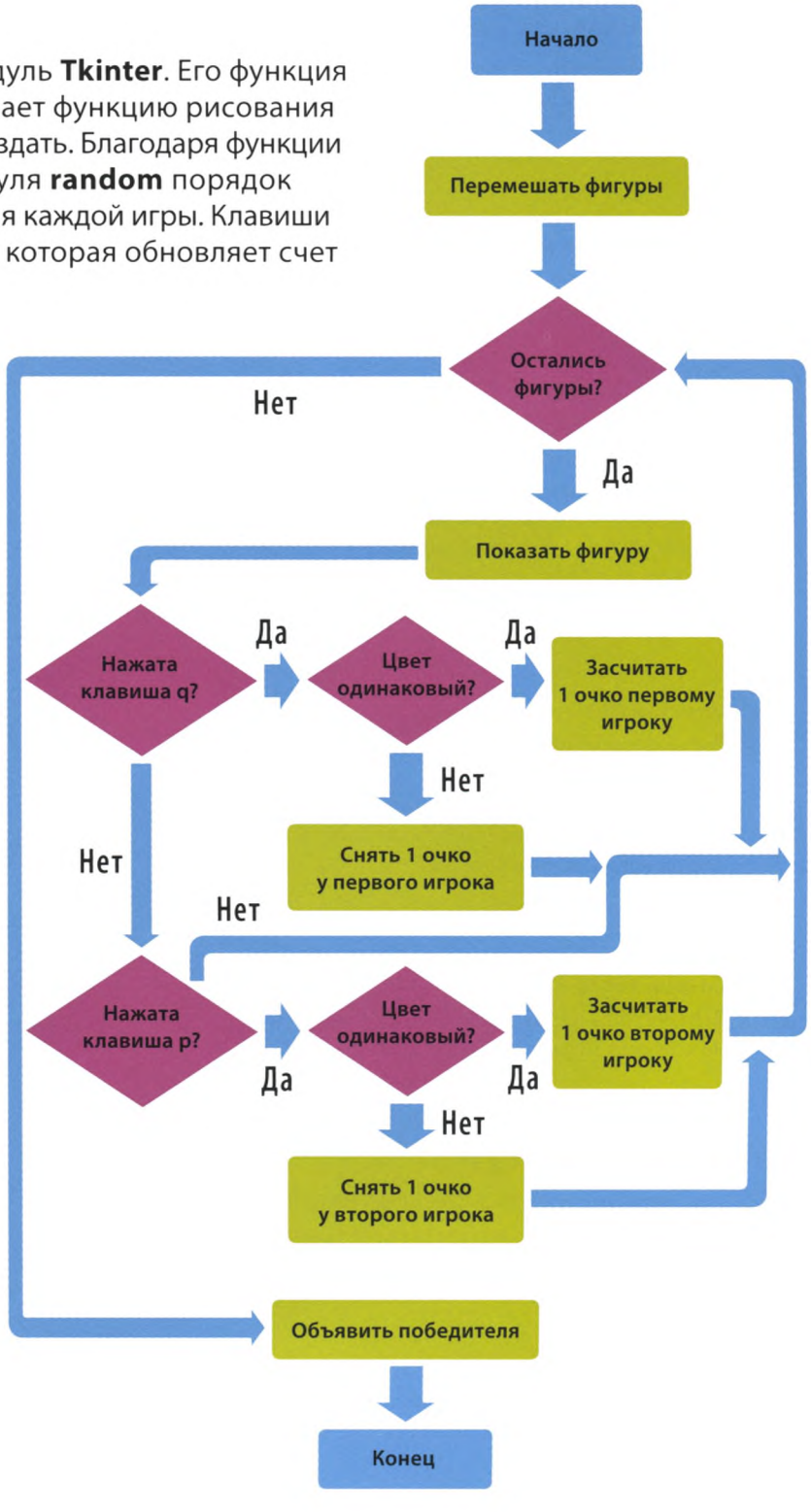
На экране сменяют друг друга фигуры разных цветов: черные, красные, зеленые, синие. Когда один цвет выпадает второй раз подряд, нужно нажать кнопку «снэп» (для первого игрока это клавиша q, а для второго p) и получить очко. Если нажать кнопку не вовремя, очко снимается. Набравший большее количество очков побеждает.





Как это работает

В этом проекте используется модуль **Tkinter**. Его функция **mainloop()** периодически вызывает функцию рисования фигур, которую тебе предстоит создать. Благодаря функции **shuffle()** («перемешать») из модуля **random** порядок появления фигур будет разным для каждой игры. Клавиши q и p связаны с функцией **snap()**, которая обновляет счет игрока, нажавшего клавишу.



▷ **Блок-схема программы «Снэп»**

Игра продолжается, пока все фигуры не будут показаны. Игроки нажимают клавиши, и программа обрабатывает эти нажатия. Когда фигур не остается, объявляется победитель, и игра заканчивается.

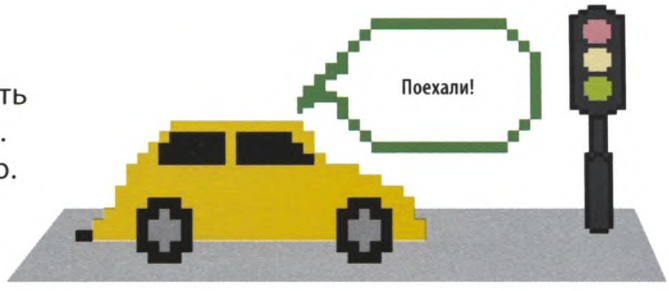
СОВЕТЫ ЭКСПЕРТА

Функция sleep()

Если ты попросишь компьютерную программу показать фигуру и тут же ее убрать, это произойдет так быстро, что пользователь ничего не заметит. Чтобы такого не произошло, в игре «Снэп» используется функция **sleep()** («спать») из модуля **time**, которая выдерживает паузу в заданное количество секунд. Так, команда time.sleep(1) приостановит выполнение программы на 1 секунду.

Приступим

Сначала нужно загрузить модули и настроить графический интерфейс пользователя (GUI). Затем — создать холст для рисования фигур.



1 Создай новый файл

Открой IDLE и создай новый файл, сохранив его как snap.py.

2 Загрузи модули

Загрузи модули **random**, **time** и части модуля **Tkinter**. **Time** позволит делать паузы, чтобы игрок успел прочитать надписи «СНЭП!» и «МИМО!» перед показом следующей фигуры. Режим HIDDEN поможет спрятать фигуры, а режим NORMAL — показывать их по одной (иначе будут видны все фигуры сразу).

Модуль random позволит «перемешать» фигуры.

```
import random
import time
from tkinter import Tk, Canvas, HIDDEN, NORMAL
```

Tkinter нужен для создания GUI.

3 Настрой GUI

Введи следующий код, в котором создается окно Tkinter с заголовком «Снэп». Проверь работу программы, запустив ее. Окно игры может открыться позади других окон — тогда сдвинь их или сверни.

```
from tkinter import Tk, Canvas, HIDDEN, NORMAL

root = Tk()
root.title('Снэп')
```

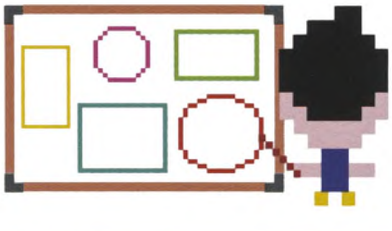
4 Создай холст

Эта строчка создает холст — пространство, на котором будут возникать фигуры.

```
root.title('Снэп')
c = Canvas(root, width=400, height=400)
```

Рисование фигур

Создать фигуры тебе поможет функция виджета **Canvas**. Она нарисует круги, квадраты и прямоугольники четырех разных цветов.



5 Создай список для фигур

Для хранения фигур понадобится список. Добавь эту строчку кода в самый конец программы.

```
c = Canvas(root, width=400, height=400)

shapes = []
```

6 Создай круги

Нарисовать круг может функция **create_oval()** («создать овал») виджета **Canvas**. Добавь в конец программы этот код: он создаст 4 круга (черный, красный, зеленый и синий) и добавит их в список **shapes** («фигуры»).

Не забудь сохранить свою работу.

```
shapes = []

circle = c.create_oval(35, 20, 365, 350, outline='black', fill='black', state=HIDDEN)
shapes.append(circle)
circle = c.create_oval(35, 20, 365, 350, outline='red', fill='red', state=HIDDEN)
shapes.append(circle)
circle = c.create_oval(35, 20, 365, 350, outline='green', fill='green', state=HIDDEN)
shapes.append(circle)
circle = c.create_oval(35, 20, 365, 350, outline='blue', fill='blue', state=HIDDEN)
shapes.append(circle)
c.pack()
```

Устанавливает режим HIDDEN, чтобы фигура была невидимой: ее время еще не пришло.

Координаты левого верхнего угла прямоугольника — *см.* «Советы эксперта».

Координаты правого нижнего угла прямоугольника — *см.* «Советы эксперта».

Располагает фигуры на холсте. Без этой команды они не будут отображаться.

Цвет круга определяется цветами контура и заливки.

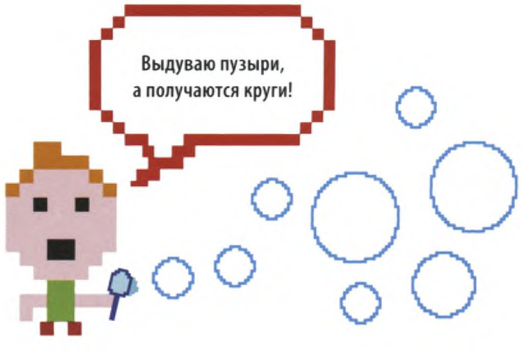
СОВЕТЫ ЭКСПЕРТА

Овалы и окружности

Функция **create_oval()** рисует овал, который как будто вписан в невидимый прямоугольник. Значения в скобках — это координаты двух противоположных углов прямоугольника. Чем дальше они друг от друга, тем больше овал. Если стороны прямоугольника равны, функция нарисует окружность.

7 Покажи круги

Запусти программу. Видишь фигуры? Они скрыты, поскольку ты создал их в режиме HIDDEN. Измени режим одного из кругов на NORMAL и снова запусти программу. Теперь этот круг должен появиться на экране. Имей в виду: если задать режим NORMAL для нескольких кругов, они отобразятся все сразу, один поверх другого.



8 Добавь прямоугольники

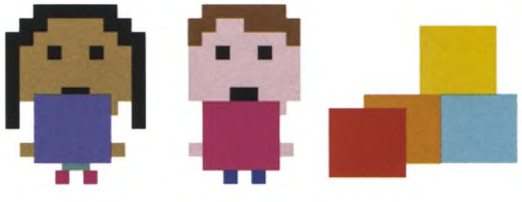
Создай четыре прямоугольника при помощи функции виджета **Canvas create_rectangle()** («создать прямоугольник»). Введи этот код после команд рисования кругов, но перед вызовом **c.pack()**. Чтобы много не печатать, набери первые две строки, скопируй их, трижды вставь и поменяй цвета.

Не забудь сохранить свою работу.

```
shapes.append(circle)

rectangle = c.create_rectangle(35, 100, 365, 270, outline='black', fill='black', state=HIDDEN)
shapes.append(rectangle)
rectangle = c.create_rectangle(35, 100, 365, 270, outline='red', fill='red', state=HIDDEN)
shapes.append(rectangle)
rectangle = c.create_rectangle(35, 100, 365, 270, outline='green', fill='green', state=HIDDEN)
shapes.append(rectangle)
rectangle = c.create_rectangle(35, 100, 365, 270, outline='blue', fill='blue', state=HIDDEN)
shapes.append(rectangle)
c.pack()
```

9 Добавь квадраты

Теперь создай квадраты. Их можно рисовать той же функцией, что и прямоугольники, сделав все стороны одинаковыми. Введи эти строчки после кода, добавленного на шаге 8, и перед вызовом **c.pack()**.



```
shapes.append(rectangle)

square = c.create_rectangle(35, 20, 365, 350, outline='black', fill='black', state=HIDDEN)
shapes.append(square)
square = c.create_rectangle(35, 20, 365, 350, outline='red', fill='red', state=HIDDEN)
shapes.append(square)
square = c.create_rectangle(35, 20, 365, 350, outline='green', fill='green', state=HIDDEN)
shapes.append(square)
square = c.create_rectangle(35, 20, 365, 350, outline='blue', fill='blue', state=HIDDEN)
shapes.append(square)
c.pack()
```

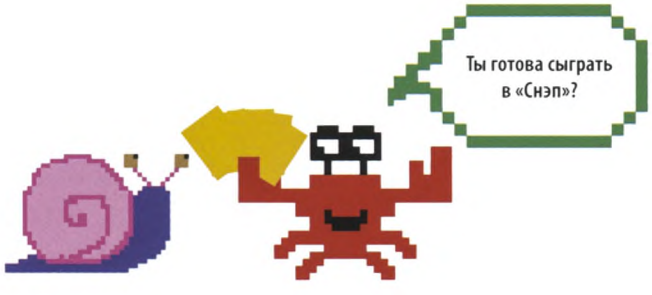
10 **Перемешай фигуры**

Чтобы фигуры появлялись вразнобой, их нужно «перемешать» — как колоду карт. Для этого подойдет функция **shuffle()** из модуля **random**. Добавь эту строчку кода после вызова **c.pack()**.

```
random.shuffle(shapes)
```

Подготовка к игре

Следующий этап — это создание нескольких переменных и написание вспомогательных частей кода. Однако до запуска самой игры дело пока не дойдет.



СОВЕТЫ ЭКСПЕРТА

Ничто имеет значение

Программисты часто присваивают переменным начальное значение 0, например переменной, хранящей счет. Но что делать, если в переменной должна быть строка, а не число? Тогда после «равно» можно поставить пару кавычек, означающих пустую строку. Однако есть переменные, для которых не подойдет ни 0, ни пустая строка. Используй для них ключевое слово **None** («Ничто»), как в коде ниже.

11 **Настрой переменные**

Тебе понадобятся переменные для отслеживания разных игровых параметров: формы фигуры, предыдущего и текущего цветов, а также счета обоих игроков.

```
random.shuffle(shapes)

shape = None
previous_color = ''
current_color = ''
player1_score = 0
player2_score = 0
```

У переменной shape пока нет значения.

В переменных, хранящих названия предыдущего и текущего цветов фигур, сейчас пустые строки.

Переменные для счета первого и второго игрока вначале содержат 0.

12 **Добавь паузу**

Добавь трехсекундную паузу перед показом первой фигуры. За это время игрок сможет найти окно Tkinter, если оно скрыто под другими окнами. Функцию **next_shape()** («следующая фигура») ты создашь позже, на шагах 16 и 17.

```
player2_score = 0

root.after(3000, next_shape)
```

Перед тем как показать фигуру, программа ждет 3 секунды (3000 миллисекунд).

