ПРОГРАММИРОВАНИЕ

на

PYTHON

ДЛЯ АБСОЛЮТНЫХ НОВИЧКОВ

Программирование на Python для абсолютных новичков

Иллюстрированное руководство с лабораторными упражнениями

Кевин Уилсон

Программирование на Python для абсолютных новичков

Издатель: Elluminet Пресс-директор: Кевин Уилсон

Ведущий редактор: Стивен Эшмор

Технический обозреватель: Майк Тейлор, Роберт Эшкрофт

Редакторы: Джоэнн Тейлор, Джеймс Марш

Корректор: Стивен Эшмор

Индексатор: Джеймс Марш

Дизайнер обложки: Кевин Уилсон

Версии электронных книг и лицензии также доступны для большинства изданий. Любой исходный код или другие дополнительные материалы, на которые ссылается автор в этом тексте, доступны читателям по адресу

www.elluminetpress.com/resources

Подробную информацию о том, как найти ресурсы вашей книги, можно найти на сайте

www.elluminetpress.com/resources

Оглавление

Об авторе

Имея более чем 20-летний опыт работы в компьютерной индустрии, Кевин Уилсон сделал карьеру в сфере технологий и показал другим, как их использовать. Получив степень магистра в области компьютерных наук, разработки программного обеспечения и мультимедийных систем, Кевин занимал различные должности в ИТ-индустрии, включая связанные с графическим и веб-дизайном, программированием, созданием и управлением корпоративными сетями, а также ИТ-поддержкой.

Он работает писателем и директором в Elluminet Press Ltd, периодически преподает информатику в колледже и работает тренером по информационным технологиям в Англии, одновременно работая над докторской диссертацией. Его книги стали ценным ресурсом для студентов Англии, Южной Африки, Канады и США.

Девиз Кевина ясен: «Если ты не можешь объяснить что-то просто, значит, ты недостаточно хорошо это понял». С этой целью он создал серию «Изучение технических вычислений», в которой разбивает сложные технологические предметы на более мелкие и простые для понимания шаги, которые студенты и обычные пользователи компьютеров могут применить на практике.

Слова благодарности

Спасибо всему персоналу Luminescent Media и Elluminet Press за их энтузиазм, самоотверженность и упорный труд при подготовке и выпуске этой книги.

Всем моим друзьям и семье за их постоянную поддержку и одобрение во всех моих писательских проектах.

Всем моим коллегам, студентам и тестировщикам, которые нашли время протестировать процедуры и оставить отзыв о книге.

Наконец, спасибо читателю за выбор этой книги. Надеюсь, это поможет вам изучить компьютерное оборудование.

Наслаждайтесь!

Введение в программирование

Что такое компьютерная программа? Компьютерная программа — это набор кратких инструкций, написанных на языке программирования, которые последовательно выполняются для достижения задачи.

Компьютерная программа обычно берет некоторые данные, например строку или число, и выполняет некоторую их обработку для получения результатов. Обычно мы называем такие данные входными данными программы, а результаты — выходными данными программы.

Для написания компьютерных программ мы используем язык программирования. Существует множество различных языков, таких как BASIC, C, C++ и Python. В этой книге мы сосредоточимся на языке программирования Python.

Каждая компьютерная программа манипулирует данными для получения результата, поэтому большинство языков позволяют программисту выбирать имена для каждого элемента данных.

Эти элементы называются переменными. Переменная, как следует из названия, представляет собой элемент, который во время выполнения программы может содержать разные значения. Переменные могут хранить данные разных типов, а разные типы могут выполнять разные действия. Например, переменная может быть целочисленной для хранения целого числа, с плавающей запятой для хранения чисел с десятичными знаками, строкой для хранения текста или списком для хранения нескольких элементов данных.

Если бы мы написали программу для вычисления площади треугольника, мы могли бы создать переменные целочисленного типа или с плавающей запятой для длины и высоты и одну для результата, поскольку все они являются числами. Исполняемая часть программы будет использовать числа, хранящиеся в переменных длины и высоты, а затем присваивать результат переменной результата. Более подробно переменные и типы данных мы рассмотрим в главе 3.

В более крупных программах нам часто приходится принимать

решения на основе данных пользователя, вычисленного результата или условия. В таком случае мы используем оператор if. Это называется выбором. Также может потребоваться повторить некоторые блоки кода, в этом случае мы используем цикл. Это называется повторением. Мы подробнее рассмотрим это в главе 4.

Многие современные компьютерные программы имеют необычный графический интерфейс пользователя, который позволяет пользователю взаимодействовать с кнопками, окнами и меню. Это известно как программные приложения или приложения. В главе 11 мы рассмотрим создание простого приложения с графическим пользовательским интерфейсом.

Язык программирования Python обладает особыми возможностями, позволяющими реализовать концепции, изложенные выше. Многие из них будут представлены в этой книге.

Знакомство с Python

Python — это язык высокого уровня, разработанный Гвидо ван Россумом в конце 80-х годов и который используется в веб-разработке, научных приложениях, играх, искусственном интеллекте, а также хорошо подходит для обучения компьютерному программированию.

Python спроектирован как легко читаемый язык. Поэтому он использует лаконичный стиль форматирования и часто использует осмысленные английские ключевые слова и имена функций.

Python — это интерпретируемый язык программирования, то есть программы Python пишутся в текстовом редакторе, а затем для выполнения обрабатываются интерпретатором Python.

Python используется в области искусственного интеллекта и его можно найти во многих повседневных приложениях. Стриминговые сервисы, такие как Spotify, используют Python для анализа данных, в частности, привычек пользователя в отношении прослушивания, чтобы предлагать рекомендации о том, за каким исполнителем следить, либо о другой музыке, которая может быть интересна конкретному пользователю, и так далее. Python также используется в алгоритмах машинного обучения Netflix для рекомендации пользователям соответствующего контента, мониторинга привычек просмотра и маркетинга.

В мире разработки игр Python используется в качестве сопутствующего языка, то есть сценарии Python используются для добавления настроек к основному игровому движку, сценариям поведения ИИ или элементам на стороне сервера. Производительность Python недостаточна для кодирования высокопроизводительных игр с интенсивным использованием графики, однако вы можете создавать простые игры с помощью Python, используя модуль pygame. Мы рассмотрим это в главе 12.

Python используется в веб-разработке и позволяет веб-разработчику очень быстро разрабатывать динамические веб-приложения. Подробнее об этом в главе 13.

Python — это кроссплатформенный язык, доступный для Windows, MacOS, Linux и Raspberry Pi.

Настройка

Чтобы начать программировать, вам понадобится компьютер под управлением Windows, MacOS или Linux, а также интегрированная среда разработки (IDE) с интерпретатором Python. Давайте установим Python.

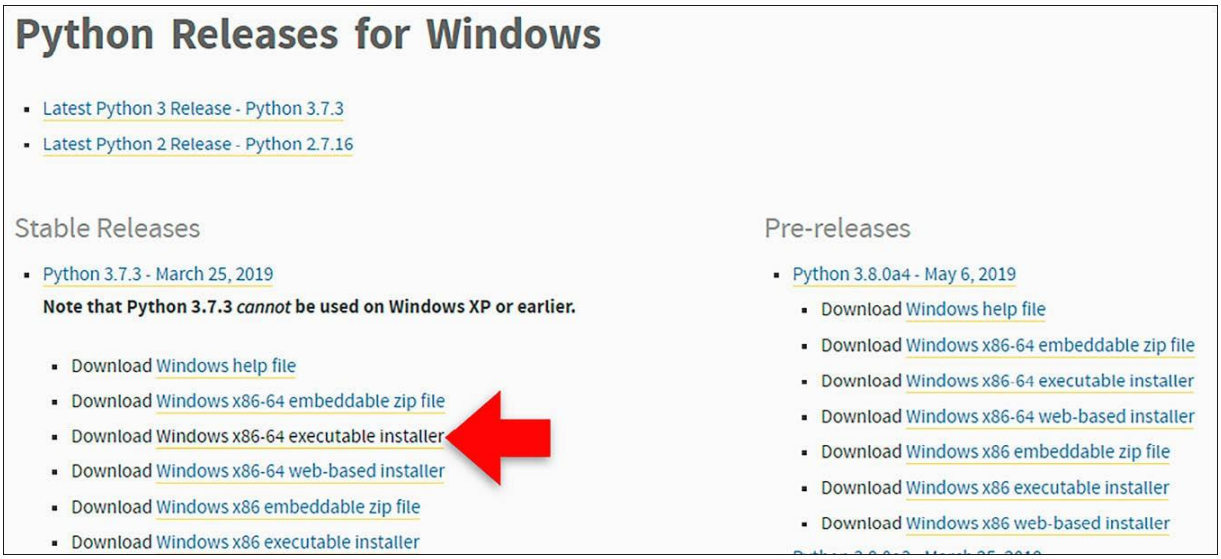
Установка в среде Windows

В нашей лаборатории мы используем рабочие станции под управлением Windows, поэтому нам необходимо установить среду разработки Python для Windows.

Откройте веб-браузер и перейдите на следующий веб-сайт

`www.python.org/downloads/windows`

На странице загрузок выберите ‘executable installer’ последней стабильной версии.



Нажмите «Выполнить» при появлении запроса в браузере. Или нажмите «python-x.x.x-amd64.exe», если вы используете Chrome.

Stable Releases

Python 3.7.3- March 25,2019

Note that Python 3.7.3 *cannot* **be used on Windows XP or earlier.**

- Download Windows help file
- Download Windows xBG-64 embeddable zip file
- Download Windows xSG-64 executable installer
- Download Windows x86·64 web-based installer
- Download Windows x86 embeddable zip file
- Download Windows x86 executable installer

python 3.7.3·amd64.exe "'

Pre-releases

Python 3.8.0a4- May 6,2019

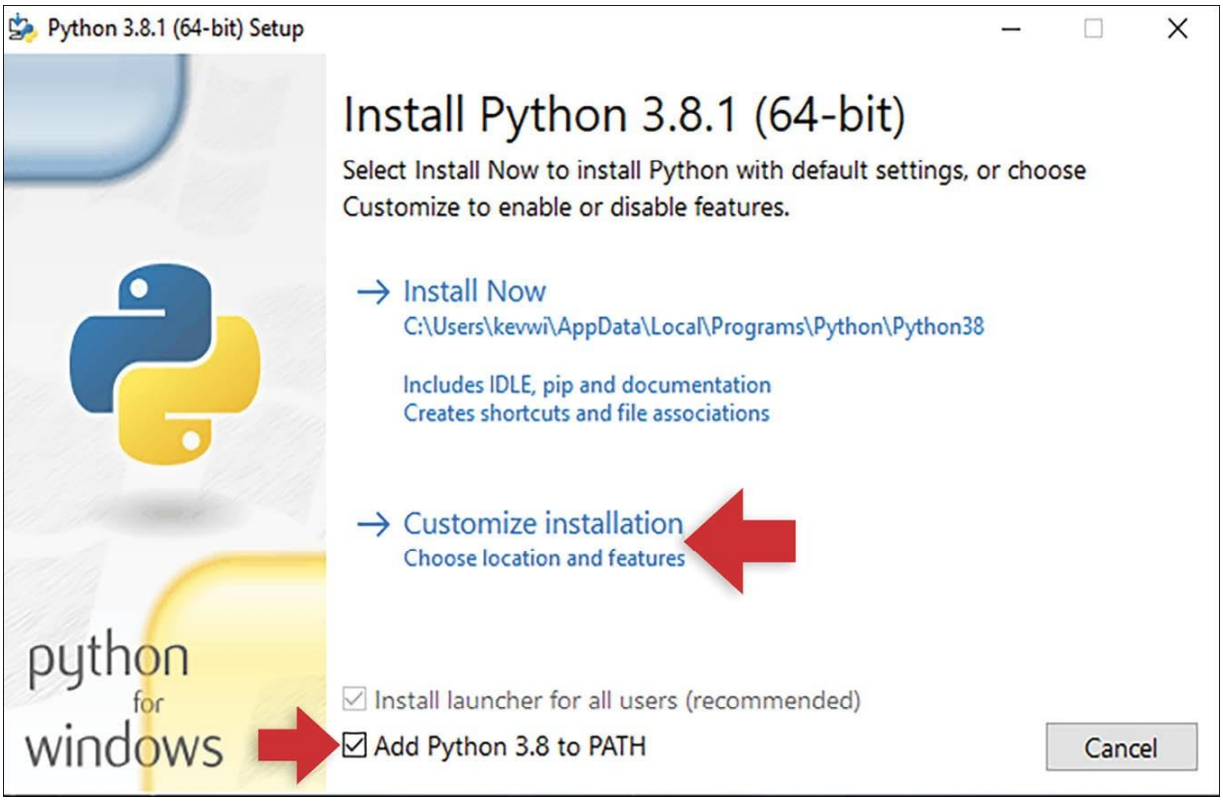
- Download Windows help file
- Download Windows x86-64 embeddable zip file
- Download Windows x8664 executable installer
- Download Windows xBG-64 web-based installer
- Download Windows x86 embeddable zip file
- Download Windows x86 executable installer
- Download Windows x86 web-based installer

Python 38.0a3- March 25,2019

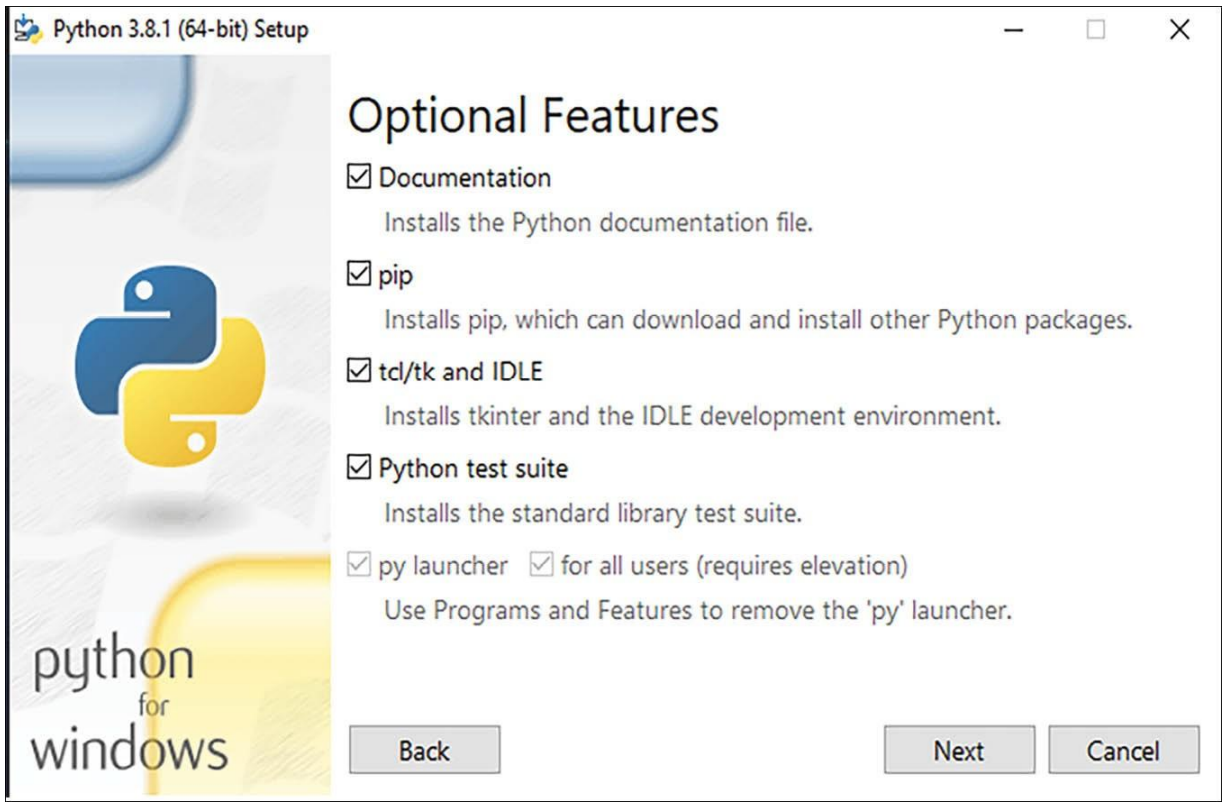
- Download Windows help file

Show all X

После запуска установщика убедитесь, что выбрано «Add python 3.x to path», затем нажмите «Customize installation», чтобы выполнить шаги для завершения установки.

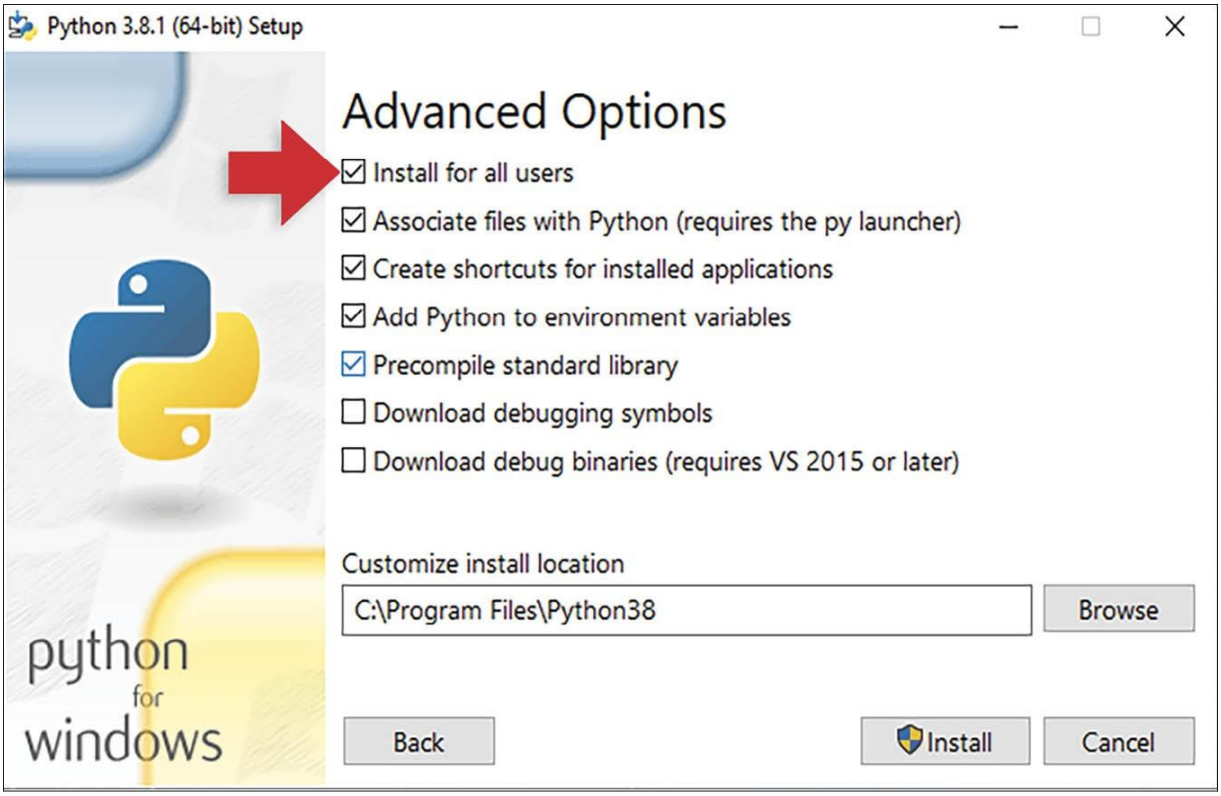


Убедитесь, что вы установили все флажки для всех дополнительных функций.

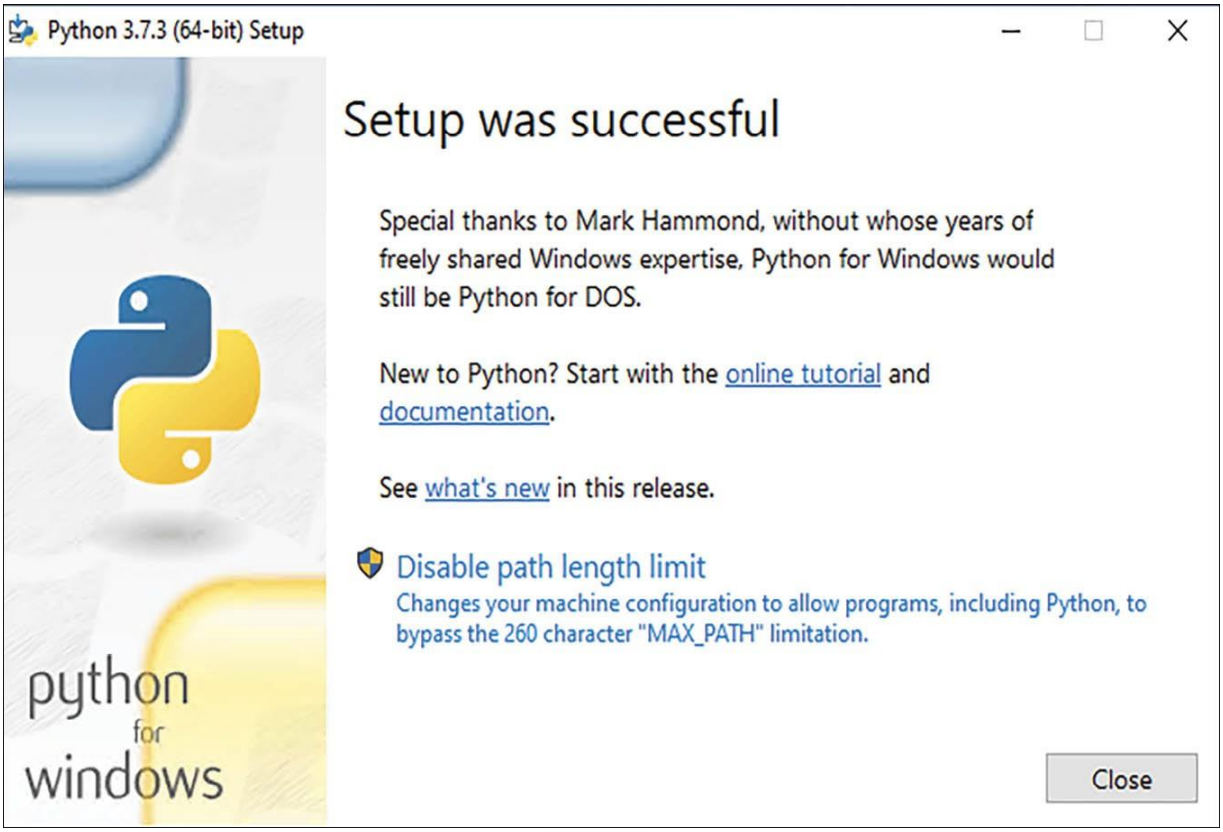


Нажмите «Next».

Убедитесь, что в верхней части диалогового окна выбран вариант «Install for all users». Нажмите «Install», чтобы начать процесс установки.

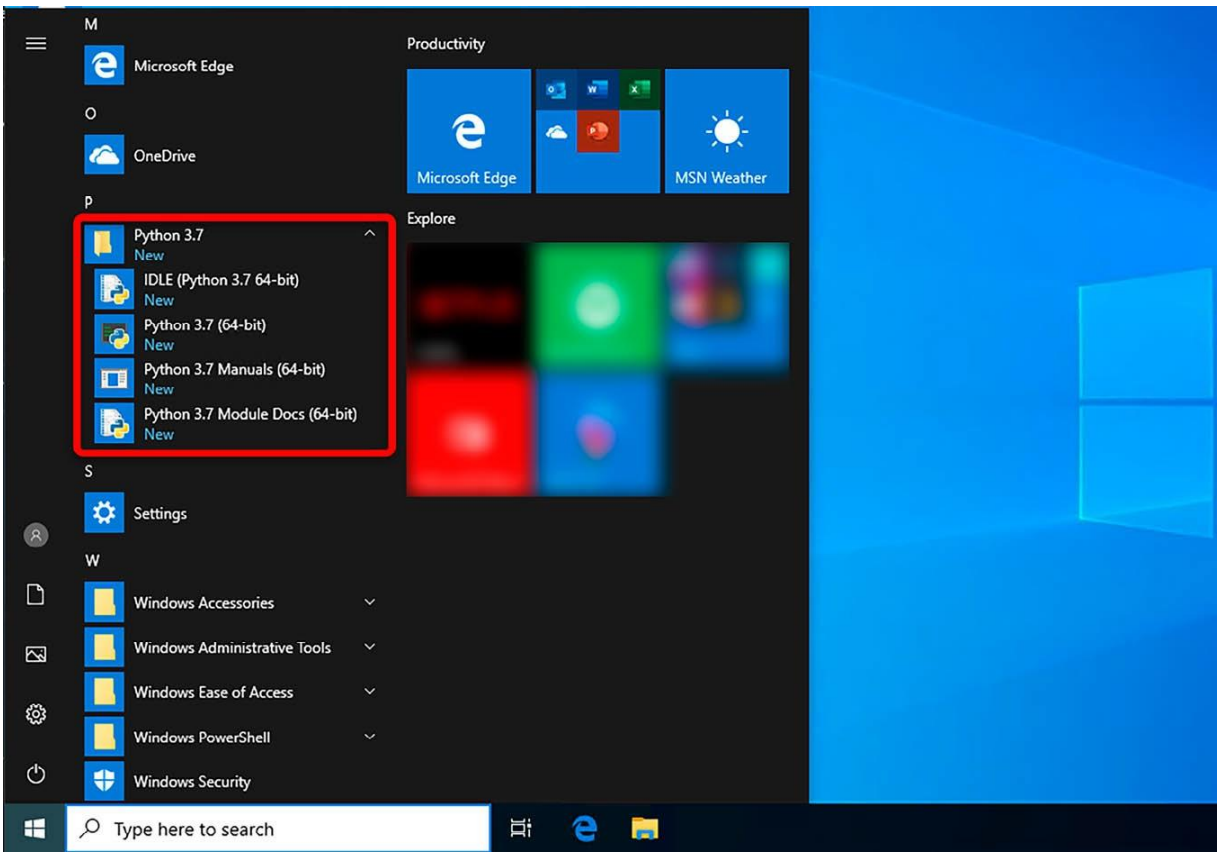


Нажмите «Disable path length limit», чтобы убедиться, что Python работает без проблем в Windows и разрешает длинные имена файлов..



Нажмите «Close», чтобы завершить процесс установки.

Вы найдете среду разработки Python (IDLE) и интерпретатор Python в папке Python в меню «Пуск».



Установка в среде MacOS

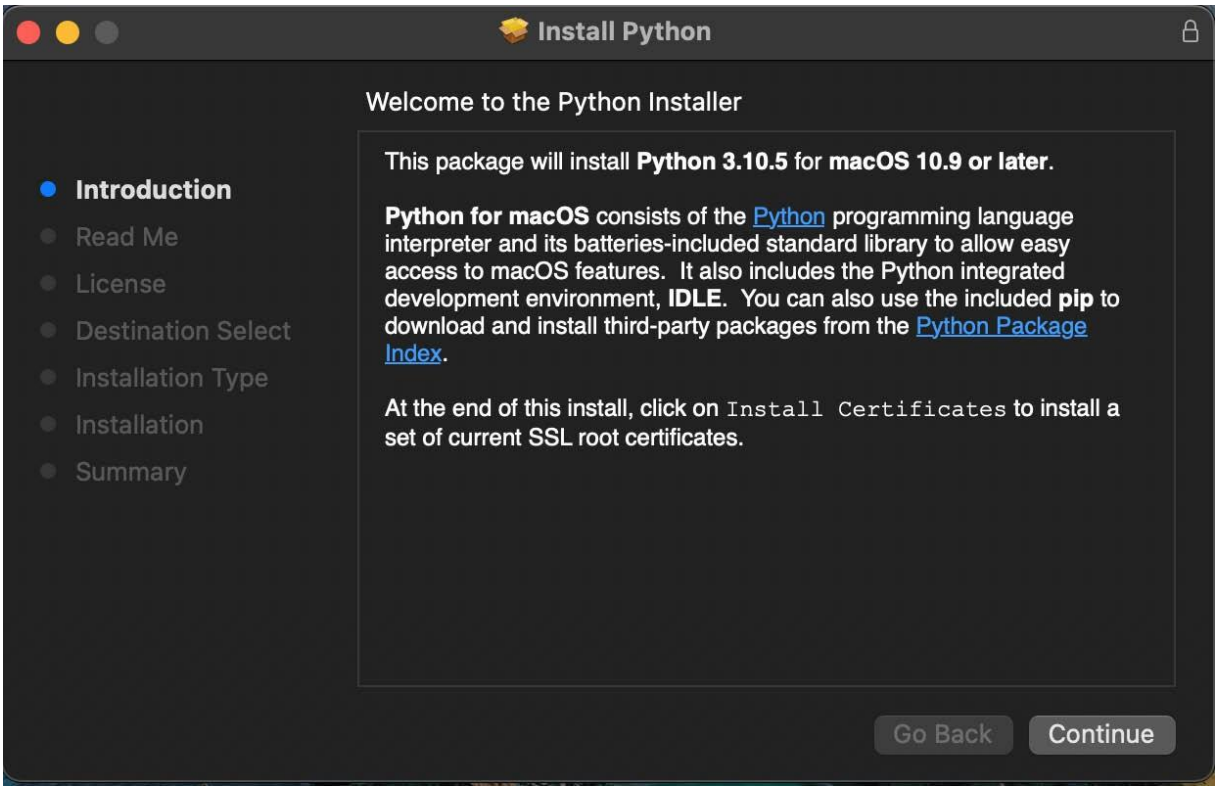
Чтобы установить Python 3 с помощью официального установщика, откройте веб-браузер и перейдите на следующий веб-сайт

`www.python.org/downloads/macos`

Нажмите Download Python.

Вы найдете пакет в папке загрузок. Дважды щелкните мышкой на пакете, чтобы начать установку

Запустите мастер установки. Нажмите «Continue».



После завершения установки вы найдете Python в папке приложений в Проводнике или на панели Пуск.

Установка в среде Linux

Если вы используете дистрибутив Linux, такой как Ubuntu, или у вас Raspberry Pi, вы можете установить Python с помощью терминального режима. Вы можете найти терминальное приложение в своих приложениях. Также вы можете нажать кнопки Control Alt T на клавиатуре.

В командной строке терминального окна введите следующие команды. Нажимайте Enter после ввода каждой строки.

```
sudo apt update
sudo apt upgrade
```

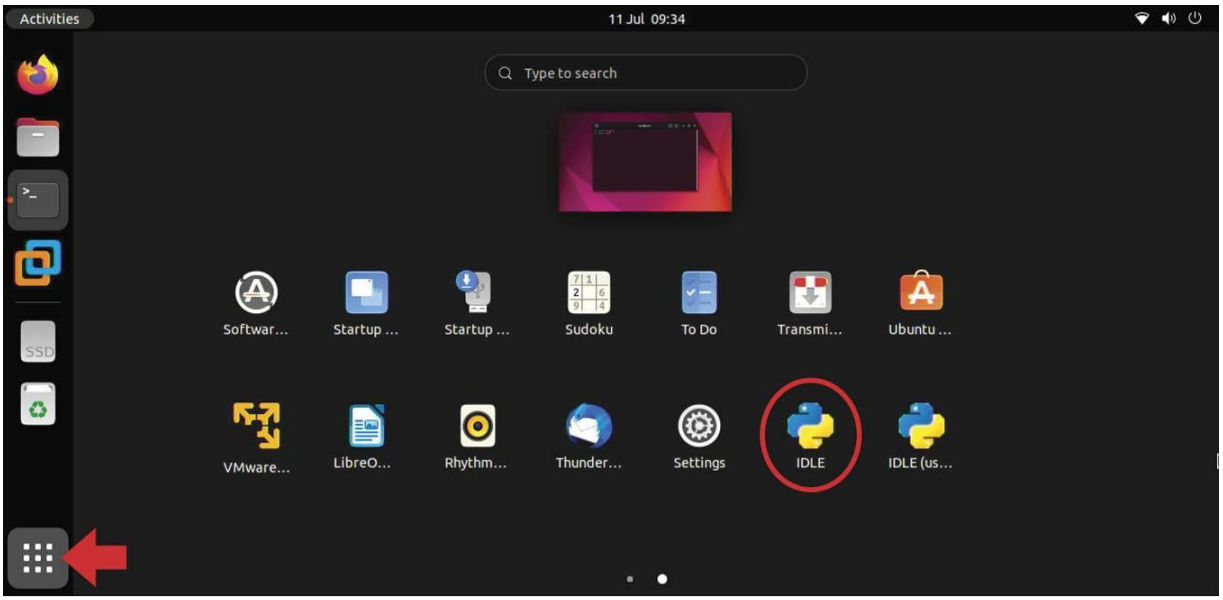
Введите следующую команду, чтобы установить Python.

```
sudo apt install python3 -y
```

После установки Python нам нужно установить IDLE, среду разработки. Для этого введите следующую команду в командной строке

```
sudo apt-get install idle3 -y
```

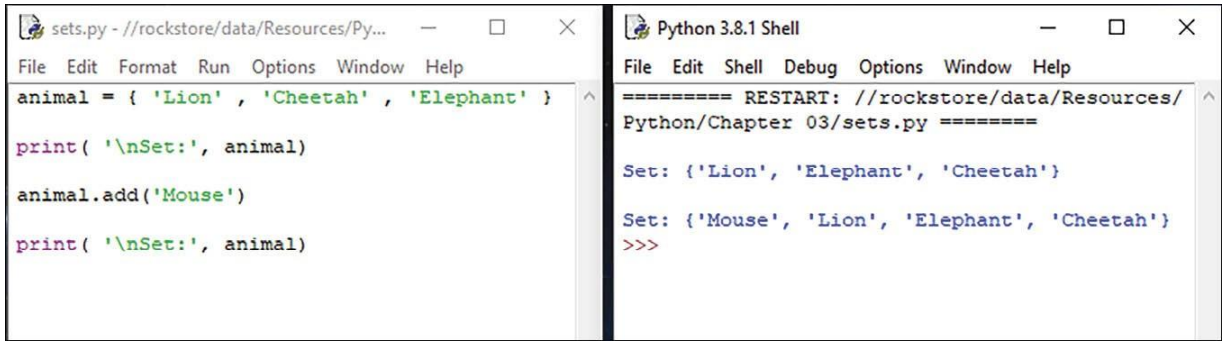
После установки вы найдете IDLE в своих приложениях.



Либо вы можете ввести следующую команду в командной строке

```
idle
```

Расположите окна так, чтобы вы могли видеть окно кода слева, а оболочку справа.



Здесь, в редакторе, вы можете написать свой код, а затем запустить его выполнение и отладку. Вы также заметите, что редактор кода обеспечивает подсветку синтаксиса, что означает, что ключевые слова и текст выделяются разными цветами, что упрощает чтение кода.

Настройка среды написания кода

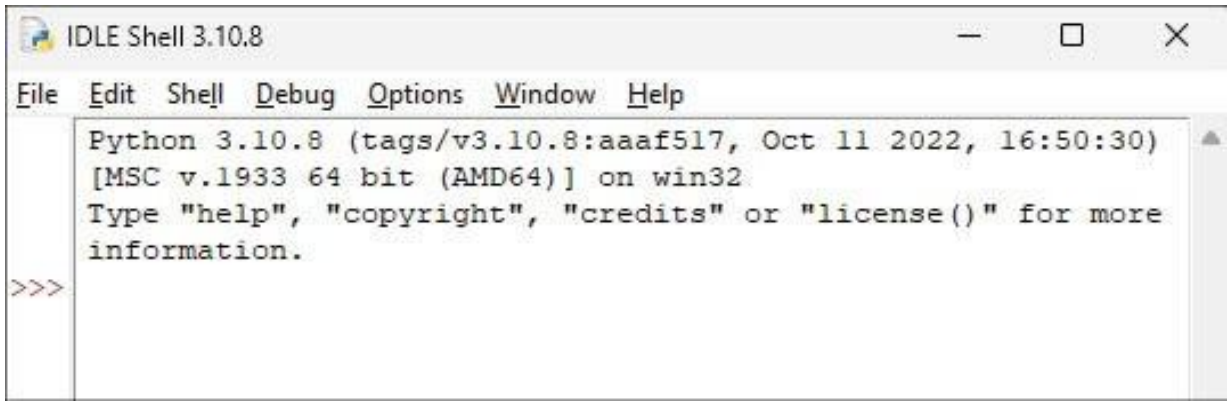
Существует бесчисленное множество сред разработки и редакторов кода, таких как

- Sublime Text
- IDLE
- Atom
- PyCharm
- Thonny
- PyDev и Visual Studio Code

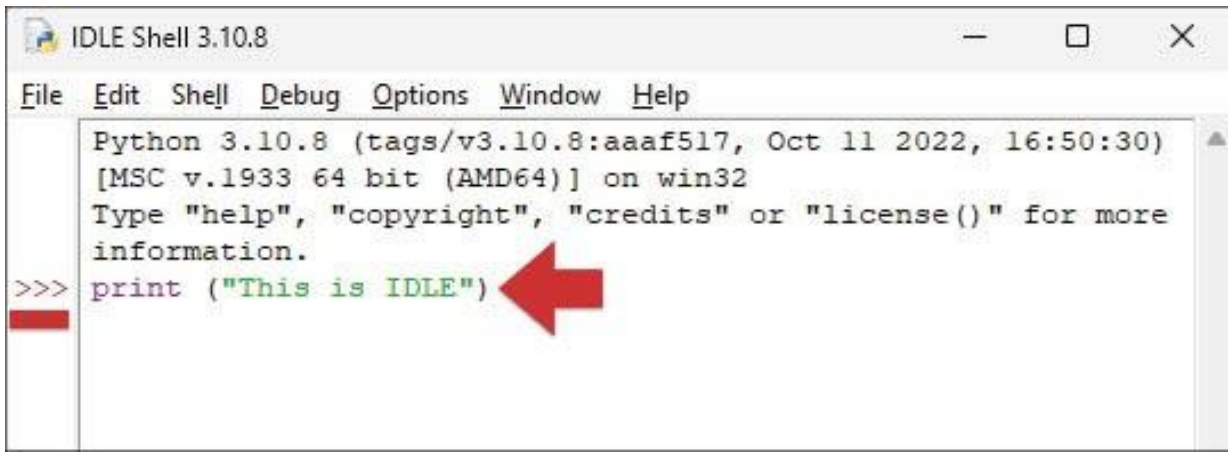
Для наших целей мы будем использовать IDLE, интегрированную среду разработки (IDE), поставляемую с Python.

Вы найдете IDLE в меню «Пуск» в Windows или в Finder/Launch Pad на Mac.

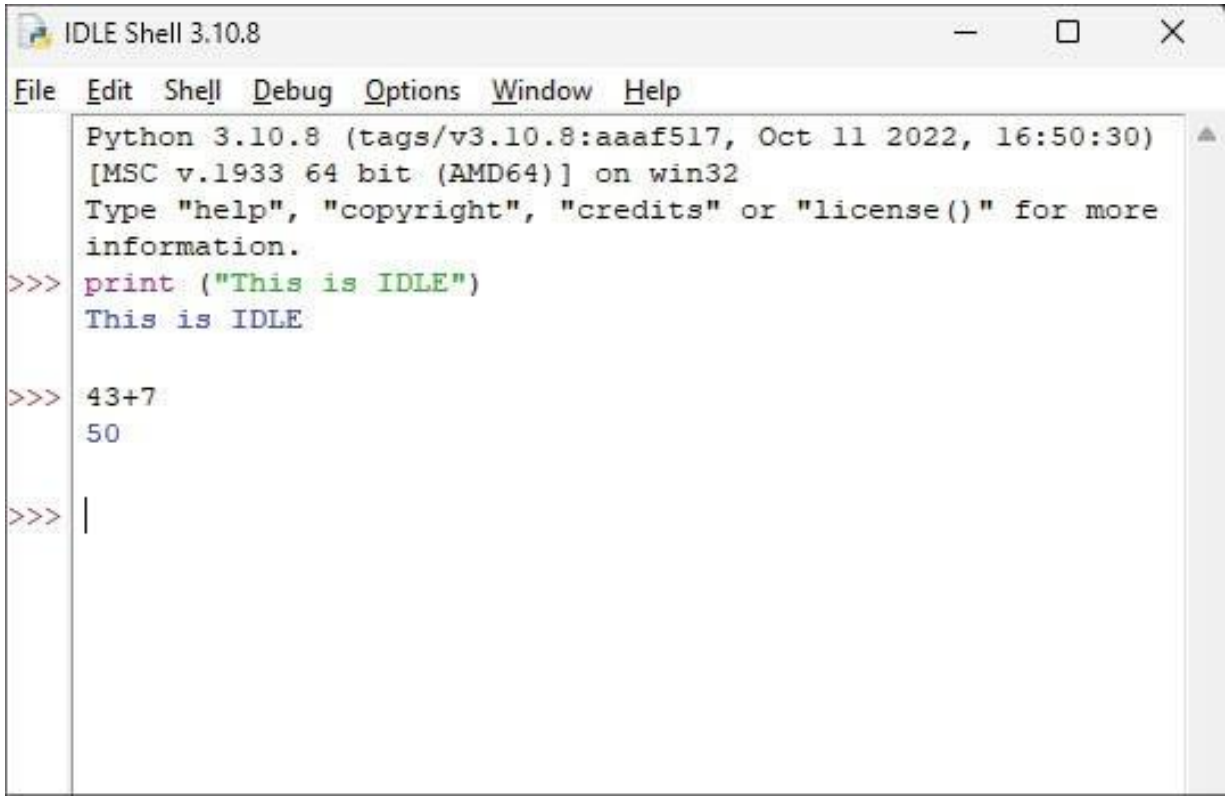
После запуска IDLE вы увидите окно оболочки.



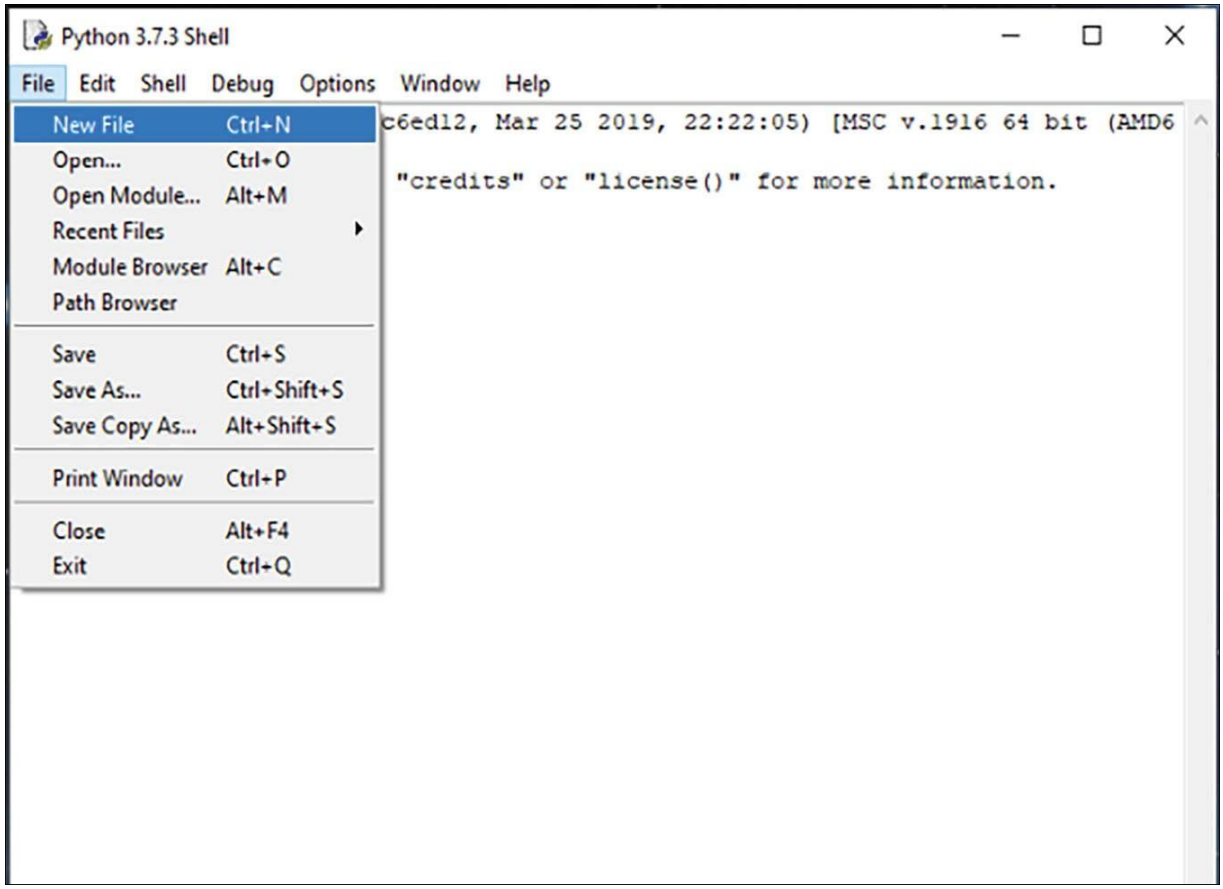
В ответ на приглашение '>>>' вы можете вводить и выполнять инструкции. Вы можете вводить математические операции, операции сравнения, получать вводимые пользователем данные, а интерпретатор Python будет их выполнять.



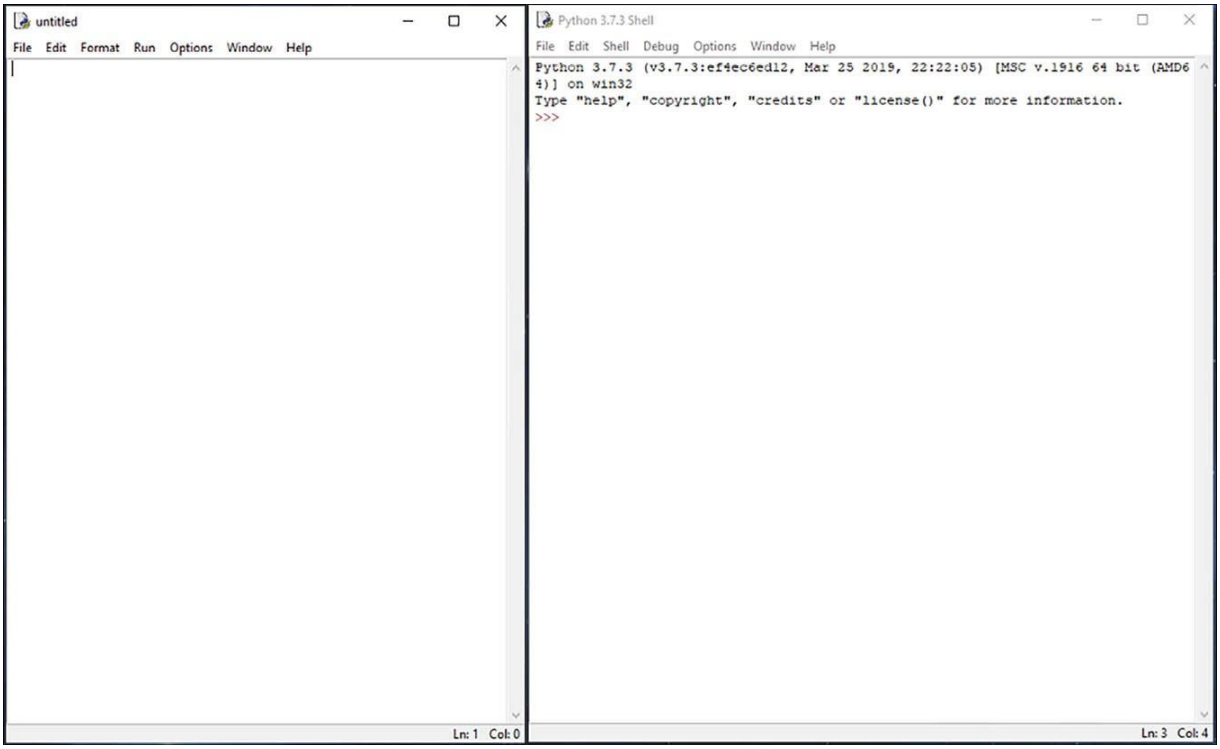
Здесь мы выполнили оператор печати и вычислили математическое выражение. Как только вы нажмете Enter, вы увидите ниже результат, отображаемый синим цветом.



Чтобы создать файл Python, выберите меню ‘File’ в верхнем левом углу окна оболочки.



Выровняйте окно редактора слева от окна оболочки, как это показано ниже.



Теперь вы можете ввести текст программы Python в окне редактора. Для сохранения файла выберите меню ‘File’, нажмите ‘Save’. Чтобы запустить программу, выберите меню ‘Run’ в окне редактора, нажмите ‘Run module’ (или нажмите F5).

Лабораторная работа

1. Настройте и установите Python и IDLE на свой компьютер.
2. Создайте на своем компьютере рабочий каталог для хранения всех ваших программ Python. Он может располагаться в папке с вашими документами, в зависимости от того, какую операционную систему вы используете.
3. Загрузите исходные файлы кода для этой книги и извлеките файл pythonfiles.zip в каталог, созданный вами выше. `elluminetpress.com/python`
4. Что такое компьютерная программа?
5. Как открыть новый файл в IDLE? Как сохранить программу в IDLE? В каком каталоге вы сохраняете свою программу? Как вы запускаете программу?
6. Где можно использовать язык Python?

Основы

Программы Python пишутся в текстовом редакторе, таком как Блокнот, PyCharm, или в редакторе кода в среде разработки Python (IDLE) и сохраняются с расширением файла .py.

Затем вы используете интерпретатор Python для выполнения кода, сохраненного в файле. Для этого раздела посмотрите демонстрационные видео

`elluminetpress.com/pybasics`

Посмотрите файлы в каталоге Chapter 02.

Давайте начнем с самого начала и узнаем, что такое язык программирования.

Язык программирования — это метод выражения набора кратких инструкций, которые должны выполняться компьютером.

Классификация языков

Существуют языки программирования разных уровней: языки низкого уровня и языки высокого уровня.

Язык низкого уровня

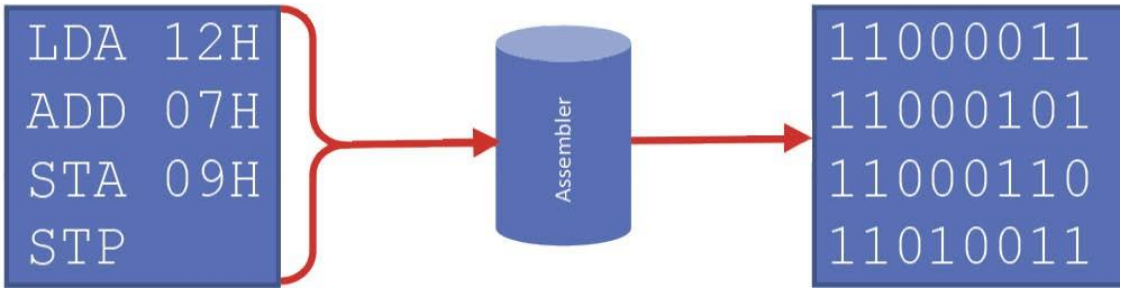
Язык низкого уровня — это язык программирования, функции которого часто относятся непосредственно к инструкциям процессора и обычно пишутся на машинном коде или языке ассемблера.

Язык ассемблера известен как язык программирования второго поколения, причем машинный код является первым поколением.

Давайте рассмотрим простую программу. Здесь у нас приведена небольшая программа суммирования, написанная на языке ассемблера для нашего процессора, и она может выглядеть примерно так

```
LDA 12H
ADD 07H
STA 09H
STP
```

Код пишется на языке ассемблера, а затем перед его выполнением компилируется в машинный код с помощью ассемблера.



Каждая инструкция языка ассемблера соответствует последовательности двоичных чисел в машинном коде. Числа, символы, адреса и другие данные преобразуются в их эквиваленты в машинном коде.

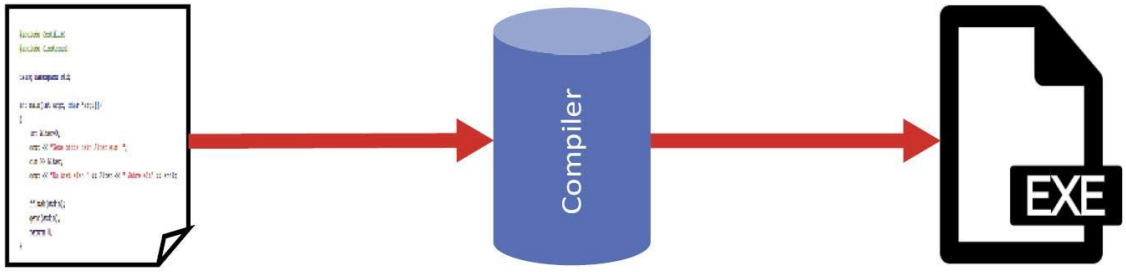
Итак, LDA можно представить двоичным кодом: 11000011, число 12_{10} в двоичном виде равно 00001100.

Собранный машинный код затем выполняется процессором.

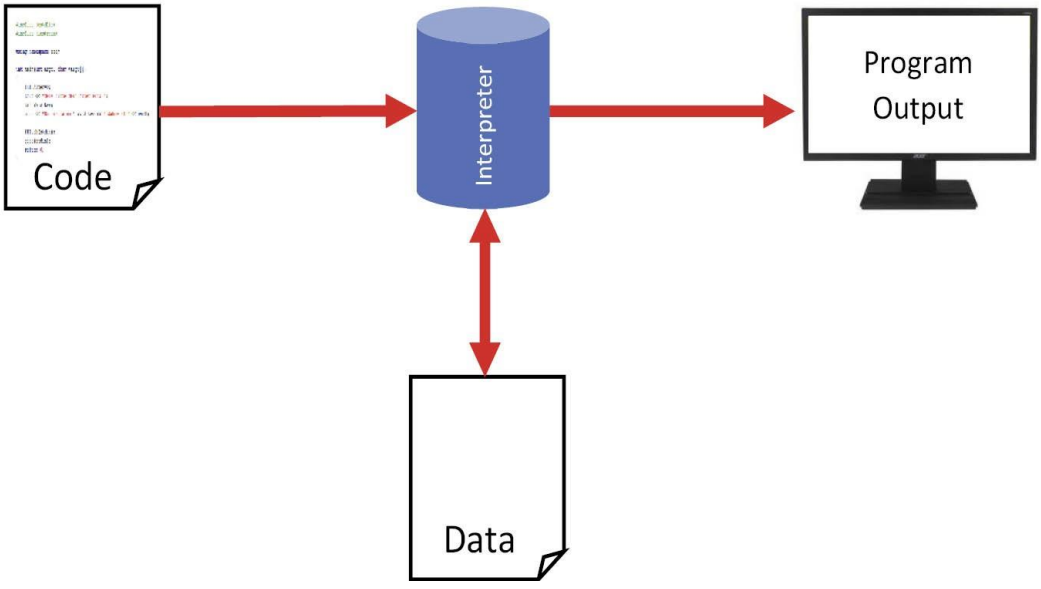
Язык высокого уровня

Python является примером языка высокого уровня. Вместо того чтобы напрямую работать с регистрами процессора и адресами памяти, языки высокого уровня имеют дело с переменными, удобочитаемыми операторами, циклами и функциями.

Код языка высокого уровня либо компилируется в исполняемую программу машинного кода, либо интерпретируется. Такие языки, как C или C++, часто компилируются, то есть код пишется, а затем преобразуется в исполняемый файл. Это делает их идеальными для разработки программного обеспечения, позволяющего писать такие приложения, как Microsoft Word, которые работают на компьютере.



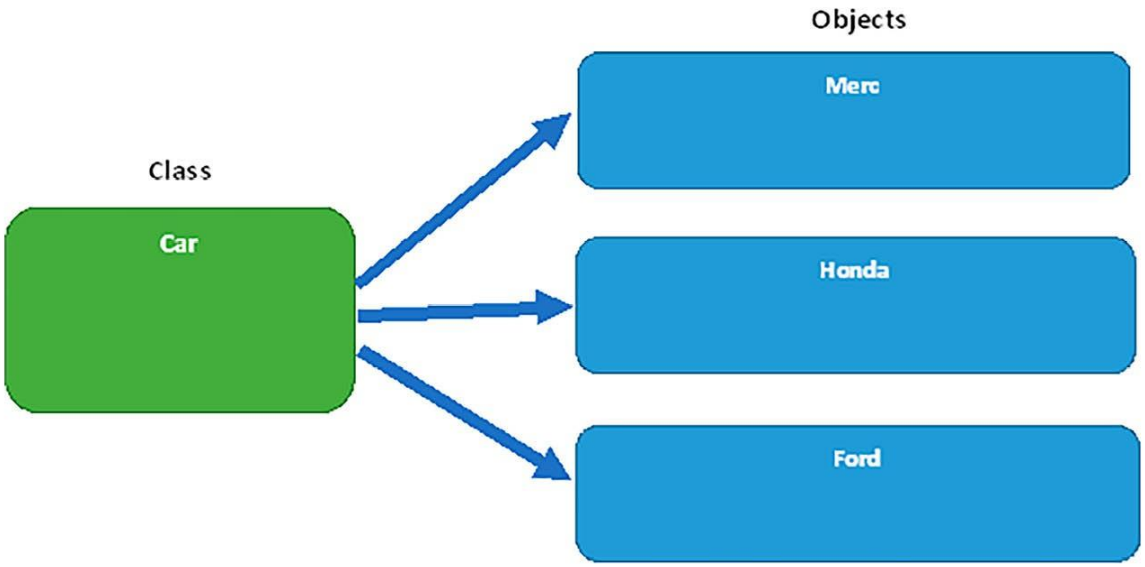
Python — это интерпретируемый язык, то есть код, который вы пишете, напрямую переводится в машинный код, что делает его хорошо подходящим для веб-разработки.



Здесь вы можете видеть, как интерпретатор выполняет код построчно, получая доступ к любым данным, необходимым программе, а затем отображает результат прямо на экране.

Объектно-ориентированное программирование

Python — объектно-ориентированный язык программирования. Это означает, что стрктура программы основан на объектах, а не на функциях и логике.



Мы рассмотрим это более подробно в главе 9, а пока просто ознакомьтесь с терминологией.

Класс

Класс — это определяемая пользователем схема или шаблон, который определяет атрибуты (переменные) и методы (функции) и содержит их все в одном модуле, называемом классом.

Объект

Объект — это экземпляр класса. Итак, вы можете иметь класс под названием ‘car’ и использовать его для определения объекта с наименованием ‘merc’, ‘taxi’ и т.д.

Атрибут

An attribute, is similar to a variable and is used temporarily store data The attribute can only be accessed from within the object where it’s defined. For example: `object.attribute`

Метод

Метод — это функция, которая используется объектом для выполнения действия. На методы обычно ссылаются, указывая наименование объекта и через точку наименование метода. Например: `object.method()`

Синтаксис языка Python

Синтаксис определяет, как пишется и интерпретируется программа, и составляет основу написания кода.

Зарезервированные слова

Это слова, зарезервированные языком программирования, и которые определяют синтаксис и структуру. Вот некоторые из наиболее распространенных из них:

and	Логический оператор, обычно используемый в операторах if
as	Создает псевдоним
assert	Для условий тестирования и использования при отладке
break	Выход из цикла
class	Создает класс
continue	Продолжает переход к следующей итерации цикла
def	Определяет функцию
del	Удаляет объект
elif	Используется в условных операторах, так же, как и else if
else	Используется в условных операторах
except	Определяет код, который будет запускаться при возникновении ошибки (исключения)
false	Логическое значение, результат операций сравнения
finally	Используется с исключениями, блок кода, который будет выполнен независимо от того, есть исключение или нет
for	Создает цикл for
from	Используется для импорта только указанного раздела из модуля.
global	Объявляет глобальную переменную
if	Создает условный оператор
import	Импортирует модуль
in	Используется для проверки наличия значения в последовательности
is	Используется для проверки того, относятся ли две переменные к одному и тому же объекту
lambda	Используется для создания небольших анонимных функций
None	Представляет собой нулевое значение
nonlocal	Используется для работы с переменными внутри вложенных функций,
not	Логический оператор
or	Логический оператор
pass	Ничего не делает, используется как заполнитель
raise	Используется для создания исключения.
return	Используется, чтобы выйти из функции и вернуть значение
true	Логическое значение, результат операций сравнения
try	Используется для создания выражения try...except

while	Создает цикл while
with	Используется для упрощения обработки исключений
yield	Используется для завершения функции, возвращает генератор

Например, слово while указывает на цикл while. Слово «if» определяет «оператор if». Вы не можете использовать зарезервированное слово в качестве имени переменной или наименования функции.

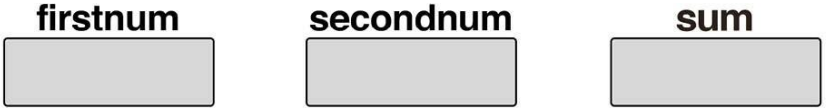
Идентификаторы

Идентификатор — это наименование, присвоенное классу, функции или переменной. Идентификаторы могут представлять собой комбинацию прописных или строчных букв, цифр или символа подчеркивания _. Идентификаторы не могут быть ключевыми словами, и вы не можете использовать в них специальные символы, такие как *, ?, !, @, # или $. Идентификаторы чувствительны к регистру, они не могут содержать пробелы, а первый символ не может быть цифрой. Постарайтесь, чтобы идентификаторы были осмысленными, чтобы они описывали, для чего они используются.

```
printData, firstVariable, _count, userCount
```

Переменные

Переменная — это помеченное место в памяти, которое используется для хранения значений компьютерной программы. Существует два типа переменных: локальные и глобальные.



Переменные, определенные внутри функции, называются локальными переменными, поскольку они являются переменными только этой конкретной функции. Эти переменные может видеть только та функция, в которой они определены. Эти переменные имеют локальную область видимости.

Здесь «sum», «firstnum» и «secondnum» являются локальными переменными функции «addsum».

```
def addsum(firstnum, secondnum):
```

```
sum = firstnum + secondnum

return sum
```

Глобальные переменные определяются в основной части программы, вне каких-либо конкретных функций. Эти переменные могут быть просмотрены любой функцией, и про них говорят, что они имеют глобальную область видимости.

В этом примере «a» и «b» являются глобальными переменными.

```
a = int(2)

b = int(3)

def addsum(firstnum, secondnum):

sum = ffirstnum + secondnum

return sum
```

Отступ

Большинство других языков программирования, таких как C и C++, используют фигурные скобки { } для определения блока кода. Python использует отступы. Используйте для этого клавишу табуляции.

C++

```
If test condition {
   execute this block if true;
} else {
   otherwise execute this block;
}
```

Python

```
if test condition:
   execute this block if true
else:
   otherwise execute this block
```

Это станет более понятным, когда мы начнем использовать операторы и циклы if/else в главе 4 и функции в главе 6.

Комментарии

Комментарии очень важны при написании программы. Вы должны четко документировать весь свой код с помощью комментариев, чтобы другие разработчики, работающие над проектом, могли лучше понять, что делает ваш код.

Используйте символ решетки (#) для написания однострочных комментариев.

```
# Prompt user for two numbers
a = input (‘Enter first number: ’)
b = input (‘Enter second number: ’)
```

Если вам нужно написать многострочный комментарий с описанием функциональности, используйте тройные кавычки до и после блока комментариев.

Например:

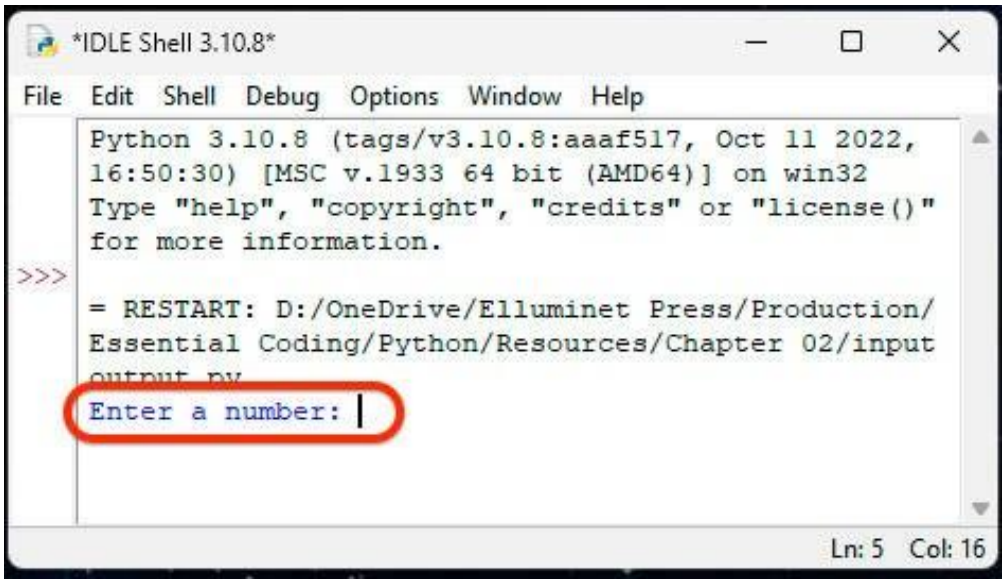
```
“““ Prompt user for two numbers
one after the other using a text input ”””
a = input (‘Enter first number: ’)
b = input (‘Enter second number: ’)
```

Ввод

Вы можете получить ввод от пользователя, используя функцию `input()`. Эта функция предлагает пользователю ввести некоторые данные и присваивает их переменной «number».

```
number = input (‘Enter a number: ’)
```

Когда мы выполняем эту строку, функция ввода отображает приглашение на консоли оболочки. Введенные данные будут присвоены переменной.



Вывод

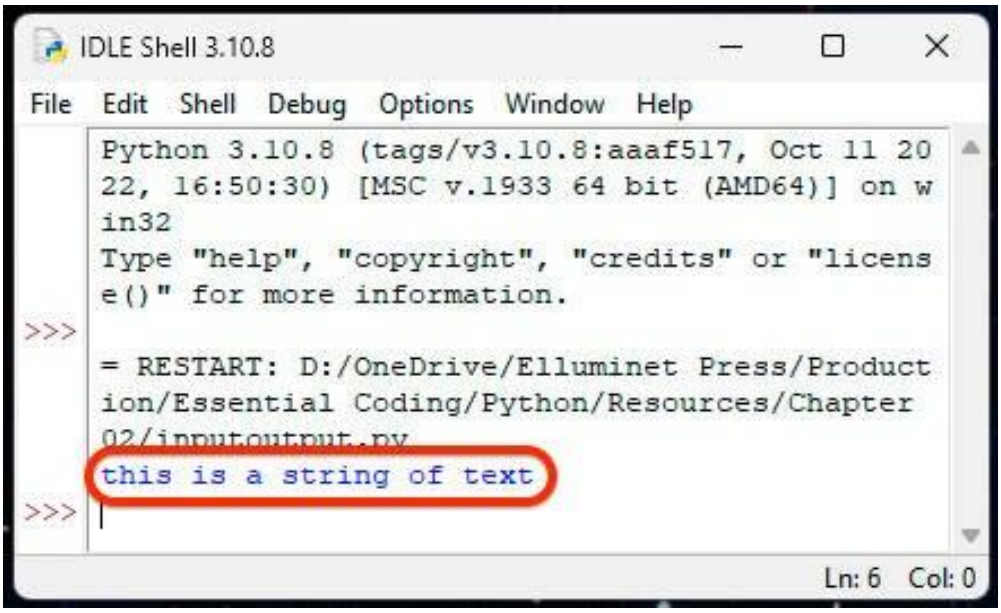
Вывести информацию на экран можно с помощью функции `print()`. Вы можете распечатать содержимое переменной или использовать строку в качестве параметров функции `print()`. Например:

```
print (number)
```

...или...

```
print ('this is a string of text')
```

Когда мы выполняем эту строку, текст выводится на консоли оболочки.



Функции

Функция — это блок кода, который выполняется только при его вызове. Python поставляется с различными встроенными функциями, такими как

```
print(), input(), format(), int()
```

Функция принимает параметры или аргументы, заключенные в круглые скобки, и возвращает результат. Кажется, что большинство людей используют оба этих термина как взаимозаменяемые, но между ними есть разница.

- Параметр — это переменная в определении функции.
- Аргумент — это значение, передаваемое во время вызова функции.

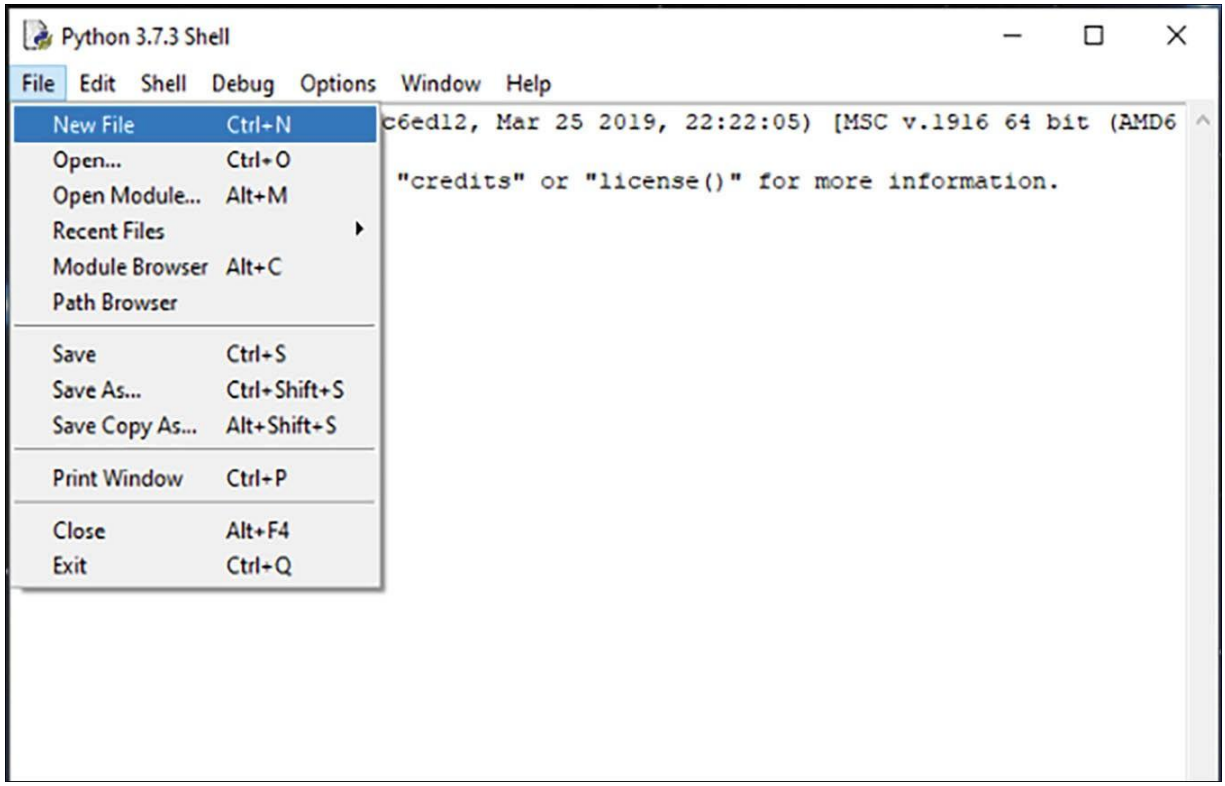
Вы также можете создавать свои собственные функции. Эти функции называются пользовательскими функциями. Например:

```
def functionName (parameters):
<function code>
```

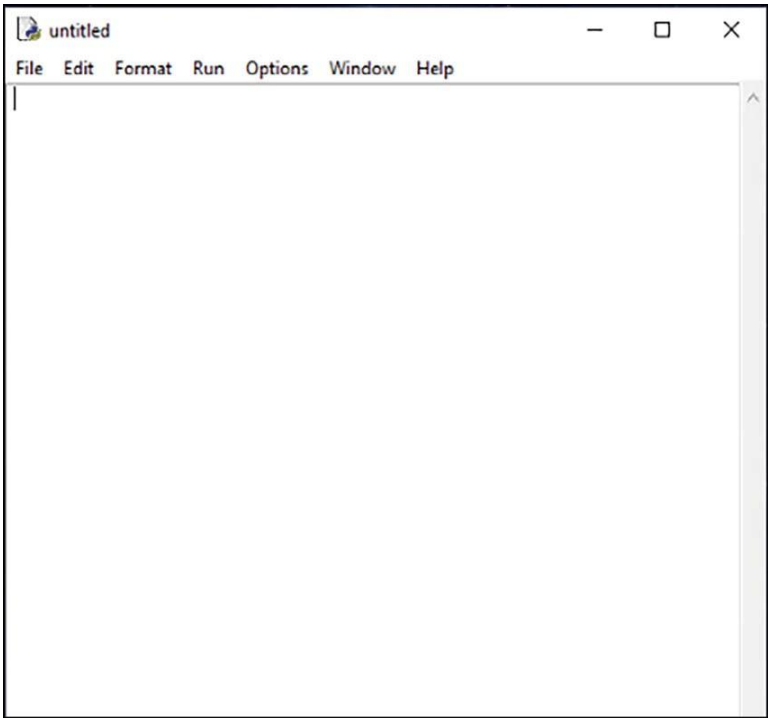
Мы рассмотрим функции в главе 6.