13 **Свяжи клавиши и «снэп»**
Добавь в программу эти две строчки. Функция **bind()** связывает нажатие клавиш q и p с функцией **snap()**, которую ты создашь позже.

```
root.after(3000, next_shape)
c.bind('q', snap)
c.bind('p', snap)
```

14 **Заставь холст слушаться клавиш**
Вызов функции **focus_set()** («задать фокус») заставляет холст слушаться нажатия клавиш. Введи эту строчку после вызовов **bind()**.

```
c.bind('q', snap)
c.bind('p', snap)
c.focus_set()
```

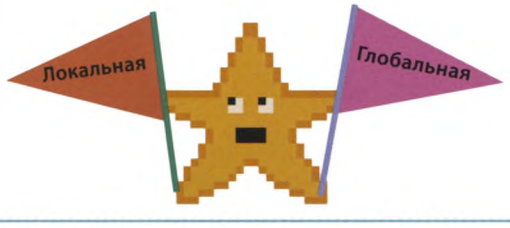
15 **Запусти основной цикл**
Добавь эту строчку в конец программы. Когда ты создашь функции **next_shape()** и **snap()**, основной цикл начнет менять фигуры и обрабатывать нажатия клавиш.

```
c.focus_set()


root.mainloop()
```

СОВЕТЫ ЭКСПЕРТА

Локальные и глобальные

Переменные бывают либо локальными, либо глобальными. Локальная переменная существует только внутри отдельной функции, и больше нигде ее использовать нельзя. А переменная, созданная в основном коде программы, называется глобальной, и к ней можно обращаться из какой угодно части кода. Чтобы работать с глобальной переменной в коде функции, при ее объявлении обязательно нужно добавить в начале ключевое слово **global**. Это и происходит на шаге 16.



Создание функций

И наконец, нужно создать 2 функции: одну для показа фигур, а другую для обработки совпадений. Введи код функций сразу после команд **import**.

16 **Создай функцию показа фигур**
Функция **next_shape()** показывает цветные фигуры одну за другой, как при раздаче карт. Для начала введи код, который вызывает функцию, помечает переменные как глобальные (*см.* «Советы эксперта») и обновляет переменную **previous_color** («предыдущий цвет»).

Благодаря ключевому слову global изменившееся значение переменной будет видно во всей программе.

```
def next_shape():
    global shape
    global previous_color
    global current_color

    previous_color = current_color
```

Сохраняет значение переменной previous_color в переменной current_color перед показом следующей фигуры.

17 Допиши функцию

Теперь напиши оставшийся код функции **next_shape()**. Чтобы показать фигуру, нужно изменить режим HIDDEN на NORMAL. Следующий код делает это с помощью функции виджета **Canvas itemconfigure()** («конфигурация элемента»). Еще одна функция, **itemcget()** («получить цвет элемента»), используется для обновления текущего цвета в **current_color** («текущий цвет»), который понадобится при проверке на совпадение.

```
previous_color = current_color

c.delete(shape)
if len(shapes) > 0:
    shape = shapes.pop()
    c.itemconfigure(shape, state=NORMAL)
    current_color = c.itemcget(shape, 'fill')
    root.after(1000, next_shape)
else:
    c.unbind('q')
    c.unbind('p')
    if player1_score > player2_score:
        c.create_text(200, 200, text='Победил игрок 1')
    elif player2_score > player1_score:
        c.create_text(200, 200, text='Победил игрок 2')
    else:
        c.create_text(200, 200, text='Ничья')
    c.pack()
```

Удаляет текущую фигуру, чтобы на нее не наслоилась новая.

Получает из списка следующую фигуру, если фигуры еще остались.

Делает новую фигуру видимой.

Сохраняет цвет новой фигуры в current_color.

Ждет 1 секунду перед тем, как показать следующую фигуру.

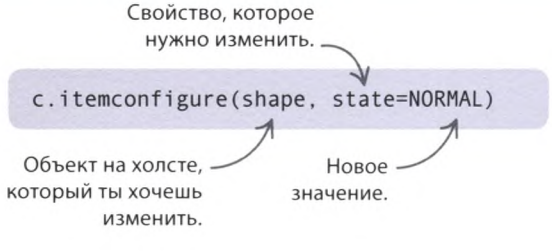
Отключают обработку нажатия клавиш, если игра закончилась.

Этот код объявляет победителя или ничью.

СОВЕТЫ ЭКСПЕРТА

Настройка объектов холста

Вид отображаемых на холсте рисунков можно настраивать с помощью функции **itemconfigure()**. В нашей программе она делает скрытые фигуры видимыми, а также позволяет менять их цвет и другие свойства. Для этого нужно указать в скобках элемент модуля **Canvas**, который нужно настроить, а затем через запятую ввести свойство и его новое значение.



18 Это «снэп»?

Сейчас игре не хватает только функции **snap()**, которая проверяет, какой игрок нажал клавишу и действительно ли цвета текущей и предыдущей фигур совпадают. После этого функция обновляет счет и выводит на экран соответствующее сообщение. Добавь этот код после функции **next_shape()**.

Не забудь сохранить свою работу.

```
def snap(event):
    global shape
    global player1_score
    global player2_score
    valid = False

    c.delete(shape)

    if previous_color == current_color:
        valid = True

    if valid:
        if event.char == 'q':
            player1_score = player1_score + 1
        else:
            player2_score = player2_score + 1
        shape = c.create_text(200, 200, text='СНЭП! Вы получаете 1 очко!')
    else:
        if event.char == 'q':
            player1_score = player1_score - 1
        else:
            player2_score = player2_score - 1
        shape = c.create_text(200, 200, text='МИМО! Вы теряете 1 очко!')
    c.pack()
    root.update_idletasks()
    time.sleep(1)
```

Пометим переменные как глобальные, чтобы потом их можно было менять из функции.

Проверяют, совпадает ли цвет предыдущей фигуры с цветом текущей.

Если цвета совпадают, этот код проверяет, какой игрок нажал кнопку, и зачисляет ему 1 очко.

Выводит сообщение о том, что игрок заработал очко.

Иначе (else) снимает со счета игрока 1 очко.

Выводит сообщение о том, что игрок промахнулся и потерял очко.

Ускоряет обновление GUI, чтобы без задержек отобразить нужное сообщение.

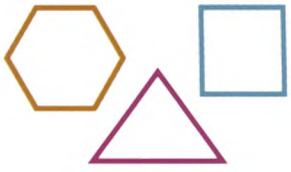
Ждет 1 секунду, чтобы игрок успел прочитать сообщение.

19 Проверь работу программы

Запусти программу и убедись, что она работает. Помни: чтобы окно Tkinter реагировало на нажатия клавиш, по нему нужно кликнуть.

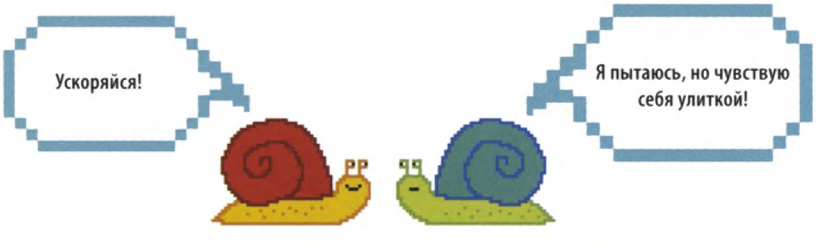
Что бы изменить?

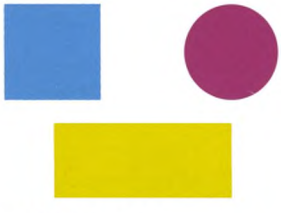
Помимо кругов, квадратов и прямоугольников Tkinter может отображать самые разные фигуры любых цветов. Это дарит нам массу возможностей для доработки программы. Кроме того, игру стоит защитить от жульничества!



△ **Цветной контур**
Сравнивая цвета фигур, программа учитывает лишь цвет заливки (свойство fill), но не контура (outline). Ты можешь создавать фигуры с контуром разных цветов, и это никак не повлияет на ход игры.

▽ **Ускорь игру**
Чтобы сделать игру сложнее, постепенно уменьшай паузу между показом фигур. Подсказка: храни это число миллисекунд в переменной. Сначала присвой ей значение 1000, а затем вычитай по 25 миллисекунд. Так ты попробуешь разные значения и найдешь те, которые тебе по душе.





△ **Больше цветов**
Сейчас игра заканчивается быстро. Чтобы сделать ее продолжительнее, добавь побольше квадратов, прямоугольников и кругов разных цветов.

Новые типы фигур

Изменив аргументы функции **create_oval()**, можно получить овал вместо круга. Также **Tkinter** позволяет рисовать сегменты, линии и многоугольники. Испытай эти примеры кода и поиграй со значениями. Не забывай задавать режим HIDDEN, чтобы фигуры не появлялись раньше времени.

1 **Нарисуй дуги, секторы и сегменты**
Функция **create_arc()** («создать дугу») служит для рисования секторов, сегментов и дуг. Если аргумент **style** («стиль») не указан, на экране появится сектор. Добавь в третьей строчке программы стили CHORD («хорда») и ARC («дуга»), как показано ниже, а в список фигур — секторы, сегменты и дуги.

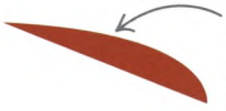
Загружает стили для create_arc().

```
from tkinter import Tk, Canvas, HIDDEN, NORMAL, CHORD, ARC
```

```
arc = c.create_arc(-235, 120, 365, 370, outline='black', \
                   fill='black', state=HIDDEN)
```

Поскольку стиль не указан, функция рисует сектор.

```
arc = c.create_arc(-235, 120, 365, 370, outline='red', \
                   fill='red', state=HIDDEN, style=CHORD)
```



Если задать стиль CHORD, получится сегмент.

```
arc = c.create_arc(-235, 120, 365, 370, outline='green', \
                   fill='green', state=HIDDEN, style=ARC)
```

Если задать стиль ARC, получится дуга.

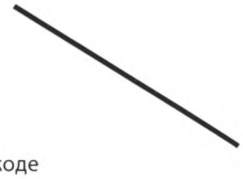
2 Нарисуй линии

Воспользуйся функцией **create_line()** («создать линию») и добавь в список фигур несколько линий.

```
line = c.create_line(35, 200, 365, 200, fill='blue', state=HIDDEN)
```



```
line = c.create_line(35, 20, 365, 350, fill='black', state=HIDDEN)
```



3 Нарисуй многоугольники

Пополнить твою коллекцию фигур многоугольниками поможет функция **create_polygon()** («создать многоугольник»), принимающая координаты всех углов фигуры.

3 пары чисел в этом коде соответствуют координатам вершин треугольника.

```
polygon = c.create_polygon(35, 200, 365, 200, 200, 35, \
                           outline='blue', fill='blue', state=HIDDEN)
```

Долой жульничество!

Если 2 игрока разом нажмут «снэп», оба получат по очку. Более того, можно накрутить счет, продолжая нажимать кнопки до появления следующей фигуры, ведь значения цветов останутся прежними. Вот доработка, которая позволит закрыть эту лазейку.

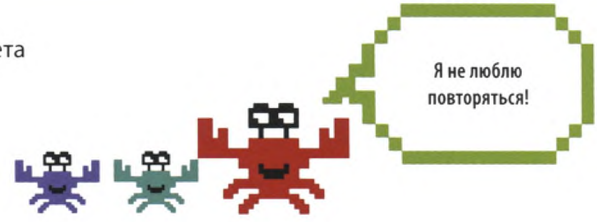
1 Сделай переменную глобальной

Объяви **previous_color** глобальной переменной в коде функции **snap()**, чтобы функция могла менять ее значение. Добавь эту строчку после других глобальных переменных.

```
    global previous_color
```

2 **Запрети повторные нажатия**

Добавь в функцию **snap()** следующую строчку. Если цвета фигур совпали, она запишет в **previous_color** пустую строку (''). Теперь игрок, повторно нажавший кнопку «снэп» на той же фигуре, потеряет очко, ведь пустая строка может быть значением **current_colour** только до начала отображения фигур.




```
    shape = c.create_text(200, 200, text='СНЭП! Вы получаете 1 очко!'
previous_color = ''
```

3 **Запрети досрочные нажатия**

Поскольку сначала значения переменных **previous_color** и **current_color** равны, можно сжульничать, нажимая кнопку до того, как будет показана первая фигура. Так что лучше присвоить этим переменным разные начальные значения — любые, например a и b.

```
previous_color = 'a'
current_color = 'b'
```

Разные строки в качестве начальных значений переменных не позволят счету увеличиваться при досрочных нажатиях клавиши «снэп».

4 **Измени сообщения**

Если оба игрока нажмут свои клавиши почти одновременно, будет неясно, кто из них должен получить очко. Чтобы это исправить, можно изменить сообщения, возникающие при нажатии кнопок.

Не забудь сохранить свою работу.

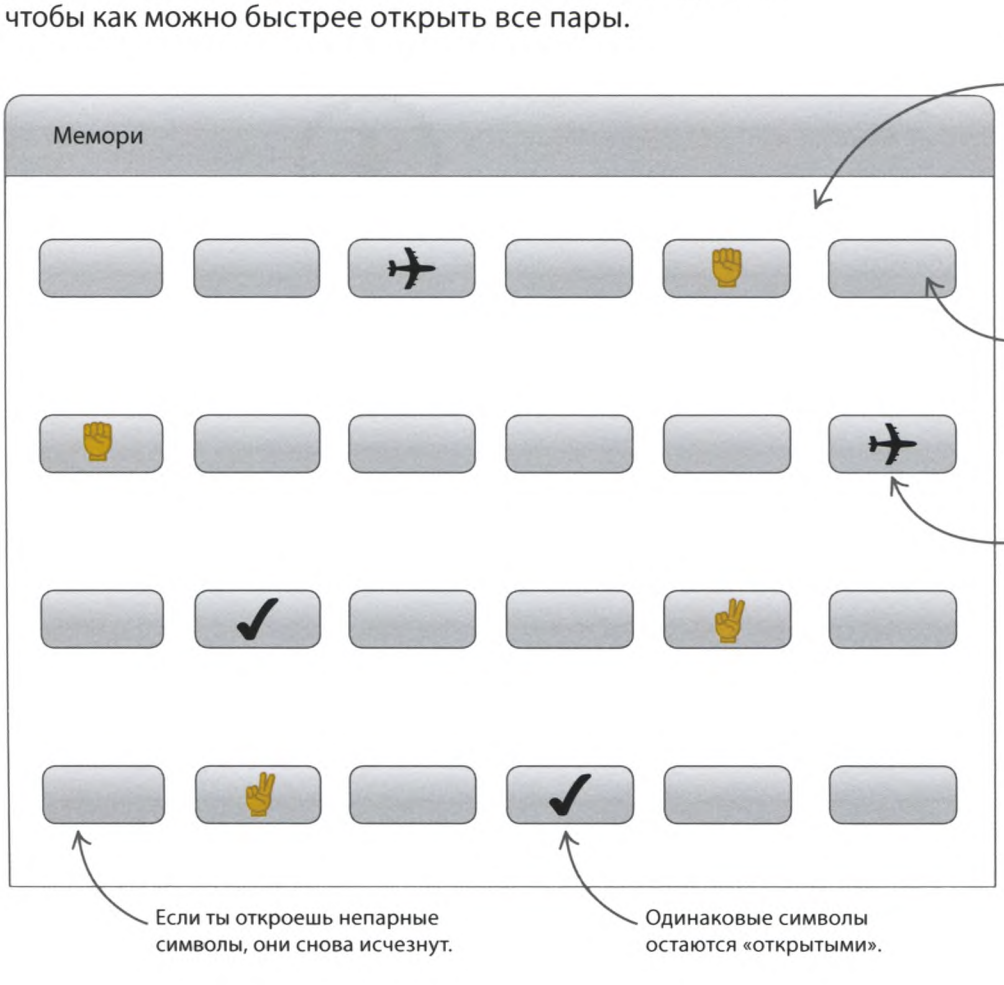
```
if valid:
    if event.char == 'q':
        player1_score = player1_score + 1
        shape = c.create_text(200, 200, text='СНЭП! Игрок 1 получает очко!')
    else:
        player2_score = player2_score + 1
        shape = c.create_text(200, 200, text='СНЭП! Игрок 2 получает очко!')
    previous_color = ''
else:
    if event.char == 'q':
        player1_score = player1_score - 1
        shape = c.create_text(200, 200, text='МИМО! Игрок 1 теряет очко!')
    else:
        player2_score = player2_score - 1
        shape = c.create_text(200, 200, text='МИМО! Игрок 2 теряет очко!')
```

«Мемори»

У тебя хорошая память? Узнать это поможет веселая игра на поиск парных символов. Поглядим, насколько быстро ты отыщешь все 12 пар!