Написание программы

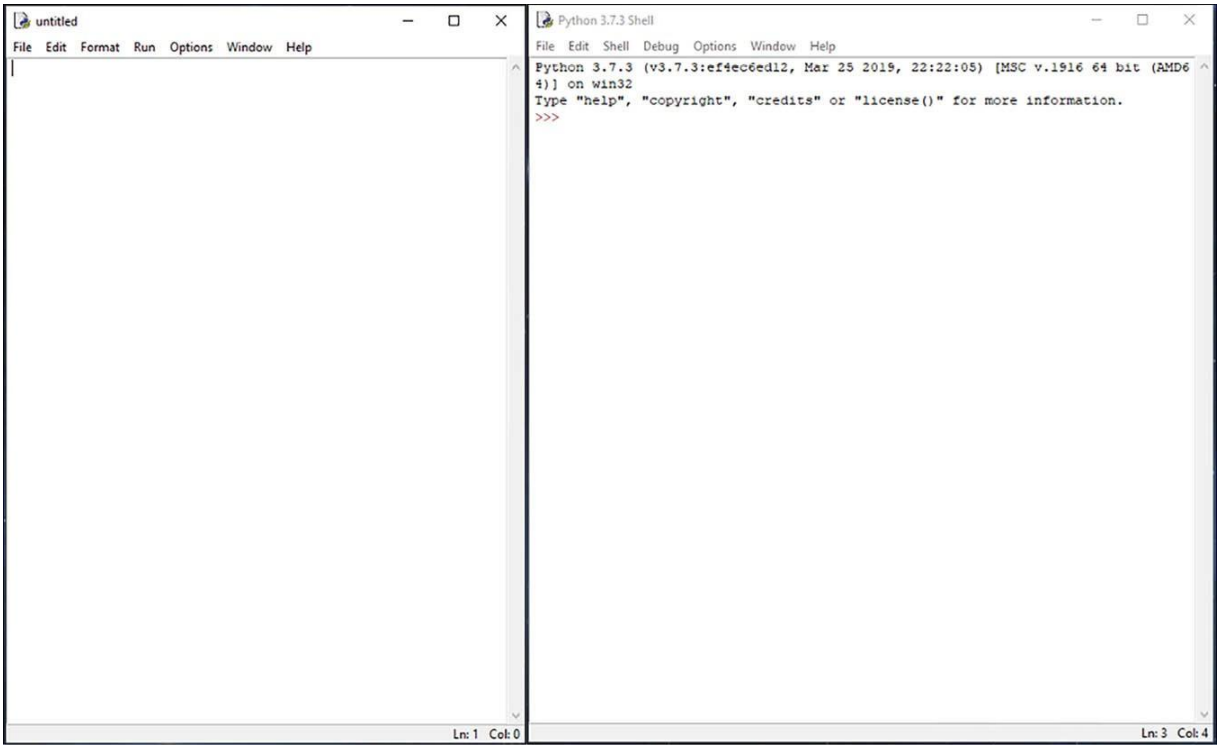
Чтобы написать программу, откройте IDLE Python из меню «Пуск». Выберите меню «File», затем нажмите «New File».



Появится новое пустое окно. Это редактор кода. Здесь вы можете написать весь свой код на Python.



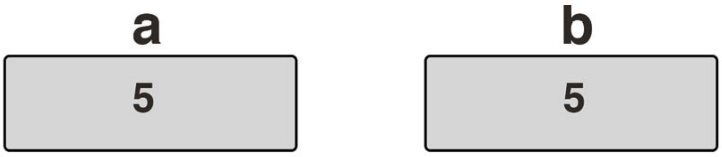
Расположите окна, как это показано ниже, расположив оболочку Python с правой стороны — здесь вы увидите результаты работы своих программ. Окно редактора кода поместите рядом с окном оболочки Python слева.



В качестве нашей первой программы мы собираемся написать код,

который складывает два числа, а затем отображает результат.

Во-первых, нам нужны две переменные для хранения чисел.

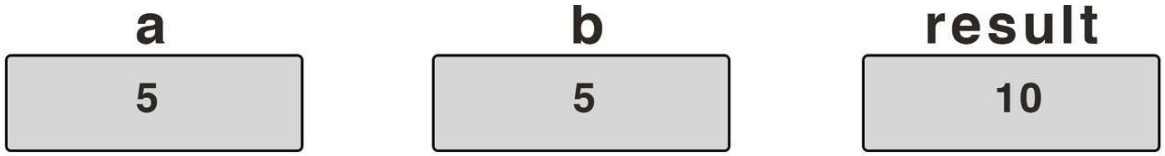


Мы будем использовать «a» и «b». Каждой переменной мы присвоим число 5.

```
a = 5
b = 5
```

Далее нам нужен фрагмент кода, который будет складывать два числа и сохранять результат.

В этом случае значения, присвоенные переменным «a» и «b», будут суммироваться и сохраняться в переменной ‘result’.



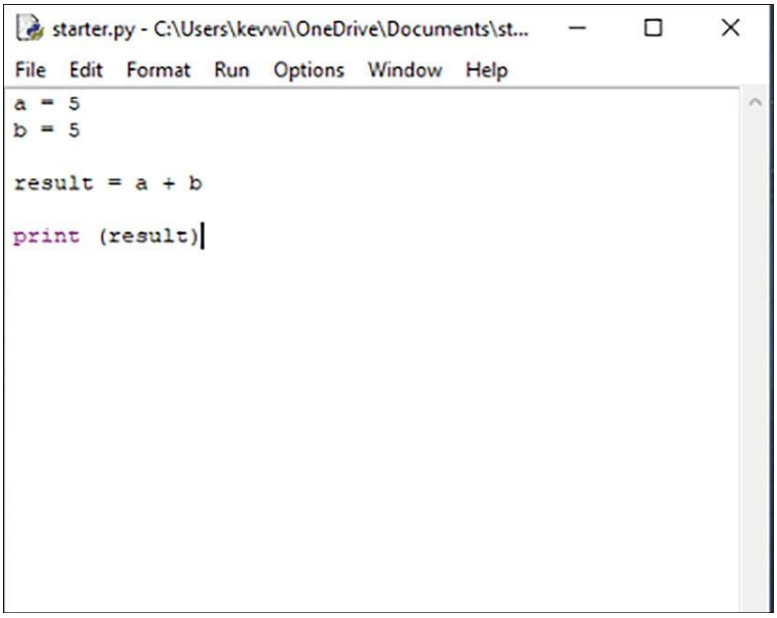
Итак...

```
result = a + b
```

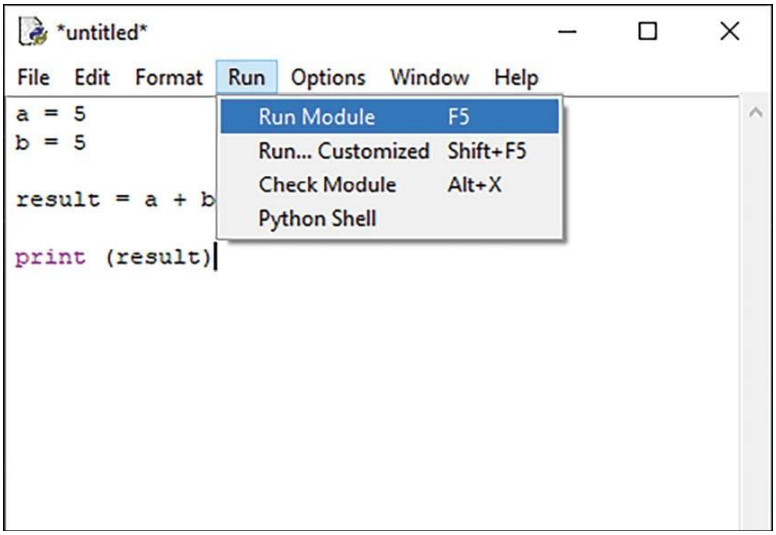
Далее нам понадобится функция для вывода результата на экран.

```
print (result)
```

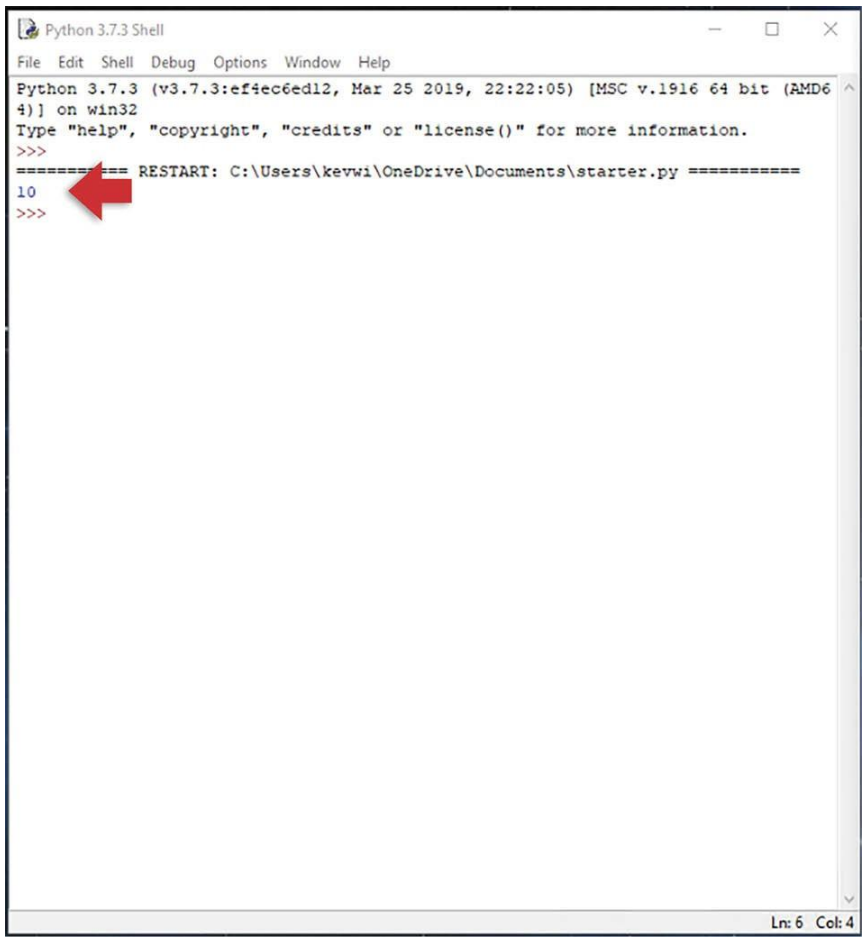
Давайте объединим все это в программе.



Чтобы запустить программу, нажмите F5 или перейдите в меню «Run» в редакторе кода и нажмите «Run Module».



На изображении ниже вы можете увидеть результат работы программы. В данном случае «10».



Эта конкретная программа не очень полезна. Было бы намного лучше, если бы мы могли позволить пользователю вводить числа, которые он хочет сложить. Для этого нам нужно добавить функцию, которая будет запрашивать у пользователя значения.

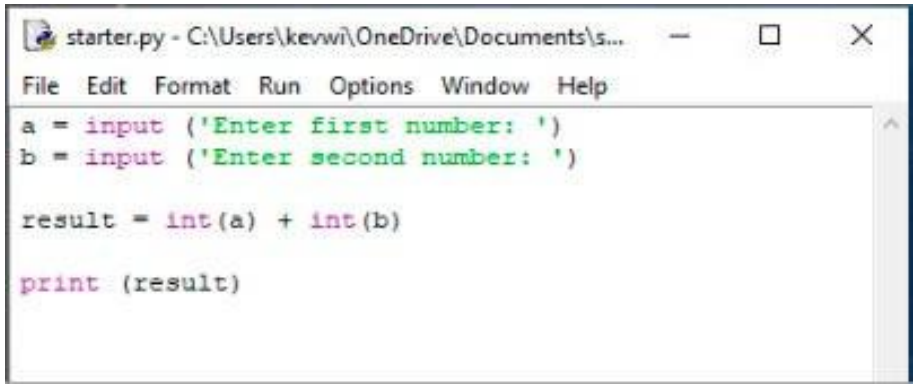
Мы будем использовать функцию `input()`. Мы можем заменить переменные «a» и «b» из предыдущей программы функцией ввода.

```
a = input ('Enter first number: ')
b = input ('Enter second number: ')
```

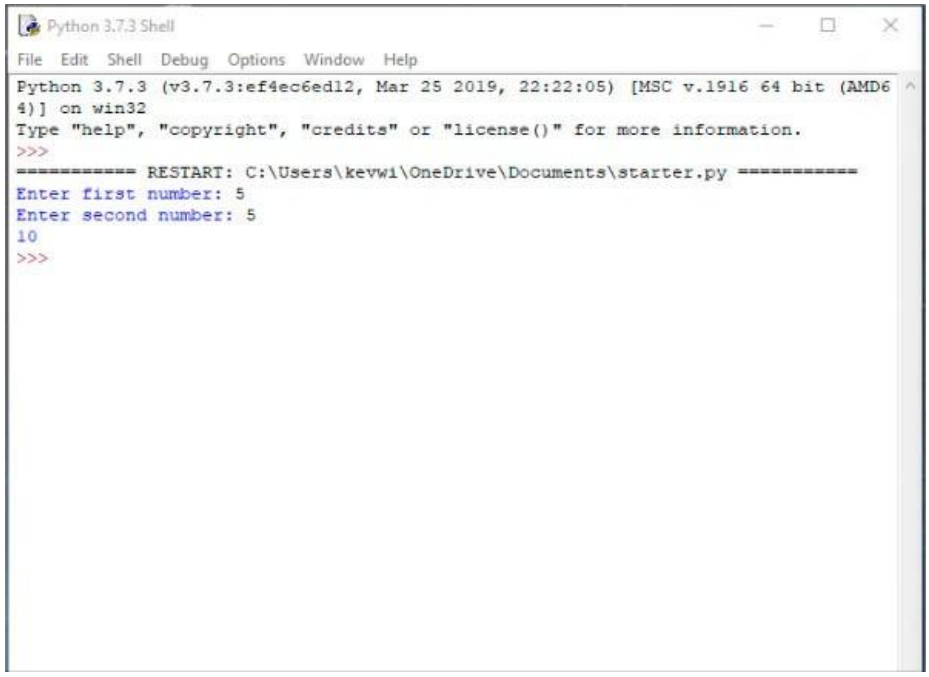
Теперь, поскольку функция ввода считывает введенные значения в виде текста (называемого строкой), нам необходимо преобразовать их в числа. Поэтому нам нужно изменить код, который складывает два числа. Мы можем использовать функцию `int()` — она преобразует текст в целое число.

```
result = int(a) + int(b)
```

Давайте объединим все это в программе.



На изображении ниже вы можете увидеть результат работы программы. Программа запросила у пользователя два числа, сложила их вместе, а затем отобразила результат в нижней строке.



Лабораторная работа

1. Какой результат дает следующий фрагмент кода?

```
num1 = 2
num2 = 3
print (num1 + num2)
```

2. Какой результат дает следующий фрагмент кода?

```
num1 = 2
num2 = 3
print (“num 1 + num 2 = “, num1 + num2)
```

3. Найдите ошибки в следующей программе

```
Num1 = 2
num2 := 3
Sum = num1 + num2;
printf(sum)
```

4. Что такое идентификатор?

5. Какие из приведенных ниже идентификаторов действительны, а какие недействительны? Почему?

```
Num1
time-of-day
tax_rate
x5
int
7th_Rec
yield
```

6. Как вы пишете комментарии в своем коде? Объясните на примере.
7. Почему вам следует включать комментарии?
8. Что такое зарезервированное слово?
9. Что такое язык низкого и высокого уровня?

10. Что такое объектно-ориентированное программирование?

Работа с данными

Вы можете хранить и манипулировать всеми типами данных: числом, строкой, списком и т. д. Для хранения данных мы используем переменную, которая является временным контейнером для хранения значений данных.

В Python вам не нужно объявлять все переменные перед их использованием, поскольку они автоматически объявляются при первом назначении каких-либо данных.

Переменные могут содержать различные типы данных, такие как строка, число, список элементов и т. д.

Для этого раздела посмотрите демонстрационные видео

`elluminetpress.com/pybasics`

Вам также понадобятся исходные файлы из каталога Chapter 03.

Основные типы данных

Переменная может хранить различные виды данных, называемые типом данных. Давайте представим некоторые основные типы, которые мы будем рассматривать.

Целые числа

Целое число (int) — это целое число, которое может быть положительным или отрицательным. Целые числа обычно могут иметь неограниченную длину.

```
score = 45
```

Числа с плавающей запятой

Число с плавающей запятой (float), иногда называемое действительным числом, представляет собой число, имеющее десятичную точку.

```
temperature = 44.7
```

Строки

В коде Python строка (str) должна быть заключена в кавычки ‘’...’’ или ‘...’, и вы можете присвоить строку переменной, используя знак «=».

```
name = “John Myers”
```

Если вы хотите присвоить многострочную строку, заключите ее в тройные кавычки """

```
message = """Hi, I would like to know
where I can find..."""
```

Конкатенация

Конкатенация означает объединение двух строк. Для этого можно использовать оператор +.

```
word0ne = "Hi! "

wordTwo = "My name is..."

sentence = word0ne + wordTwo
```

Результатом будет предложение, представляющее собой объединение двух строк:

```
"Hi! My name is..."
```

Срез

Вы можете разрезать строку, то есть вернуть из строки диапазон символов. Например, если мы хотим разрезать следующее

Index:	0	1	2	3	4	5	6	7	8
Str:	E	L	L	U	M	I	N	E	T

Мы указываем начальный индекс и конечный индекс, разделенные двоеточием.

```
sliced = Str[4:8]
```

Здесь мы вырезаем с 4 до 8 (последняя позиция не учитывается)

Index:	0	1	2	3	4	5	6	7	8
Str:	E	L	L	U	M	I	N	E	T

Это вернет результат

```
MINE
```

Строковые методы

Python предоставляет множество методов для работы со строками. Например, если мы хотим посчитать, сколько раз в строке появляется буква «L»

```
Str = 'ELLUMINET'
```

Вы можете использовать метод count()

```
charCount = Str.count('L')
```

Вот несколько других распространенных строковых методов. Попробуйте их.

Функция	Описание	Пример
count()	Возвращает, сколько раз в строке встречается указанное значение	`Str.count("string to count")`
find()	Ищет в строке указанное значение и возвращает позицию, где оно было найдено	`Str.find("chars to find")`
format()	Форматирует указанные значения в строке	`Str.format()`

join()	Возвращает новую строку, которая представляет собой объединение строк.	`Str.join("string 1", "string 2"...)`
lower()	Преобразует строку в нижний регистр.	`Str.lower()`
replace()	Создает новую строку, заменяя некоторые части другой строки.	`Str.replace("value to replace", "new value")`
split()	Разбивает строку на список строк на основе разделителя, например запятой.	`Str.split()`
upper()	Преобразует строку в верхний регистр.	`Str.upper()`

Управляющие символы

Управляющий символ предписывает интерпретатору выполнить определенную операцию, такую как разрыв строки или табуляция, или зарезервированный символ, такой как кавычка или апостроф. Управляющие символы начинаются с обратной косой черты (\) и используются для форматирования строки.

Управляющий символ	Функция
\n	Разрыв строки
\t	Табуляция (горизонтальный отступ)
\	Новая строка в многострочной строке
\\	Обратная косая черта
\’	Апостроф или одинарная кавычка
\"	Двойная кавычка

Например, вы можете использовать символ табуляции и разрыва строки, чтобы отформатировать текст.

```
print(“John \t 45 \nJoanne \t 15”)
```

Вывод в этой строке будет выглядеть примерно так:

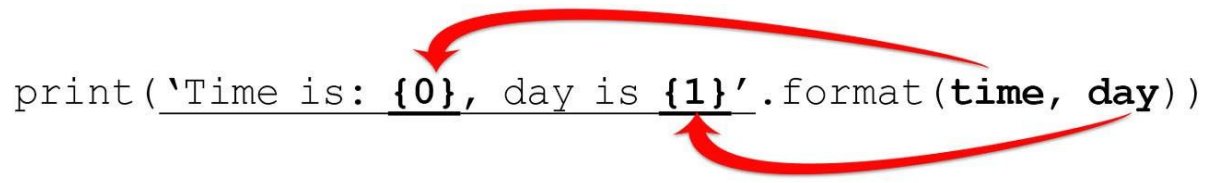
```
John 45
Joanne 15
```

Форматирование строк

Форматирование вывода с использованием метода `.format()` использует { } для обозначения замены переменной в выходной строке функции `print()`.

```
print(`Time is: {0}`.format(var))
```

Они известны как поля формата и заменяются объектами, передаваемыми в метод `.format()`. Мы можем использовать число в скобках для обозначения положения объекта, переданного в метод `format()`.



Строковые символы

Мы можем получить доступ к отдельным символам, используя индекс. Помните, что индекс начинается с 0.

Если бы мы объявили

```
Str = 'ELLUMINET'
```

Мы получили бы что-то вроде этого::

Index:	0	1	2	3	4	5	6	7	8
Str:	E	L	L	U	M	I	N	E	T

Для индексации любого из символов необходим элемент Str с индексом в квадратных скобках.

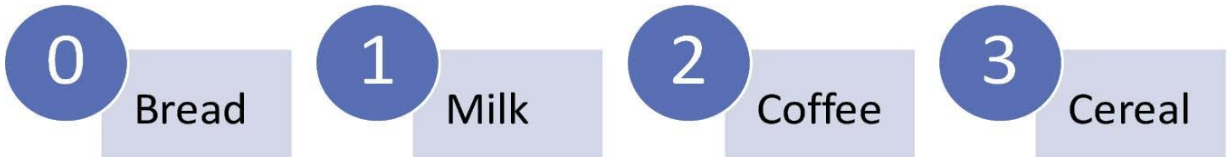
```
Str[index]
```

Итак, в приведенном ниже примере Chr = 'L'

```
Chr = Str[2]
```

Списки

Список — это упорядоченная последовательность элементов данных, обычно одного типа, каждый из которых идентифицируется индексом (показанным ниже в кружках). Этот пример известен как одномерный список.



В других языках программирования списки называются массивами, и вы можете создать их следующим образом: элементы списка заключаются в квадратные скобки []:

```
shoppingList = ['bread', 'milk', 'coffee', 'cereal']
```

Чтобы сослаться на элемент в списке, поместите ссылку на него в квадратные скобки:

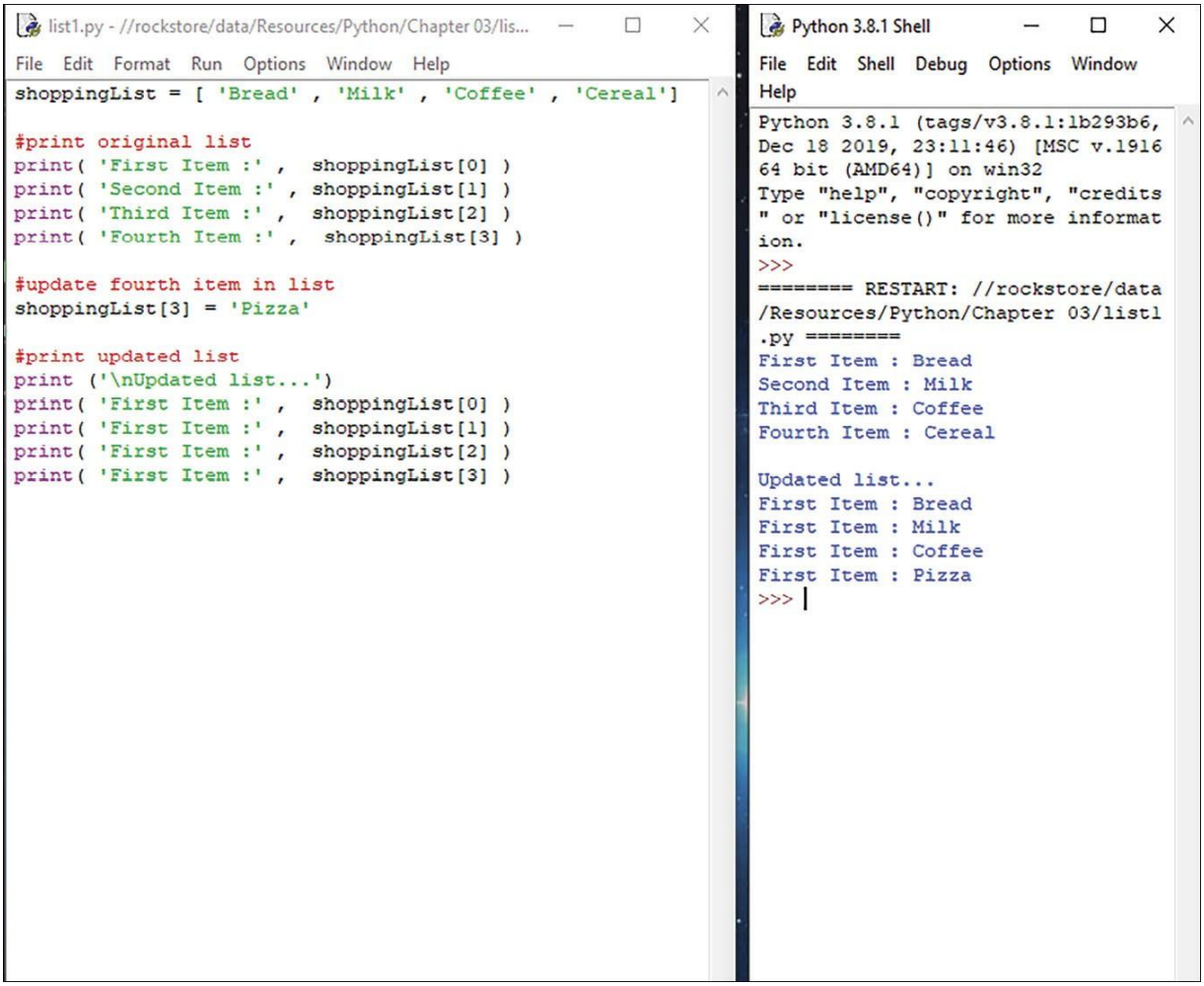
```
print (shoppingList[1])
```

Вы можете присвоить другое значение элементу в списке (например, заменить хлопья или Cereal)

```
shoppingList[3] = "chocolate"
```

В итоге у вас получится что-то вроде этого

Давайте рассмотрим программу. Откройте файл list1.py. Здесь мы создали список и наполнили его некоторыми данными.



Мы можем вывести данные из списка, используя оператор печати и ссылку на элемент в списке покупок.

```
print (shoppingList[3])
```

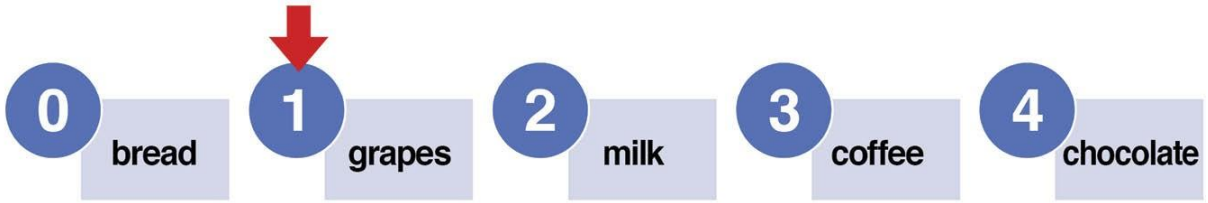
Мы также можем обновить элемент в списке, используя ссылку.

```
shoppingList[3] = 'pizza'
```

Чтобы вставить элемент – укажите индекс, по которому нужно вставить элемент. Например, чтобы вставить слово «grapes» по индексу 1

```
shoppingList.insert(1, 'grapes')
```

Этот код вставляет «grapes» в положение индекса [1] и сдвигает элементы после места вставки вправо.



Чтобы добавить элемент, используйте `append()`. Он добавит элемент в конец списка.

```
shoppingList.append('blueberry')
```

Чтобы удалить элемент по его значению

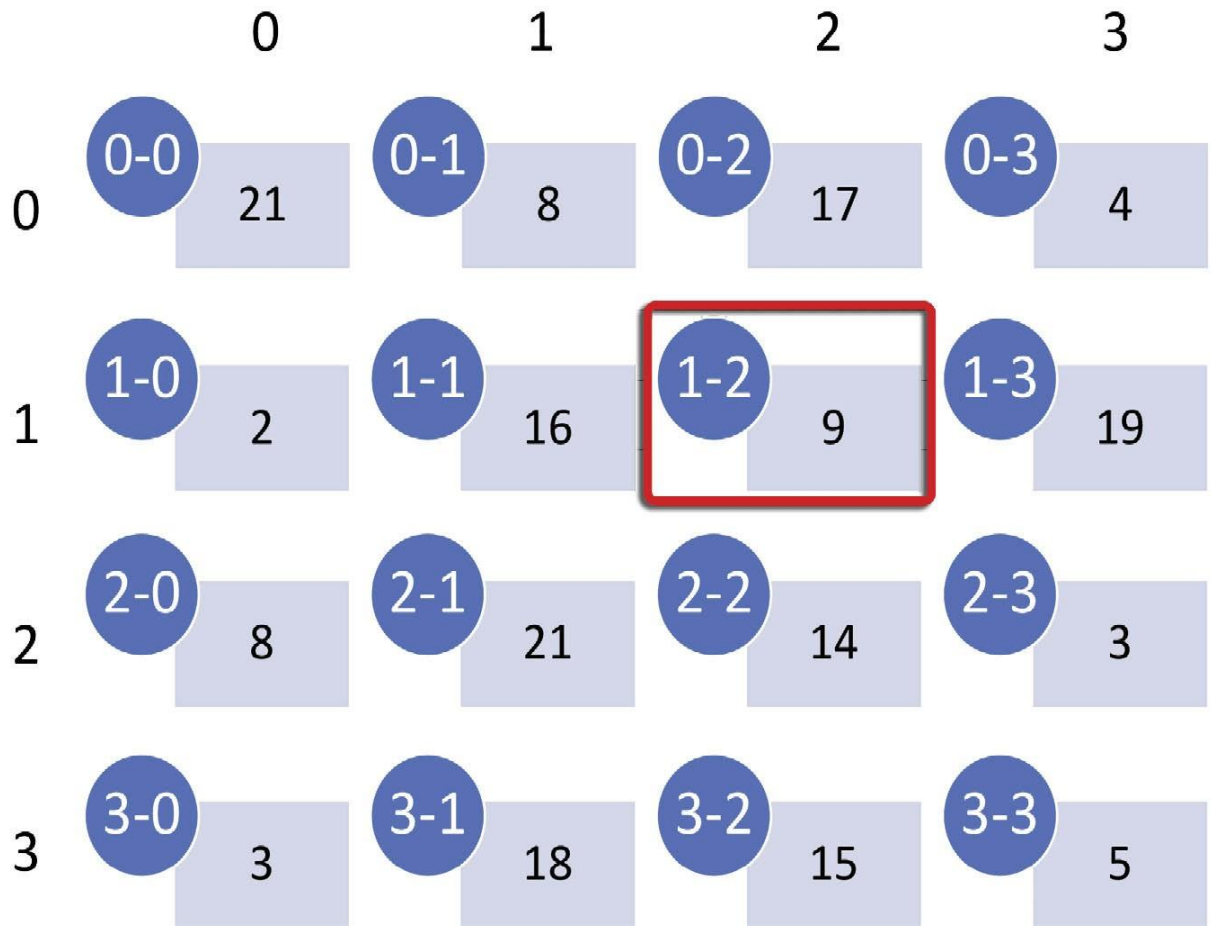
```
shoppingList.remove('blueberry')
```

Удаление элемента по заданному индексу

```
shoppingList.pop(1)
```

Двумерные списки

Двумерные списки визуализируются как сетка, похожая на таблицу со строками и столбцами. Каждый столбец в сетке нумеруется, начиная с 0. Аналогично, каждая строка нумеруется, начиная с 0. Каждая ячейка в сетке идентифицируется индексом — индексом строки, за которым следует индекс столбца (показан ниже в кружках).



Вы можете объявить приведенный выше список как

```
scoreSheet   =
[
[ 21, 8, 17, 4 ],
[ 2, 16, 9, 19 ],
[ 8, 21, 14, 3 ],
[ 3, 18, 15, 5 ]
]
```

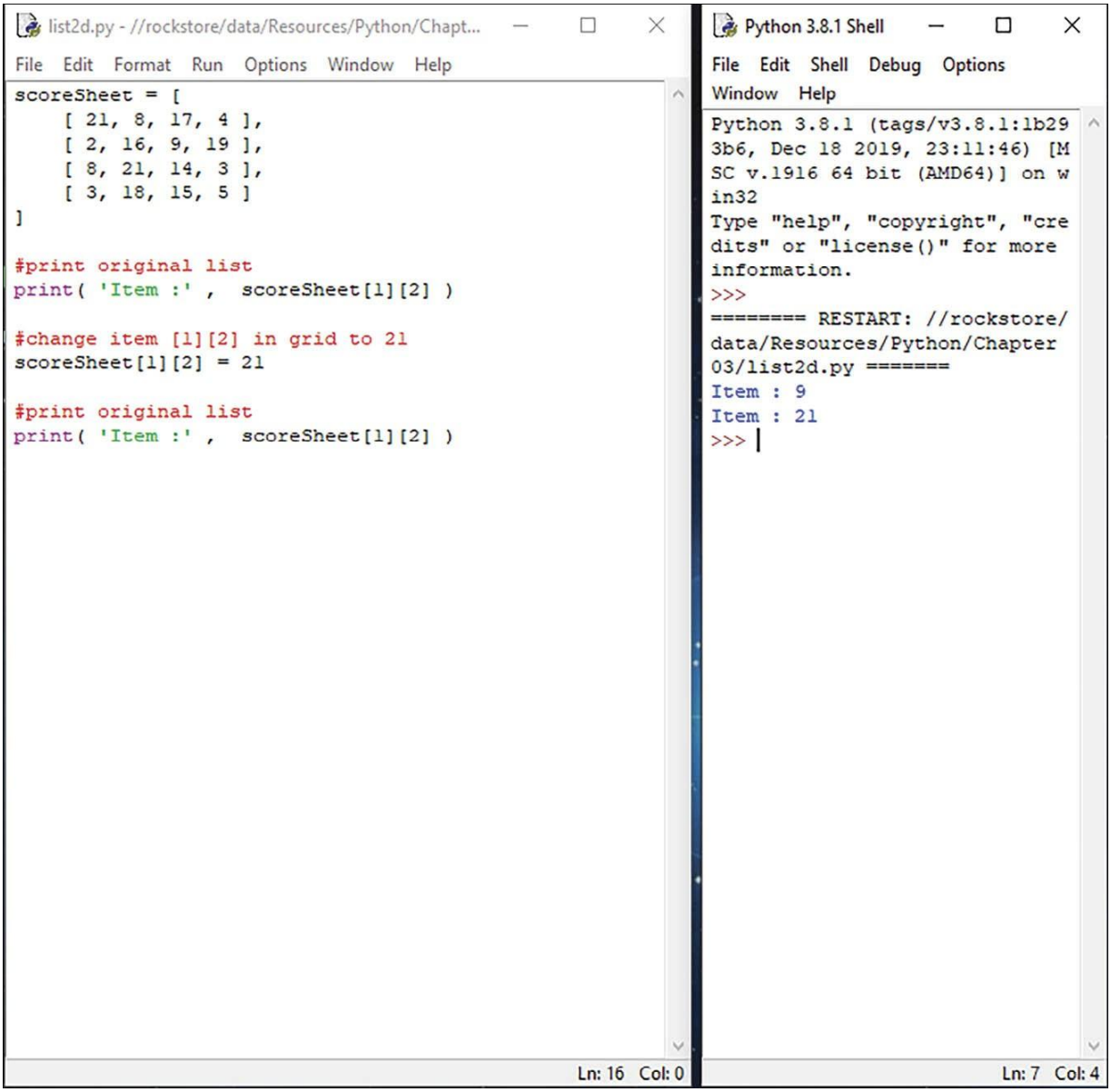
Чтобы сослаться на элемент двумерного списка, поместите обе ссылки в квадратные скобки (сначала индекс строки, затем индекс столбца):

```
print (scoreSheet[1][2]) #circled above
```

Вы можете изменить элементы в списке, поместив обе ссылки в квадратные скобки (сначала индекс строки, затем индекс столбца), а затем присвоив значение:

```
scoreSheet [0][3] = 21
```

Давайте рассмотрим программу. Откройте файл list2d.py. Здесь мы объявили наш список ShoreSheet и наполнили его некоторыми данными.