Что происходит

Игрок видит на экране окно с кнопками. После клика по двум любым на их месте появляются символы. Если оба символа одинаковые, они остаются «открытыми», а если нет, программа снова их прячет. Постарайся запомнить, под какой кнопкой какой символ находится, чтобы как можно быстрее открыть все пары.



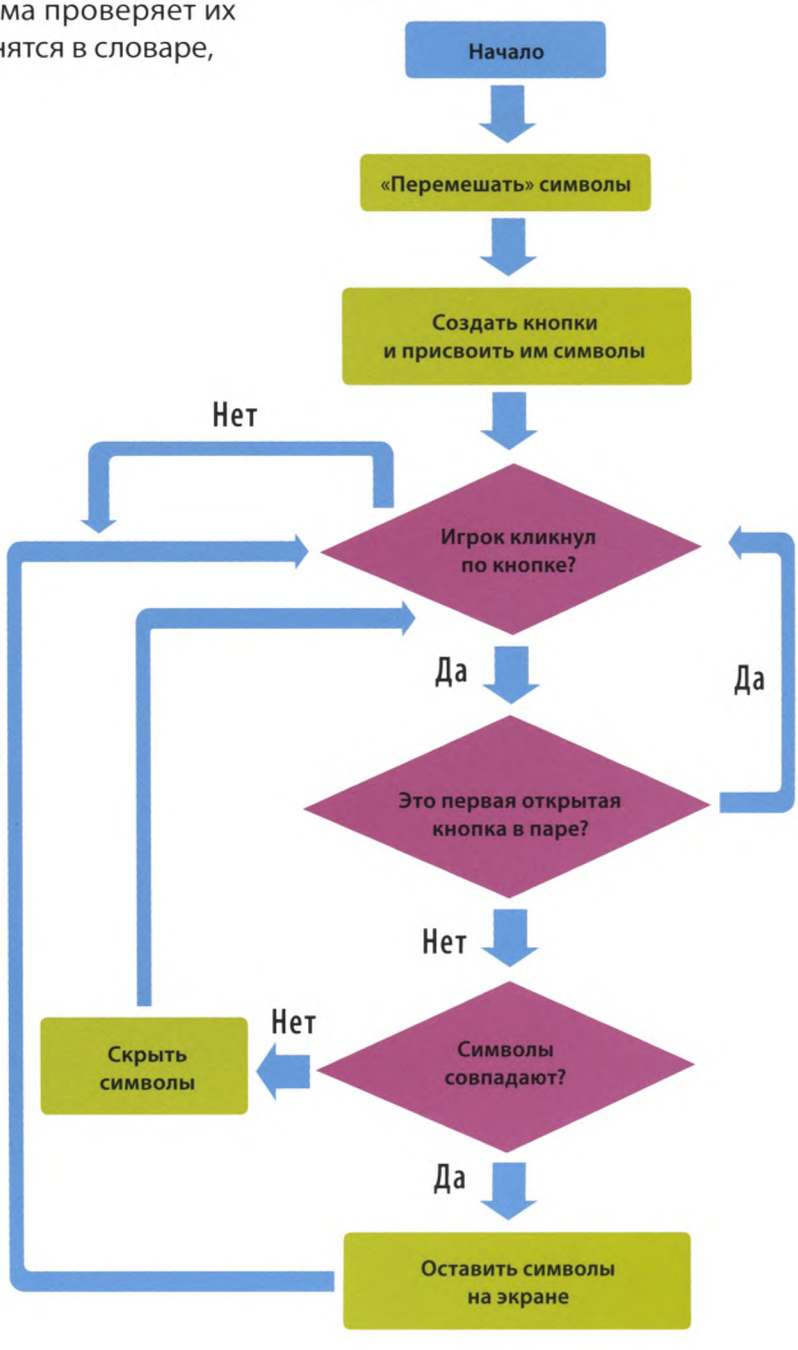
◁ **Окно GUI**

Окно и кнопки относятся к графическому интерфейсу пользователя (GUI). Создает их модуль Tkinter.

Как это работает

Для отображения кнопок на экране используется модуль **Tkinter**. Его функция **mainloop()** отслеживает клики по кнопкам и обрабатывает их с помощью особой лямбда-функции, которая показывает символ. Если «открыты» 2 символа, программа проверяет их на совпадение. При этом кнопки хранятся в словаре, а символы — в списке.

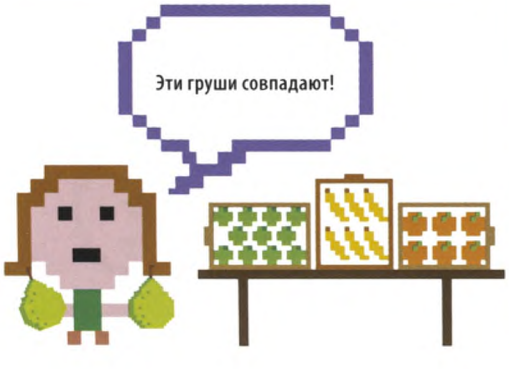
▽ **Блок-схема программы «Мемори»**
Программа «перемешивает» символы, создает элементы интерфейса, а затем ожидает кликов по кнопкам. Игра завершается, когда все пары символов «открыты».



СОВЕТЫ ЭКСПЕРТА

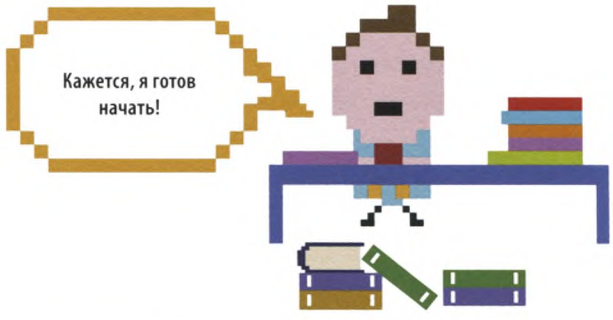
Лямбда-функция

Подобно **def**, ключевое слово **lambda** служит для создания функций. Такие лямбда-функции пишутся в одну строку, не имеют имени, а использовать их можно в любом месте программы. Например, функция lambda x: x*2 удваивает число. Можно сохранить ее в переменной, вот так: double = lambda x: x*2, а потом вызывать, вот так: double(x), где **x** — это любое число. Например, double(2) вернет 4. Лямбда-функции удобны при создании элементов GUI, чтобы при нажатии на кнопки вызывалась одна функция с разными аргументами, а не множество разных (в нашем примере — 24).



Приступим?

На первом этапе создания программы необходимо настроить GUI и связать кнопки с символами, которые сначала будут скрыты.



1 Создай новый файл

Открой IDLE, создай новый файл и сохрани его как memory.py («память»).

2 Загрузи модули

Введи эти строчки, чтобы загрузить необходимые модули. Модуль **random** поможет «перемешать» символы, **time** — сделать паузу, а **Tkinter** — создать игровой GUI.

Состояние DISABLED («заблокировано») делает кнопку неактивной, если для символа уже найдена пара.

```
import random
import time
from tkinter import Tk, Button, DISABLED
```

Виджет Button служит для создания кнопок в окне Tkinter.

3 Настрой GUI

Сразу после команд **import** введи код настройки GUI. Функция **root.resizable()** («основа. изменяемый») позволяет сделать размер окна неизменным. Это важно, поскольку, изменив размер окна, игрок собьет позиции кнопок, которые тебе скоро предстоит создать.

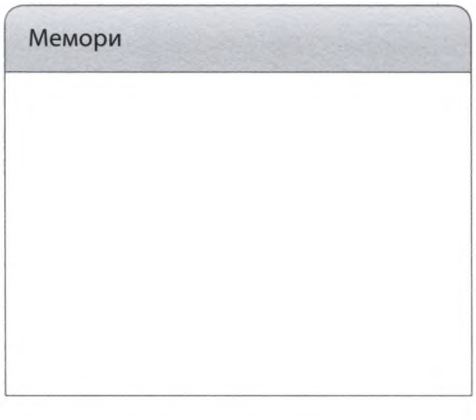
Создают окно Tkinter с нужным заголовком.

```
root = Tk()
root.title('Мемори')
root.resizable(width=False, height=False)
```

Делает размер окна неизменяемым.

4 Проверь работу программы

Запусти программу. Должно появиться пустое окно Tkinter с заголовком «Мемори». Если окна не видно, вероятно, его заслоняют другие окна.



Не забудь сохранить свою работу.

5 Создай переменные

Создай в конце программы необходимые переменные, включая словарь для хранения кнопок. После каждого клика программа должна определять, какой символ из пары открылся: первый или второй. Также необходимо сохранять координаты первой нажатой кнопки, чтобы затем сравнивать их с координатами второй.

```
root.resizable(width=False, height=False)

buttons = {}
first = True
previousX = 0
previousY = 0
```

Словарь для кнопок.

Эта переменная служит для проверки, первый ли это символ в паре.

Переменные для хранения координат первой из двух нажатых кнопок.

6 Добавь символы

Чтобы добавить в игру символы, введи следующий код. Как и в проекте «Девять жизней», здесь используются знаки Юникода. В игре 12 пар символов. Добавь этот код после переменных, созданных на шаге 5.


```
previousY = 0

button_symbols = {}
symbols = [u'\u2702', u'\u2702', u'\u2705', u'\u2705', u'\u2708', u'\u2708',
           u'\u2709', u'\u2709', u'\u270A', u'\u270A', u'\u270B', u'\u270B',
           u'\u270C', u'\u270C', u'\u270F', u'\u270F', u'\u2712', u'\u2712',
           u'\u2714', u'\u2714', u'\u2716', u'\u2716', u'\u2728', u'\u2728']
```

Словарь для хранения символа каждой кнопки.

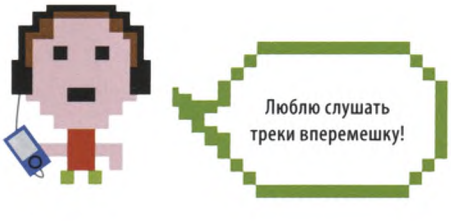
В этом списке хранятся 12 пар игровых символов.

7 «Перемешай» символы

Нельзя, чтобы символы всегда находились на одних и тех же позициях, иначе игрок запомнит их расположение и в следующий раз сможет открыть все пары с первой попытки. Поэтому перед началом каждой игры символы нужно «перемешать». Для этого введи следующую строчку после списка **symbols**.

Функция shuffle() из модуля random «перемешивает» символы.

```
random.shuffle(symbols)
```



Время кнопок

Теперь твоя задача — создать кнопки и разместить их в окне. А еще — функцию **show_symbol()** («показать символ»), которая обрабатывает клики по кнопкам.

8 **Размести кнопки**

Кнопки разместим в окне в 4 ряда по 6 штук. Расставить их помогут вложенные циклы. Внешний цикл (с переменной цикла **x**) пройдет 6 столбцов слева направо, а внутренний цикл (с переменной цикла **y**) — каждый столбец сверху вниз. В ходе работы циклов каждая кнопка получит пару координат *x* и *y*, задающих ее позицию в окне. Добавь этот код после вызова **shuffle()**.

СОВЕТЫ ЭКСПЕРТА

Виджет Button

В **Tkinter** есть виджет под названием **Button** («Кнопка»), который служит для создания кнопок. Ему можно передавать разные аргументы, из которых для нас важны **command**, **width** и **height**. Аргумент **command** («команда») указывает, какую функцию вызывать при клике (у нас это будет лямбда-функция), а **width** («ширина») и **height** («высота») задают ширину и высоту кнопки.

```
random.shuffle(symbols)

for x in range(6):
    for y in range(4):
        button = Button(command=lambda x=x, y=y: show_symbol(x, y), \
                        width=3, height=3)
        button.grid(column=x, row=y)
        buttons[x, y] = button
        button_symbols[x, y] = symbols.pop()
```

Это вложенные циклы.

Создает кнопку определенного размера и задает действие при клике по ней.

Размещает кнопку в окне.

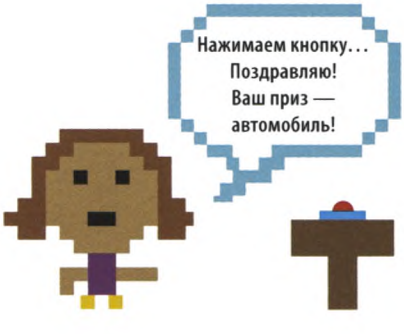
С помощью обратного слеша длинную строку кода можно разделить надвое.

Сохраняет кнопку в словаре buttons.

Привязывает символ к этой кнопке.

△ Как это работает

На каждом проходе цикла создается кнопка, а обработчиком клика по ней назначается лямбда-функция, которая сохраняет *x*- и *y*-координаты кнопки (строку и столбец). При клике значения этих **x** и **y** будут переданы еще не написанной функции **show_symbol()**, чтобы она знала, какой из символов открывать.



ЗАПОМНИ

Вложенные циклы

О вложенных циклах уже говорилось на с. 35. Циклы можно вкладывать друг в друга без ограничений. В этом коде внешний цикл повторяется 6 раз, и во время каждого его прохода внутренний цикл повторяется 4 раза. Значит, в сумме код внутреннего цикла выполняется 6 × 4 = 24 раза.

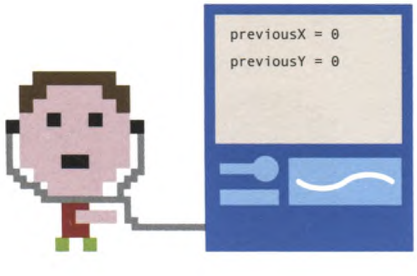
9 **Запусти основной цикл**

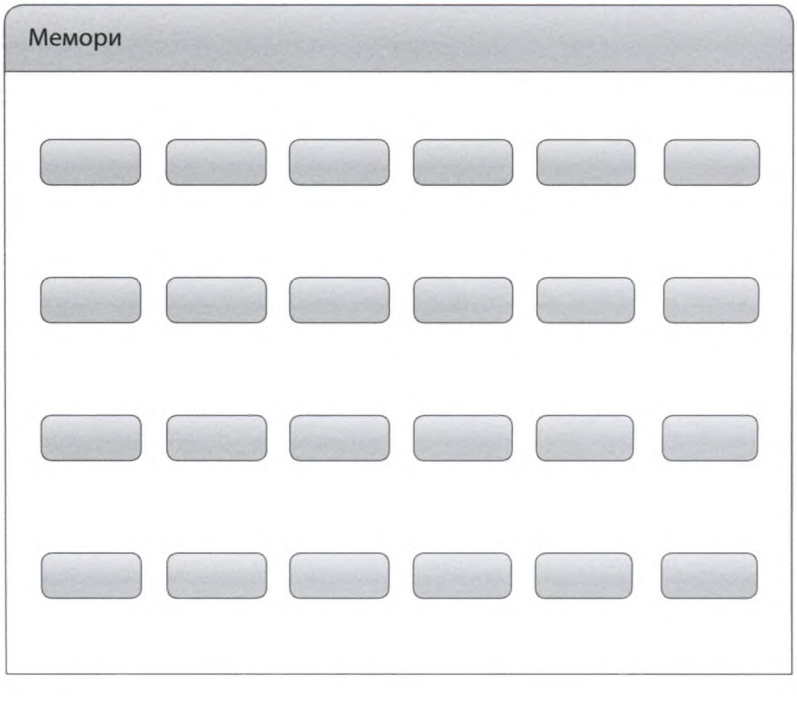
Пришло время запускать основной цикл **Tkinter**, который отобразит GUI и начнет отслеживать нажатия клавиш. Введи эту строчку после кода, добавленного на шаге 8.

```
        button_symbols[x, y] = symbols.pop()


root.mainloop()
```

10 **Проверь работу программы**

Снова запусти программу. Теперь в окне должны появиться 24 кнопки: 4 ряда по 6 штук. Если твое окно не похоже на это, внимательно проверь код на наличие ошибок.





11 Покажи символ

Настало время создать функцию, обрабатывающую клики по кнопкам. Сначала она покажет символ на экране, а дальше будет действовать в зависимости от того, первый он в паре или второй. Если первый, функция просто запомнит, какая кнопка была нажата, а если второй, то проверит «открытые» символы на совпадение. Несовпадающие символы исчезнут, а одинаковые останутся на экране, причем их кнопки станут неактивными.

```
from tkinter import Tk, Button, DISABLED

def show_symbol(x, y):
    global first
    global previousX, previousY
    buttons[x, y]['text'] = button_symbols[x, y]
    buttons[x, y].update_idletasks()

    if first:
        previousX = x
        previousY = y
        first = False
    elif previousX != x or previousY != y:
        if buttons[previousX, previousY]['text'] != buttons[x, y]['text']:
            time.sleep(0.5)
            buttons[previousX, previousY]['text'] = ''
            buttons[x, y]['text'] = ''
        else:
            buttons[previousX, previousY]['command'] = DISABLED
            buttons[x, y]['command'] = DISABLED
        first = True
```

По значениям *x* и *y* функция понимает, какая кнопка была нажата.

Объявляют глобальные переменные.

Отображают символ.

Если это первый символ из двух «открытых», программа запоминает x- и y-координаты нажатой кнопки.

Если открыт второй символ, условие не даст игроку смухлевать, повторно кликнув ту же кнопку!

Если символы не совпадают...

Если символы совпадают...

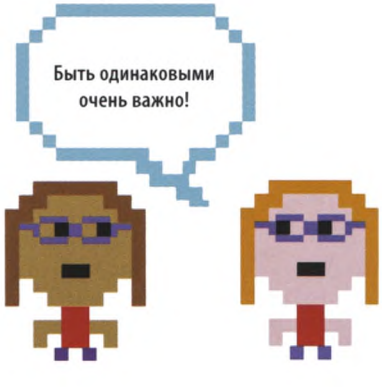
Делают 2 кнопки, соответствующие угаданным символам, неактивными, чтобы игрок не мог снова по ним кликнуть.

Делают паузу в полсекунды, чтобы игрок успел рассмотреть символы, а затем скрывают их.

Запоминает, что первое нажатие произошло.

△ Как это работает

Функция **show_symbol()** отображает картинку на экране, меняя текст кнопки с пустой строки на символ Юникода. Вызов функции **update_idletasks()** («обработать невыполненные задачи») позволяет сразу же показать эти изменения. Если открыт первый из двух символов, программа сохраняет координаты его кнопки. Если открыт второй символ в паре, проверяет, не выбрал ли игрок одну кнопку дважды. Если нет, проверяет символы на совпадение. Если символы разные, прячет их, меняя текст кнопок на пустые строки. Если же символы совпали, программа оставляет их на экране и делает нажатые кнопки неактивными.



Что бы изменить?

Если есть желание усовершенствовать эту игру, в конце можно вывести на экран количество ходов. Тогда игроки будут стремиться улучшить свой результат и соревноваться друг с другом. А еще можно усложнить игру, добавив больше символов.

Количество ходов

Сейчас у игрока нет возможности оценить свое мастерство или померяться силой с друзьями. Как сделать игру более соревновательной? Добавь в код переменные для подсчета ходов!

1 **Добавь новый виджет**

Чтобы показать в конце игры количество сделанных игроками ходов, нам понадобится виджет **messagebox** («окно сообщения»). Измени строку с загрузкой **Tkinter**, дописав слово messagebox после DISABLED.

```
from tkinter import Tk, Button, DISABLED, messagebox
```

2 **Создай новые переменные**

Также тебе понадобятся 2 переменные. Одна будет учитывать общее количество ходов, а другая — количество открытых пар. Пусть вначале значения обеих будут равны 0. Введи эти строчки после переменной **previousY**.

```
previousY = 0
moves = 0
pairs = 0
```

Игрок еще не сделал ходов и ничего не открыл, поэтому здесь нули.

3 **Пометь их как глобальные**

Переменные **moves** («ходы») и **pairs** («пары») — глобальные, их нужно будет менять из функции **show_symbol()**. Дай функции об этом знать, добавив в ее начало следующие строчки.

```
def show_symbol(x, y):
    global first
    global previousX, previousY
    global moves
    global pairs
```

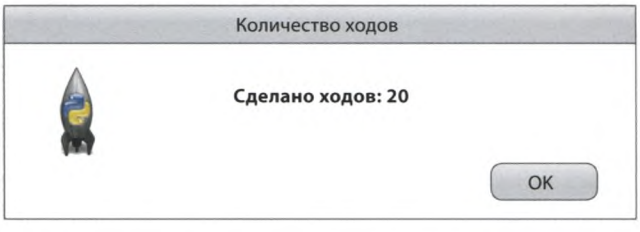
4 Сосчитай ходы

Один ход — это 2 клика по кнопкам (попытка открыть пару). Так что прибавлять 1 к значению переменной **moves** нужно при вызове **show_symbol()** либо для первого, либо для второго клика, но не для обоих. Пусть это будет первый клик. Измени код функции **show_symbol()** следующим образом.

```
    if first:
        previousX = x
        previousY = y
        first = False
        moves = moves + 1
```

5 Выведи сообщение

Теперь добавь ближе к концу **show_symbol()** такой код. Он будет считать открытые пары, а когда игра завершится, покажет окошко с количеством сделанных ходов и кнопкой OK. При клике OK будет вызвана функция **close_window()** («закрыть окно») — ее мы создадим дальше.




```
buttons[x, y]['command'] = DISABLED
pairs = pairs + 1
if pairs == len(buttons)/2:
    messagebox.showinfo('Количество ходов', 'Сделано ходов: ' +
                        str(moves), command=close_window)
```

Увеличивает количество открытых пар на 1.

Показывает окошко с количеством сделанных ходов.

Если все пары открыты, запустится тело if.

△ Как это работает

Поскольку в игре 12 пар символов, можно было просто написать **pairs == 12**, однако наша программа хитрее — она получает количество пар так: len(buttons)/2. Это позволит добавлять в игру новые кнопки без необходимости менять код функции **show_symbol()**.

6 Закрой окно

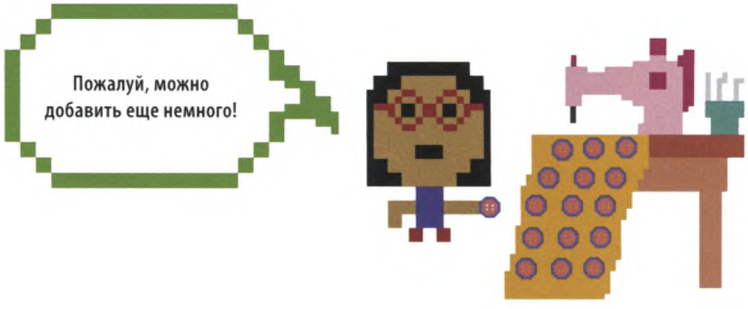
И наконец, нужно создать функцию **close_window()**, чтобы программа завершалась при клике кнопки OK в окошке «Количество ходов». Добавь этот код сразу после команд **import**.

```
def close_window(self):
    root.destroy()
```

Эта команда закрывает окно.

Еще больше кнопок

Чтобы проверить память игрока по полной программе, добавь в игру еще больше кнопок и символов.



1 Новые символы

Чтобы добавить новые пары символов, дополни код списка **symbols** следующим образом.


U+2733


U+2734


U+2744

```
symbols = [u'\u2702', u'\u2702', u'\u2705', u'\u2705', u'\u2708', u'\u2708',
           u'\u2709', u'\u2709', u'\u270A', u'\u270A', u'\u270B', u'\u270B',
           u'\u270C', u'\u270C', u'\u270F', u'\u270F', u'\u2712', u'\u2712',
           u'\u2714', u'\u2714', u'\u2716', u'\u2716', u'\u2728', u'\u2728',
           u'\u2733', u'\u2733', u'\u2734', u'\u2734', u'\u2744', u'\u2744']
```

Добавь в конец списка
3 новые пары символов.

2 Новые кнопки

Добавь в список символов новый ряд кнопок, указав в скобках функции **range ()** вложенного цикла число 5 вместо 4.

```
for x in range(6):
    for y in range(5):
```

Теперь в игре будет 5 рядов кнопок, а не 4.

3 Еще больше?

Сейчас в игре 30 кнопок. Если ты захочешь добавить еще, помни, что количество новых кнопок должно быть кратно 6, то есть длине полного ряда. Или можешь поэкспериментировать, изменив коды циклов и расставив кнопки по-другому.


U+2747


U+274C


U+274E


U+2753


U+2754


U+2755


U+2757


U+2764


U+2795


U+2796

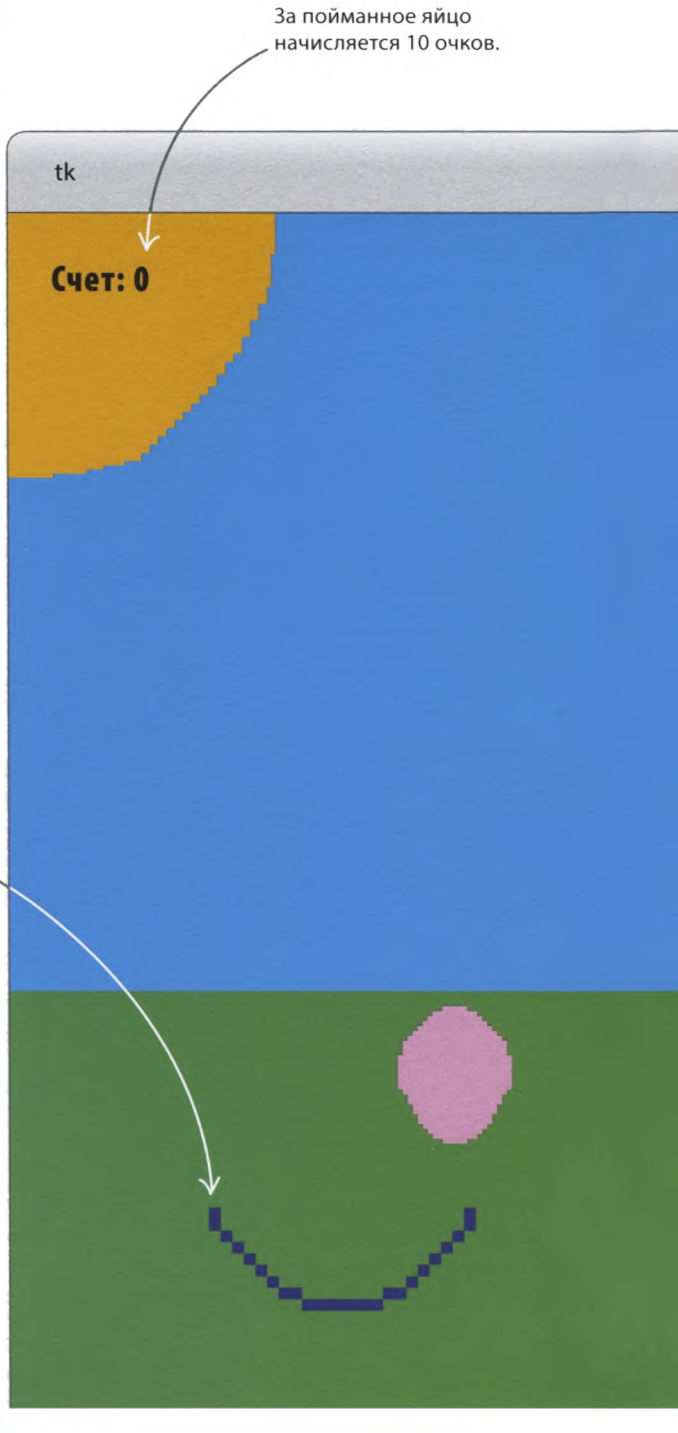

U+2797


U+27A1


U+27B0

«Яйцелов»

Это игра на внимательность и быстроту реакции. Собери волю в кулак и поймай столько яиц, сколько сможешь, заработав максимум очков. А потом устрой с друзьями турнир на звание лучшего яйцелова!



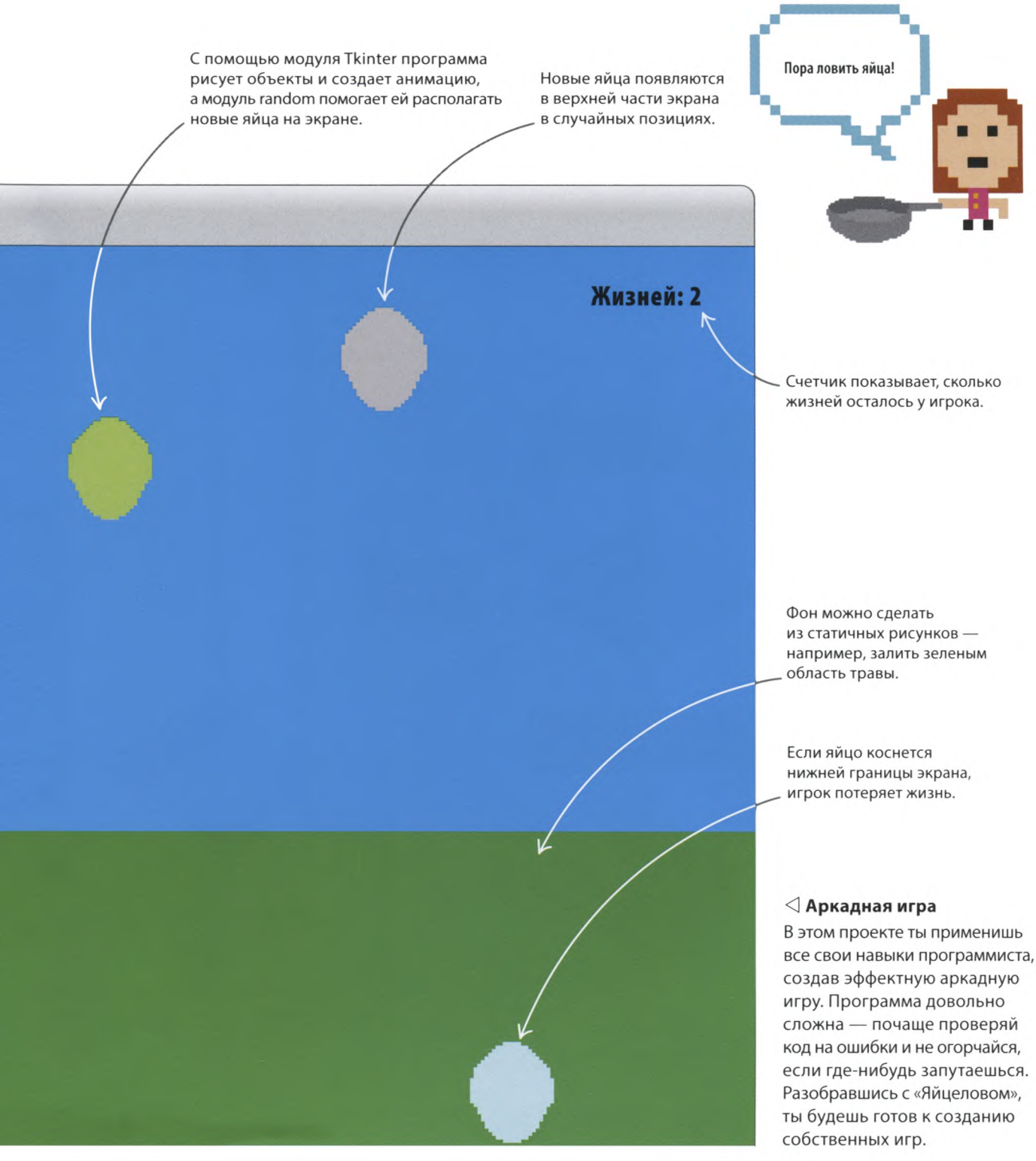
Что происходит

Перемещая корзину в нижней части экрана вправо-влево, старайся поймать падающие яйца до того, как они коснутся земли. Если яйцо окажется в корзине, ты заработаешь очки, а упустив его, потеряешь жизнь. Внимание: чем больше яиц ты поймаешь, тем чаще и быстрее будут падать новые. Если потеряешь все 3 жизни, игра закончится.