Мы можем добавить элемент в определенное место в списке

```
scoreSheet [1][2] = 21
```

Мы также можем выводить данные, хранящиеся в определенном месте в списке

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

Наборы

Набор — это неупорядоченная коллекция уникальных элементов, заключенных в фигурные скобки { }. Наборы могут содержать разные типы.

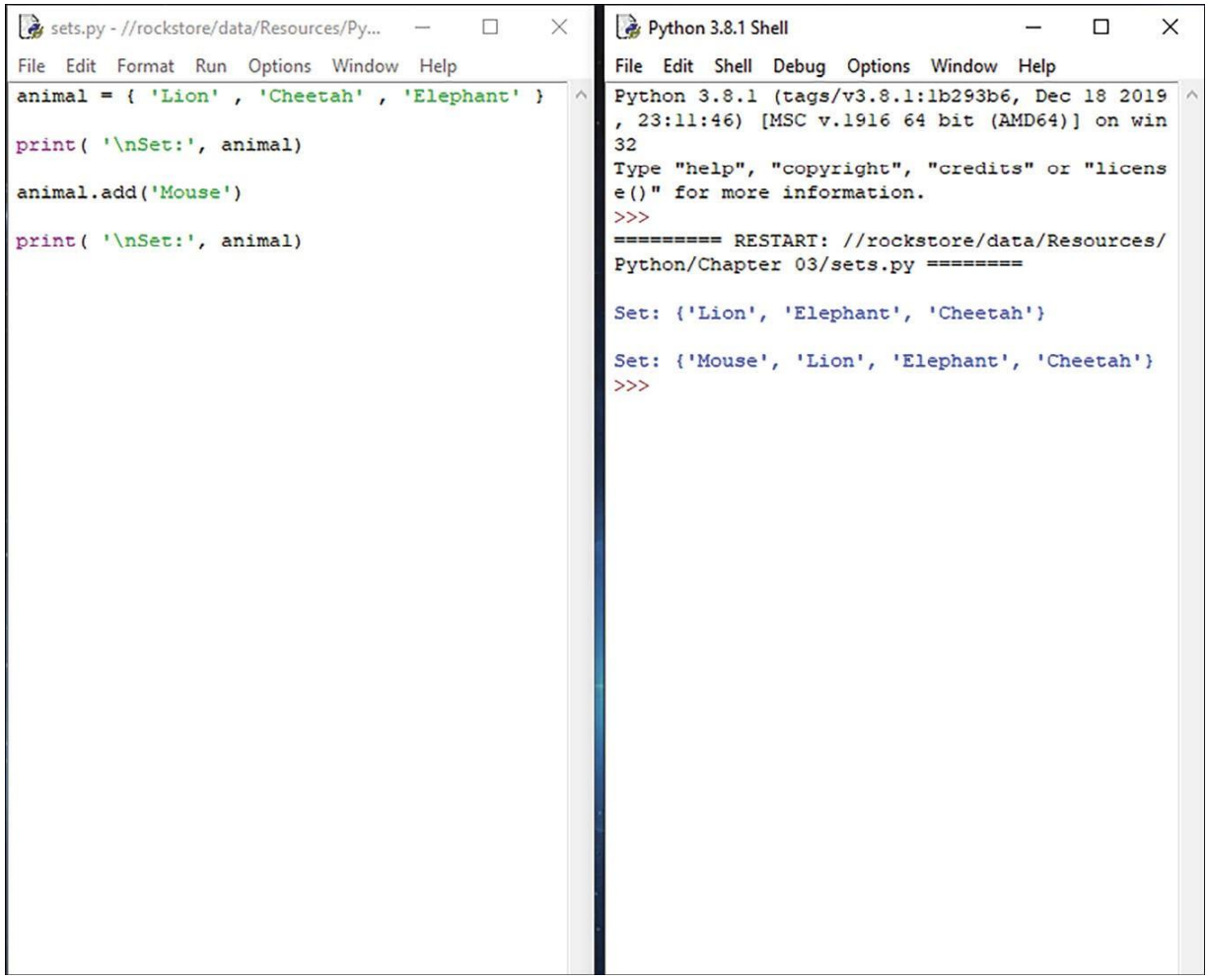
Вы можете создать такой набор:

```
setName = {1, 6, 2, 9}
```

Две вещи, которые следует отметить относительно наборов. Во-первых, вы не можете индексировать отдельные элементы набора, поскольку это неупорядоченный тип данных. Во-вторых, вы не можете изменять отдельные значения в наборе, как это можно делать со списками. Однако вы можете добавлять или удалять элементы. Используйте метод `.add()` и введите данные для добавления в скобках.

```
setName.add('item to add')
```

Давайте рассмотрим программу. Откройте файл set.py. Здесь мы создали набор с названиями животных. Мы можем вывести данные из набора.



Мы также можем добавить элемент в набор, используя метод `.add()`.

Кортежи

Кортеж похож на список и представляет собой последовательность элементов, каждый из которых идентифицируется индексом. Как и в списках, индекс начинается с 0, а не с 1.

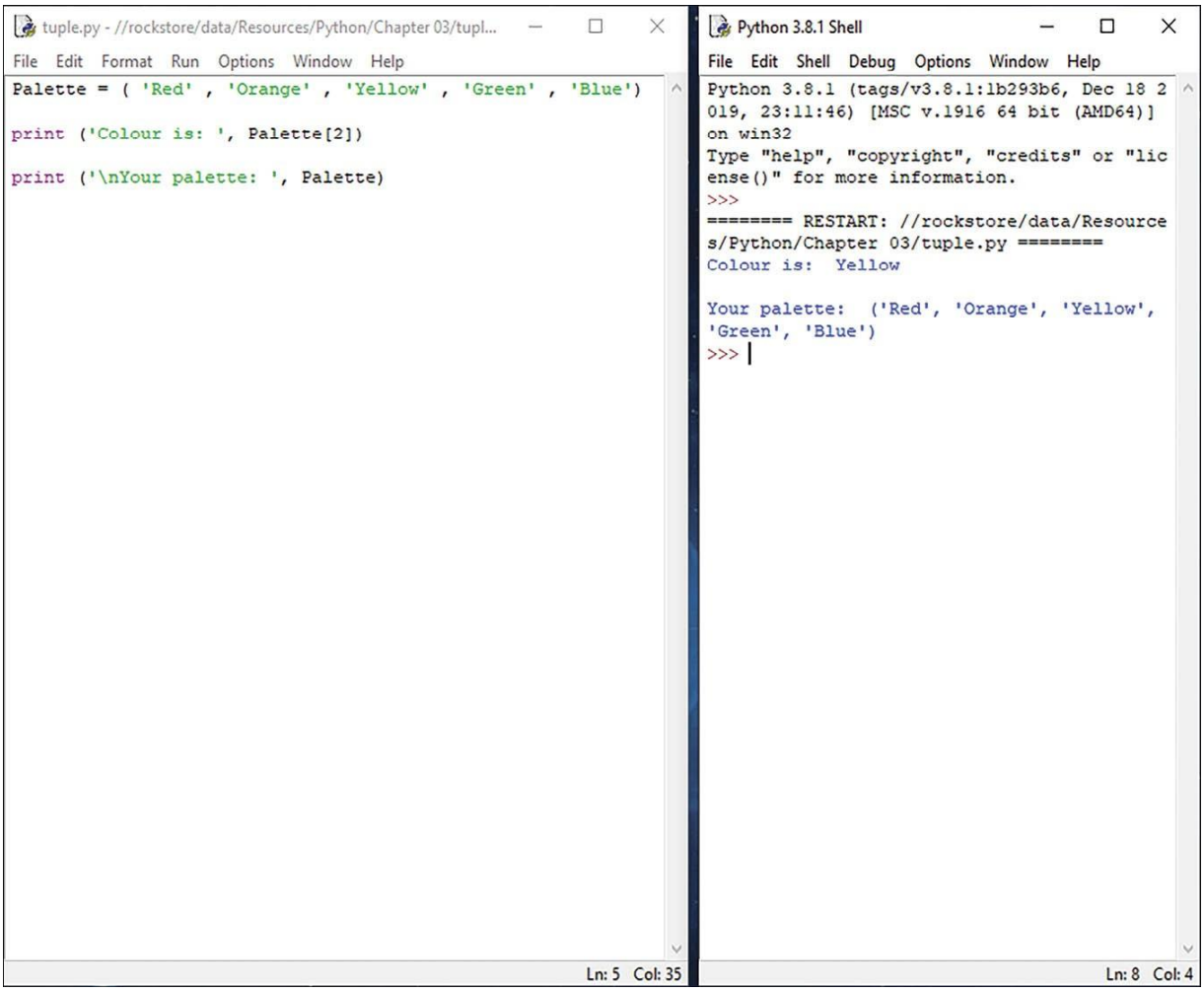
В отличие от списков, элементы в кортеже нельзя изменить после их назначения, а сами кортежи могут содержать разные типы данных.

Чтобы создать кортеж, заключите элементы в круглые скобки ( ).

```
userDetails = (1, 'John', '123 May Road')
```

Используйте кортеж, если хотите сохранить данные другого типа, например данные для входа на ваш веб-сайт.

Давайте рассмотрим программу. Откройте файл tuple.py. Здесь мы создали кортеж из нескольких цветов.



Мы можем вывести данные из кортежа с помощью оператора печати.

```
print (Palette[2])
```

Словари

Словарь — это неупорядоченная коллекция элементов, каждый из которых идентифицируется ключом. Чтобы создать словарь, заключите элементы в фигурные скобки { }. Определите каждый элемент с помощью ключа, используя двоеточие.

```
capitalCities = {'UK': 'London',
'Australia': 'Canberra',
'USA': 'Washington DC',
'Spain': 'Madrid'}
```

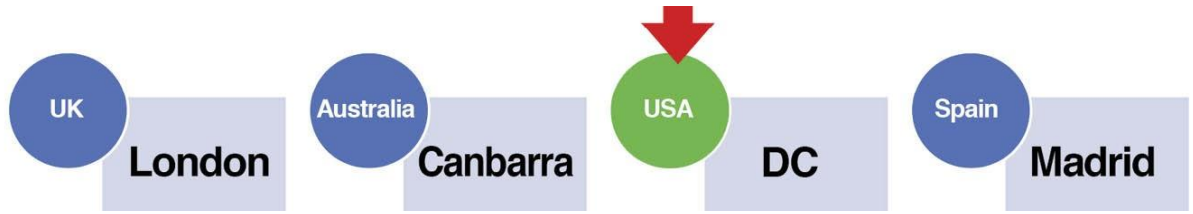
Когда мы объявляем вышеизложенное, мы можем визуализировать это следующим образом:

Чтобы прочитать значение, укажите наименование словаря, затем поместите ключ в квадратные скобки.

```
print (capitalCities['USA'])
```

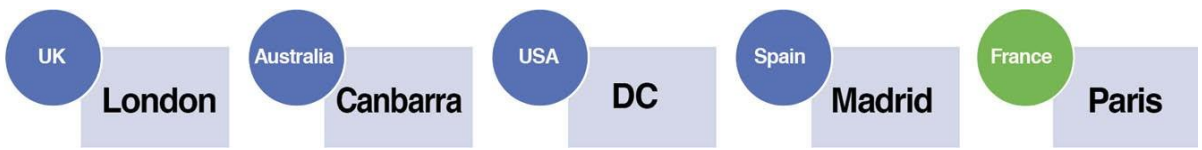
Чтобы изменить значение, укажите наименование словаря, поместив ключ в квадратные скобки. Затем присвойте значение, используя '='.

```
capitalCities['USA']      = 'DC'
```



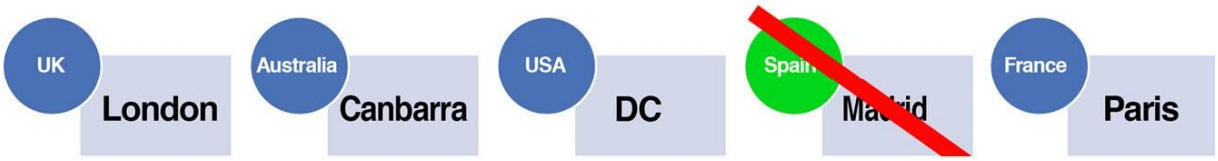
Чтобы добавить элемент, используйте метод .update(). Заключите данные, которые нужно добавить, в фигурные скобки `{'Key' : 'Data'}`. Если ключ не существует, он добавит элемент в конец.

```
capitalCities.update( {'France': 'Paris'} )
```



Чтобы удалить элемент, укажите ключ элемента для удаления 'spain'.

capitalCities.pop('Spain')



В итоге у нас получится следующее:

Давайте рассмотрим программу. Откройте файл Capitals.py. Здесь мы создали словарь с некоторыми данными. Мы прошлись по словарю, используя цикл for, чтобы распечатать каждый элемент словаря.

capitals.py - D:/OneDrive/Apress/Python/Code/Chapter 03/capitals.py (3.10.8)

File Edit Format Run Options Window Help

```
capitalCities = {'UK': 'London',
                 'Australia': 'Canberra',
                 'USA': 'Washington DC',
                 'Spain': 'Madrid'}


for i in capitalCities:
   print ("Capital of ", i, " is ", capitalCities[i])
print()

capitalCities['USA'] = 'DC'

capitalCities.update( {'France': 'Paris'} )

capitalCities.pop('Spain')
```

Мы заменили 'USA' на 'DC', присвоив 'DC' словарю по ключу 'USA'. Мы также использовали метод `.update()`, чтобы добавить 'France'.

Наконец, мы использовали метод `.pop()` для удаления слова 'Spain'.

Приведение типов данных

Переменные могут содержать различные типы данных, такие как текст (называемый строкой), целое число (называемое целым числом) или число с плавающей запятой (числа с десятичной точкой).

В Python вам не нужно объявлять все переменные перед их использованием. Однако вам может потребоваться преобразовать переменные в другие типы. Эта процедура известна как приведение типов.

В Python есть два типа преобразования типов: неявное и явное.

При неявном преобразовании типов Python автоматически преобразует один тип данных в другой.

В случае явного преобразования типов программист преобразует тип данных в требуемый тип данных с помощью определенной функции. Вы можете использовать следующие функции для приведения типов данных:

- `int( )` преобразует данные в целое число
- `long( )` преобразует данные в длинное целое число
- `float( )` преобразует данные в число с плавающей запятой
- `str( )` преобразует данные в строку

Например, вы можете использовать функцию **input( )**, чтобы запросить у пользователя некоторые данные

```
a = input (`Enter first number: `)
```

В этом примере пользователю будет предложено ввести некоторые данные, а затем сохранить эти данные в переменной «a» в виде строки.

Это может и звучит нормально, но что, если мы захотим выполнить некоторые арифметические действия с данными? Мы не сможем этого сделать, если данные хранятся в виде строки. Нам придется ввести данные в переменную как целое число, либо число с плавающей запятой.

```
int(a)
```

либо

```
float(a)
```

Арифметические операторы

В языке Python существует несколько арифметических операторов, которые вы можете использовать.

Оператор	Описание
**	Возведние в стпень
/	Деление
*	Умножение
+	Сложение
-	Вычитание

Приоритет операторов

BIDMAS (иногда называемый BODMAS) — это аббревиатура, обычно используемая для запоминания приоритета математических операторов, то есть порядка, в котором вы оцениваете каждый оператор.

1. Brackets **( )** или скобки
2. Indices ****** или возведение в степень
3. Divide **/** или деление
4. Multiply ***** или умножение
5. Add **+** или сложение
6. Subtract **–** или вычитание

Выполнение арифметических операций

Если вы хотите добавить 20% налога с продаж к цене в 12,95 долларов, вы можете написать что-то вроде этого...

```
total = 12.95 + 12.95 * 20 / 100
```

Согласно приведенному выше перечню приоритетов, вы должны сначала выполнить оператор деления:

```
20 / 100 = 0.2
```

Затем - умножения

```
12.95 * 0.2 = 2.59
```

И наконец, сложения

```
12.95 + 2.59 = 15.54
```

Операторы сравнения

Эти операторы используются для сравнения значений и обычно используются в условных операторах или при построении циклов.

Оператор	Описание
==	Равно
!=	Неравно
>	Больше чем
<	Меньше чем
>=	Больше чем или равно
>=	Меньше чем или равно

Например, сравнивая два значения в операторе «if», вы можете написать что-то вроде этого:

```
if a > 10:
print (“You’ve gone over 10...“)
```

Логические операторы

Также известны как логические операторы и обычно используются в условных операторах (if...) или создании циклов (while... for...). Мы рассмотрим операторы if и циклы в Главе 4.

Оператор	Описание
and	Возвращает `верно`, если оба операнда являются истинными
or	Возвращает `верно`, если какой-либо из операндов является истинным
not	Возвращает `верно`, если операнд является истинным

Например, вы можете объединить два сравнения в выражении ‘if’ , используя ‘and’, как это показано ниже:

```
if a >= 0 and a <= 10:
print (“Your number is between 0 and 10”)
else
print (“0ut of range - must be between 0 & 10”)
```

Использование оператора ‘and’ будет означать, что оба условия (a >= 0) и (a <= 10) должны быть истинными.

Побитовые операторы

Побитовые операторы используются для сравнения двоичных чисел:

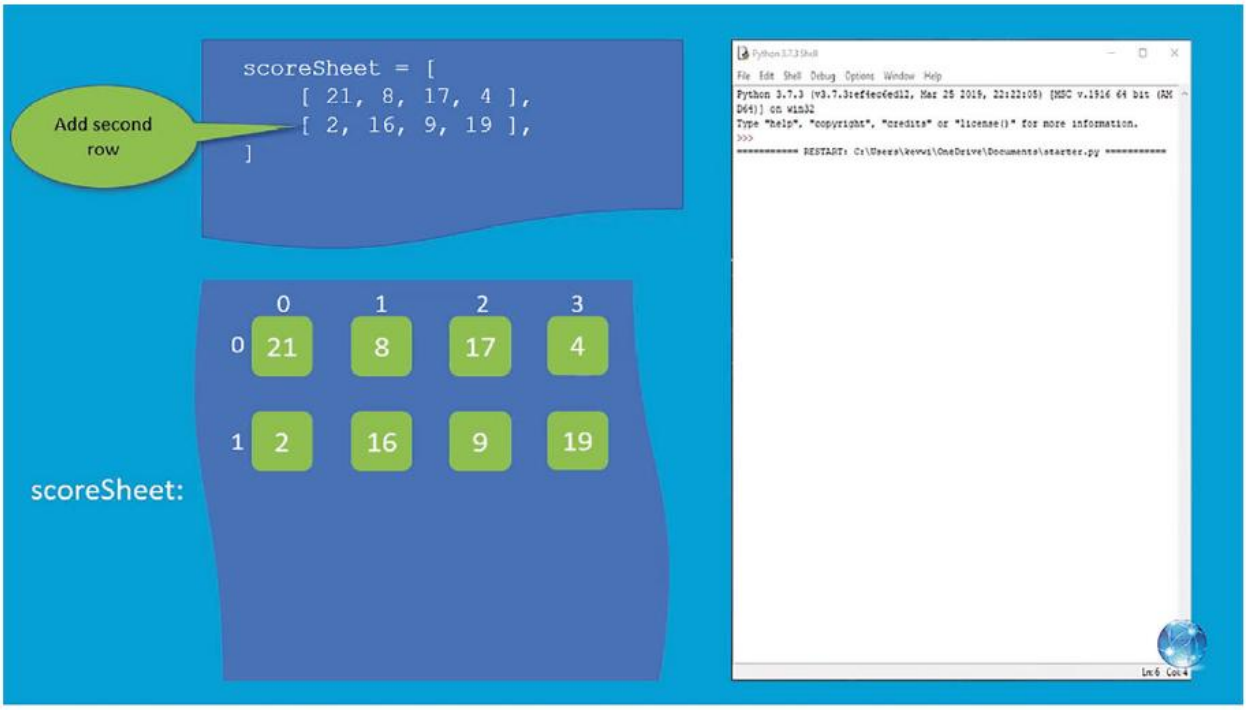
Оператор	Наимнование	Описание
&	И	Устанавливает каждый бит в 1, если оба бита равны 1
|	ИЛИ	Устанавливает каждый бит в 1, если один из двух бит равен 1
^	Исключающее ИЛИ	Устанавливает каждый бит в 1, если только один из двух бит равен 1
~	НЕ	Инвертирует все биты
<<	Сдвиг влево	Сдвигает биты влево
>>	Сдвиг вправо	Сдвигает биты вправо

Вы можете применять побитовые операторы

```
a >> 2 #сдвинуть биты 'a' влево на 2 позиции

a << 2 #сдвинуть биты 'a' вправо на 2 позиции

a & b #выполнить операцию AND над битами
```

Посмотрите демонстрационные видеоролики и ознакомьтесь с назначением переменных, типов данных, списков, кортежей и словарей.

```
www.elluminetpress.com/pybasics
```


Add second row
scoreSheet = [
[21, 8, 17, 4],
[2, 16, 9, 19],
]
0 1 2 3
0 21 8 17 4
1 2 16 9 19
scoreSheet:
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\kevvs\OneDrive\Documents\starter.py ==========
Ln: 6 Col: 4


Variables
Lists
Assign Value to List
NOW PLAYING
2D List Assign Value

Лабораторная работа

1. Напишите программу, которая принимает длину в дюймах и печатает ее в сантиметрах (1 дюйм = 2,54 см).
2. Напишите программу, которая принимает ваше имя, фамилию и год рождения и добавляет их в массив.
3. Напишите программу, которая добавляет в словарь данные о сотрудниках. Используйте номер сотрудника в качестве ключа.
4. Напишите программу, преобразующую температуру из градусов по Цельсию в градусы по Фаренгейту.

   $F = C \times 9/5 + 32$

5. Напишите программу, вычисляющую объём сферы

   $V = 4/3\ \pi r^3$

6. Напишите программу для расчета и отображения валовой и чистой заработной платы сотрудника. В этом сценарии из валовой заработной платы вычитается налог по ставке 20%, чтобы получить чистую заработную плату.
7. Напишите программу, хранящую список покупок из 10 товаров. Выведите весь список на экран, затем напечатайте пункты 2 и 8.
8. Расширьте предыдущую программу таким образом, чтобы вставить элемент в список.
9. Что такое логический оператор? Напишите программу, демонстрирующую его.
10. Что такое оператор сравнения? Напишите программу, демонстрирующую его.
11. Что такое приведение типов данных? Зачем нам это нужно? Напишите программу, демонстрирующую это.

Управление потоком

Управление потоком контролирует порядок выполнения операторов или вызовов функций программы.

Существует три структуры управления: последовательность, выбор и итерация.

Python обладает различными структурами управления, такими как циклы while, for и операторы if, которые используются для определения того, какой фрагмент кода выполняется в соответствии с определенными условиями.

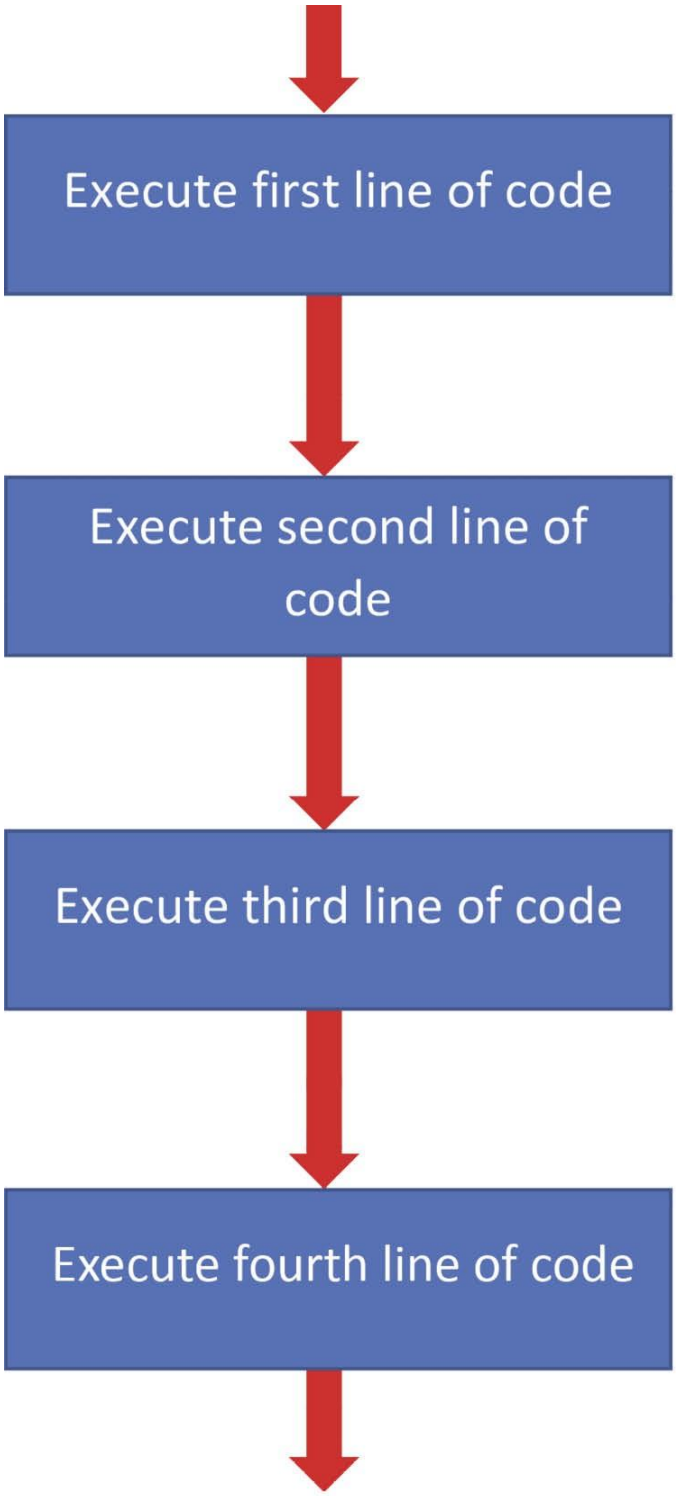
Для этого раздела посмотрите демонстрационные видео

`elluminetpress.com/pythonflow`

Вам также понадобятся исходные файлы из каталога Chapter 04.

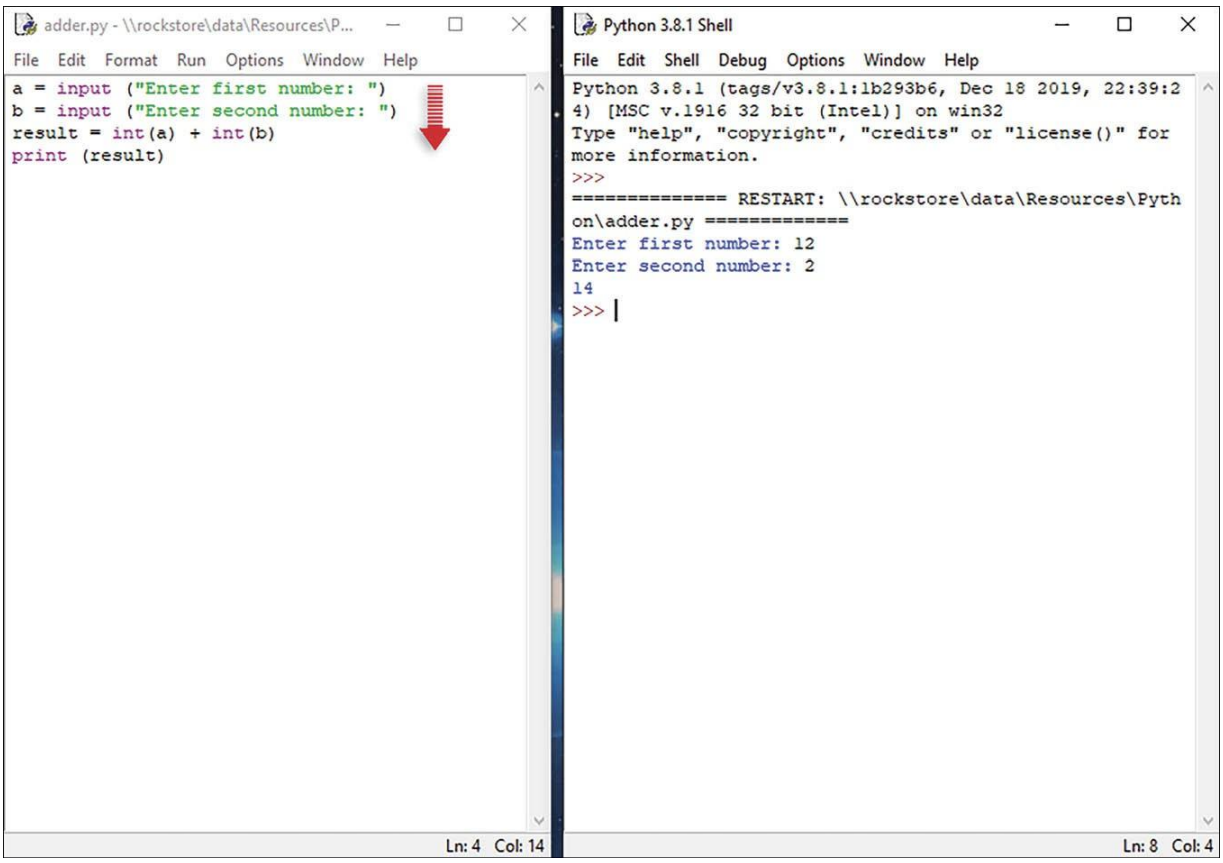
Последовательность

Компьютерная программа — это набор пошаговых инструкций, которые последовательно выполняются для достижения цели задачи или решения проблемы. Последовательность может содержать любое количество инструкций, но ни одна инструкция в последовательности не может быть пропущена.

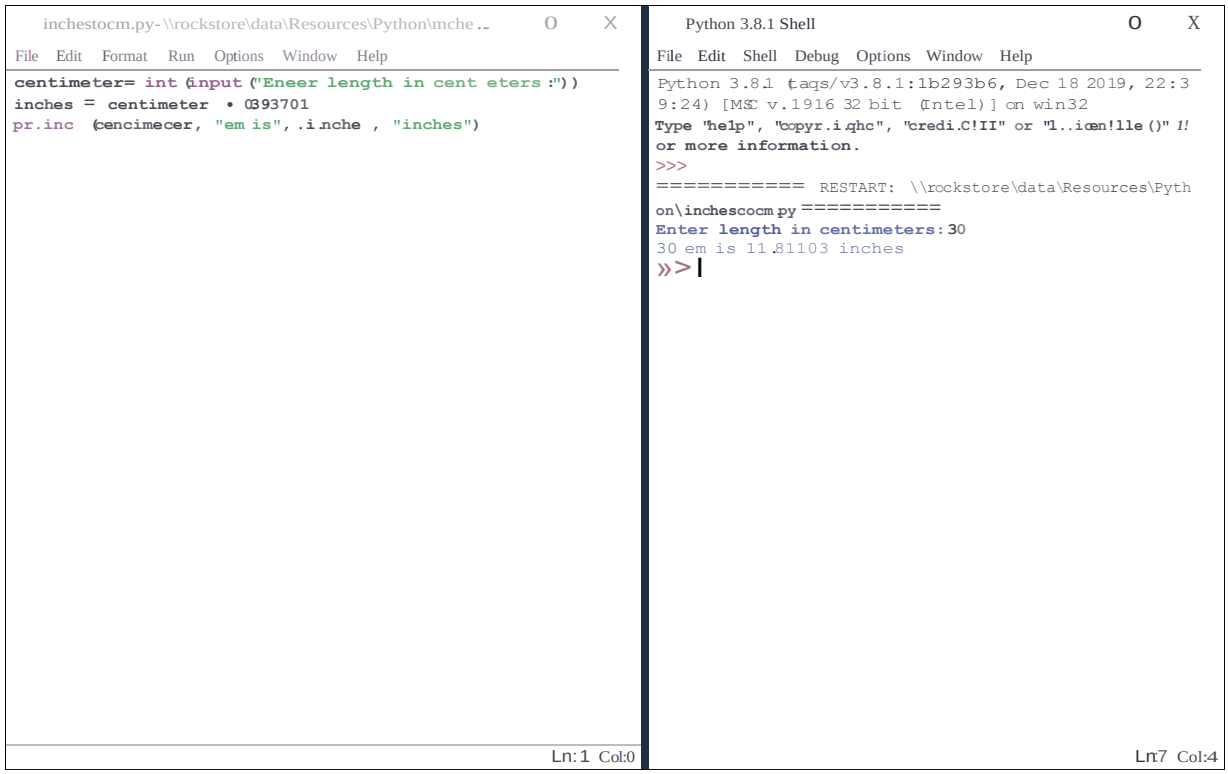


Интерпретатор будет последовательно перебирать и выполнять каждую строку кода от начала до конца программы.

Давайте рассмотрим программу. Откройте adder.py. Эта программа содержит четыре оператора.