СОВЕТЫ ЭКСПЕРТА

Тайминг

Тайминг событий очень важен. Вначале яйца должны появляться раз в 4 секунды (иначе их будет слишком много на экране) и сдвигаться вниз каждые полсекунды. Если бы они падали быстрее, игра оказалась бы чересчур сложной. Программа определяет, поймал ли игрок яйцо, 10 раз в секунду. Делай она это реже, проверка давала бы осечки. По мере набора очков увеличивается количество новых яиц и их скорость падения.

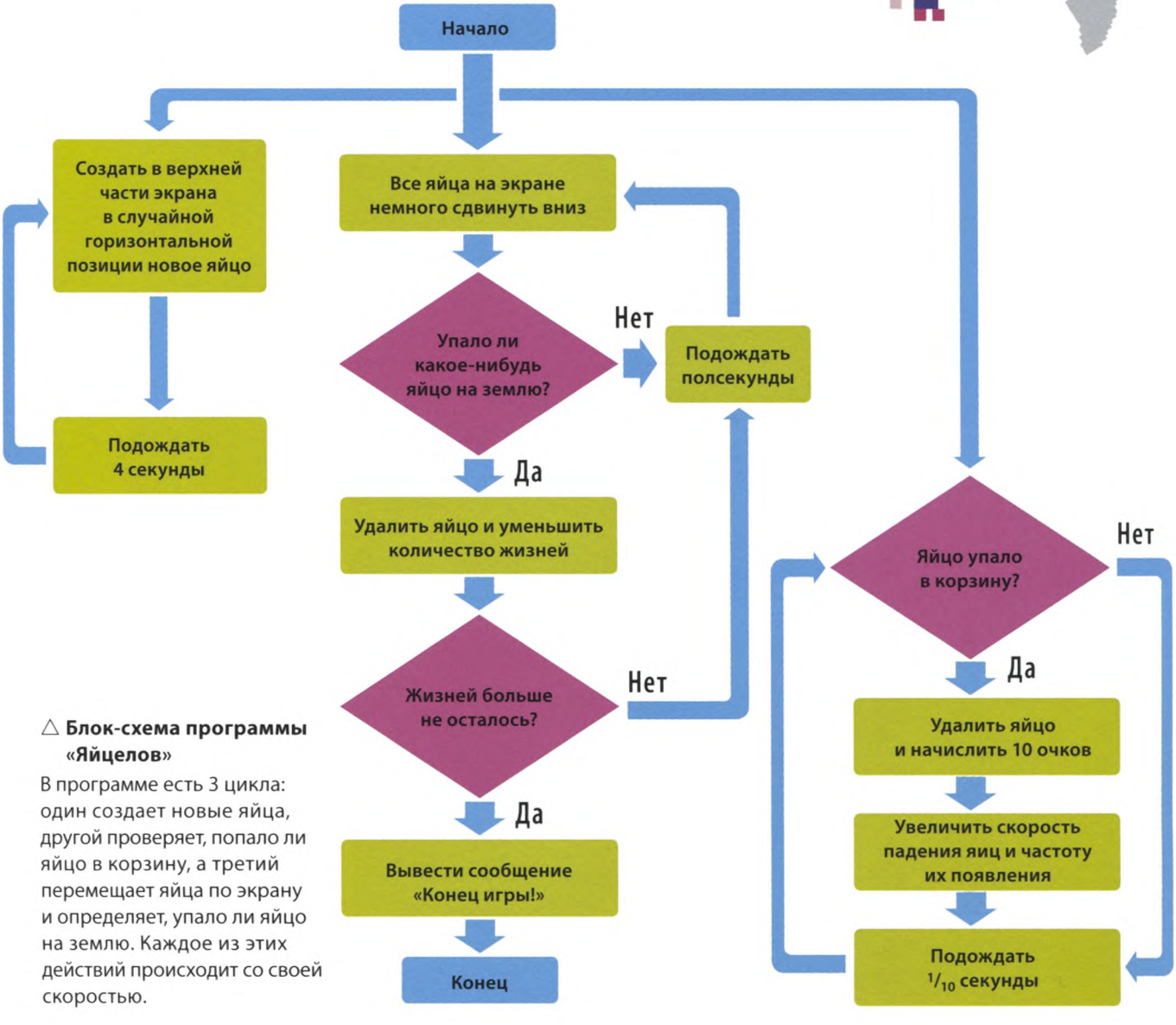


◁ **Аркадная игра**

В этом проекте ты применишь все свои навыки программиста, создав эффектную аркадную игру. Программа довольно сложна — почаще проверяй код на ошибки и не огорчайся, если где-нибудь запутаешься. Разобравшись с «Яйцеловом», ты будешь готов к созданию собственных игр.

Как это работает

Когда на экране появится фон, возникшие в верхней части экрана яйца начнут медленно смещаться вниз, что создаст иллюзию падения. При помощи циклов код снова и снова будет проверять координаты яиц, сопоставляя их с координатами нижней границы экрана и корзины. Если яйцо попало в корзину или разбилось, оно исчезает с экрана, а программа обновляет счет и число оставшихся жизней.



△ **Блок-схема программы «Яйцелов»**

В программе есть 3 цикла: один создает новые яйца, другой проверяет, попало ли яйцо в корзину, а третий перемещает яйца по экрану и определяет, упало ли яйцо на землю. Каждое из этих действий происходит со своей скоростью.

Подготовка

Давай сначала загрузим необходимые для этого проекта модули, а затем подготовим все, что требуется для создания основных функций.

1 Создай файл

Открой IDLE и создай новый файл, сохранив его как egg_catcher.py («яйцелов»).

2 Загрузи модули

В программе «Яйцелов» используются 3 модуля: **itertools** для перебирания цветов, **random** для создания яиц в случайных позициях и **Tkinter** для отрисовки графики и анимации. Добавь в программу эти 3 строчки кода , чтобы загрузить части этих модулей.

```
from itertools import cycle
from random import randrange
from tkinter import Canvas, Tk, messagebox, font
```

Импортирует только те части модулей, которые нам нужны.

3 Настрой холст

Введи этот код после команд **import**. В нем создаются переменные, хранящие ширину и высоту холста, а затем с их помощью создается и сам холст. Чтобы оживить фон, на холсте можно изобразить траву в виде зеленого прямоугольника и солнце в виде оранжевого овала.

```
from tkinter import Canvas, Tk, messagebox, font

canvas_width = 800
canvas_height = 400

root = Tk()
c = Canvas(root, width=canvas_width, height=canvas_height, \
           background='deep sky blue')
c.create_rectangle(-5, canvas_height - 100, canvas_width + 5, \
                   canvas_height + 5, fill='sea green', width=0)
c.create_oval(-80, -80, 120, 120, fill='orange', width=0)
c.pack()
```

Создает окно.

Холст будет синего цвета, размером 800 × 400 пикселей.

С помощью обратного слеша раздели длинную строку кода надвое.

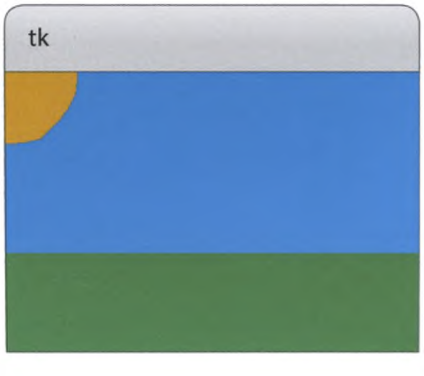
Создает траву.

Функция pack() нужна для отрисовки главного окна и его содержимого.

Создает солнце.

4 Оцени свой холст

Запусти код. На экране должен появиться пейзаж с зеленой травой, голубым небом и ярким солнцем. Если чувствуешь в себе силы, создай свою картинку с фигурами других цветов и размеров. В случае чего ты всегда сможешь вернуться к первоначальной версии кода.



5 Настрой параметры яиц

Создай переменные для хранения цвета, ширины и высоты яиц. Также нужны переменные для счета, скорости падения яиц и частоты появления новых. Изменением этих параметров управляет переменная **difficulty_factor** («коэффициент сложности»): чем меньше ее значение, тем сложнее игра.

Функция cycle() позволяет перебирать цвета по порядку.

```
c.pack()

color_cycle = cycle(['light blue', 'light green', 'light pink', 'light yellow', 'light cyan'])
egg_width = 45
egg_height = 55
egg_score = 10
egg_speed = 500
egg_interval = 4000
difficulty_factor = 0.95
```

Поймав яйцо, игрок получает 10 очков.

Новое яйцо возникает каждые 4000 миллисекунд (4 секунды).

Коэффициент сложности показывает, с какой частотой должны появляться яйца на экране и с какой скоростью падать (чем ближе к 1, тем проще играть).

6 Настрой параметры корзины

Создай несколько переменных для описания корзины: помимо цвета и размера понадобятся ее начальные координаты. Для расчета местоположения дуги, означающей корзину, используй размеры окна.

Не забудь сохранить свою работу.

```
difficulty_factor = 0.95

catcher_color = 'blue'
catcher_width = 100
catcher_height = 100
catcher_start_x = canvas_width / 2 - catcher_width / 2
catcher_start_y = canvas_height - catcher_height - 20
catcher_start_x2 = catcher_start_x + catcher_width
catcher_start_y2 = catcher_start_y + catcher_height

catcher = c.create_arc(catcher_start_x, catcher_start_y, \
                       catcher_start_x2, catcher_start_y2, start=200, extent=140, \
                       style='arc', outline=catcher_color, width=3)
```

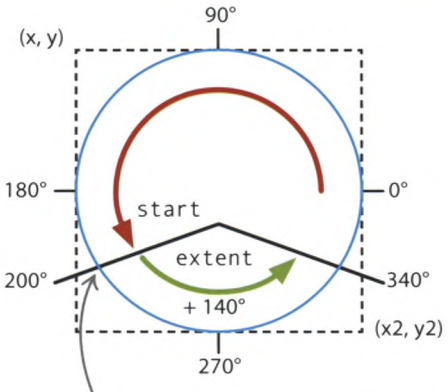
Диаметр окружности, которая используется для рисования дуги.

Эти строчки задают начальную позицию корзины (у нижнего края по центру холста).

Начинает рисовать дугу на отметке 200 градусов.

Заканчивает рисовать дугу на отметке 340 градусов.

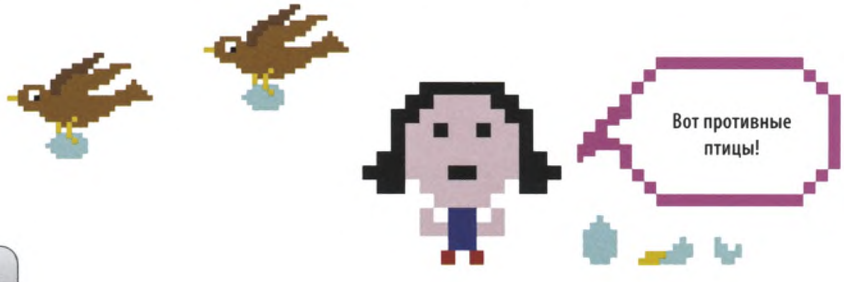
Рисует корзину.

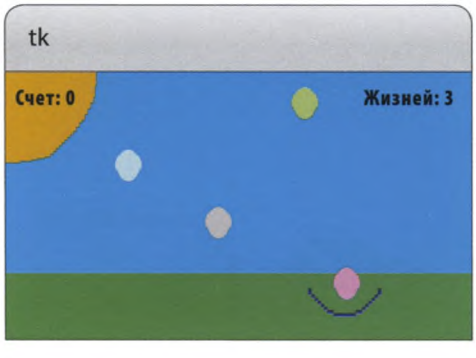


Полная окружность — это 360°. Наша дуга начинается чуть дальше половины окружности, на отметке 200°.

◁ Как это работает

Корзина представляет собой дугу, то есть часть окружности. **Tkinter** рисует окружности, вписывая их в невидимый прямоугольник. Значения переменных **catcher_start_x** и **catcher_start_y** задают координаты одного угла этого прямоугольника, а значения **catcher_start_x2** и **catcher_start_y2** — противоположного ему угла. У функции **create_arc()** есть 2 аргумента (число градусов), определяющих, какую часть окружности нужно взять для создания дуги: **start** («начало») — это место, в котором дуга начинается, а **extent** («протяженность») — протяженность дуги в градусах.





7 Добавь счетчики очков и жизней

Добавь этот код после настройки параметров корзины. Он выставляет нулевой начальный счет и создает текст, который будет виден на экране. Также код присваивает счетчику жизней значение 3 и отображает это число. Проверь работу программы, добавив в ее конец вызов **root.mainloop()** и запустив. Убедившись, что все в порядке, удали эту строчку: пока что она не нужна.

```
catcher = c.create_arc(catcher_start_x, catcher_start_y, \
                       catcher_start_x2, catcher_start_y2, start=200, extent=140,
                       style='arc', outline=catcher_color, width=3)

game_font = font.nametofont('TkFixedFont')
game_font.config(size=18)

score = 0
score_text = c.create_text(10, 10, anchor='nw', font=game_font, fill='darkblue', \
                           text='Счет: ' + str(score))

lives_remaining = 3
lives_text = c.create_text(canvas_width - 10, 10, anchor='ne', font=game_font, \
                           fill='darkblue', text='Жизней: ' + str(lives_remaining))
```

Задает клевый «компьютерный» шрифт.

Меняя это число, можно увеличивать или уменьшать размер надписи.

Вначале у игрока 3 жизни.

Главные игровые функции

Все настройки сделаны, а значит, пришло время написать код самой игры. Для этого нам понадобятся функции, создающие яйца и заставляющие их падать, а также функции, которые обрабатывают пойманные и упущенные яйца.

8 Создай яйца

Введи этот код. Для учета яиц служит список **eggs**. Функция **create_egg()** выбирает позицию (координата *x* всегда случайная), создает в этом месте яйцо в виде овала и добавляет его в список яиц. В конце функция ставит таймер, вызывающий ее после паузы.

```
lives_text = c.create_text(canvas_width - 10, 10, anchor='ne', font=game_font, fill='darkblue', \
                           text='Жизней: ' + str(lives_remaining))

eggs = []

def create_egg():
    x = randrange(10, 740)
    y = 40
    new_egg = c.create_oval(x, y, x + egg_width, y + egg_height, fill=next(color_cycle), width=0)
    eggs.append(new_egg)
    root.after(egg_interval, create_egg)
```

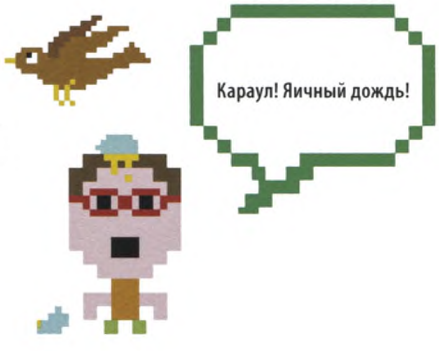
Список для учета яиц.

Выбирает случайную горизонтальную позицию для нового яйца.

Создает овал.

Добавляет овал в список яиц.

Вызывает функцию create_egg() снова спустя egg_interval миллисекунд.



9 Заставь яйца падать

Добавь в код функцию **move_eggs()** («переместить яйца»). Она перебирает все яйца в списке **eggs** и увеличивает *y*-координату каждого из них, сдвигая яйцо вниз. Затем функция проверяет, не достигло ли яйцо нижнего края холста. Если достигло (игрок его упустил), вызывается функция **egg_dropped()** («упущенные яйца»), которая удаляет яйцо из списка и уменьшает количество жизней. И наконец, **move_eggs()** ставит таймер, чтобы снова вызвать себя после небольшой паузы.

```
    root.after(egg_interval, create_egg)

def move_eggs():
    for egg in eggs:
        (egg_x, egg_y, egg_x2, egg_y2) = c.coords(egg)
        c.move(egg, 0, 10)
        if egg_y2 > canvas_height:
            egg_dropped(egg)
    root.after(egg_speed, move_eggs)
```

Цикл проходит по списку яиц.

Получает координаты яйца.

Сдвигает яйцо на 10 пикселей вниз.

Яйцо достигло нижнего края холста?

Если так, нужно вызвать функцию для упущенных яиц.

Вызывает эту функцию снова через egg_speed миллисекунд.

10 Ой, яйцо разбилось!

Добавь после **move_eggs()** код функции **egg_dropped()**. Если яйцо упало и разбилось, оно удаляется из списка и исчезает с холста, после чего вызывается функция **lose_a_life()** («потеря жизни»), которая уменьшает количество жизней на 1 (ее ты создашь на шаге 11). Если жизней больше не осталось, на экран выводится сообщение «Конец игры!».

```
    root.after(egg_speed, move_eggs)

def egg_dropped(egg):
    eggs.remove(egg)
    c.delete(egg)
    lose_a_life()
    if lives_remaining == 0:
        messagebox.showinfo('Конец игры!', 'Итоговый счет: ' \
                            + str(score))
        root.destroy()
```

Удаляет яйцо из списка eggs.

Удаляет яйцо с холста.

Вызывает функцию lose_a_life().

Если жизней не осталось, сообщает, что игра закончилась.

Завершает программу.

11 Потеря жизни

При потере жизни нужно вычесть 1 из значения переменной **lives_remaining** («осталось жизней») и обновить счетчик жизней на экране. Введи этот код после функции **eggs_dropped()**.

```
        root.destroy()

def lose_a_life():
    global lives_remaining
    lives_remaining -= 1
    c.itemconfigure(lives_text, text='Жизней: ' \
                    + str(lives_remaining))
```

Эта переменная должна быть глобальной, чтобы функция могла ее изменить.

Игрок теряет 1 жизнь.

Обновляет счетчик оставшихся жизней на экране.

12 Яйцо поймано?

Создай функцию **check_catch()** («проверить пойманные»). Яйцо считается пойманным, если оно попало в дугу корзины. Чтобы отследить это событие, цикл **for** получает координаты каждого яйца и сравнивает их с координатами корзины. Если проверка дает положительный результат, яйцо удаляется из списка и исчезает с экрана, а счет игрока увеличивается.

```
    c.itemconfigure(lives_text, text='Жизней: ' + str(lives_remaining))

def check_catch():
    (catcher_x, catcher_y, catcher_x2, catcher_y2) = c.coords(catcher)
    for egg in eggs:
        (egg_x, egg_y, egg_x2, egg_y2) = c.coords(egg)
        if catcher_x < egg_x and egg_x2 < catcher_x2 and catcher_y2 - egg_y2 < 40:
            eggs.remove(egg)
            c.delete(egg)
            increase_score(egg_score)
    root.after(100, check_catch)
```

Получает координаты корзины.

Получает координаты яйца.

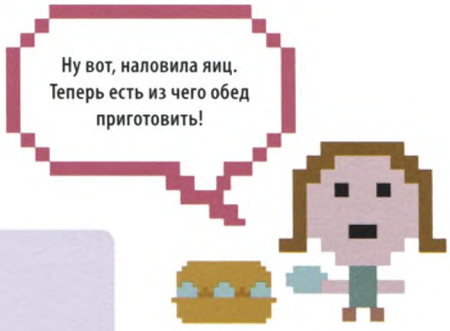
Проверяет, находится ли яйцо в корзине как по вертикали, так и по горизонтали.

Увеличивает счет на 10 очков.

Вызывает функцию check_catch() снова через 100 миллисекунд ($^1/_{10}$ секунды).

13 Увеличь счет

Сначала к значению **score** («счет») прибавляется значение аргумента **points** («очки»). Затем скорость падения и частота появления яиц обновляются путем умножения на коэффициент сложности **difficulty_factor**. И наконец, новый счет выводится на экран. Добавь функцию **increase_score()** («увеличить счет») после **check_catch()**.




```
    root.after(100, check_catch)

def increase_score(points):
    global score, egg_speed, egg_interval
    score += points
    egg_speed = int(egg_speed * difficulty_factor)
    egg_interval = int(egg_interval * difficulty_factor)
    c.itemconfigure(score_text, text='Счет: ' + str(score))
```

Увеличивает счет игрока.

Обновляет счетчик очков на экране.

Поймай эти яйца!

Теперь у тебя есть все необходимые картинки и функции. Осталось добавить управление корзиной и ввести команды, запускающие игру.

14 Управляй корзиной

Функции **move_left()** и **move_right()** проверяют координаты корзины, чтобы не дать ей выйти за пределы экрана. Если до левой или правой границы окна еще есть место, корзина смещается на 20 пикселей по горизонтали. Команды **bind()** связывают эти две функции с клавишами-стрелками ВЛЕВО и ВПРАВО на клавиатуре, а вызов **focus_set()** позволяет программе отслеживать нажатия клавиш. Добавь этот код после функции **increase_score()**.

```
    c.itemconfigure(score_text, text='Счет: ' \
                    + str(score))

def move_left(event):
    (x1, y1, x2, y2) = c.coords(catcher)
    if x1 > 0:
        c.move(catcher, -20, 0)

def move_right(event):
    (x1, y1, x2, y2) = c.coords(catcher)
    if x2 < canvas_width:
        c.move(catcher, 20, 0)

c.bind('<Left>', move_left)
c.bind('<Right>', move_right)
c.focus_set()
```

Дошла ли корзина до левого края холста?

Если нет, сдвигает корзину влево.

Дошла ли корзина до правого края холста?

Если нет, сдвигает корзину вправо.

Эти строчки связывают вызовы функций с нажатиями клавиш.

15 **Запусти игру**

Три зацикленные функции запускаются с помощью таймеров. Это нужно для того, чтобы они не начали работать раньше, чем основной цикл. Вызов **mainloop()** запускает основной цикл модуля **Tkinter**, обслуживающий наши циклы и таймеры. Игра готова — вперед, яйцам от тебя не уйти!

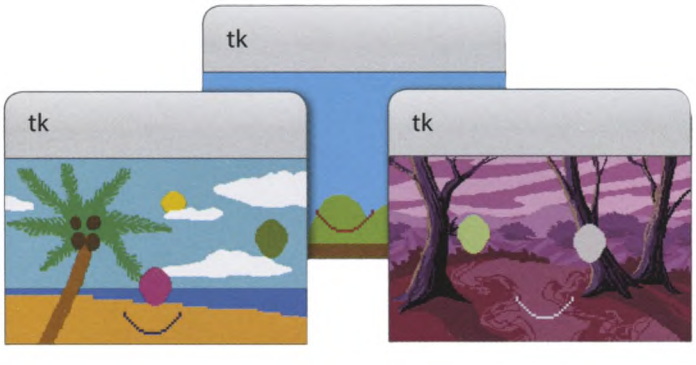
```
c.focus_set()

root.after(1000, create_egg)
root.after(1000, move_eggs)
root.after(1000, check_catch)
root.mainloop()
```

3 игровых цикла запускаются после паузы в 1000 миллисекунд (1 секунду).

Запускает основной цикл модуля Tkinter.

Что бы изменить?

Чтобы игра выглядела еще лучше, можно добавить в нее собственные классные фоны. Музыка и забавные звуки тоже придадут ей индивидуальности.



◁ **Фоновые картинки**

Tkinter позволяет отображать на холсте графические файлы. Для загрузки файлов GIF подойдет класс **tkinter.PhotoImage**. Если же тебя интересует файл другого формата, попробуй **Pillow** — это удобный модуль для работы с изображениями.

СОВЕТЫ ЭКСПЕРТА

Установка модулей

Некоторые полезные модули — например, **Pygame** («python-игра») — не входят в стандартную библиотеку Python. Их нужно устанавливать отдельно. Инструкции по установке лучше всего искать на сайте выбранного модуля или по ссылке https://docs.python.org/3/installing/.

▷ **Пошумим?**

Чтобы оживить игру, добавь в нее фоновую музыку и звуковые эффекты для пойманного и разбитого яйца. Проигрывать звуки тебе поможет виджет **pygame.mixer**, однако сначала его придется установить. Еще понадобится файл со звуком, который ты хочешь включать, — положи его в одну папку с файлом программы. Теперь, чтобы услышать звук, тебе остается написать лишь несколько строчек кода.

```
import time

from pygame import mixer

mixer.init()
beep = mixer.Sound("beep.wav")
beep.play()
time.sleep(5)
```

Подготавливает виджет к проигрыванию звука.

Загружает звуковой файл.

Проигрывает звук.

Пауза нужна для того, чтобы услышать звук.

Справочные материалы

Коды проектов

Здесь ты найдешь готовые Python-коды всех проектов из этой книги (без доработок). Если какая-то программа у тебя не запускается, внимательно сверь ее с нужным кодом.

Тест «Животные» (страница 36)

```
def check_guess(guess, answer):
    global score
    still_guessing = True
    attempt = 0
    while still_guessing and attempt < 3:
        if guess.lower() == answer.lower():
            print('Ответ верный')
            score = score + 1
            still_guessing = False
        else:
            if attempt < 2:
                guess = input('Ответ неверный. Попробуйте еще раз')
            attempt = attempt + 1

    if attempt == 3:
        print('Правильный ответ: ' + answer)

score = 0
print('Тест «Животные»')
guess1 = input('Какой медведь живет за полярным кругом? ')
check_guess(guess1, 'белый медведь')
guess2 = input('Какое сухопутное животное самое быстрое? ')
check_guess(guess2, 'гепард')
guess3 = input('Какое животное самое большое? ')
check_guess(guess3, 'синий кит')

print('Вы набрали очков: ' + str(score))
```

«Генератор паролей» (страница 52)

```
import random
import string

adjectives = ['sleepy', 'slow', 'hot',
              'cold', 'big', 'red',
              'orange', 'yellow', 'green',
              'blue', 'good', 'old',
              'white', 'free', 'brave']
```

```
nouns = ['apple', 'dinosaur', 'ball',
         'cat', 'goat', 'dragon',
         'car', 'duck', 'panda']

print('Добро пожаловать!')

while True:
    adjective = random.choice(adjectives)
    noun = random.choice(nouns)
    number = random.randrange(0, 100)
    special_char = random.choice(string.punctuation)

    password = adjective + noun + str(number) + special_char
    print('Новый пароль: %s' % password)

    response = input('Сгенерировать другой пароль? Введите д или н: ')
    if response == 'н':
        break
```

«Девять жизней» (страница 60)

```
import random

lives = 9
words = ['пицца', 'ангел', 'мираж', 'носки', 'выдра', 'петух']
secret_word = random.choice(words)
clue = list('?????')
heart_symbol = u'\u2764'
guessed_word_correctly = False

def update_clue(guessed_letter, secret_word, clue):
    index = 0
    while index < len(secret_word):
        if guessed_letter == secret_word[index]:
            clue[index] = guessed_letter
        index = index + 1

while lives > 0:
    print(clue)
    print('Осталось жизней: ' + heart_symbol * lives)
    guess = input('Угадайте букву или слово целиком: ')

    if guess == secret_word:
        guessed_word_correctly = True
        break

    if guess in secret_word:
        update_clue(guess, secret_word, clue)
```

```
    else:
        print('Неправильно. Вы теряете жизнь')
        lives = lives - 1

if guessed_word_correctly:
    print('Победа! Было загадано слово ' \
          + secret_word)
else:
    print('Проигрыш! Было загадано слово ' \
          + secret_word)
```

«Сборщик роботов» (страница 72)

```
import turtle as t

def rectangle(horizontal, vertical, color):
    t.pendown()
    t.pensize(1)
    t.color(color)
    t.begin_fill()
    for counter in range(1, 3):
        t.forward(horizontal)
        t.right(90)
        t.forward(vertical)
        t.right(90)
    t.end_fill()
    t.penup()

t.penup()
t.speed('slow')
t.bgcolor('Dodger blue')

# ступни
t.goto(-100, -150)
rectangle(50, 20, 'blue')
t.goto(-30, -150)
rectangle(50, 20, 'blue')

# ноги
t.goto(-25, -50)
rectangle(15, 100, 'grey')
t.goto(-55, -50)
rectangle(-15, 100, 'grey')

# туловище
t.goto(-90, 100)
rectangle(100, 150, 'red')

# руки
t.goto(-150, 70)
rectangle(60, 15, 'grey')
t.goto(-150, 110)
rectangle(15, 40, 'grey')

t.goto(10, 70)
rectangle(60, 15, 'grey')
t.goto(55, 110)
rectangle(15, 40, 'grey')

# шея
t.goto(-50, 120)
rectangle(15, 20, 'grey')

# голова
t.goto(-85, 170)
rectangle(80, 50, 'red')

# глаза
t.goto(-60, 160)
rectangle(30, 10, 'white')
t.goto(-55, 155)
rectangle(5, 5, 'black')
t.goto(-40, 155)
rectangle(5, 5, 'black')

# рот
t.goto(-65, 135)
rectangle(40, 5, 'black')

t.hideturtle()
```

«Радуга-пружинка» (страница 82)

```
import turtle
from itertools import cycle

colors = cycle(['red', 'orange', 'yellow', \
                'green', 'blue', 'purple'])

def draw_circle(size, angle, shift):
    turtle.pencolor(next(colors))
    turtle.circle(size)
    turtle.right(angle)
    turtle.forward(shift)
    draw_circle(size + 5, angle + 1, shift + 1)

turtle.bgcolor('black')
turtle.speed('fast')
turtle.pensize(4)
draw_circle(30, 0, 1)
```