При запуске программы инструкции выполняются последовательно. Давайте рассмотрим другой пример. Откройте instocm.py

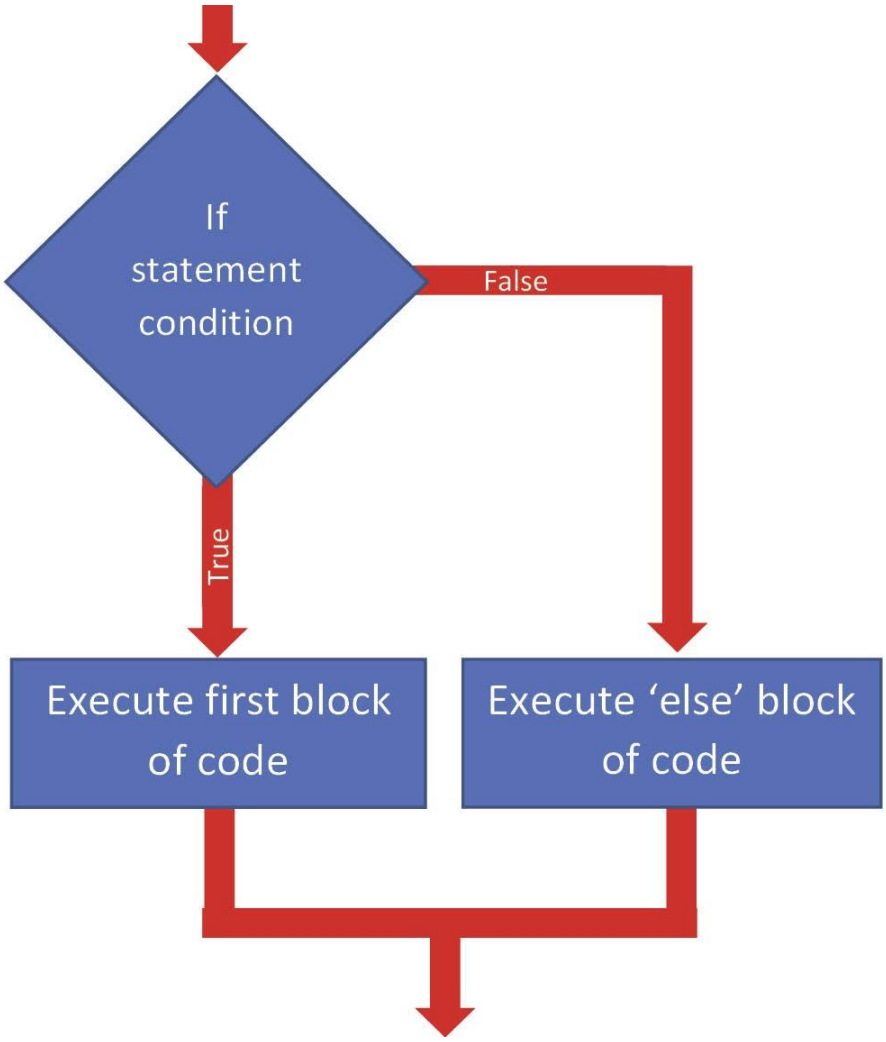

inchestocm.py- \\rockstore\data\Resources\Python\mche ...
File Edit Format Run Options Window Help
centimeter= int (input ("Eneer length in cent eters :"))
inches = centimeter • 0393701
pr.inc (cencimecer, "em is", .i.nche , "inches")
Ln: 1 Col:0
Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
Python 3.8.1 (taqs/v3.8.1:1b293b6, Dec 18 2019, 22:3
9:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyr.i.qhc", "credi.C!II" or "1..icen!lle()" !!
or more information.
>>>
========== RESTART: \\rockstore\data\Resources\Pyth
on\inchescocm py ==========
Enter length in centimeters: 30
30 em is 11.81103 inches
»>
Ln:7 Col:4

Выбор

В большинстве компьютерных программ существуют определенные моменты, когда необходимо принять решение. Это решение основано на условии, и это условие может быть либо истинным, либо ложным.

if...else

Если используются выражения при необходимости принять решение. Если условие истинно, оператор if выполнит первый блок кода, если условие ложно, оператор if выполнит блок кода «else», если он включен в код.



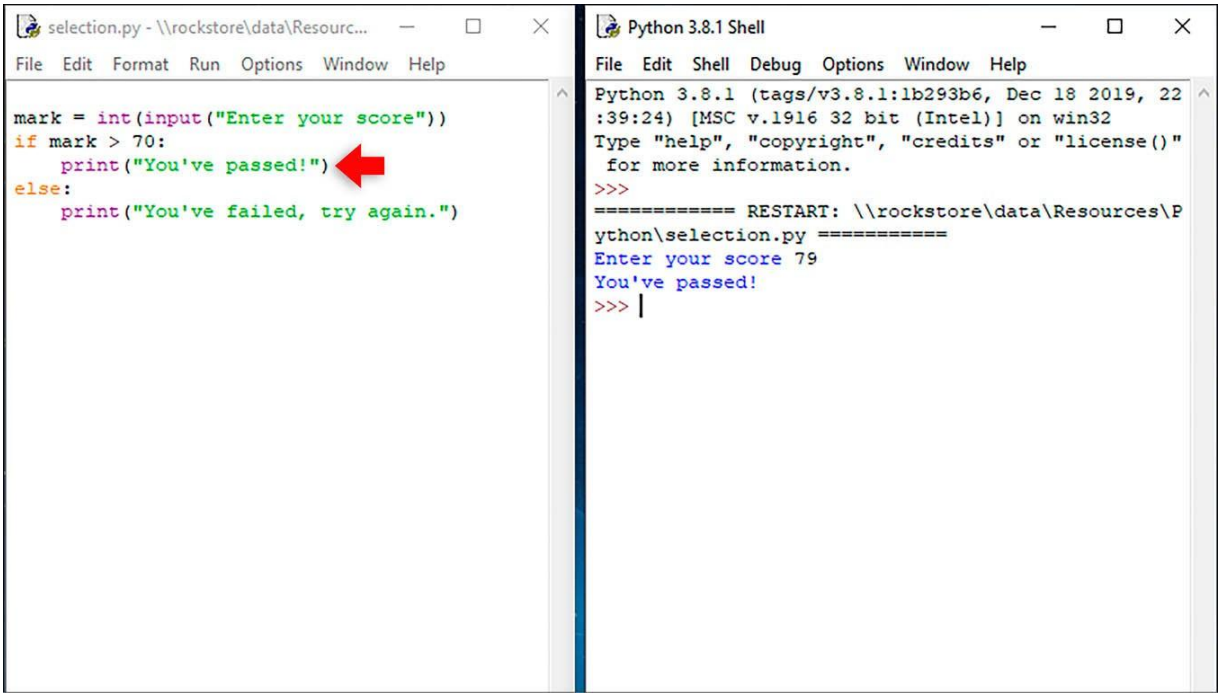
Так например:

```
if num >= 0: #условие
print("Positive or Zero") #первый блок
```

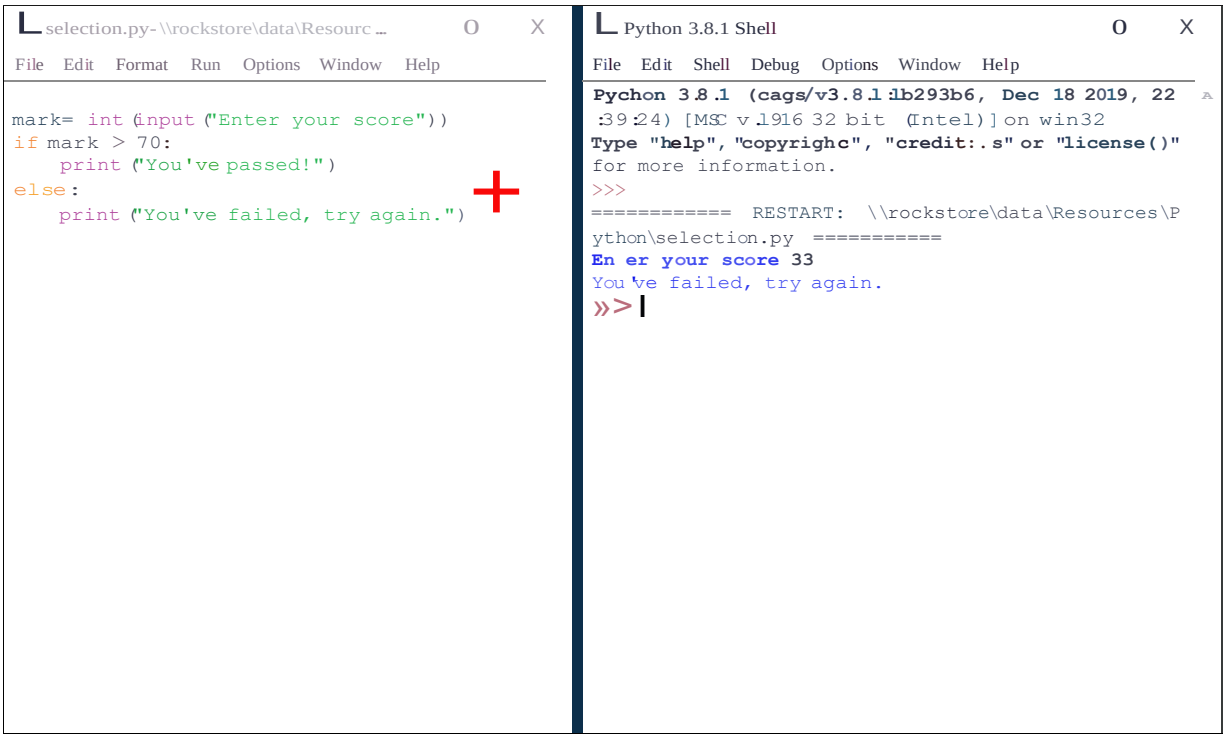
```
else:
print("Negative number") #блок else
```

Давайте рассмотрим программу. Откройте selection.py. Здесь мы видим очень простой оператор if, позволяющий определить, является ли результат теста пройденным или нет. Уровнь прохождения равен 70, поэтому нам нужен оператор if, чтобы возвращать сообщение о прохождении, если введенное значение больше 70. Помните, что нам также нужно преобразовать переменную «mark» в целое число (int).

Если вы введете значение больше 70, интерпретатор Python выполнит первый блок оператора if.

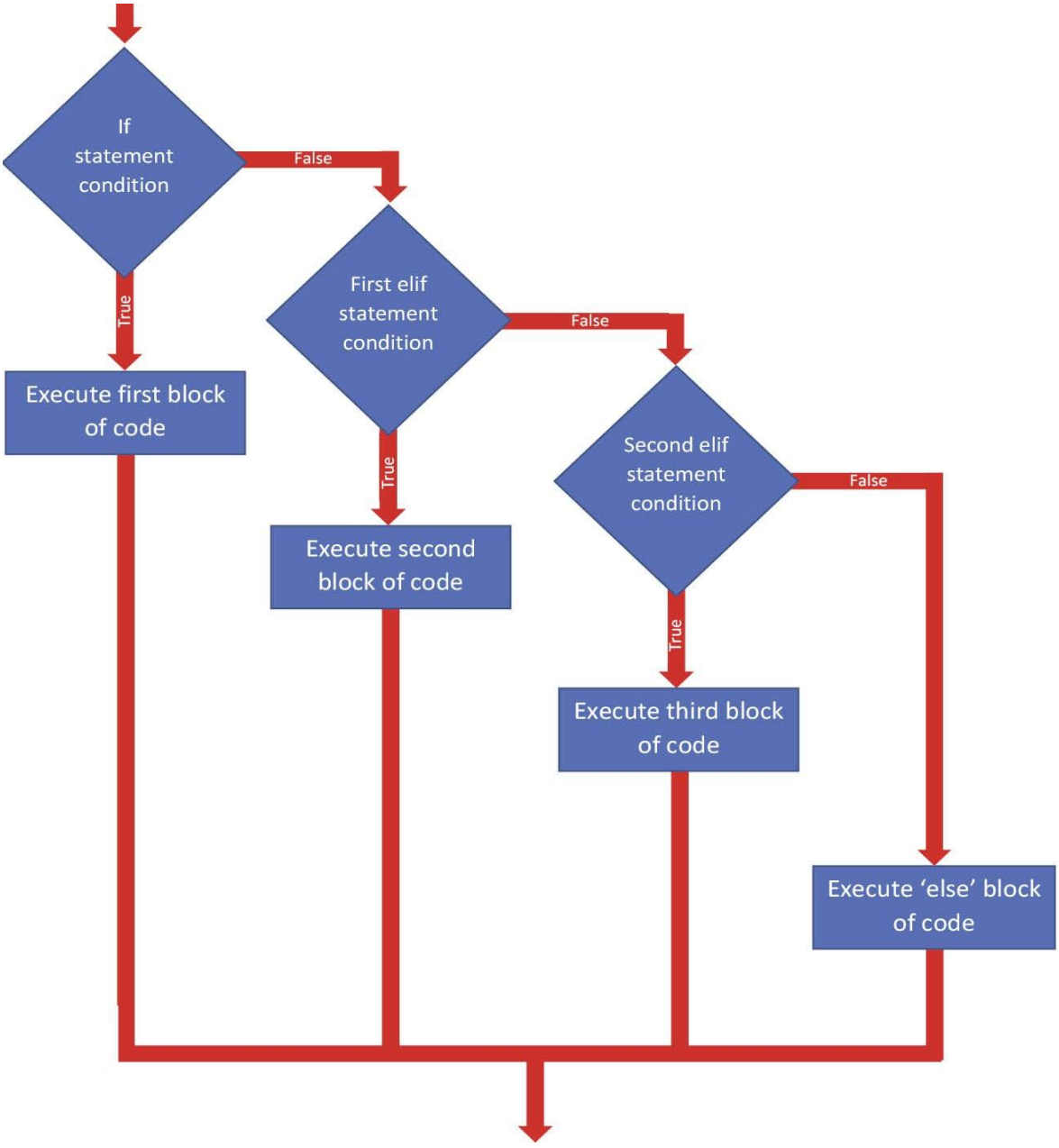


Если вы введете значение ниже 70, интерпретатор Python выполнит блок else оператора if.


selection.py-\\rockstore\data\Resourc ...
File Edit Format Run Options Window Help
mark= int (input ("Enter your score"))
if mark > 70:
print ("You've passed!")
else:
print ("You've failed, try again.")
Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
Pychon 3.8.1 (cags/v3.8.1:1b293b6, Dec 18 2019, 22
:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyrighc", "credit:.s" or "license()"
for more information.
>>>
========== RESTART: \\rockstore\data\Resources\P
ython\selection.py ==========
En er your score 33
You've failed, try again.
»>

elif

Используйте оператор `elif`, если необходимо принять несколько решений. Каждое решение (или условие) будет иметь набор инструкций, которые необходимо выполнить.

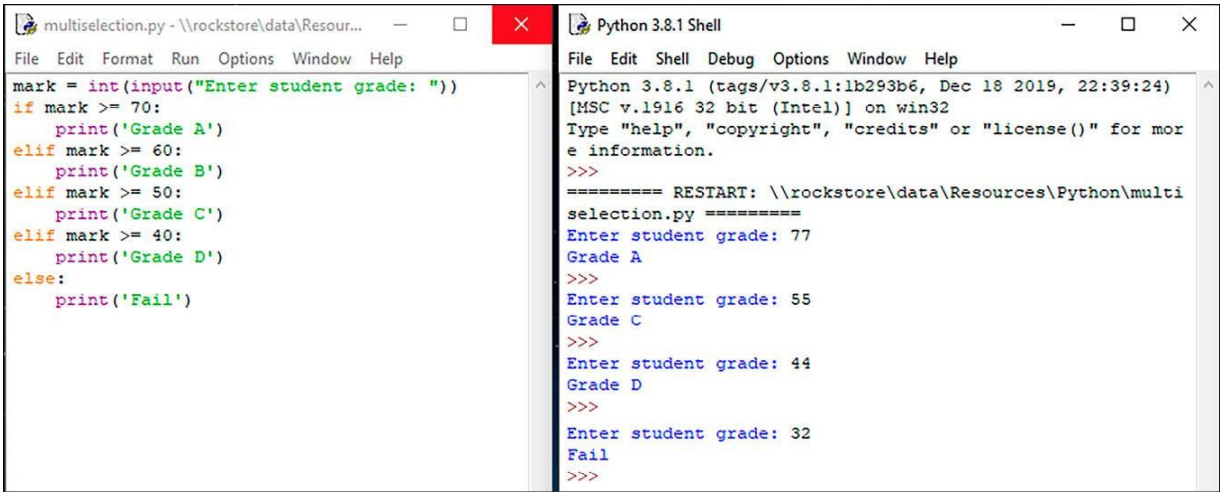


Так например:

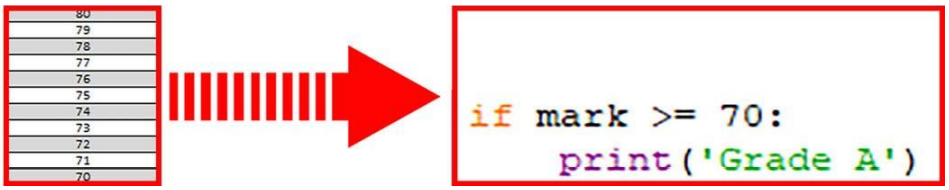
```
if condition: #условие if
[statements] #первый блок кода
```

```
elif condition: #первый оператор elif
[statements] #второй блок кода
elif condition: #второй оператор elif
[statements] #третий блок кода
else:
[statements] #блок кода else
```

Давайте рассмотрим программу. Откройте multiselection.py.

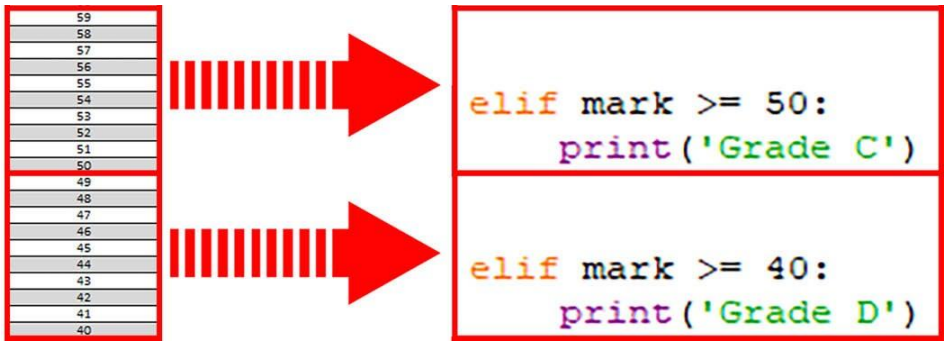


Если мы проанализируем оператор eilf, мы увидим, как он работает. Для первого условия любое введенное число, которое больше 70, приведет к выполнению первого блока.

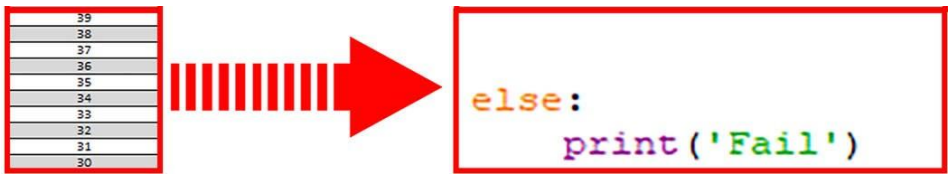


Для любого числа от 60 до 69, интерпретатор выполнит второй блок.

Аналогично и для остальных условий. 50-59 и 40-49.



При любом условии, которое не удовлетворяет приведенным выше операторам ‘elif’, интерпретатор выполнит блок else.

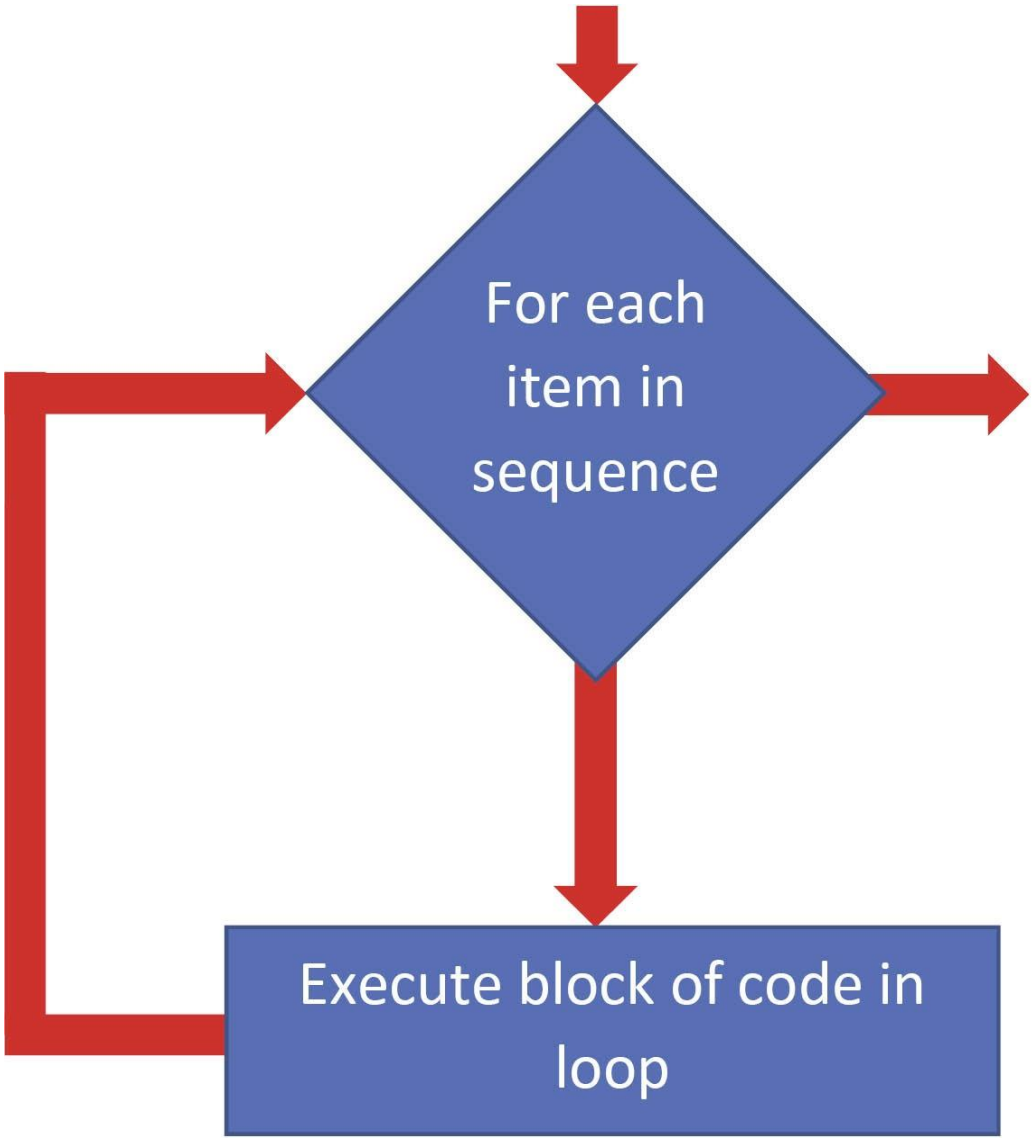


Итерация (циклы)

Цикл — это набор операторов, которые повторяются до тех пор, пока не будет выполнено определенное условие. Мы рассмотрим два типа циклов: цикл for и цикл while.

Цикл for

Цикл for выполняет набор операторов для каждого элемента последовательности, например списка, строки или диапазона. Это позволяет повторять блок кода определенное количество раз.

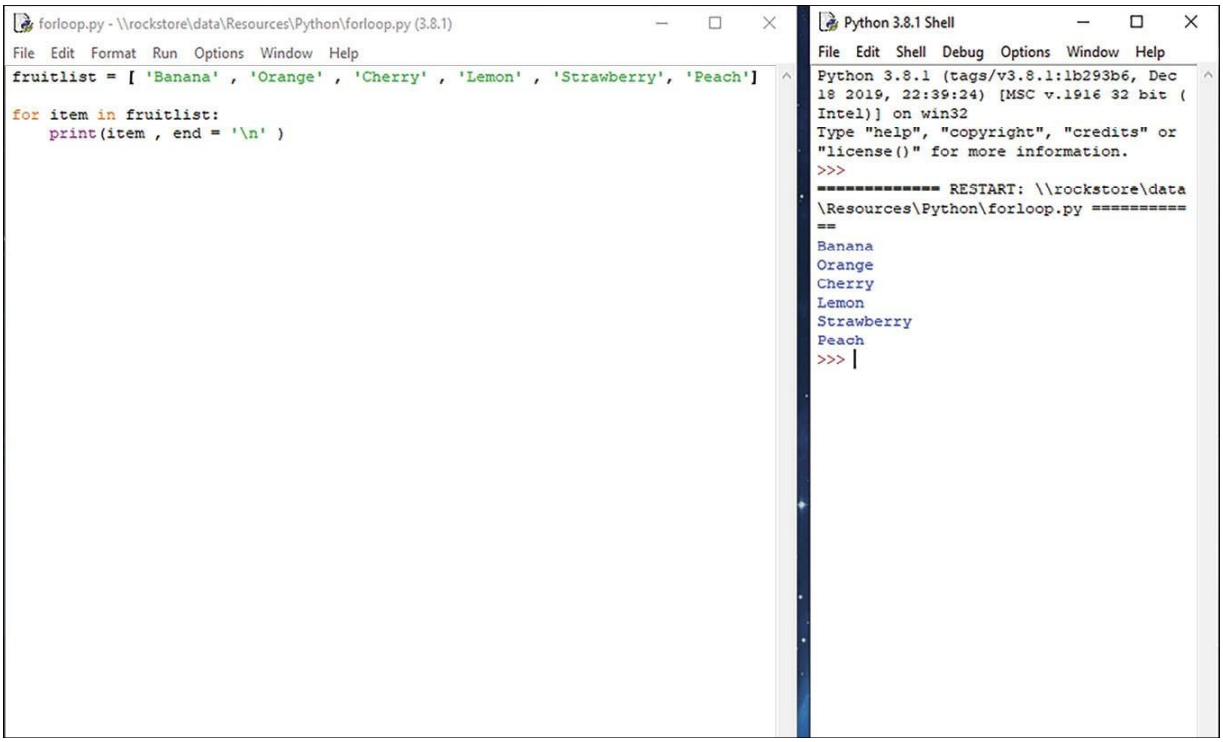


Этот конкретный цикл распечатает каждое имя из списка на новой строке

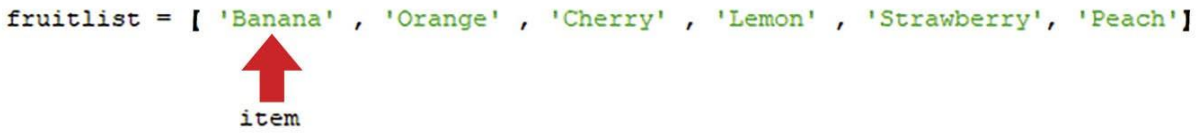
```
myList = ['john', 'lucy', 'kate', 'mike']
```

```
for val in myList:
print (val) #блок кода в цикле
```

Давайте рассмотрим программу. Откройте forloop.py. Цикл for содержит условие цикла. В этом примере цикл будет выполняться для каждого элемента списка фруктов (последовательности).



Переменная ‘item’ в операторе цикла for является указателем или счетчиком текущего значения или элемента последовательности.



Для каждого из этих элементов интерпретатор выполнит все инструкции внутри цикла. В этом примере это оператор print.

```
print (item, end='\n')
```

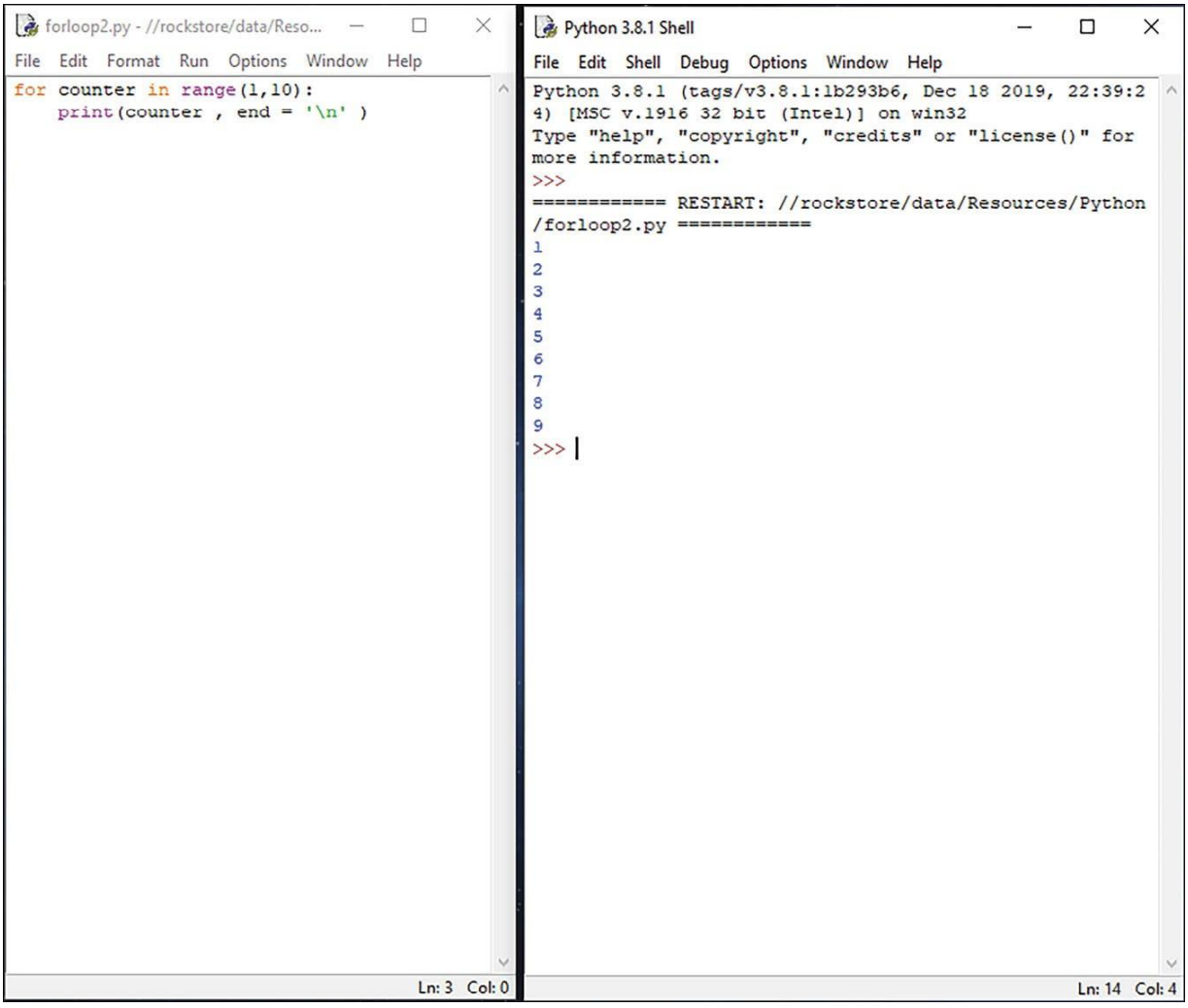
Интерпретатор снова проверит условие в цикле for и, если оно истинно, интерпретатор снова выполнит все инструкции внутри цикла. На каждой итерации цикла счетчик перемещается к следующему значению или элементу.

```
fruitlist = [ 'Banana' , 'Orange' , 'Cherry' , 'Lemon' , 'Strawberry', 'Peach']
```

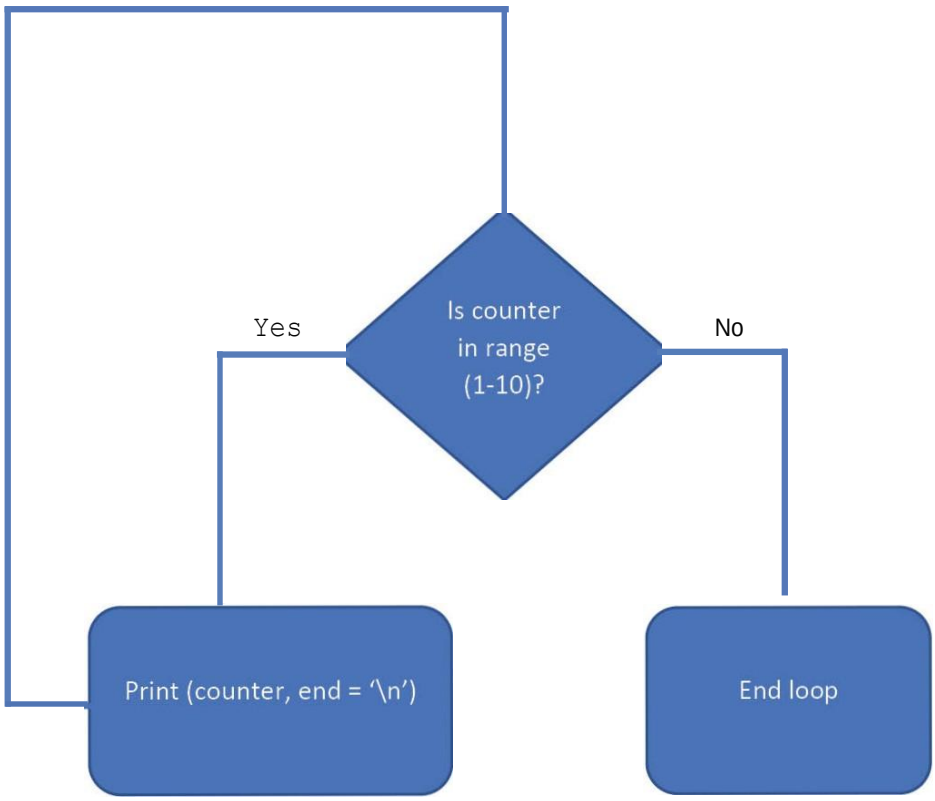
item

В конце последовательности условие цикла становится ложным, поэтому цикл завершается.

Давайте рассмотрим другой пример. Откройте forloop2.py.

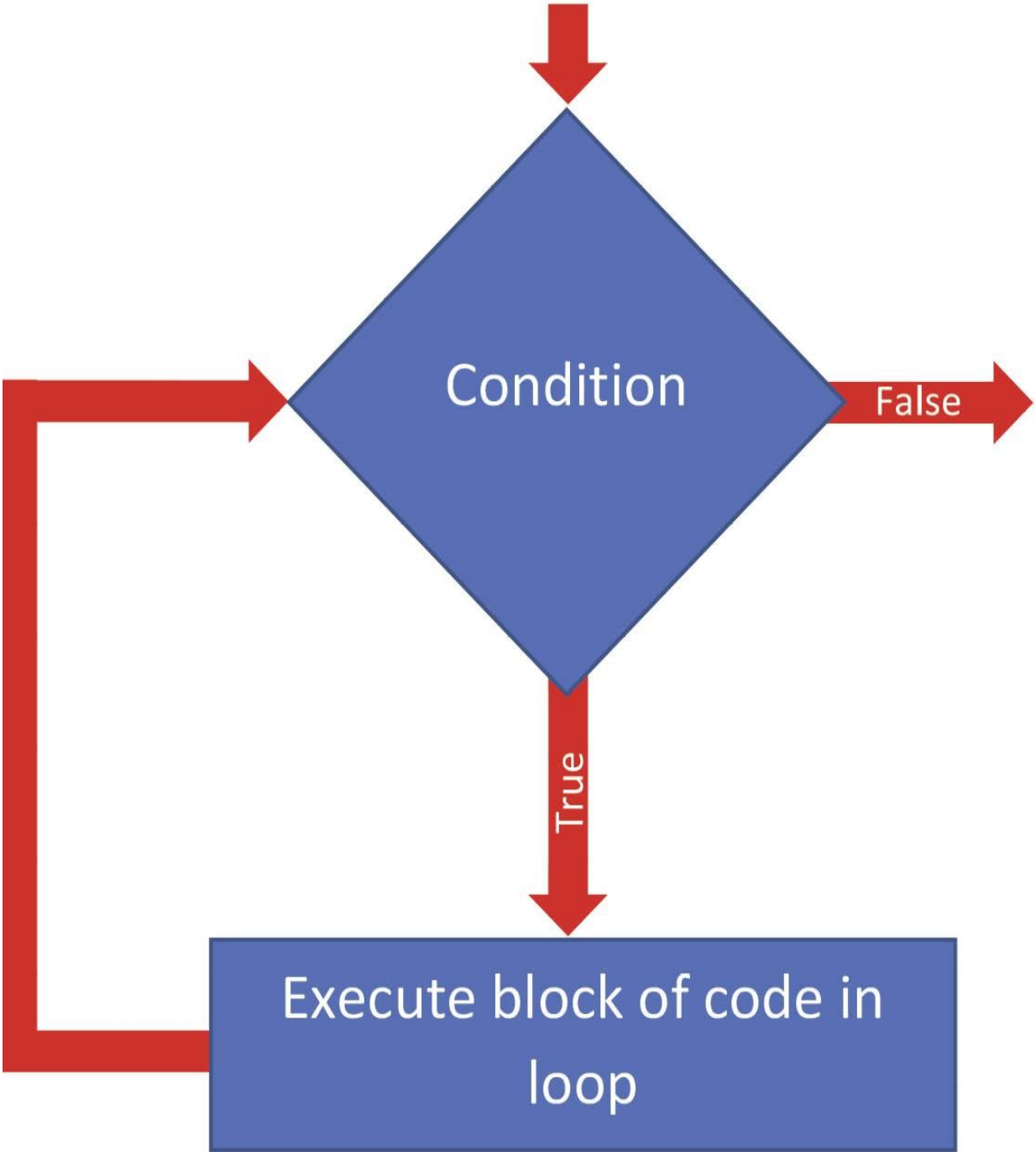


Запустив программу, вы увидите, что она делает.