«Звездное небо» (страница 90)

```
import turtle as t
from random import randint, random

def draw_star(points, size, col, x, y):
    t.penup()
    t.goto(x, y)
    t.pendown
    angle = 180 - (180 / points)
    t.color(col)
    t.begin_fill()
    for i in range(points):
        t.forward(size)
        t.right(angle)
    t.end_fill()

# Основной код
t.Screen().bgcolor('dark blue')

while True:
    ranPts = randint(2, 5) * 2 + 1
    ranSize = randint(10, 50)
    ranCol = (random(), random(), random())
    ranX = randint(-350, 300)
    ranY = randint(-250, 250)

    draw_star(ranPts, ranSize, ranCol, ranX, ranY)
```

«Безумная радуга» (страница 98)

```
import random
import turtle as t

def get_line_length():
    choice = input('Выберите длину линий (длинные, средние, короткие): ')
    if choice == 'длинные':
        line_length = 250
    elif choice == 'средние':
        line_length = 200
    else:
        line_length = 100
    return line_length

def get_line_width():
    choice = input('Выберите толщину линий (суперширокие, широкие, тонкие): ')
    if choice == 'суперширокие':
        line_width = 40
    elif choice == 'широкие':
        line_width = 25
```

```
    else:
        line_width = 10
    return line_width

def inside_window():
    left_limit = (-t.window_width() / 2) + 100
    right_limit = (t.window_width() / 2) - 100
    top_limit = (t.window_height() / 2) - 100
    bottom_limit = (-t.window_height() / 2) + 100
    (x, y) = t.pos()
    inside = left_limit < x < right_limit and bottom_limit < y < top_limit
    return inside

def move_turtle(line_length):
    pen_colors = ['red', 'orange', 'yellow', 'green', 'blue', 'purple']
    t.pencolor(random.choice(pen_colors))
    if inside_window():
        angle = random.randint(0, 180)
        t.right(angle)
        t.forward(line_length)
    else:
        t.backward(line_length)

line_length = get_line_length()
line_width = get_line_width()

t.shape('turtle')
t.fillcolor('green')
t.bgcolor('black')
t.speed('fastest')
t.pensize(line_width)

while True:
    move_turtle(line_length)
```

«Календарь ожидания» (страница 110)

```
from tkinter import Tk, Canvas
from datetime import date, datetime

def get_events():
    list_events = []
    with open('events.txt', encoding='utf-8') as file:
        for line in file:
            line = line.rstrip('\n')
            current_event = line.split(',')
            event_date = datetime.strptime(current_event[1], '%d/%m/%Y').date()
            current_event[1] = event_date
            list_events.append(current_event)
    return list_events
```

```
def days_between_dates(date1, date2):
    time_between = str(date1 - date2)
    number_of_days = time_between.split(' ')
    return number_of_days[0]

root = Tk()
c = Canvas(root, width=800, height=800, bg='black')
c.pack()
c.create_text(100, 50, anchor='w', fill='orange', font='Arial 28 bold underline', \
              text='Календарь ожидания')

events = get_events()
today = date.today()

vertical_space = 100

for event in events:
    event_name = event[0]
    days_until = days_between_dates(event[1], today)
    display = '%s через %s дн.' % (event_name, days_until)
    c.create_text(100, vertical_space, anchor='w', fill='lightblue', \
                  font='Arial 28 bold', text=display)

    vertical_space = vertical_space + 30
```

«Знаток» (страница 120)

```
from tkinter import Tk, simpledialog, messagebox

def read_from_file():
    with open('capital_data.txt', encoding='utf-8') as file:
        for line in file:
            line = line.rstrip('\n')
            country, city = line.split('/')
            the_world[country] = city

def write_to_file(country_name, city_name):
    with open('capital_data.txt', 'a', encoding='utf-8') as file:
        file.write('\n' + country_name + '/' + city_name)

print('Знаток — Столицы мира')
root = Tk()
root.withdraw()
the_world = {}

read_from_file()

while True:
    query_country = simpledialog.askstring('Страна', 'Введите название страны:')

    if query_country in the_world:
```

```
        result = the_world[query_country]
        messagebox.showinfo('Ответ',
                            query_country + ': столица этой страны — ' + result + '!')
    else:
        new_city = simpledialog.askstring('Научите меня',
                                          'Я не знаю, ' +
                                          'как называется столица страны ' + query_country + '!')
        the_world[query_country] = new_city
        write_to_file(query_country, new_city)

root.mainloop()
```

«Тайная переписка» (страница 130)

```
from tkinter import messagebox, simpledialog, Tk

def is_even(number):
    return number % 2 == 0

def get_even_letters(message):
    even_letters = []
    for counter in range(0, len(message)):
        if is_even(counter):
            even_letters.append(message[counter])
    return even_letters

def get_odd_letters(message):
    odd_letters = []
    for counter in range(0, len(message)):
        if not is_even(counter):
            odd_letters.append(message[counter])
    return odd_letters

def swap_letters(message):
    letter_list = []
    if not is_even(len(message)):
        message = message + 'x'
    even_letters = get_even_letters(message)
    odd_letters = get_odd_letters(message)
    for counter in range(0, int(len(message)/2)):
        letter_list.append(odd_letters[counter])
        letter_list.append(even_letters[counter])
    new_message = ''.join(letter_list)
    return new_message

def get_task():
    task = simpledialog.askstring('Задание', 'Что сделать: зашифровать или расшифровать?')
    return task
```

```
def get_message():
    message = simpledialog.askstring('Сообщение', 'Введите секретное сообщение: ')
    return message

root = Tk()

while True:
    task = get_task()
    if task == 'зашифровать':
        message = get_message()
        encrypted = swap_letters(message)
        messagebox.showinfo('Зашифрованное сообщение:', encrypted)
    elif task == 'расшифровать':
        message = get_message()
        decrypted = swap_letters(message)
        messagebox.showinfo('Расшифрованное сообщение:', decrypted)
    else:
        break
root.mainloop()
```

«Экранный питомец» (страница 142)

```
from tkinter import HIDDEN, NORMAL, Tk, Canvas

def toggle_eyes():
    current_color = c.itemcget(eye_left, 'fill')
    new_color = c.body_color if current_color == 'white' else 'white'
    current_state = c.itemcget(pupil_left, 'state')
    new_state = NORMAL if current_state == HIDDEN else HIDDEN
    c.itemconfigure(pupil_left, state=new_state)
    c.itemconfigure(pupil_right, state=new_state)
    c.itemconfigure(eye_left, fill=new_color)
    c.itemconfigure(eye_right, fill=new_color)

def blink():
    toggle_eyes()
    root.after(250, toggle_eyes)
    root.after(3000, blink)

def toggle_pupils():
    if not c.eyes_crossed:
        c.move(pupil_left, 10, -5)
        c.move(pupil_right, -10, -5)
        c.eyes_crossed = True
    else:
        c.move(pupil_left, -10, 5)
        c.move(pupil_right, 10, 5)
        c.eyes_crossed = False
```

```
def toggle_tongue():
    if not c.tongue_out:
        c.itemconfigure(tongue_tip, state=NORMAL)
        c.itemconfigure(tongue_main, state=NORMAL)
        c.tongue_out = True
    else:
        c.itemconfigure(tongue_tip, state=HIDDEN)
        c.itemconfigure(tongue_main, state=HIDDEN)
        c.tongue_out = False

def cheeky(event):
    toggle_tongue()
    toggle_pupils()
    hide_happy(event)
    root.after(1000, toggle_tongue)
    root.after(1000, toggle_pupils)
    return

def show_happy(event):
    if (20 <= event.x and event.x <= 350) and (20 <= event.y and event.y <= 350):
        c.itemconfigure(cheek_left, state=NORMAL)
        c.itemconfigure(cheek_right, state=NORMAL)
        c.itemconfigure(mouth_happy, state=NORMAL)
        c.itemconfigure(mouth_normal, state=HIDDEN)
        c.itemconfigure(mouth_sad, state=HIDDEN)
        c.happy_level = 10
    return

def hide_happy(event):
    c.itemconfigure(cheek_left, state=HIDDEN)
    c.itemconfigure(cheek_right, state=HIDDEN)
    c.itemconfigure(mouth_happy, state=HIDDEN)
    c.itemconfigure(mouth_normal, state=NORMAL)
    c.itemconfigure(mouth_sad, state=HIDDEN)
    return

def sad():
    if c.happy_level == 0:
        c.itemconfigure(mouth_happy, state=HIDDEN)
        c.itemconfigure(mouth_normal, state=HIDDEN)
        c.itemconfigure(mouth_sad, state=NORMAL)
    else:
        c.happy_level -= 1
    root.after(5000, sad)

root = Tk()
c = Canvas(root, width=400, height=400)
c.configure(bg='dark blue', highlightthickness=0)
c.body_color = 'SkyBlue1'
```

```
body = c.create_oval(35, 20, 365, 350, outline=c.body_color, fill=c.body_color)
ear_left = c.create_polygon(75, 80, 75, 10, 165, 70, outline=c.body_color, fill=c.body_color)
ear_right = c.create_polygon(255, 45, 325, 10, 320, 70, outline=c.body_color, fill=c.body_color)
foot_left = c.create_oval(65, 320, 145, 360, outline=c.body_color, fill=c.body_color)
foot_right = c.create_oval(250, 320, 330, 360, outline=c.body_color, fill=c.body_color)

eye_left = c.create_oval(130, 110, 160, 170, outline='black', fill='white')
pupil_left = c.create_oval(140, 145, 150, 155, outline='black', fill='black')
eye_right = c.create_oval(230, 110, 260, 170, outline='black', fill='white')
pupil_right = c.create_oval(240, 145, 250, 155, outline='black', fill='black')

mouth_normal = c.create_line(170, 250, 200, 272, 230, 250, smooth=1, width=2, state=NORMAL)
mouth_happy = c.create_line(170, 250, 200, 282, 230, 250, smooth=1, width=2, state=HIDDEN)
mouth_sad = c.create_line(170, 250, 200, 232, 230, 250, smooth=1, width=2, state=HIDDEN)
tongue_main = c.create_rectangle(170, 250, 230, 270, outline='red', fill='red', state=HIDDEN)
tongue_tip = c.create_oval(170, 250, 230, 300, outline='red', fill='red', state=HIDDEN)

cheek_left = c.create_oval(70, 180, 120, 230, outline='pink', fill='pink', state=HIDDEN)
cheek_right = c.create_oval(280, 180, 330, 230, outline='pink', fill='pink', state=HIDDEN)

c.pack()

c.bind('<Motion>', show_happy)
c.bind('<Leave>', hide_happy)
c.bind('<Double-1>', cheeky)

c.happy_level = 10
c.eyes_crossed = False
c.tongue_out = False

root.after(1000, blink)
root.after(5000, sad)
root.mainloop()
```

«Гусеница» (страница 158)

```
import random
import turtle as t

t.bgcolor('yellow')

caterpillar = t.Turtle()
caterpillar.shape('square')
caterpillar.color('red')
caterpillar.speed(0)
caterpillar.penup()
caterpillar.hideturtle()

leaf = t.Turtle()
```

```
leaf_shape = ((0, 0), (14, 2), (18, 6), (20, 20), (6, 18), (2, 14))
t.register_shape('leaf', leaf_shape)
leaf.shape('leaf')
leaf.color('green')
leaf.penup()
leaf.hideturtle()
leaf.speed(0)

game_started = False
text_turtle = t.Turtle()
text_turtle.write('Нажмите ПРОБЕЛ, чтобы начать игру', align='center', font=('Arial', 16, 'bold'))
text_turtle.hideturtle()

score_turtle = t.Turtle()
score_turtle.hideturtle()
score_turtle.speed(0)

def outside_window():
    left_wall = -t.window_width() / 2
    right_wall = t.window_width() / 2
    top_wall = t.window_height() / 2
    bottom_wall = -t.window_height() / 2
    (x, y) = caterpillar.pos()
    outside = \
            x < left_wall or \
            x > right_wall or \
            y < bottom_wall or \
            y > top_wall
    return outside

def game_over():
    caterpillar.color('yellow')
    leaf.color('yellow')
    t.penup()
    t.hideturtle()
    t.write('Конец игры!', align='center', font=('Arial', 30, 'normal'))

def display_score(current_score):
    score_turtle.clear()
    score_turtle.penup()
    x = (t.window_width() / 2) - 50
    y = (t.window_height() / 2) - 50
    score_turtle.setpos(x, y)
    score_turtle.write(str(current_score), align='right', font=('Arial', 40, 'bold'))

def place_leaf():
    leaf.ht()
    leaf.setx(random.randint(-200, 200))
```

```
    leaf.sety(random.randint(-200, 200))
    leaf.st()

def start_game():
    global game_started
    if game_started:
        return
    game_started = True

    score = 0
    text_turtle.clear()

    caterpillar_speed = 2
    caterpillar_length = 3
    caterpillar.shapesize(1, caterpillar_length, 1)
    caterpillar.showturtle()
    display_score(score)
    place_leaf()

    while True:
        caterpillar.forward(caterpillar_speed)
        if caterpillar.distance(leaf) < 20:
            place_leaf()
            caterpillar_length = caterpillar_length + 1
            caterpillar.shapesize(1, caterpillar_length, 1)
            caterpillar_speed = caterpillar_speed + 1
            score = score + 10
            display_score(score)
        if outside_window():
            game_over()
            break

def move_up():
    if caterpillar.heading() == 0 or caterpillar.heading() == 180:
        caterpillar.setheading(90)

def move_down():
    if caterpillar.heading() == 0 or caterpillar.heading() == 180:
        caterpillar.setheading(270)

def move_left():
    if caterpillar.heading() == 90 or caterpillar.heading() == 270:
        caterpillar.setheading(180)

def move_right():
    if caterpillar.heading() == 90 or caterpillar.heading() == 270:
        caterpillar.setheading(0)
t.onkey(start_game, 'space')
t.onkey(move_up, 'Up')
t.onkey(move_right, 'Right')
```

```
t.onkey(move_down, 'Down')
t.onkey(move_left, 'Left')
t.listen()
t.mainloop()
```