Is counter
in range
(1-10)?
Yes
No
Print (counter, end = '\n')
End loop

Цикл while

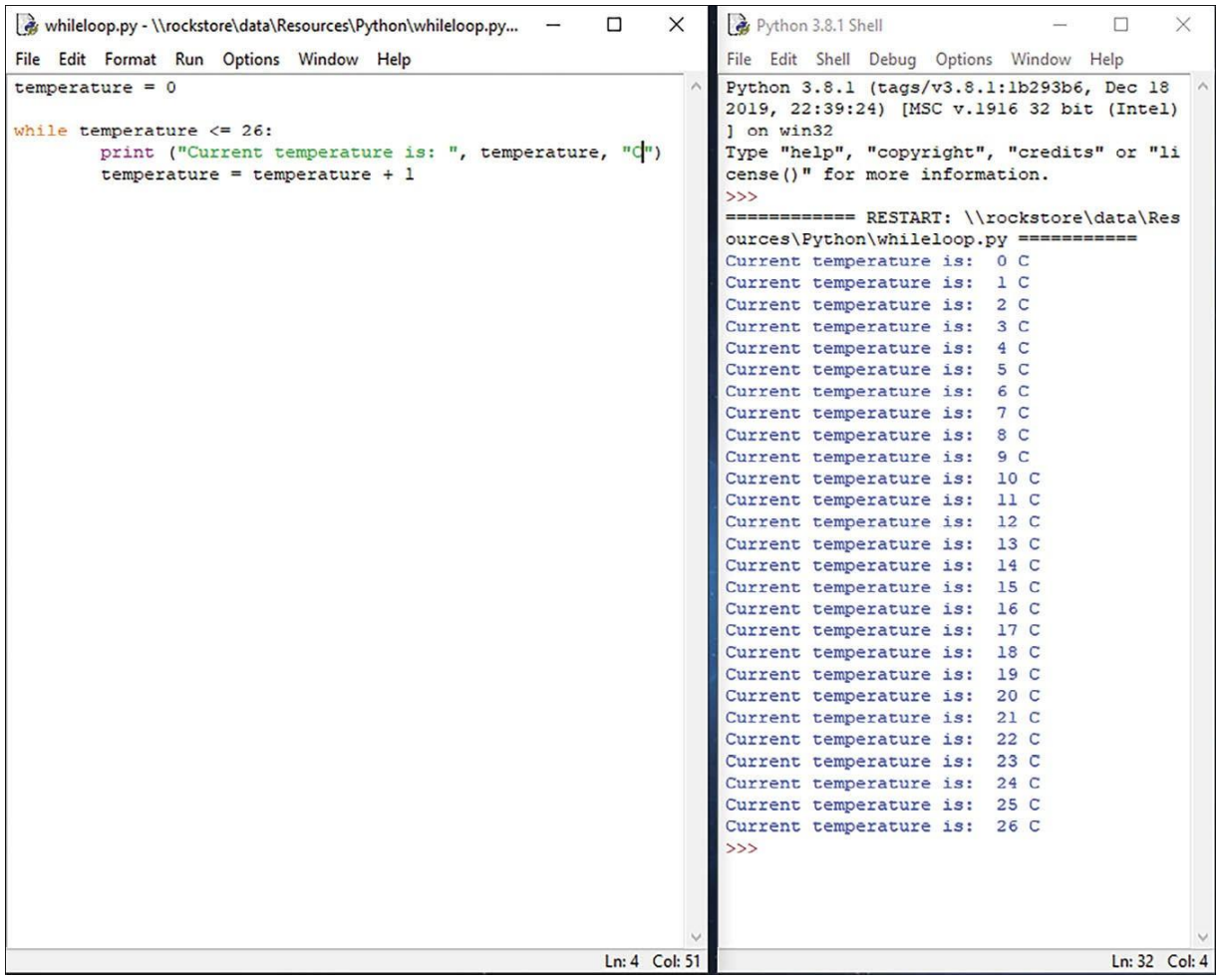
Цикл while выполняет набор операторов до тех пор, пока определенное условие истинно. Этот цикл позволяет повторять набор операторов в блоке кода неизвестное количество раз и будет продолжать повторяться, пока условие истинно.



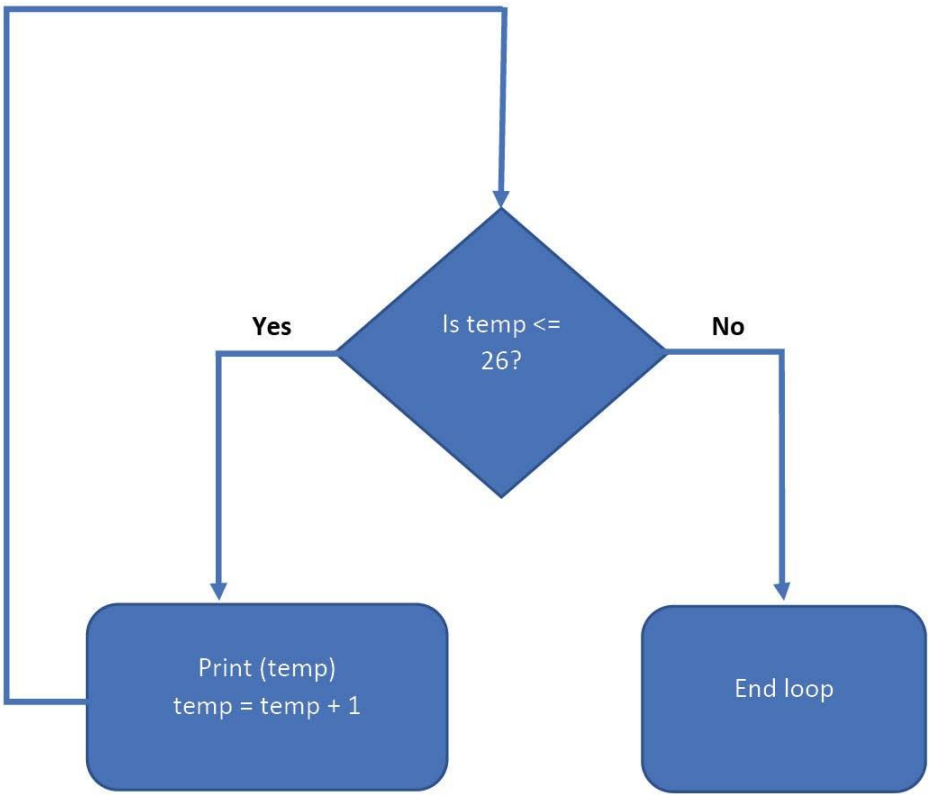
Этот конкретный цикл будет продолжать запрашивать у пользователя ввод ткста до тех пор, пока пользователь не введет слово ‘fire’.

```
userInput = ''
while userInput != 'fire':
userInput = input ('Enter passcode: ')
```

Давайте рассмотрим программу. Откройте whileloop.py. Цикл while содержит условие цикла.



Запустив программу, вы увидите, что она делает.



Break и Continue

Оператор `break` выходит из цикла. В этом примере цикл прерывается, когда счетчик равен 5.

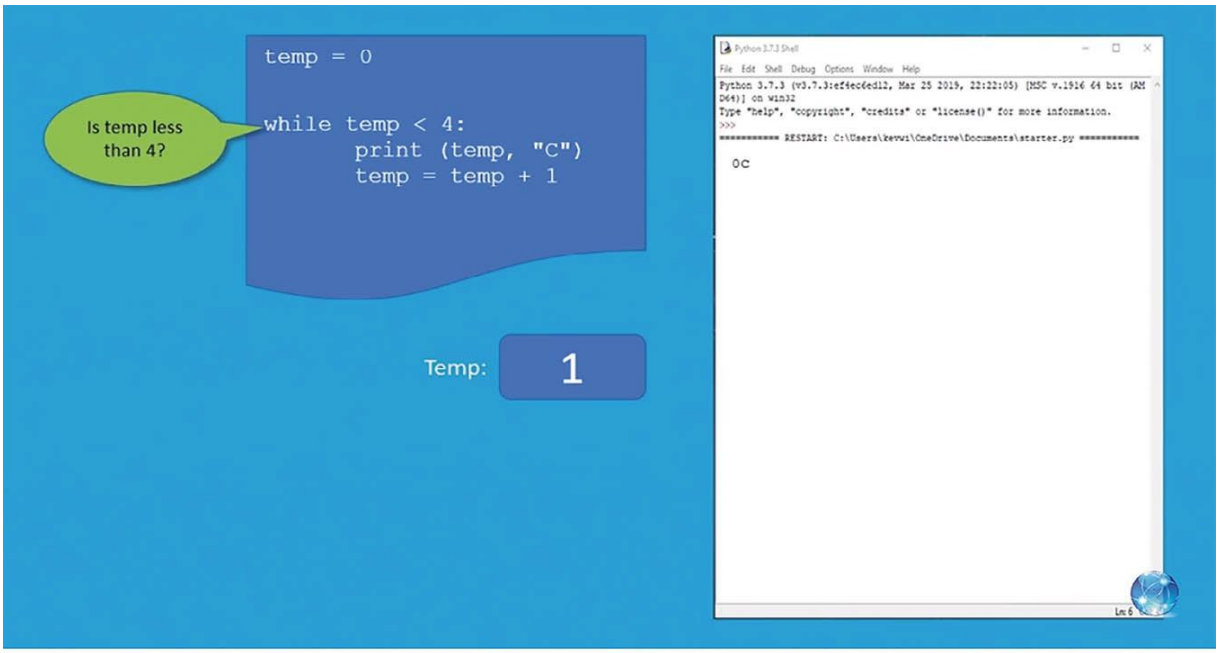
```
while (counter < 10):
if counter == 5:
break
counter = counter + 1
```

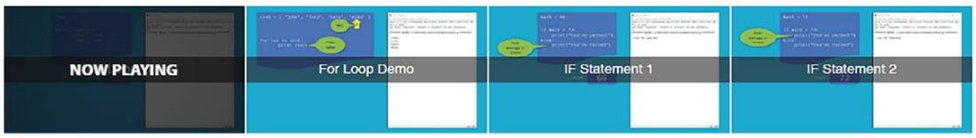
Оператор `continue` возвращает выполнение программы к началу цикла, не выполняя при этом остальную часть кода после ключевого слова continue. В данном примере цикл перезапускается, если ‘number’ являтся четным.

```
myList = [1, 2, 3, 4, 5, 6, 7, 8]
for number in myList:
if number % 2 == 0: #если число четное
continue
print(number)
```

Посмотрите демонстрационные видеоролики и ознакомьтесь с циклами и операторами if.

www.elluminetpress.com/pythonflow





Лабораторная работа

Взгляните на следующие упражнения и используйте полученные знания для решения задач.

1. Напишите программу для вывода на экран чисел от 1 до 10.
2. Напишите программу для вывода на экран списка имен.
3. Напишите программу для вычисления и печати квадратов чисел от 1 до 10. Используйте табуляции для отображения их в таблице.
4. Напишите программу, которая принимает число от пользователя до тех пор, пока не будет введено отрицательное число.
5. Напишите программу, которая принимает целое число и печатает указанный диапазон, к которому это число принадлежит.

   Диапазон 1: от 0 до 10

   Диапазон 2: от 11 до 20

   Диапазон 3: от 21 до 30

   Диапазон 4: от 31 до 40

Работа с файлами

Поскольку память компьютера (ОЗУ) является энергозависимой, все сохраненные данные теряются при отключении питания. Поэтому любые данные, которые необходимо хранить постоянно, следует сохранять в файле.

Файл — это именованное место на диске, которое используется для хранения данных. Каждый файл идентифицируется именем файла.

Python содержит встроенные функции для чтения данных из файлов, а также создания и записи файлов.

Для этого раздела посмотрите демонстрационные видео

`elluminetpress.com/pyfiles`

Вам также понадобятся исходные файлы из каталога Chapter 05.

Типы файлов

Существует два типа файлов: текстовые и двоичные. По умолчанию Python читает и записывает данные в виде текстового файла.

Текстовый файл

Текстовый файл хранит последовательности символов в формате ASCII: обычные текстовые файлы, файлы HTML, исходный код программы.

Используйте эти ключи при открытии файла в текстовом режиме:

- “r” открывает файл для чтения, возвращает сообщение об ошибке, если файл не существует
- “a” открывает файл для добавления информации, создает файл, если он не существует
- “w” открывает файл для записи, создает файл, если он не существует
- “r+” открывает файл как для чтения, так и для записи

Двоичный файл

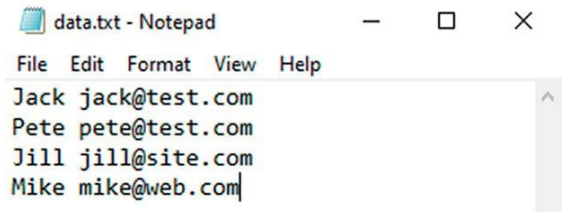
Двоичный файл хранит данные в виде последовательности байтов (единиц и нулей) — в том же формате, что и память компьютера (ОЗУ). Например, изображения, такие как JPEG или PNG, аудиофайлы, такие как WAV или MP3, видеофайлы, такие как MP4, и исполняемые файлы программ.

Используйте эти ключи при открытии файла в двоичном режиме:

- “rb” открывает файл для двоичного чтения, возвращает сообщение об ошибке, если файл не существует
- “ab” открывает файл для двоичного добавления информации, создает файл, если он не существует
- “wb” открывает файл для записи, создает файл, если он не существует
- “rb+” открывает файл как для чтения, так и для записи

Операции с текстовыми файлами

По умолчанию Python открывает файлы как текстовые файлы. Текстовые файлы содержат читаемые символы, как это показано в примере ниже.

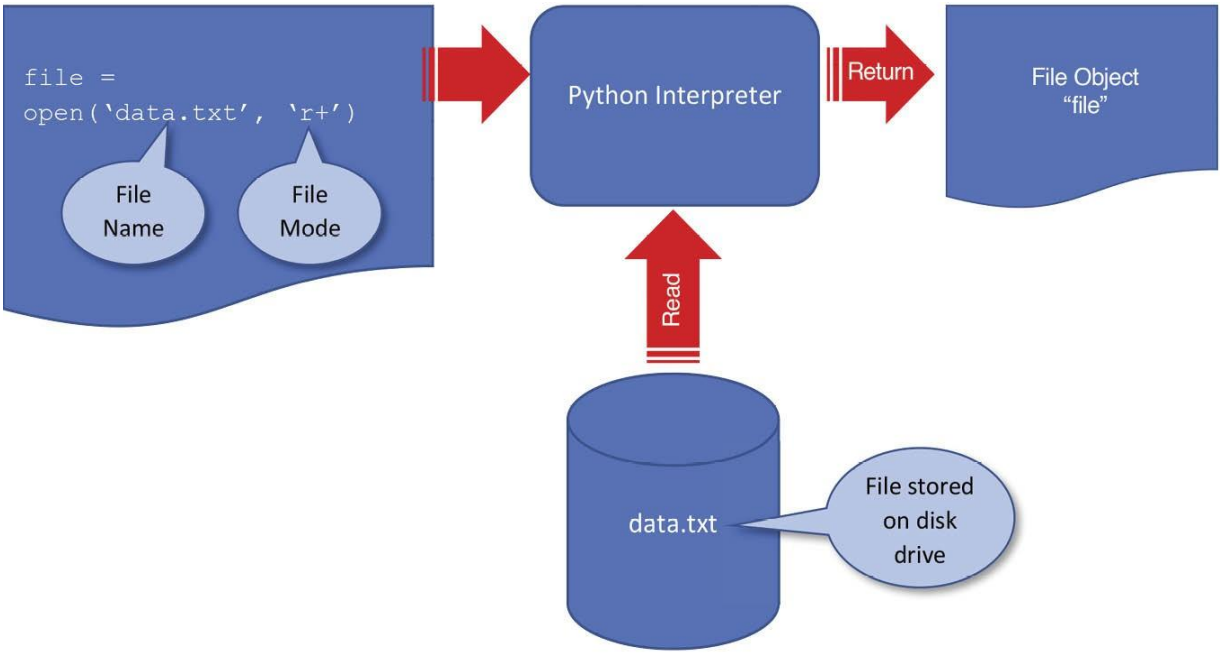


Как открыть файлы

Чтобы открыть файл, используйте функцию `open( )`. Эта функция возвращает объект файла, с которым мы можем работать (в этом примере он называется файлом).

```
file = open('data.txt', 'file mode')
```

Когда вы открываете файл, укажите имя файла и ключ файла в параметрах функции open().



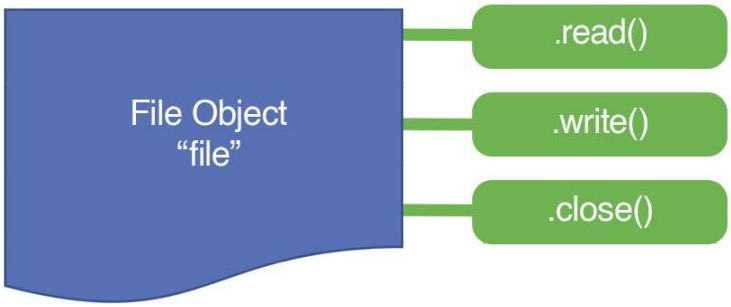
Файловый режим сообщает интерпретатору Python, что вы собираетесь делать с файлом, т. е. читать, записывать или добавлять информацию.

- “r” открывает файл для чтения, возвращает сообщение об

ошибке, если файл не существует

- “a” открывает файл для добавления информации, создает файл, если он не существует
- “w” открывает файл для записи, создает файл, если он не существует
- “r+” открывает файл как для чтения, так и для записи

Как только функция `open()` возвращает файловый объект, мы можем работать с файлом, используя методы объекта, такие как `.read()`, `.write()` или .close()



Как записать в файл

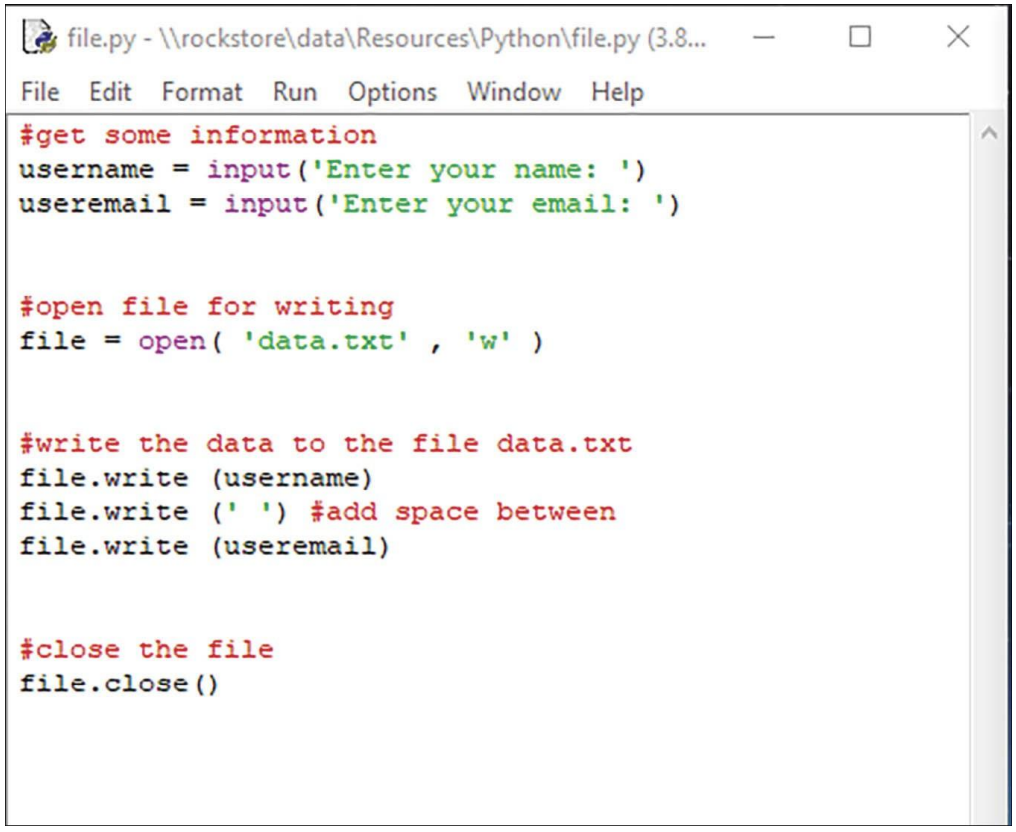
Для записи данных в файл используйте `.write()`. Этот метод записывает указанный текст в файл.

```
file.write("Data to write to the file...")
```

При открытии файла для записи используйте либо

- "a" открывает файл для добавления, создает файл, если он не существует. Новые данные добавляются в конец файла.
- "w" открывает файл для записи, создает файл, если он не существует. Перезаписывает любые существующие в файле данные.

Давайте рассмотрим программу. Откройте file.py.

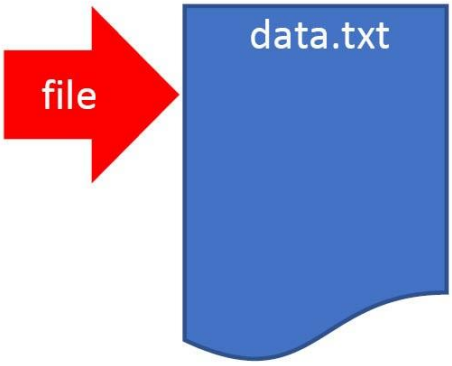


Здесь мы получаем некоторую информацию от пользователя (имя пользователя и адрес электронной почты).

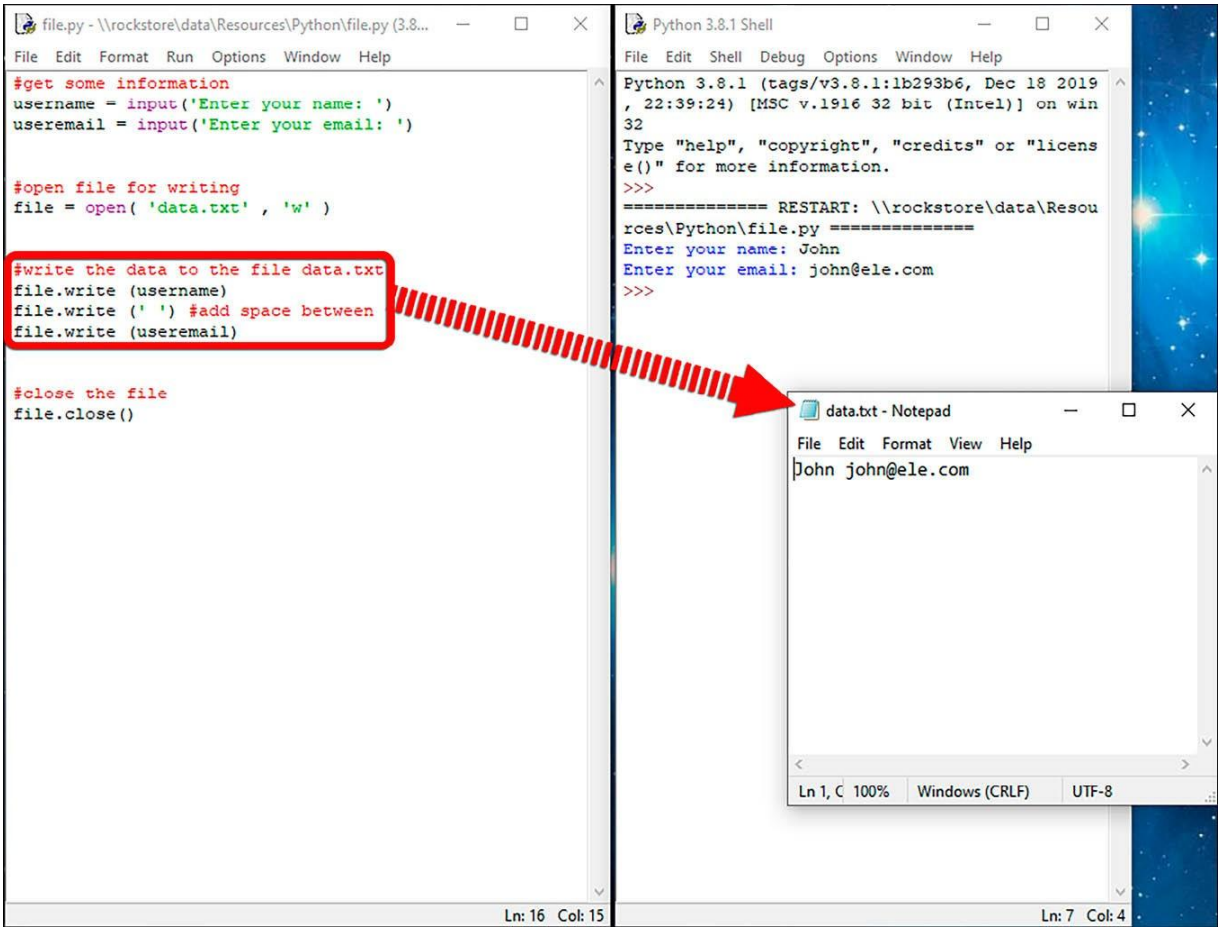
```
username = input('Enter your name: ')
useremail = input('Enter your email: ')
```

Затем мы открываем файл с именем ‘data.txt’ для записи и присваиваем его объекту с именем ‘file’.

file = open(‘data.txt’, ‘w’)



Затем мы записываем имя пользователя и адрес электронной почты в файл, используя метод `.write` для файлового объекта.



Помните, что файловый режим ‘w’ открывает файл для записи. Это также означает, что любые новые данные перезапишут любые данные, уже хранящиеся в файле. После завершения операций с файлом мы закрываем наш файл.

```
file.close()
```

Чтение из файла

Чтобы прочитать данные из файла, используйте метод `.read()` для чтения всего файла.

```
fileContent = fileName.read( )
```

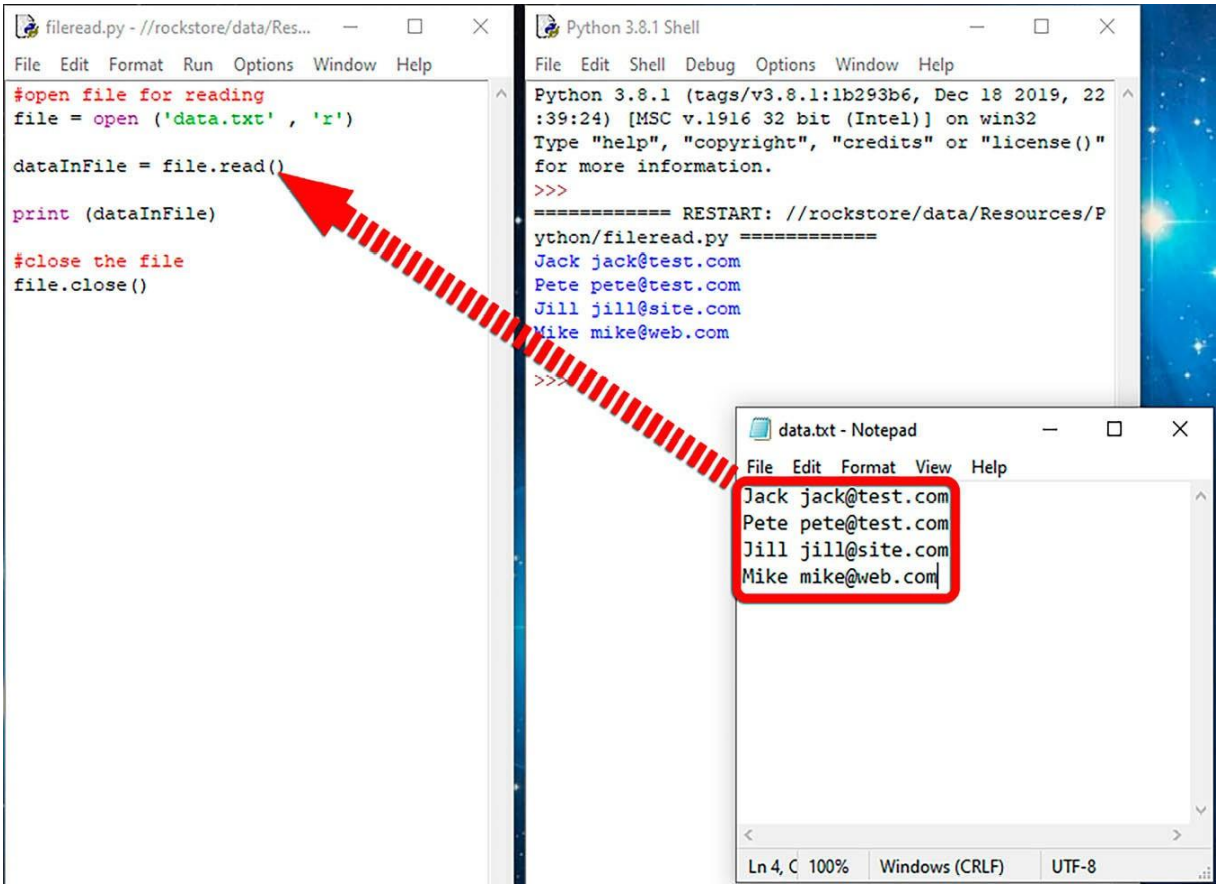
Используйте метод `.readline()` для построчного чтения.

```
nextLine = fileName.readline( )
```

При открытии файла для чтения используйте следующие ключи

- “r” открывает файл для чтения, возвращает сообщение об ошибке, если файл не существует
- “r+” открывает файл как для чтения, так и для записи

Давайте рассмотрим программу. Откройте файл fileread.py.


fileread.py - //rockstore/data/Res...
File Edit Format Run Options Window Help
#open file for reading
file = open ('data.txt' , 'r')
dataInFile = file.read()
print (dataInFile)
#close the file
file.close()
Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22
:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()"
for more information.
>>>
========== RESTART: //rockstore/data/Resources/P
ython/fileread.py ==========
Jack jack@test.com
Pete pete@test.com
Jill jill@site.com
Mike mike@web.com
>>>
data.txt - Notepad
File Edit Format View Help
Jack jack@test.com
Pete pete@test.com
Jill jill@site.com
Mike mike@web.com
Ln 4, C 100% Windows (CRLF) UTF-8

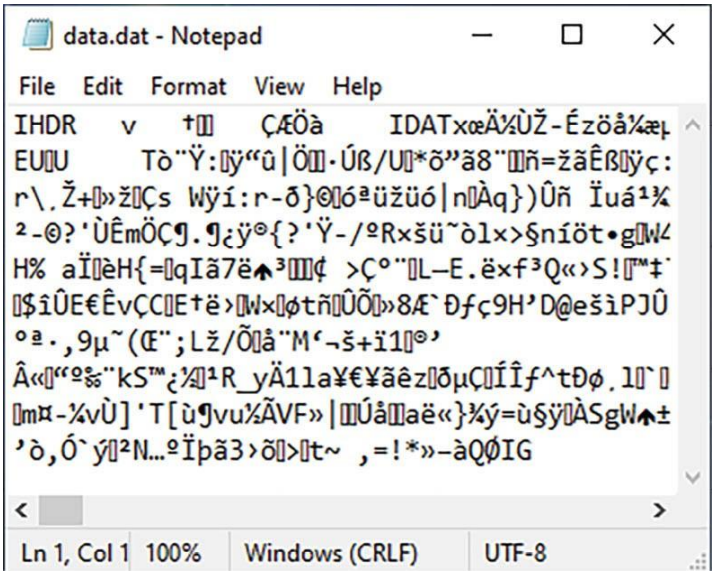
Здесь мы открываем файл с именем ‘data.txt’ и присваиваем его объекту с именем ‘file’. Затем мы читаем данные, используя метод `.read` объекта ‘file’.

Наконец мы закрываем файл.

Операции с двоичными файлами

Большая часть цифровых данных хранится в двоичных файлах, поскольку они намного меньше и быстрее текстовых файлов.

Двоичные файлы не доступны для чтения человеком, как это показано в примере ниже. Здесь мы открыли двоичный файл в текстовом редакторе.



Как открыть файлы

Чтобы открыть файл, используйте функцию `open()`.

```
file = open('data.dat', 'file mode')
```

Когда вы открываете файл, укажите имя файла и ключ файла в параметрах функции `open()`.

Ключ сообщает интерпретатору Python, что вы собираетесь делать с файлом, т. е. читать, записывать или добавлять.

- “rb” открывает файл для чтения, возвращает сообщение об ошибке, если файл не существует
- “ab” открывает файл для добавления, создает файл, если он не существует
- “wb” открывает файл для записи, создает файл, если он не существует

- “rb+” открывает файл как для чтения, так и для записи

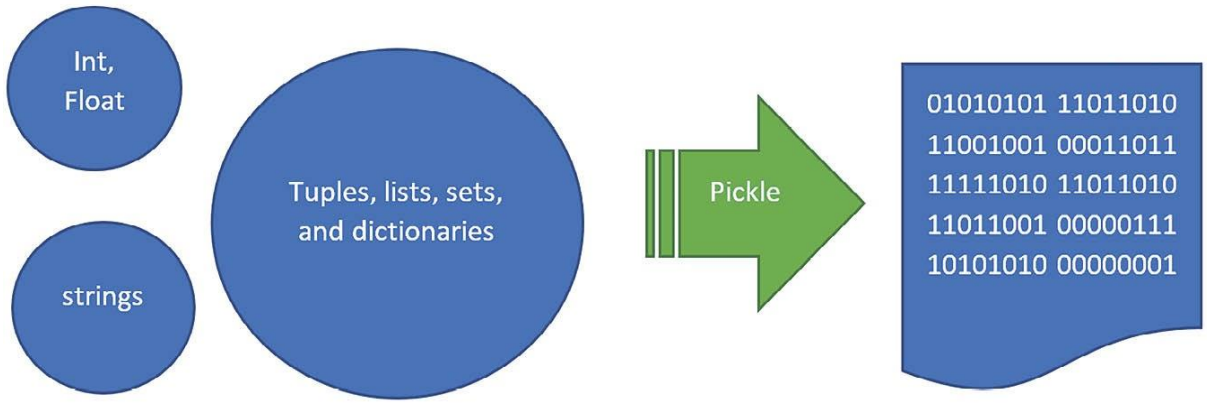
Это зависит от цели вашей программы.

Хорошей практикой является открытие файла, выполнение необходимых операций, а затем закрытие этого файла.

Запись в файл

Метод `.write()` записывает данные в текстовом формате, и если вы попытаетесь записать данные в двоичном режиме с помощью этого метода, вы получите сообщение об ошибке при запуске программы.

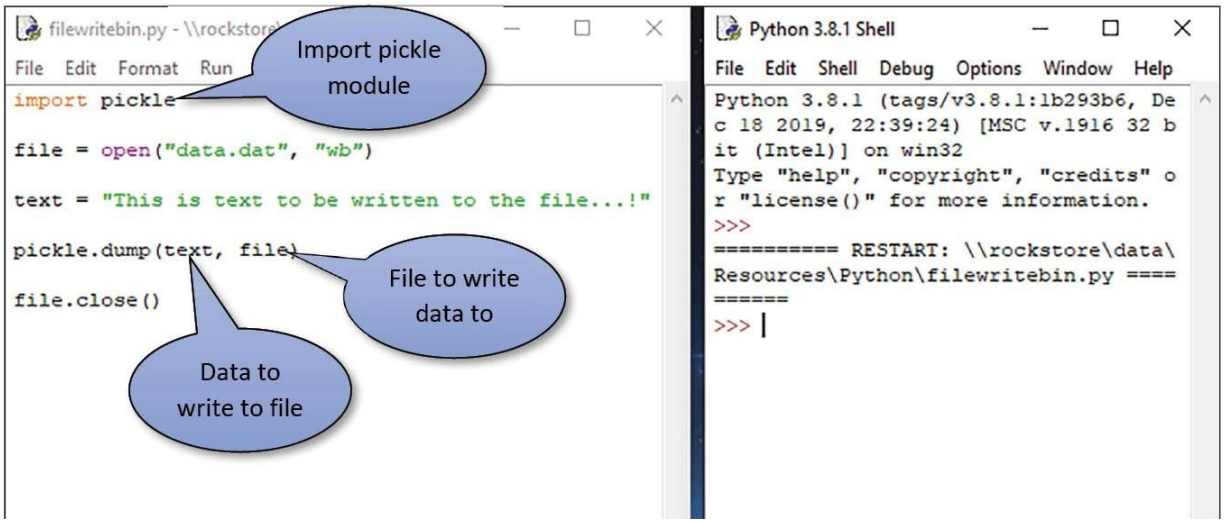
Чтобы записать данные в двоичном формате, сначала нам нужно преобразовать их в последовательность байтов. Мы можем сделать это с помощью модуля pickle, используя процесс, называемый пиклингом. Пиклинг — это процесс, при котором объекты данных, такие как целые числа, строки, списки и словари, преобразуются в поток байтов.



Для записи в файл используйте метод `pickle.dump()`

```
pickle.dump (data-to-be-written, file-to-write-
to)
```

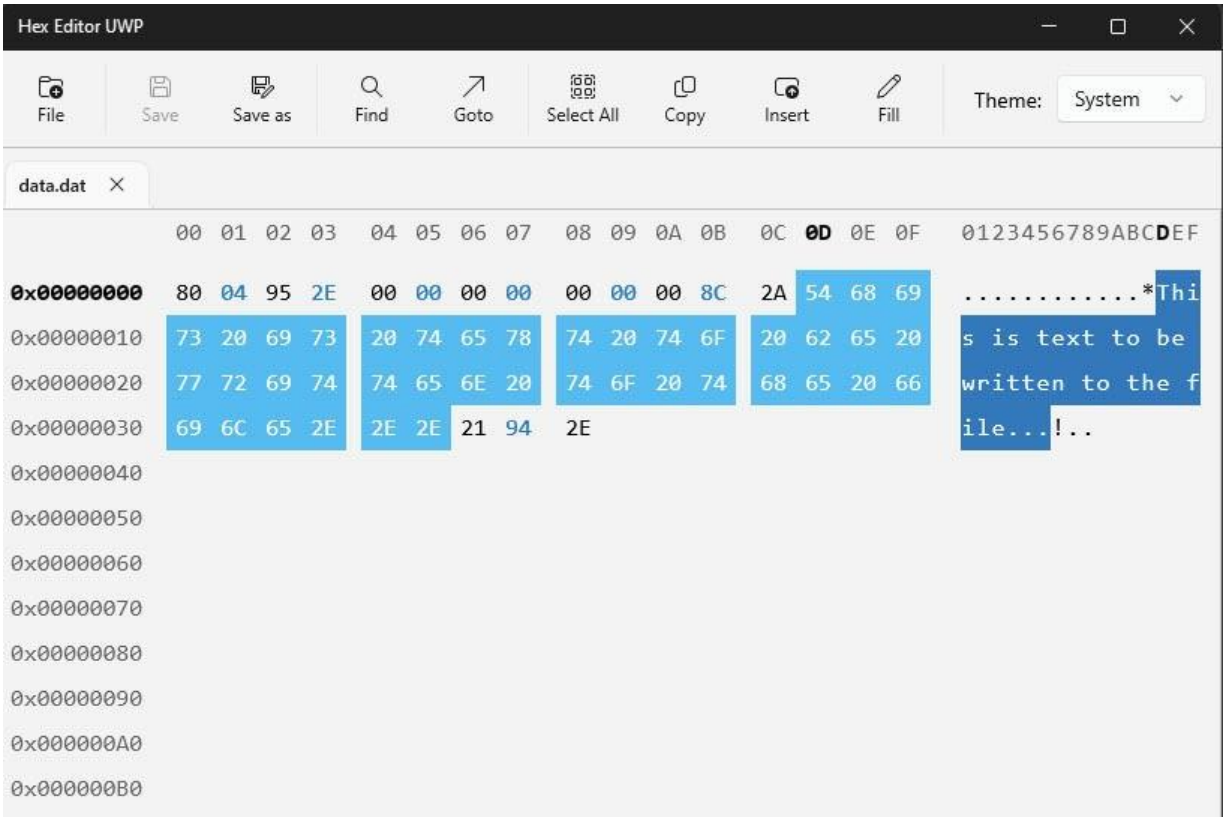
Давайте посмотрим на программу. Откройте filewritebin.py. Сначала нам нужно подключить модуль pickle. Вы можете сделать это с помощью команды импорта.



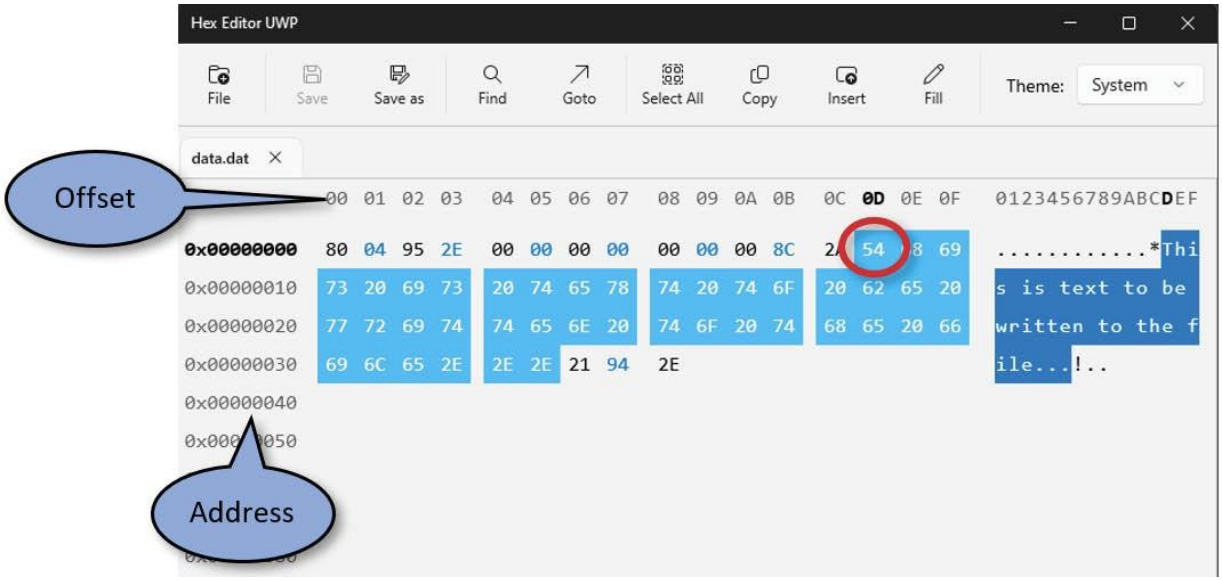
Далее открываем файл обычным способом, только используем при этом ключ двоичной записи (wb).

Теперь, чтобы записать данные в файл, мы выбираем объект ‘text’ с помощью метода `pickle.dump()`. Наконец закрываем файл.

Вот данные, записанные в файл. В шестнадцатеричном редакторе ниже вы можете видеть, что строка “Data that has been written to a binary file”, была преобразована в двоичные данные.



Обратите внимание на то, что данные посередине представлены шестнадцатеричными числами, которые часто используются для сокращения записи в двоичном формате. Каждое шестнадцатеричное число в шестнадцатеричном блоке представляет собой 1 байт.



Чтение файла

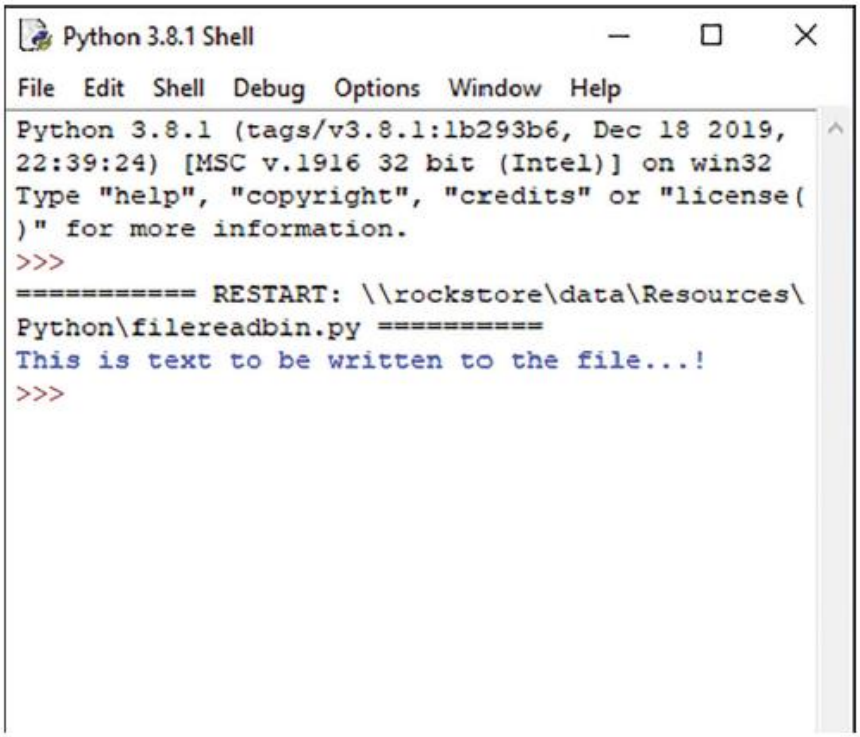
Помните, когда мы записывали данные в двоичный файл, мы использовали процесс, называемый pickle. Что ж, для чтения данных из файла мы используем аналогичный процесс.

Чтобы прочитать файл, используйте метод `pickle.load()`

```
pickle.load(file-to-read-from)
```

Давайте рассмотрим на программу. Откройте filereadbin.py. Сначала нам нужно подключить модуль Pickle. Вы можете сделать это с помощью команды импорта.

Когда мы запускаем программу, данные считываются из файла, робразуются, а затем присваиваются переменной ‘data’. Теперь мы можем вывести ‘data’ на экран.



Произвольный доступ к файлам

Когда файл открывается, интерпретатор Python выделяет указатель внутри файла. Этот указатель определяет позицию в файле, откуда будет происходить чтение или запись. Указатель можно переместить в любое место файла.

Чтобы переместить указатель файла, используйте метод `.seek()`.

```
file.seek(position-in-file, whence)
```

Первый параметр (position-in-file) определяет, на сколько байт нужно переместиться. Положительное значение переместит указатель вперед, отрицательное значение — назад. Позиция в файле называется смещением.

Второй параметр (whence) определяет начальную точку в файле, он принимает одно из трех значений:

- 0 : устанавливает начальную точку в начало файла (используется по умолчанию)
- 1 : устанавливает начальную точку в текущую позицию
- 2 : устанавливает начальную точку в конец файла

В файле каждая позиция может содержать один байт или один символ. Помните, система отсчета начинается с 0.

Используя наш текстовый файл в качестве примера, file.seek(5) переместит указатель файла на 6-й байт.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	J	a	c	k		j	a	c	k	@	t	e	s	t	.	c	o	m
18	P	e	t	e		p	e	t	e	@	t	e	s	t	.	c	o	m

```
file.seek(23, 0)
```

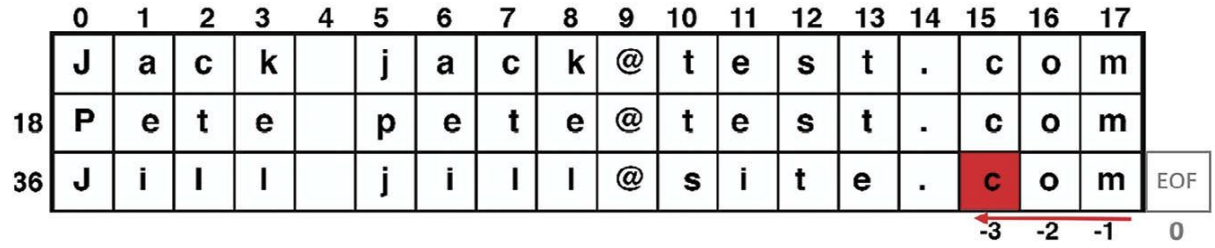
Это переместит указатель в позицию 23 от начала файла

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	J	a	c	k		j	a	c	k	@	t	e	s	t	.	c	o	m
18	P	e	t	e		p	e	t	e	@	t	e	s	t	.	c	o	m
36	J	i	l	l		j	i	l	l	@	s	i	t	e	.	c	o	m

Если вы хотите прочитать файл с конца, вам нужно будет считать в обратном направлении от конца файла, это делается с помощью отрицательного смещения. Функция `seek()` с отрицательным смещением работает только тогда, когда файл открыт в двоичном режиме.

Чтобы переместить указатель из конца файла

```
f.seek(-3, 2)
```



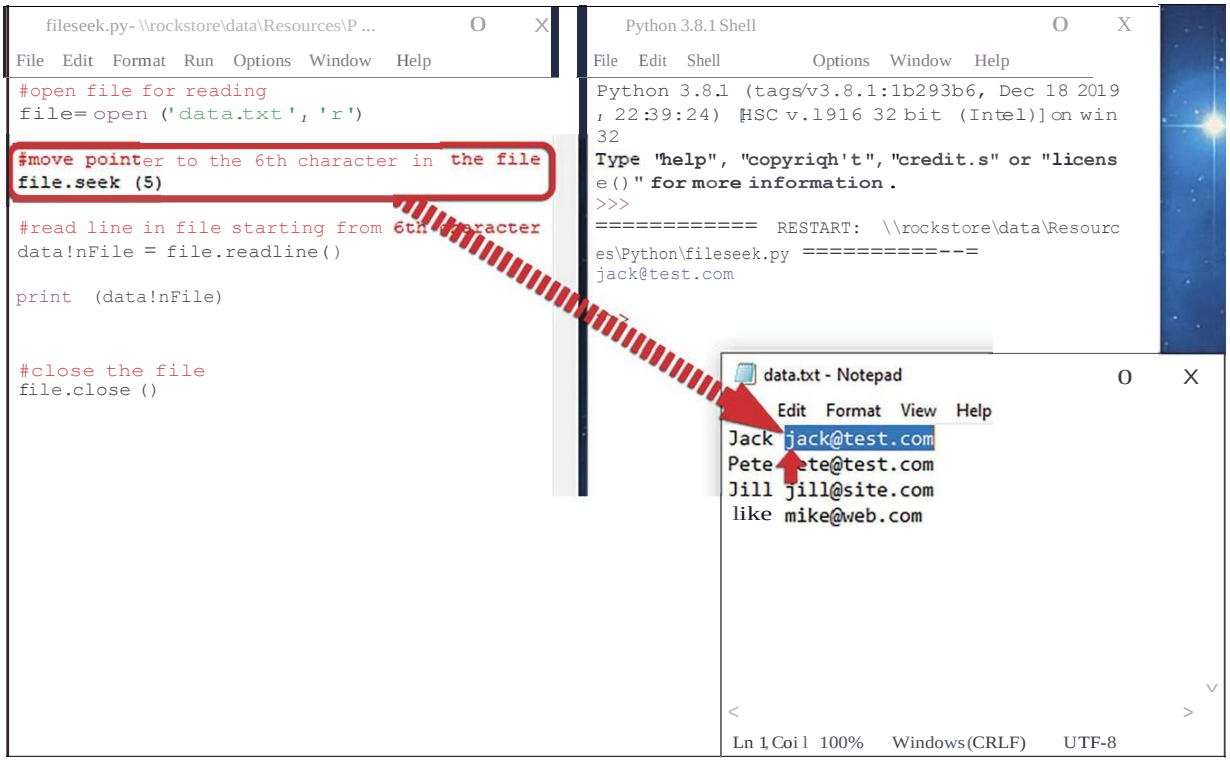
Используйте `.decode('utf-8')`, чтобы преобразовать двоичный файл обратно в текст.

```
datafromfile = f.readline().decode('utf-8')
```

Чтобы найти текущую позицию указателя файла в файле, используйте метод `.tell()`.

```
file.tell()
```

Давайте рассмотрим программу. Здесь мы начинаем чтение первой строки файла data.txt, начиная с 6-го байта или символа.


fileseek.py- \\rockstore\data\Resources\P ...
File Edit Format Run Options Window Help
#open file for reading
file= open ('data.txt', 'r')
#move pointer to the 6th character in the file
file.seek (5)
#read line in file starting from 6th character
data!nFile = file.readline()
print (data!nFile)
#close the file
file.close ()
Python 3.8.1 Shell
File Edit Shell Options Window Help
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019
, 22:39:24) [MSC v.1916 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "licens
e()" for more information.
>>>
========== RESTART: \\rockstore\data\Resourc
es\Python\fileseek.py ==========--=
jack@test.com
data.txt - Notepad
Edit Format View Help
Jack jack@test.com
Pete pete@test.com
Jill jill@site.com
like mike@web.com
Ln 1, Col 1 100% Windows (CRLF) UTF-8

Методы работы с файлами

Вот краткий обзор методов, доступных для файловых объектов. Вы можете связать наименования методов, приведенные ниже, с наименованием объекта, используя следующий синтаксис:

```
fileobject.method()
```

Здесь у нас приведены различные методы закрытия файла, а также некоторые другие распространенные методы.

Метод	Описание
`close ()`	Закрывает файл
`detach ()`	Возвращает выделенный необработанный поток из буфера
`fileno ()`	Возвращает число, представляющее поток с точки зрения операционной системы
`flush ()`	Очищает внутренний буфер
`isatty ()`	Возвращает, является ли файловый поток интерактивным или нет

Еще существуют методы чтения данных из файла. Вы можете прочитать весь файл или строки из файла.

Метод	Описание
`read ()`	Возвращает содержимое файла
`readable ()`	Возвращает, может или нет быть прочитан файловый поток
`readline ()`	Возвращает из файла одну строку
`readlines ()`	Возвращает из файла список строк

Различные способы записи данных в файл. Вы можете проверить, доступен ли файл для записи, или записать данные в файл.

Метод	Описание
`writable ()`	Возвращает, может или нет быть произведена запись в файл
`write ()`	Записывает указанную строку в файл
`writelines ()`	Записывает в файл список строк

Различные другие методы поиска позиции в файле, а также метод для возврата текущей позиции в файле и метод для усечения файла до

определенного размера.

Метод	Описание
`seek ()`	Изменяет положение в файле
`seekable ()`	Возвращает, позволяет ли файл нам именить положение в файле
`tell ()`	Возвращает текущее положение в файле
`truncate ()`	Изменяет размер файла до заданного значения

Лабораторная работа

Взгляните на следующие упражнения и используйте полученные знания для решения задач.

1. Напишите программу, которая получает строку от пользователя и записывает ее в файл вместе с именем пользователя.
2. Измените программу из упражнения 1 так, чтобы она добавляла данные в файл, а не перезаписывала их.
3. Напишите программу для записи списка имен в файл.
4. Напишите программу для чтения файла построчно и сохранения его в списке.
5. В чем разница между текстовым файлом и двоичным файлом?

Использование функций

Функции помогают разбить программу на более мелкие фрагменты. Это позволяет избежать повторения кода, что делает более крупные программы более эффективными и простыми в обслуживании.

Для этого раздела посмотрите демонстрационные видео

`elluminetpress.com/pyfunctions`

Вам также понадобятся исходные файлы из каталога Chapter 06.

Что такое функции

Функция — это блок кода, который выполняется только при его вызове. Python поставляется с библиотекой функций для выполнения широкого спектра задач. Это так называемые встроенные функции. Например:

```
print ()
input ()
open ()
```

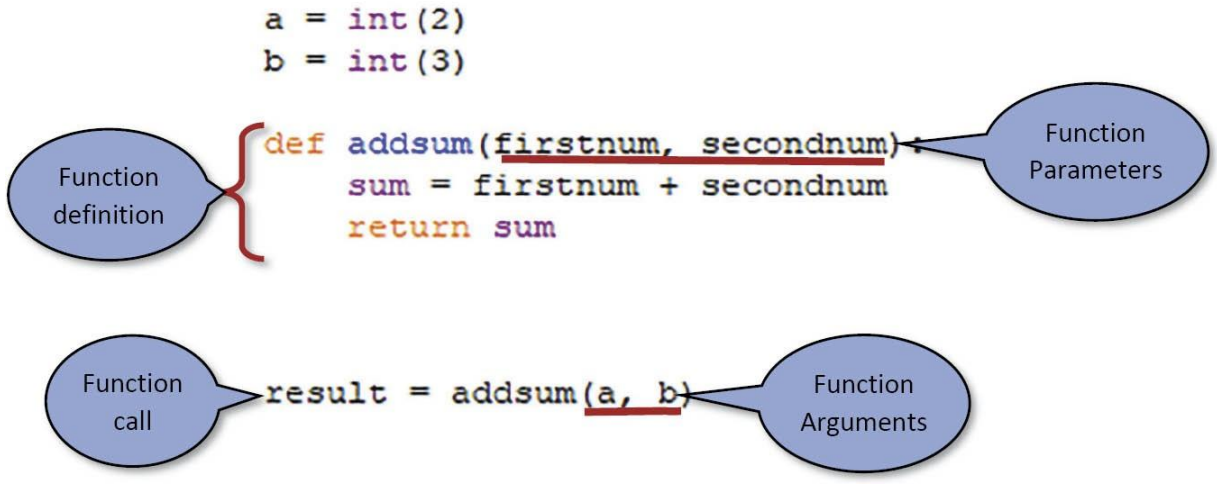
Вы также можете создавать свои собственные функции. Эти функции называются пользовательскими функциями. Например:

```
def functionName (parameters):
<function code>
```

Функция принимает параметры или аргументы, заключенные в круглые скобки, и возвращает результат. Кажется, что большинство людей используют оба этих термина как синонимы, но это не одно и то же.

- Параметр – это переменная в определении функции.
- Аргумент – это значение, передаваемое во время вызова функции.

Например, в следующем коде. Определение функции ‘addnum’ принимает два параметра ‘firstnum’ и ‘secondnum’.



В строке внизу мы вызываем функцию ‘addnum’ и передаем ей два аргумента ‘a’ и ‘b’.

Встроенные функции

Python имеет различные встроенные функции. Вот некоторые из наиболее распространенных из них.

Функция	Описание	Пример
abs()	Возвращает абсолютное значение числа	`abs(val)`
bin()	Возвращает двоичную версию числа	`bin(val)`
bool()	Возвращает логическое значение указанного объекта	`bool(val)`
bytearray()	Возвращает массив байтов	`bytearray(val)`
bytes()	Возвращает байтовый объект	`bytes(val)`
chr()	Возвращает символ из указанного Юникода	`chr(65)`
dict()	Возвращает словарь (массив)	`diet(id = "A34",name = "USA")`
divmod()	Возвращает частное и остаток при делении аргумента на аргумент	`divmod(17, 3)`
filter()	Используйте функцию фильтра, чтобы исключить элементы из итерируемого объекта	`filter (function, item-to-be-filtered)`
float()	Возвращает число с плавающей запятой	`float(val)`
format()	Форматирует указанное значение Замените слово «формат» на 'b' - двоичный формат 'd' - десятичный формат 'e' — научный формат, с маленькой буквой e. 'E' — научный формат, с заглавной буквой E. 'f - формат номера фиксированной точки 'F' — формат номера фиксированной точки, верхний регистр. 'o' - восьмеричный формат 'x' — шестнадцатеричный формат, строчные буквы. 'X' — шестнадцатеричный формат, верхний регистр. 'n' — числовой формат '%' - процентный формат	`format(value, format)`

help()	Запускает встроенную справочную систему	
hex()	Преобразует число в шестнадцатеричное значение	`hex(val)`
input()	Обеспечивает пользовательский ввод	`val=input ( 'enter../ )`
int()	Возвращает целое число	`int(val)`
len()	Возвращает длину объекта	`len(list)`
list()	Возвращает список	`list (vals)`
max()	Возвращает наибольший элемент	`max(2,43)`
min()	Возвращает наименьший элемент	`min (2,43)`
oct()	Преобразует число в восьмеричное	`oct(val)`
open()	Открывает файл и возвращает файловый объект	`open("file", "mode")`
pow()	Возвращает значение x в степени y	`pow(2, 3)`
print()	Печатает на стандартном устройстве вывода	`print(val to print)`
range()	Возвращает последовательность чисел, начиная с 0 и увеличивая ее на 1 (по умолчанию)	`range(start, stop, incr)`
round()	Округляет число	`round(number, digits)`
set()	Возвращает новый заданный объект	
slice()	Возвращает объект в виде фрагмента	`slice(start , end , step)`
sorted()	Возвращает отсортированный список	`sorted (iterable, key, reverse)`
str()	Возвращает объект в виде строки	`str(object, encoding)`
sum()	Суммирует элементы итератора	`sum(vals)`
super()	Возвращает объект, представляющий родительский класс	`super ()  init ()`

Пользовательские функции

Вы можете объявить новую функцию, используя ключевое слово def, за которым следует имя функции.

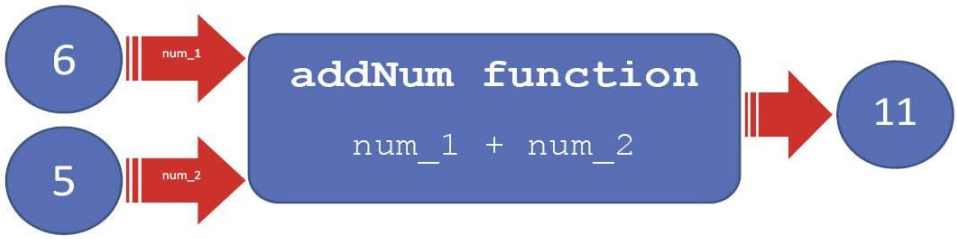
```
def functionName(parameters):

code to be executed in function
```

Если функция принимает параметры, вы можете включить их в круглые скобки рядом с наименованием функции.

Например, если бы мы написали функцию для сложения двух чисел, мы могли бы написать что-то вроде этого:

```
def addNum(num_1, num_2):

return num_1 + num_2
```

Эта функция принимает два числа в качестве параметров ‘num1’ и ‘num2’, складывает их и возвращает результат.



Вы можете вызвать функцию следующим образом с аргументами 6 и 5 в скобках:

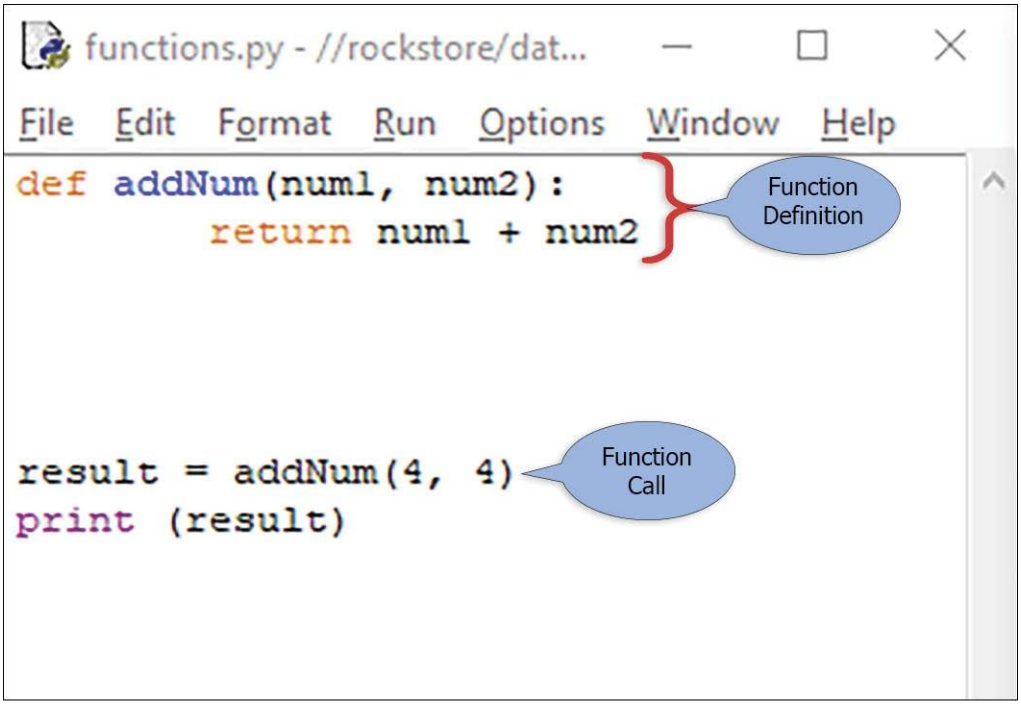
```
result = addNum(6, 5)
```

Для небольших программ вы можете объявить свои функции в одном и том же файле - обычно вверху, но по мере того, как программы становятся больше и сложнее, вам следует объявить свои функции в отдельном файле, а затем включить этот файл в свой основной скрипт. Это позволяет вам модульизировать и повторно использовать код — это хорошая практика программирования для более крупных проектов.

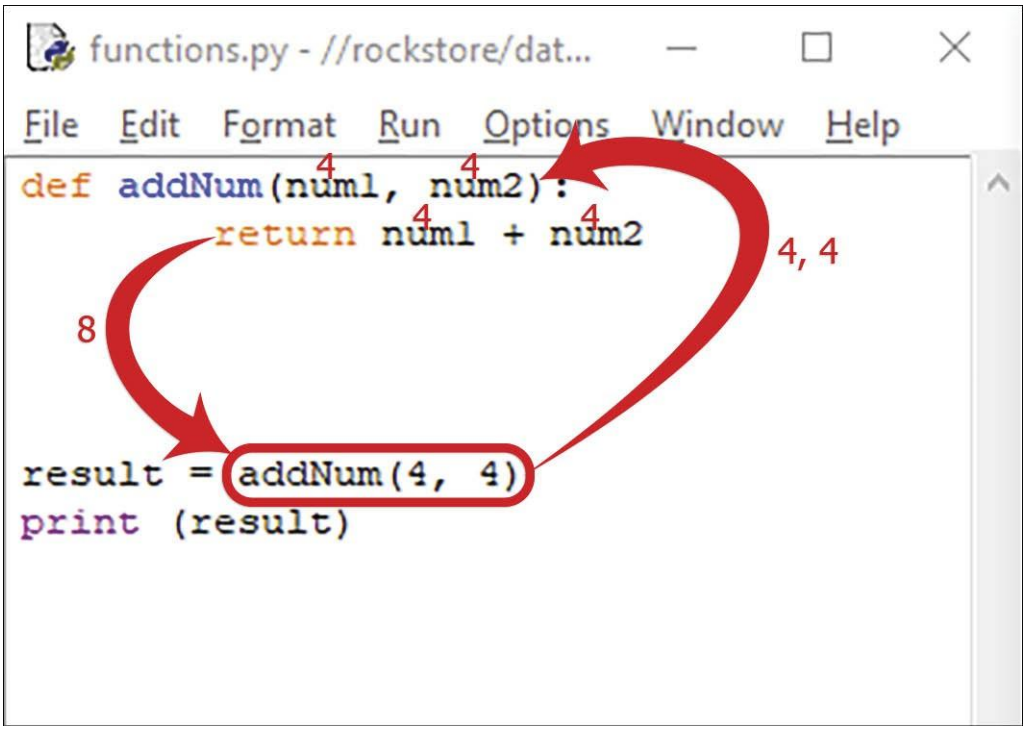
Мы можем объявить нашу функцию addNum в файле myfunctions.py и включить ее в файл functionmain.py. Это называется модулем (подробнее о модулях см. в Главе 7). Чтобы включить функции в другой скрипт, используйте ключевое слово import.

```
import myfunctions
```

Давайте рассмотрим программу. Откройте файл function.py. Здесь, в верхней части скрипта, мы определили простую функцию для сложения двух чисел.



В нижней части скрипта мы вызываем нашу функцию addNum и передаем два значения в качестве аргументов (4, 4). Мы используем ключевое слово return для возврата результата.

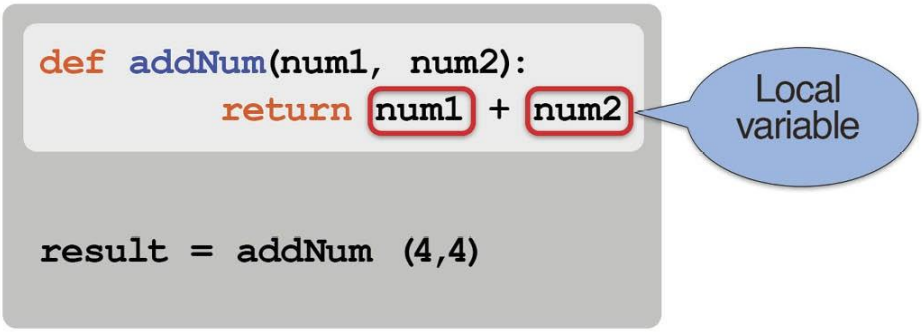


Результат, возвращаемый функцией, затем присваивается переменной ‘result’. Наконец, мы выводим на экран содержимое переменной ‘result’ и видим, что происходит.

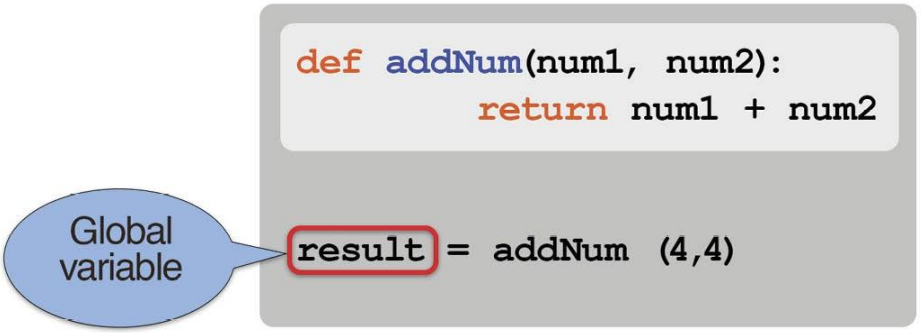
Область видимости

Часть программы, где доступна переменная, называется ее областью видимости. В этом разделе мы рассмотрим локальную и глобальную область видимости.

Если переменная доступна только внутри области, в которой она создана (например, переменная, созданная внутри функции), то она принадлежит локальной области видимости этой функции и может использоваться только внутри этой функции. Это называется локальной областью видимости.



Переменная, созданная в основной части кода Python, является глобальной переменной и принадлежит глобальной области видимости. Глобальные переменные доступны из любой области, глобальной и локальной.



Период, в течение которого переменная существует в памяти, называется ее временем существования. Переменные, определенные внутри функции, существуют только во время выполнения функции.

Как только функция возвращается, переменные внутри функции уничтожаются.

Рекурсия

Рекурсивная функция — это функция, которая может вызывать сама себя. Это позволяет функции повторяться несколько раз.

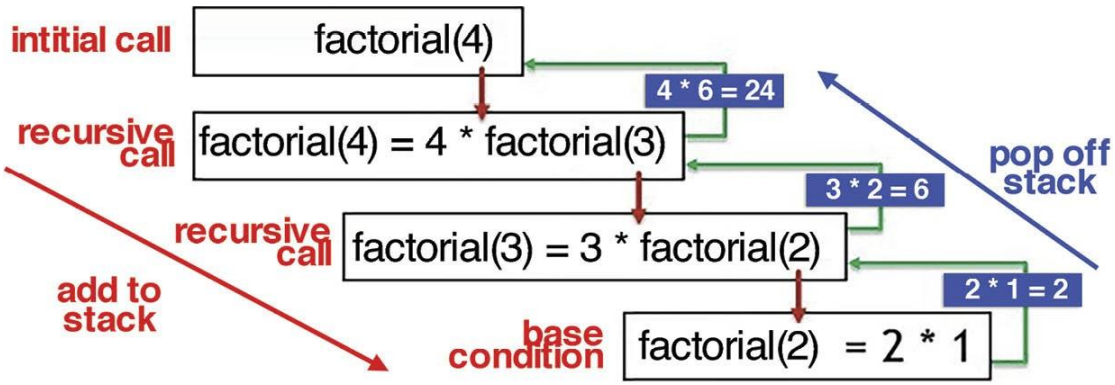
Откройте файл recursion1.py. Здесь у нас есть рекурсивная функция, которая вычисляет факториал числа. Помните, что для вычисления факториала необходимо перемножить все числа от 1 до заданного числа.

```
def factorial(n):
if n <= 1:
return 1
else:
return n * factorial(n-1)
```

Если мы вызываем функцию факториала и передаем положительное целое число (n)

```
factorial(4)
```

Она будет рекурсивно вызывать сама себя, каждый раз уменьшая число, передаваемое функции (n), на единицу, а затем добавляя вызов в стек вызовов.