«Снэп» (страница 168)

```
import random
import time
from tkinter import Tk, Canvas, HIDDEN, NORMAL

def next_shape():
    global shape
    global previous_color
    global current_color

    previous_color = current_color

    c.delete(shape)
    if len(shapes) > 0:
        shape = shapes.pop()
        c.itemconfigure(shape, state=NORMAL)
        current_color = c.itemcget(shape, 'fill')
        root.after(1000, next_shape)
    else:
        c.unbind('q')
        c.unbind('p')
        if player1_score > player2_score:
            c.create_text(200, 200, text='Победил игрок 1')
        elif player2_score > player1_score:
            c.create_text(200, 200, text='Победил игрок 2')
        else:
            c.create_text(200, 200, text='Ничья')
        c.pack()

def snap(event):
    global shape
    global player1_score
    global player2_score
    valid = False

    c.delete(shape)
    if previous_color == current_color:
        valid = True

    if valid:
        if event.char == 'q':
            player1_score = player1_score + 1
```

```
        else:
            player2_score = player2_score + 1
        shape = c.create_text(200, 200, text='СНЭП! Вы получаете 1 очко!')
    else:
        if event.char == 'q':
            player1_score = player1_score - 1
        else:
            player2_score = player2_score - 1
        shape = c.create_text(200, 200, text='МИМО! Вы теряете 1 очко!')
    c.pack()
    root.update_idletasks()
    time.sleep(1)

root = Tk()
root.title('Снэп')
c = Canvas(root, width=400, height=400)

shapes = []

circle = c.create_oval(35, 20, 365, 350, outline='black', fill='black', state=HIDDEN)
shapes.append(circle)
circle = c.create_oval(35, 20, 365, 350, outline='red', fill='red', state=HIDDEN)
shapes.append(circle)
circle = c.create_oval(35, 20, 365, 350, outline='green', fill='green', state=HIDDEN)
shapes.append(circle)
circle = c.create_oval(35, 20, 365, 350, outline='blue', fill='blue', state=HIDDEN)
shapes.append(circle)

rectangle = c.create_rectangle(35, 100, 365, 270, outline='black', fill='black', state=HIDDEN)
shapes.append(rectangle)
rectangle = c.create_rectangle(35, 100, 365, 270, outline='red', fill='red', state=HIDDEN)
shapes.append(rectangle)
rectangle = c.create_rectangle(35, 100, 365, 270, outline='green', fill='green', state=HIDDEN)
shapes.append(rectangle)
rectangle = c.create_rectangle(35, 100, 365, 270, outline='blue', fill='blue', state=HIDDEN)
shapes.append(rectangle)

square = c.create_rectangle(35, 20, 365, 350, outline='black', fill='black', state=HIDDEN)
shapes.append(square)
square = c.create_rectangle(35, 20, 365, 350, outline='red', fill='red', state=HIDDEN)
shapes.append(square)
square = c.create_rectangle(35, 20, 365, 350, outline='green', fill='green', state=HIDDEN)
shapes.append(square)
square = c.create_rectangle(35, 20, 365, 350, outline='blue', fill='blue', state=HIDDEN)
shapes.append(square)
c.pack()

random.shuffle(shapes)

shape = None
```

```
previous_color = ''
current_color = ''
player1_score = 0
player2_score = 0

root.after(3000, next_shape)
c.bind('q', snap)
c.bind('p', snap)
c.focus_set()

root.mainloop()
```

«Мемори» (страница 180)

```
import random
import time
from tkinter import Tk, Button, DISABLED

def show_symbol(x, y):
    global first
    global previousX, previousY
    buttons[x, y]['text'] = button_symbols[x, y]
    buttons[x, y].update_idletasks()

    if first:
        previousX = x
        previousY = y
        first = False
    elif previousX != x or previousY != y:
        if buttons[previousX, previousY]['text'] != buttons[x, y]['text']:
            time.sleep(0.5)
            buttons[previousX, previousY]['text'] = ''
            buttons[x, y]['text'] = ''
        else:
            buttons[previousX, previousY]['command'] = DISABLED
            buttons[x, y]['command'] = DISABLED
        first = True

root = Tk()
root.title('Мемори')
root.resizable(width=False, height=False)
buttons = {}
first = True
previousX = 0
previousY = 0
button_symbols = {}
symbols = [u'\u2702', u'\u2702', u'\u2705', u'\u2705', u'\u2708', u'\u2708',
           u'\u2709', u'\u2709', u'\u270A', u'\u270A', u'\u270B', u'\u270B',
```

```
          u'\u270C', u'\u270C', u'\u270F', u'\u270F', u'\u2712', u'\u2712',
          u'\u2714', u'\u2714', u'\u2716', u'\u2716', u'\u2728', u'\u2728']
random.shuffle(symbols)

for x in range(6):
    for y in range(4):
        button = Button(command=lambda x=x, y=y: show_symbol(x, y), width=3, height=3)
        button.grid(column=x, row=y)
        buttons[x, y] = button
        button_symbols[x, y] = symbols.pop()

root.mainloop()
```

«Яйцелов» (страница 190)

```
from itertools import cycle
from random import randrange
from tkinter import Canvas, Tk, messagebox, font

canvas_width = 800
canvas_height = 400

root = Tk()
c = Canvas(root, width=canvas_width, height=canvas_height, background='deep sky blue')
c.create_rectangle(-5, canvas_height - 100, canvas_width + 5, canvas_height + 5, \
                   fill='sea green', width=0)
c.create_oval(-80, -80, 120, 120, fill='orange', width=0)
c.pack()

color_cycle = cycle(['light blue', 'light green', 'light pink', 'light yellow', 'light cyan'])
egg_width = 45
egg_height = 55
egg_score = 10
egg_speed = 500
egg_interval = 4000
difficulty_factor = 0.95

catcher_color = 'blue'
catcher_width = 100
catcher_height = 100
catcher_start_x = canvas_width / 2 - catcher_width / 2
catcher_start_y = canvas_height - catcher_height - 20
catcher_start_x2 = catcher_start_x + catcher_width
catcher_start_y2 = catcher_start_y + catcher_height

catcher = c.create_arc(catcher_start_x, catcher_start_y, \
                       catcher_start_x2, catcher_start_y2, start=200, extent=140, \
                       style='arc', outline=catcher_color, width=3)
```

```
game_font = font.nametofont('TkFixedFont')
game_font.config(size=18)

score = 0
score_text = c.create_text(10, 10, anchor='nw', font=game_font, fill='darkblue', \
                           text='Счет: ' + str(score))

lives_remaining = 3
lives_text = c.create_text(canvas_width - 10, 10, anchor='ne', font=game_font, fill='darkblue', \
                           text='Жизней: ' + str(lives_remaining))

eggs = []

def create_egg():
    x = randrange(10, 740)
    y = 40
    new_egg = c.create_oval(x, y, x + egg_width, y + egg_height, fill=next(color_cycle), width=0)
    eggs.append(new_egg)
    root.after(egg_interval, create_egg)

def move_eggs():
    for egg in eggs:
        (egg_x, egg_y, egg_x2, egg_y2) = c.coords(egg)
        c.move(egg, 0, 10)
        if egg_y2 > canvas_height:
            egg_dropped(egg)
    root.after(egg_speed, move_eggs)

def egg_dropped(egg):
    eggs.remove(egg)
    c.delete(egg)
    lose_a_life()
    if lives_remaining == 0:
        messagebox.showinfo('Конец игры!', 'Итоговый счет: ' + str(score))
        root.destroy()

def lose_a_life():
    global lives_remaining
    lives_remaining -= 1
    c.itemconfigure(lives_text, text='Жизней: ' + str(lives_remaining))

def check_catch():
    (catcher_x, catcher_y, catcher_x2, catcher_y2) = c.coords(catcher)
    for egg in eggs:
        (egg_x, egg_y, egg_x2, egg_y2) = c.coords(egg)
        if catcher_x < egg_x and egg_x2 < catcher_x2 and catcher_y2 - egg_y2 < 40:
            eggs.remove(egg)
            c.delete(egg)
            increase_score(egg_score)
```

```
    root.after(100, check_catch)

def increase_score(points):
    global score, egg_speed, egg_interval
    score += points
    egg_speed = int(egg_speed * difficulty_factor)
    egg_interval = int(egg_interval * difficulty_factor)
    c.itemconfigure(score_text, text='Счет: ' + str(score))

def move_left(event):
    (x1, y1, x2, y2) = c.coords(catcher)
    if x1 > 0:
        c.move(catcher, -20, 0)

def move_right(event):
    (x1, y1, x2, y2) = c.coords(catcher)
    if x2 < canvas_width:
        c.move(catcher, 20, 0)

c.bind('<Left>', move_left)
c.bind('<Right>', move_right)
c.focus_set()

root.after(1000, create_egg)
root.after(1000, move_eggs)
root.after(1000, check_catch)
root.mainloop()
```

Глоссарий

ASCII
American Standard Code for Information Interchange (Американский стандартный код для обмена информацией) — способ представления символов в двоичном коде.

GUI
Graphic User Interface (графический интерфейс пользователя) — кнопки, окна и другие элементы управления программой, с которыми пользователь может взаимодействовать.

Python
Популярный язык программирования, который создал Гвидо ван Россум. Отлично подходит для начинающих.

turtle
Модуль Python, позволяющий создавать на экране графику и анимацию с помощью робота-черепашки.

Аргумент
Значение, передаваемое в функцию при вызове.

Баг
Ошибка в коде, из-за которой программа работает не так, как задумано, или не работает вовсе.

Библиотека
Набор функций, который можно повторно использовать в разных программах.

Блок-схема
Диаграмма, которая описывает программу в виде последовательности шагов и решений.

Булево значение
Ответ True («правда») или False («ложь») на вопрос, который задает программа.

Ввод
Данные, которые пользователь вводит в компьютер через специальные устройства: клавиатуру, мышку, микрофон.

Ветвление
Место в коде, в котором у программы есть выбор из двух вариантов действий.

Вещественное число
Дробное число с десятичной точкой.

Виджет
Независимый элемент графического интерфейса, который имеет стандартный вид и выполняет стандартные действия. В книге используются виджеты модуля Tinker.

Вложенный цикл
Цикл, помещенный внутрь другого цикла.

Возвращаемое значение
Значение, которое функция передает обратно в вызвавший ее код.

Вывод
Данные, которые программа показывает пользователю.

Вызов
Обращение к функции из кода программы.

Глобальная переменная
Переменная, значение которой доступно всем частям программы. *См. также* Локальная переменная.

Графика
Элементы на экране, которые не являются текстом, например картинки, иконки и символы.

Данные
Информация, представленная в виде текста, символов или чисел.

Запуск
Передача кода программы на выполнение.

Индекс
Номер элемента в списке. В Python первый элемент имеет индекс 0, второй — 1 и т. д.

Инструкция
Наименьшая автономная часть языка программирования: команда или набор команд.

Интерфейс
Механизм взаимодействия пользователя с программой или устройством. *См. также* GUI.

Комментарий
Текстовое примечание в коде, которое помогает понять, как устроена программа. Не влияет на ее работу.

Константа
Значение, которое невозможно изменить.

Координаты
Пара чисел, задающая положение точки на экране по горизонтали и вертикали. Обычно записываются как (x, y).

Кортеж
Набор разделенных запятыми элементов в круглых скобках. Кортеж похож на список, однако после создания его элементы нельзя изменить.

Локальная переменная
Переменная, значение которой доступно лишь одной части кода, например коду функции. *См. также* Глобальная переменная.

Модуль
Набор готового к использованию кода, который можно импортировать в свои программы.

Оператор
Символ, позволяющий выполнить определенное действие в коде, например сложение (+) или вычитание (–).

Операционная система
Программа, управляющая компьютером, такая как Windows, macOS, Linux и т. д.

Отладка
Поиск и исправление ошибок в программе.

Отступ
Способ сместить блок кода вправо относительно других блоков. В Python отступ равен 4 пробелам. Все строки в отдельно взятом блоке нужно вводить с одинаковым отступом.

Переключение
Смена двух возможных состояний.

Переменная
Именованное место для хранения данных, которые можно изменять.

Пиксель
Мельчайшая точка цифрового изображения.

Программа
Набор инструкций, которым следует компьютер, решая ту или иную задачу.

Программное обеспечение (ПО)
Совокупность программ, работающих на компьютере.

Рекурсия
Цикл, возникающий, когда функция вызывает саму себя.

Синтаксис
Правила написания кода. Несоблюдение синтаксиса приводит к ошибкам.

Словарь
Набор парных значений, таких как страны и их столицы.

Случайные значения
Значения, которые нельзя предсказать заранее. Часто используются при создании игр.

Событие
Действие, на которое программа может отреагировать, например нажатие клавиши или клик мышкой.

Список
Набор элементов, хранящихся в определенном порядке.

Строка
Последовательность символов, таких как цифры, буквы, знаки препинания, пробелы.

Условие
Конструкция, которая служит для принятия решения в программе и возвращает True или False. *См. также* Булево значение.

Файл
Именованный набор данных.

Флаговая переменная
Переменная, принимающая одно из двух состояний, таких как True и False.

Функция
Выполняющий конкретную задачу код, который работает как программа в программе.

Целое число
Число, которое не является дробью и записывается без десятичной точки.

Цикл
Код, который выполняется повторно, так что его не нужно каждый раз вводить заново.

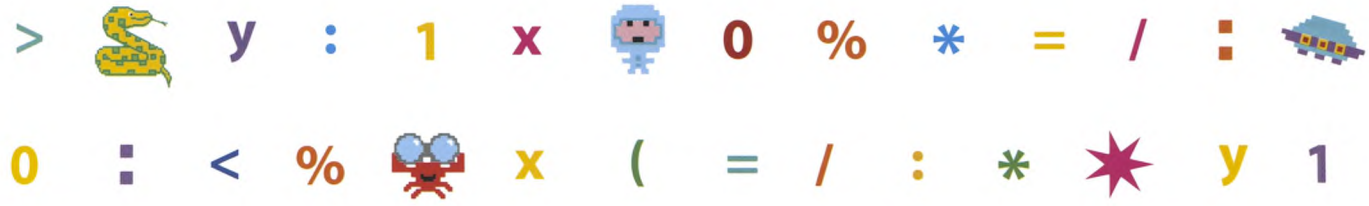
Шифрование
Кодирование данных с целью сделать их доступными лишь конкретным людям.

Юникод
Универсальная кодировка символов, поддерживающая тысячи разных букв и знаков.

Язык программирования
Язык, на котором можно писать инструкции для компьютера.

Предметный указатель

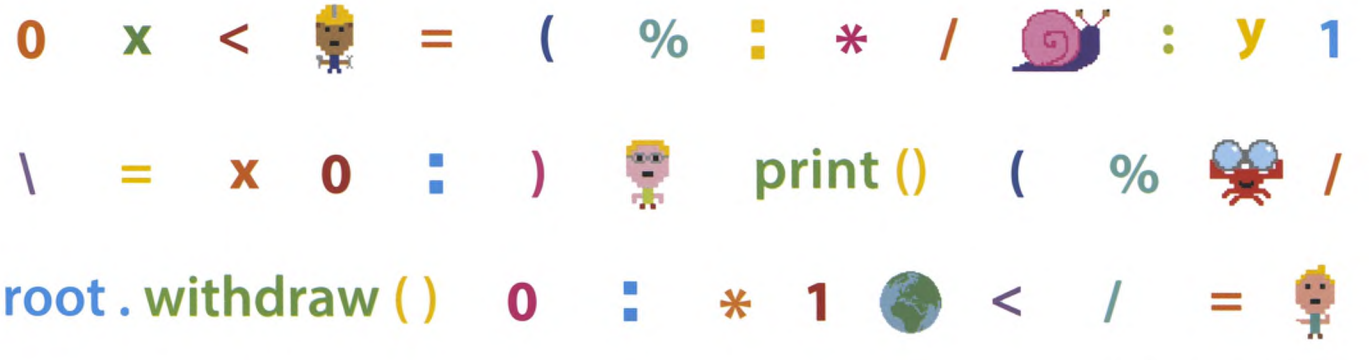
Жирным шрифтом выделены номера страниц с важными сведениями.



С помощью этой книги научиться программировать сможет любой. Все, что тебе нужно, — это компьютер, интернет (для скачивания установочных файлов) и желание творить. Создавай полезные программы и веселые игры на одном из самых популярных языков программирования — Python!

Внутри тебя ждут пошаговые инструкции по созданию 16 программ, примеры кода, иллюстрации, блок-схемы и забавные пиксельные герои. Следуя подробным описаниям, ты научишь компьютер рисовать робота и звездное небо, создашь тест на эрудицию и адвент-календарь, программу-шифровальщик и экранного питомца. А если устанешь программировать, отвлекись и поиграй в игры на внимание и быстроту реакции, которые сам же и создашь!

Книга идеально подходит для детей 10 лет и старше, а также для взрослых, которые делают первые шаги в программировании.



Детские книги на сайте
mann-ivanov-ferber.ru

facebook.com/mifdetstvo
vk.com/mifdetstvo
instagram.com/mifdetstvo

ISBN 978-5-00117-399-1