Если мы введем 4, функция факториала вызовет саму себя и передаст (n-1, что равно 3) в качестве аргумента.

```
return 4 * factorial(3)
```

При следующем вызове функция пройдет (n-1, что равно 2)

```
return 3 * factorial(2)
```

При следующем вызове функция пройдет (n-1, что равно 1)

```
return 2 * factorial(1)
```

Рекурсия заканчивается, когда число (n) уменьшается до 1.

```
return 1
```

Это называется базовым условием (помните, что в функции мы имели **if n <= 1**). Это возвращает 1 и завершает рекурсивные вызовы.

Как только мы достигаем базового условия, каждый вызов извлекается из стека и оценивается.

Сначала извлекаем factorial(1), который в настоящее время равен 1, затем умножьте на 2 = 2

```
2 * factorial(1)
```

Затем извлекаем factorial(2), который теперь равен 2, затем умножьте на 3 = 6

```
3 * factorial(2)
```

Наконец, извлекаем factorial(3), который теперь равен 6, затем умножьте на 4 = 24

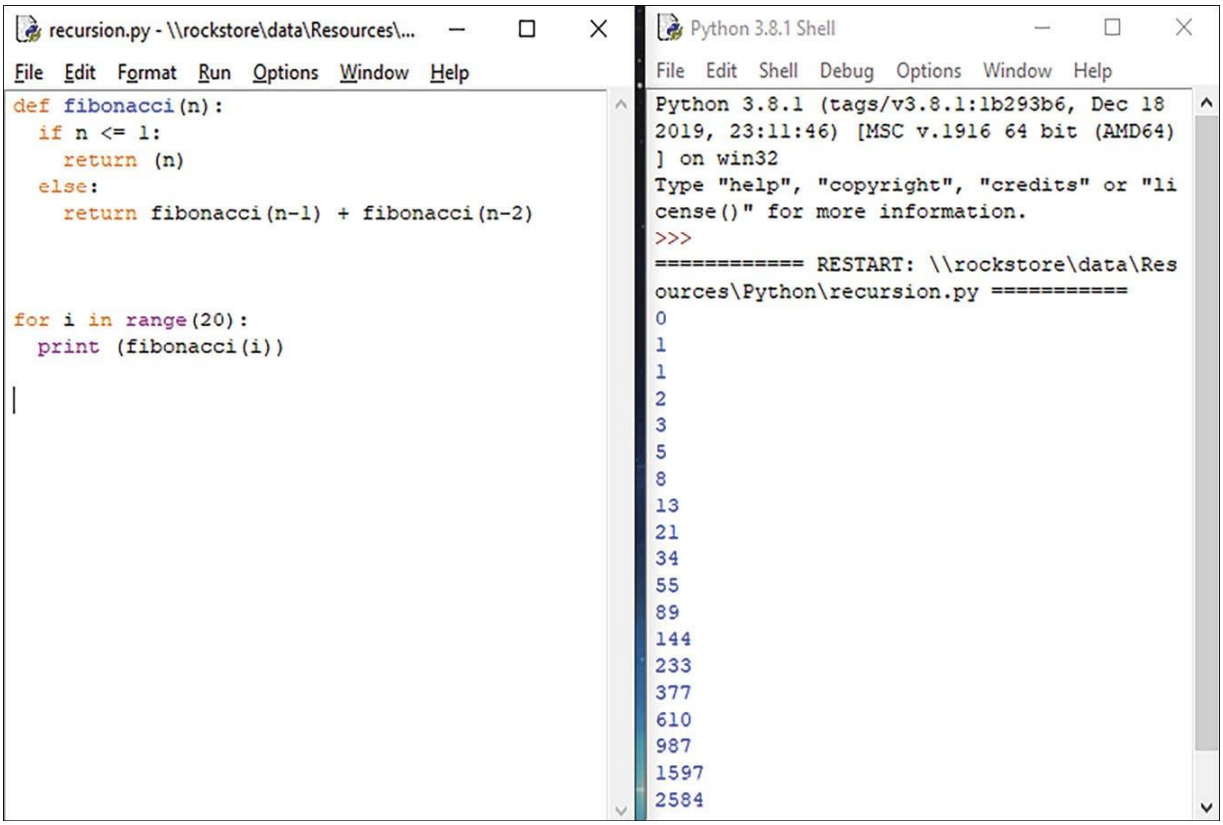
```
4 * factorial(3)
```

Посмотрите демо-версию рекурсии. Здесь мы записали рекурсивную функцию по мере прохождения кода, чтобы мы могли видеть стек и переменные во время ее выполнения построчно. Перейдите на следующий веб-сайт:

```
elluminetpress.com/pyfunctions
```

Выберите ‘recursion functions’. Изучите процедуру, чтобы увидеть, как она работает.

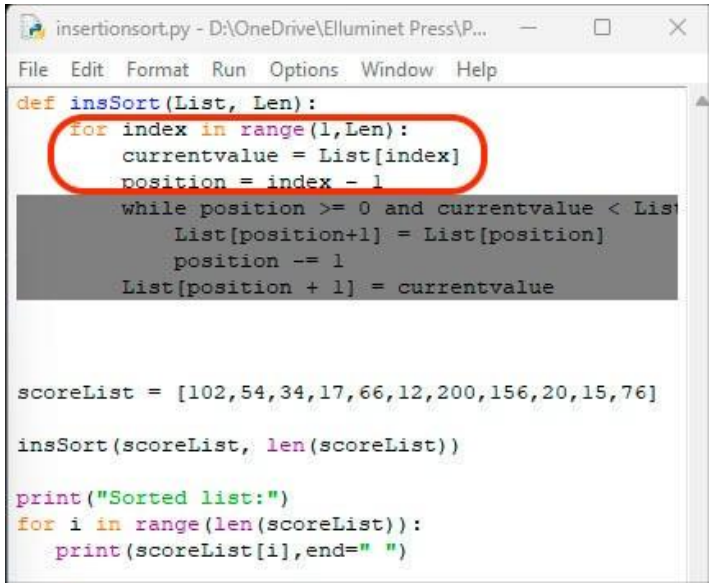
Вот еще один пример: рекурсивная функция для вывода чисел Фибоначчи. Посмотрите recursion.py. Пройдитесь по коду и изучите, как он работает.



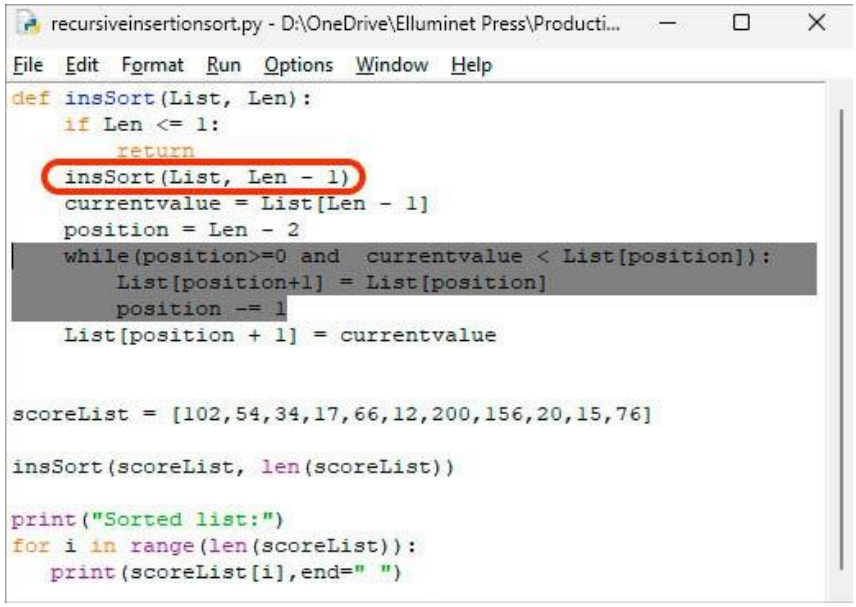
Рекурсивные программы также можно писать с использованием итераций, так зачем заморачиваться рекурсией? Рекурсивные программы позволяют программистам писать эффективные функции с использованием минимального объема кода и являются полезным методом, позволяющим уменьшить длину кода и облегчить его чтение и запись.

Рекурсия хорошо работает для таких алгоритмов, как обход двоичного дерева или алгоритм сортировки и генерации фракталов. Однако, если производительность имеет решающее значение, лучше использовать итерацию, поскольку рекурсия может быть намного медленнее.

Вот пример сортировки вставками, написанный с использованием итерации (цикл while). Взгляните на insertionsort.py.



Вот тот же алгоритм, написанный с использованием рекурсии. Взгляните на recursiveinsertionsort.py.



Найдите отличие.

Лабораторная работа

1. Напишите программу, которая принимает число от пользователя и использует функцию для возведения числа в квадрат, а затем возвращает результат. Распечатайте результат на экране.
2. Напишите функцию, возвращающую наибольшее из двух чисел. Проверьте функцию и распечатайте результаты на экране.
3. В чем разница между локальной и глобальной переменной?
4. Напишите программу, которая печатает первые 10 положительных чисел, используя рекурсивную функцию.
5. В чем разница между параметром и аргументом?
6. Что такое встроенная функция?
7. Что такое пользовательские функции?
8. Что делает функцию рекурсивной?
9. Каковы преимущества и недостатки рекурсии?
10. Взгляните на следующую программу

```
def fibonacci(n):
if n <= 1:
return (n)
else:
return fibonacci(n-1) + fibonacci(n-2)
for i in range(5):
print (fibonacci(i))
```

Отследите вызовы функций программы.

Использование модулей

При разработке более сложных приложений Python по мере увеличения размера программы рекомендуется разделить программу на несколько файлов для упрощения обслуживания и возможности повторного использования кода. Для этой цели используются модули.

Модули — это просто файлы с расширением .py, содержащие код, который можно импортировать в другую программу.

При этом мы можем создать библиотеку кода, содержащую набор функций, которые вы хотите включить при разработке более крупных приложений.

В этом разделе мы рассмотрим, как создавать модули и включать их в наши программы на Python. Посмотрите демо видео.

`elluminetpress.com/pyfunctions`

Вам также понадобятся исходные файлы из каталога Chapter 07.

Импорт модулей

Python has a whole library of modules you can import into your programs. Here are some common built in modules you can use.

- **math** — Математические функции
- **turtle** — Черепашья графика
- **tkinter** — Набор инструментов графического интерфейса
- **pygame** — Инструментарий для создания игр и других мультимедийных приложений.

Чтобы импортировать модули в ваш код, используйте ключевое слово import. В этом примере я собираюсь использовать ключевое слово import для импорта графического модуля Turtle в программу Python. Для этого вводим следующую строку в начале программы:

```
import moduleName
```

Чтобы вызвать функцию из импортированного модуля, используйте

```
moduleName.function()
```

Например, если бы мы хотели использовать черепашью графику, мы бы импортировали Turtle

```
import turtle
```

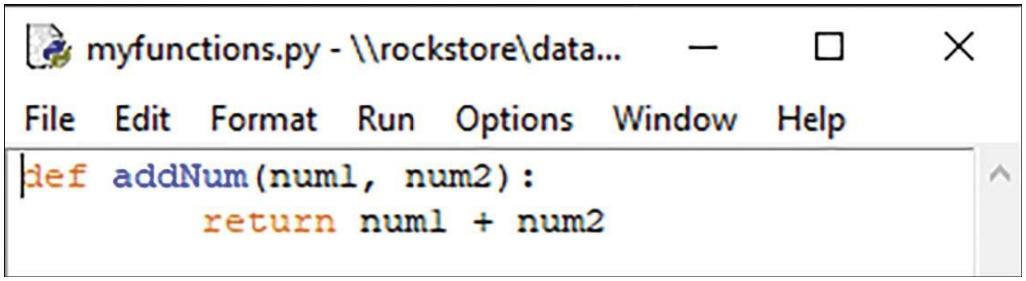
Чтобы использовать функцию из этого модуля

`turtle.functionname()` **например:** `turtle.forward(100)`

Создание собственных модулей

Вы можете объявить и сохранить свои функции в отдельном файле и импортировать их в основную программу.

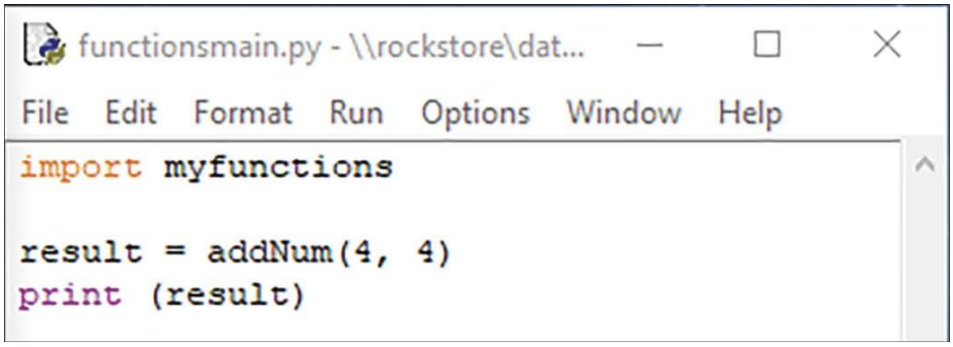
Все определения функций могут храниться в файле, например, myfunctions.py



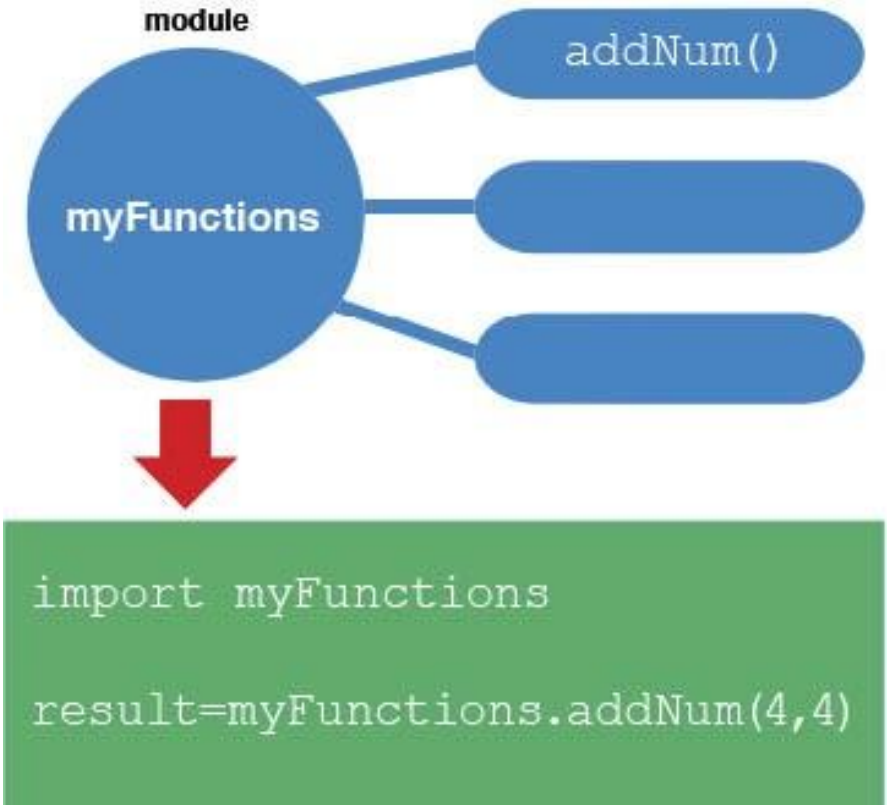
Здесь мы получаем два файла: functionmain.py и myfunctions.py

Основную программу можно назвать functionmain.py. Все функции этой программы хранятся в файле myfunctions.py

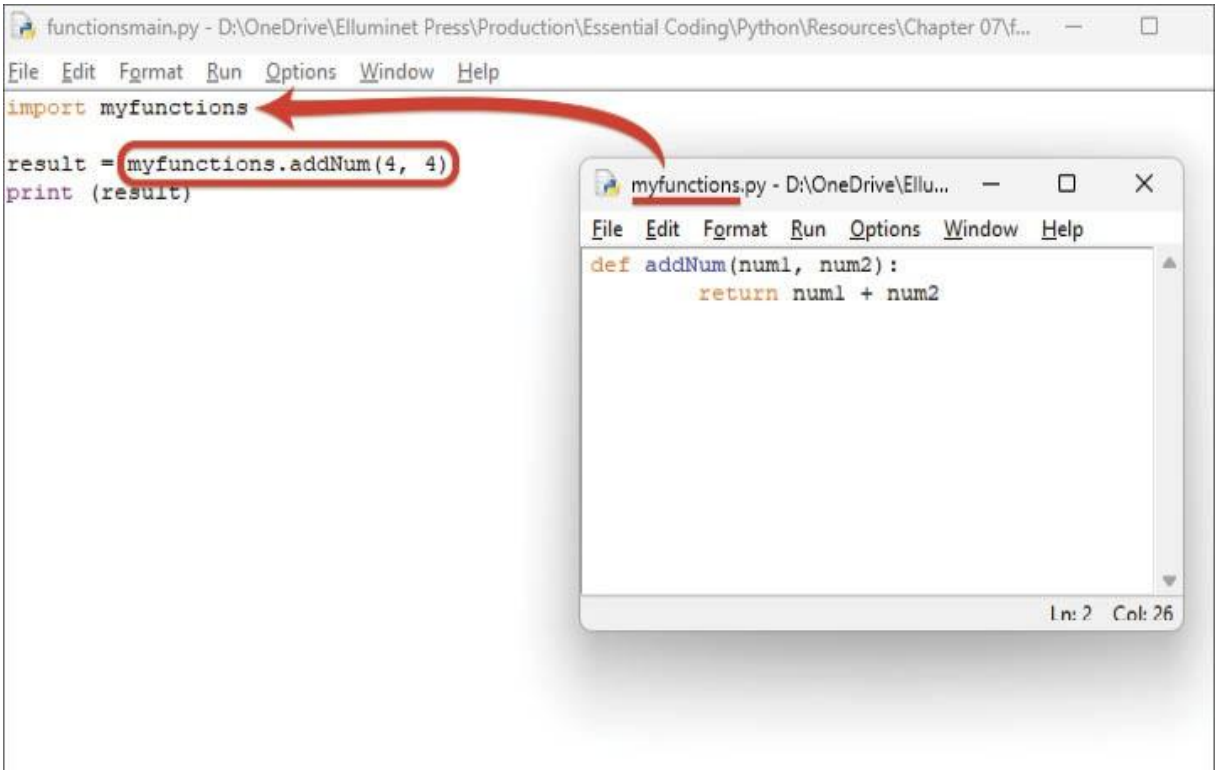
В начале основной программы нам будет нужно импортировать функции, хранящиеся в другом файле (myfunctions.py). Удалите расширение файла (.py).



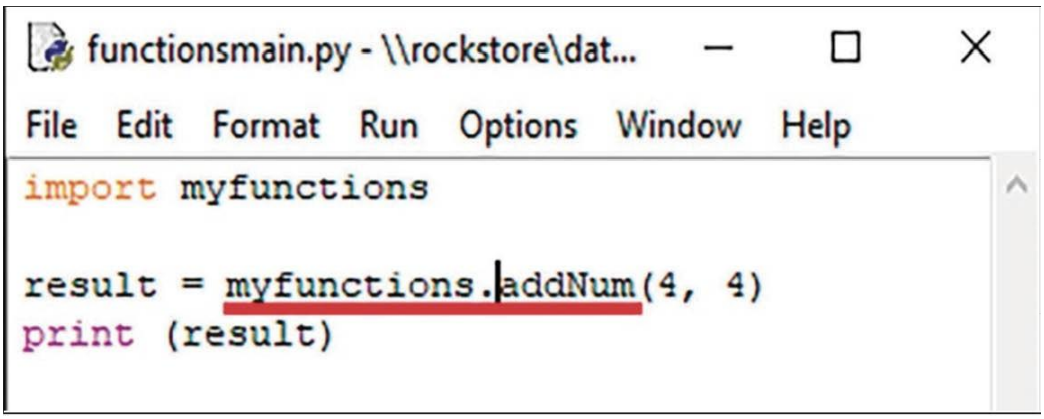
Это вызывает модуль. Любые объявленные функции будут включены в основную программу.



Вы можете включить эти функции в любую необходимую вам программу. Это упрощает ее обслуживание.



Теперь, чтобы вызвать любую функцию из этого модуля, вам нужно указать имя модуля, а затем имя функции.



Лабораторная работа

1. Напишите функцию, которая принимает число от пользователя и использует функцию для возведения числа в квадрат, а затем возвращает результат.
2. Сохраните этот файл как модуль
3. Импортируйте только что созданный модуль в новую программу.
4. Вызовите функцию в модуле

Обработка исключений

Исключением является ошибка, возникающая во время выполнения программы, иногда называемая ошибкой при выполнении. Это может быть ошибка «файл не найден», если вы пытаетесь загрузить несуществующий файл, или «ошибка типа», если вы вводите текст в поле, когда программа ожидает ввод числа.

Исключения полезны для обработки ошибок, возникающих при обработке файлов, доступе к сети и вводе данных.

Эти ошибки могут быть корректно обработаны с помощью процедур обработки исключений Python.

Вам также понадобятся исходные файлы из каталога Chapter 08.

Типы исключений

Вот список встроенных исключений согласно документации Python.

Исключение	Причина
AssertionError	Возникает при сбое оператора контроля ошибок.
AttributeError	Возникает при сбое назначения атрибута или ссылки.
EOFError	Возникает, когда функции input() достигают состояния конца файла.
FloatingPointError	Возникает в случае сбоя операции с плавающей запятой.
GeneratorExit	Возникает при вызове метода генератора close().
ImportError	Возникает, когда импортированный модуль не найден.
IndexError	Возникает, когда индекс последовательности выходит за пределы допустимого диапазона.
KeyError	Возникает, если в словаре не найден ключ.
Keyboardinterrupt	Возникает, когда пользователь нажимает клавишу прерывания (Ctrl+c или Delete)
MemoryError	Возникает, если для операции не хватает памяти.
NameError	Возникает, если переменная не найдена в локальной или глобальной области видимости.
NotImplementedError	Вызывается абстрактными методами
OSError	Возникает, если работа системы вызывает системную ошибку.
OverflowError	Возникает, если результат арифметической операции слишком велик для представления
ReferenceError	Возникает, если для доступа к референту, собирающему мусор, используется слабый ссылочный прокси.
RuntimeError	Возникает, если ошибка не подпадает под какую-либо другую категорию.
Stopiteration	Вызывается функцией next() чтобы указать, что итератор больше не может возвращать элементы.
SyntaxError	Вызывается синтаксическим анализатором при обнаружении синтаксической ошибки.
IndentationError	Возникает при неправильном отступе.
TabError	Возникает, если отступ состоит из несогласованных табуляции и пробелов.
SystemError	Возникает, если интерпретатор обнаруживает внутреннюю ошибку.
SystemExit	Вызывается функцией sys.exit()
TypeError	Возникает, если функция или операция применяется к объекту неправильного типа.
UnboundLocalError	Возникает, если делается ссылка на локальную переменную в функции или методе, но к этой переменной не привязано никакое значение.

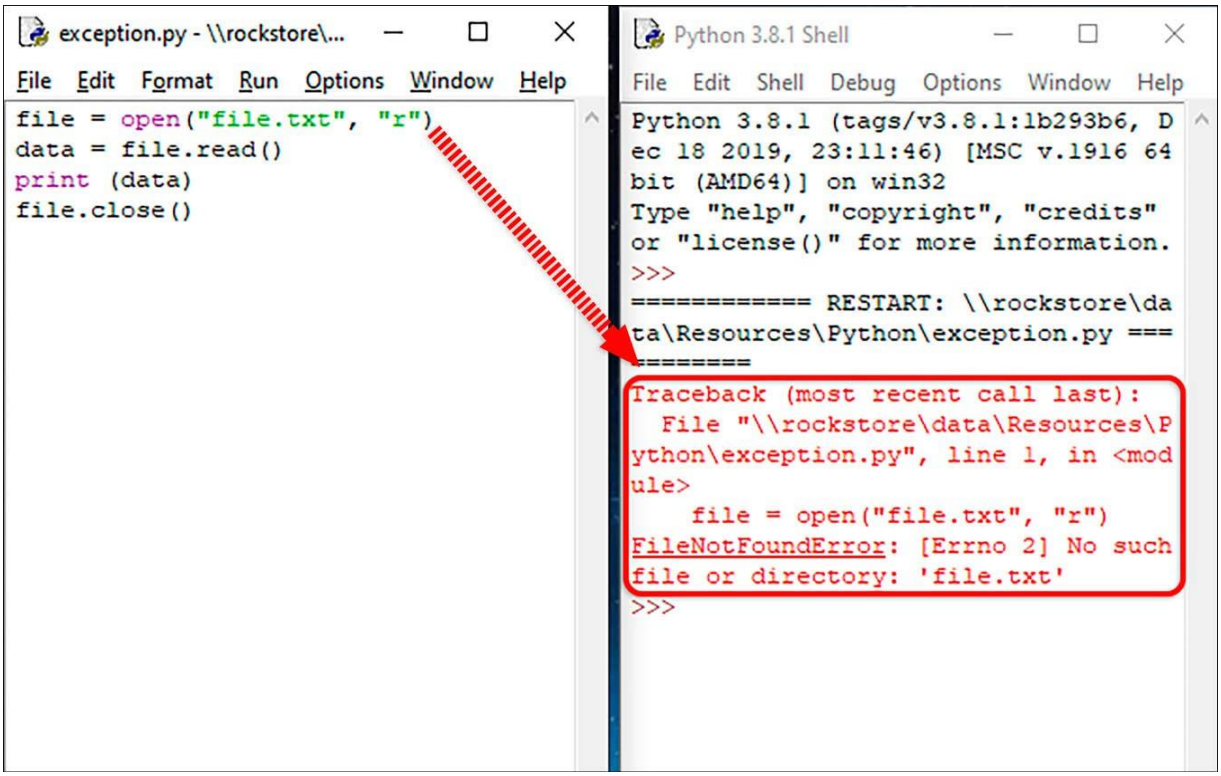
UnicodeError	Возникает при возникновении ошибки кодирования или декодирования, связанной с Unicode.
UnicodeEncodeError	Возникает, если во время кодирования возникает ошибка, связанная с Unicode.
UnicodeDecodeError	Возникает, если во время декодирования возникает ошибка, связанная с Unicode.
UnicodeTranslateError	Возникает, если во время трансляции возникает ошибка, связанная с Юникодом.
ValueError	Возникает, если функция получает аргумент правильного типа, но неправильного значения.
ZeroDivisionError	Возникает, если второй операнд деления или операции по модулю равен нулю.

Всякий раз, когда возникает исключение, интерпретатор останавливает выполнение программы и выдает ошибку исключения, как это показано в таблице выше.

Вы можете перехватить эти исключения, используя ключевые слова `try` и `except`, и предоставить код для обработки ошибки.

Перехват исключений

Если мы запустим следующий код, интерпретатор Python выдаст исключение `FileNotFoundError`, поскольку файла с именем ‘file.txt’ нет. Это приведет к сбою программы.



Вы можете перехватывать исключения, используя ключевые слова `try` и `except`. Просто поместите свой код в блок ‘try’ и код обработки ошибок для каждого исключения в блок ‘exception’, как это показано ниже.

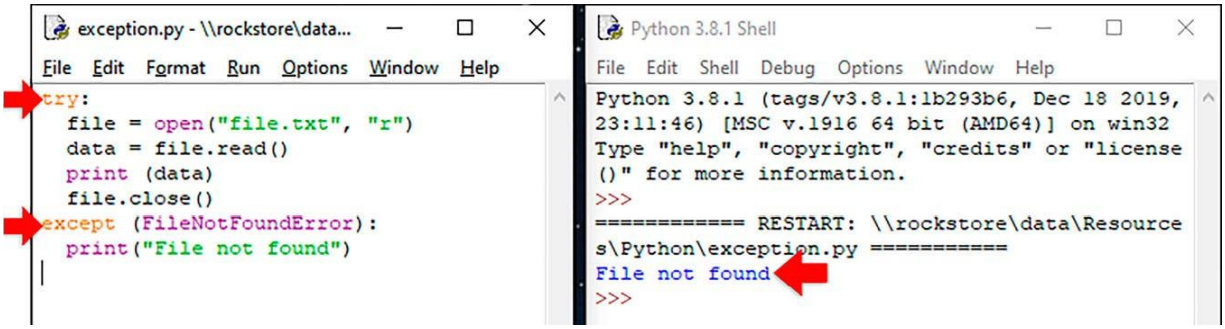
```
try:
# Code to execute as normal
except [exception (see table above)]:
# Code to deal with exception
```

Блок `try` содержит код для выполнения. Блок `except` содержит код для обработки ошибки.

Давайте еще раз посмотрим на программу. Мы можем взять наш код и поместить его в блок ‘try’. Затем добавим блок ‘except’ для обработки ошибок. Если мы посмотрим на сообщение об ошибке в оболочке, то

увидим, что это `FileNotFoundError`. Мы можем добавить это после ключевого слова ‘except’.

Теперь, когда мы запускаем программу, мы получаем простое сообщение, а не ужасную ошибку.



Используйте блок `finally` для выполнения очистки. Оператор `finally` выполняется независимо от того, создает ли оператор `try` исключение или нет.

```
try:
file = open(“file.txt”, “r”)
data = file.read()
except (FileNotFoundError):
print(“File not found”)
```

finally:

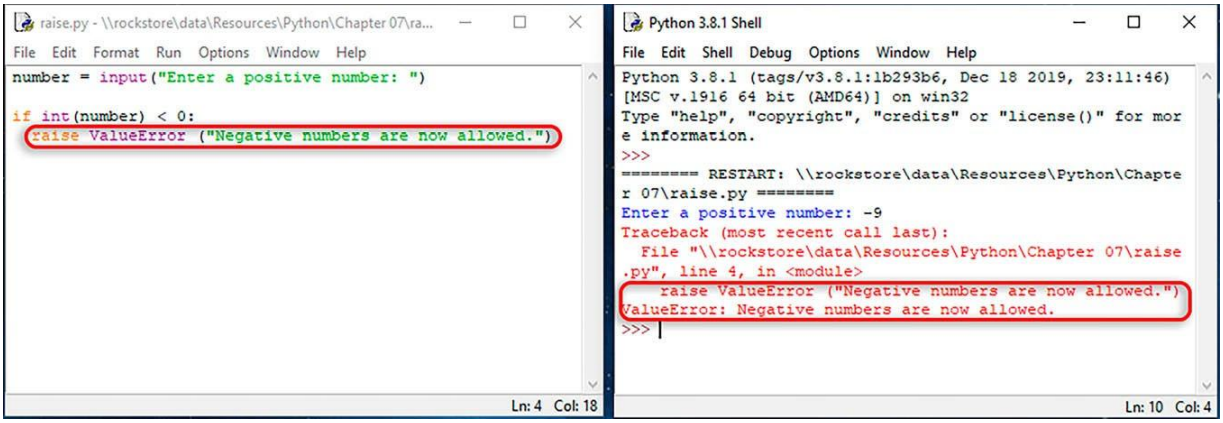
f.close()

Генерирование собственных исключений

Используйте ключевое слово `raise`, чтобы вызвать появление указанного исключения, а затем укажите тип ошибки, используя таблицу на стр. 103

```
if number < 0:
    raise ValueError("Negative numbers only.")
```

Здесь, в файле raise.py, мы вызвали исключение ValueError.



Объектно-ориентированное программирование

Python — объектно-ориентированный язык программирования. Это означает, что структура программы основана на объектах, а не на функциях и логике.

Каждый объект создается с использованием схемы, известной как класс. Каждый класс имеет атрибуты, содержащие данные, с которыми объект должен работать.

Каждый класс также содержит функции, называемые методами, которые выполняют операции с объектами.

Объект — это экземпляр класса. Итак, вы можете иметь класс под названием ‘car’ и использовать его для определения объекта под названием ‘merc’.

Для этого раздела посмотрите демонстрационные видео.

`elluminetpress.com/pyoop`

Вам понадобятся исходные файлы из каталога Chapter 09.

Класс

Класс — это определяемая пользователем схема или шаблон, который определяет атрибуты (переменные) и методы (функции) и содержит их все в одном модуле, называемом классом.

```
class ClassName :
```

Например, мы могли бы создать класс под названием Person.

```
class Person :
```

Объект

Объект — это экземпляр класса. Итак, из объявленного выше класса ‘person’ мы можем использовать его для создания объектов с наименованиями ‘staff’, ‘student’, ‘lecturer’ и т.д.

```
objectName = className(parameters)
```

Например

```
student = Person(...)
```

Атрибут

Атрибут аналогичен переменной и используется для временного хранения данных. Например, имя, дата рождения и адрес электронной почты.

```
class Person :
def   init  (self, name, dob, email):
self.name=name
self.dob  =dob
self.email=email
```

Атрибуты определяются внутри специального метода, называемого конструктором.

```
def  init (self, name, dob, email):
```

Этот метод вызывается автоматически при создании объекта. Все параметры, передаваемые объекту, заключаются в круглые скобки.

‘self’ — это ссылка на текущий объект и используется для доступа к атрибутам, принадлежащим этому объекту.

Метод

Метод — это функция, которая используется объектом для выполнения действия. На методы обычно ссылаются, указывая имя объекта через точку и имя метода. Например, `getAge()`

```
class Person :
...
def getAge(self):
currentDate = date.today()
age = currentDate.year - self.dob.year
returnage
```

Принципы ООП

Четыре принципа ООП: инкапсуляция, наследование, полиморфизм и абстракция.

Инкапсуляция

С помощью инкапсуляции вы ограничиваете доступ к методам и атрибутам внутри определенного класса. Это предотвращает случайное изменение данных и нежелательные изменения других объектов.

Наследование

Класс может наследовать все методы и атрибуты другого класса. Если бы класс под названием ‘person’ имел имя, возраст и дату рождения. Мы могли бы определить два других дочерних класса, называемых ‘student’ и ‘staff’. Оба могут наследовать методы и атрибуты класса ‘person’.

Полиморфизм

Полиморфизм позволяет нам определять методы в дочернем классе с тем же именем, что и в родительском классе. Это известно как переопределение метода. Полиморфизм также позволяет нам определять методы, которые могут принимать разные формы.

Абстракция

Абстракция — это процесс сведения объектов к их сути так, чтобы были представлены только необходимые элементы. Другими словами, вы удаляете всю ненужную информацию об объекте, чтобы уменьшить его сложность.

Классы и объекты

Как мы видели во введении, вы можете определить класс, используя ключевое слово `class`. Давайте рассмотрим подробнее.

```
class <class-name> :
<class attributes and methods>
```

Здесь мы создали класс под названием Person.

```
class Person :
```

Все классы обладают функцией `init ()`, которая автоматически выполняется при запуске класса. Используйте функцию `init ()` для инициализации атрибутов.

```
def  init (self, name, dob, email):
self.name = name
self.dob = dob
self.email = email
```

Помните, что ключевое слово `self` представляет текущий экземпляр класса (т. е. объект, созданный на основе класса). Используя ключевое слово `self`, вы можете получить доступ к атрибутам самого объекта.

Когда вы объявляете метод, вы передаете ему текущий экземпляр класса (т.е. сам объект) вместе с любыми другими необходимыми параметрами.

```
def getAge(self):
currentDate = date.today()
age = currentDate.year - self.dob.year
return age
```

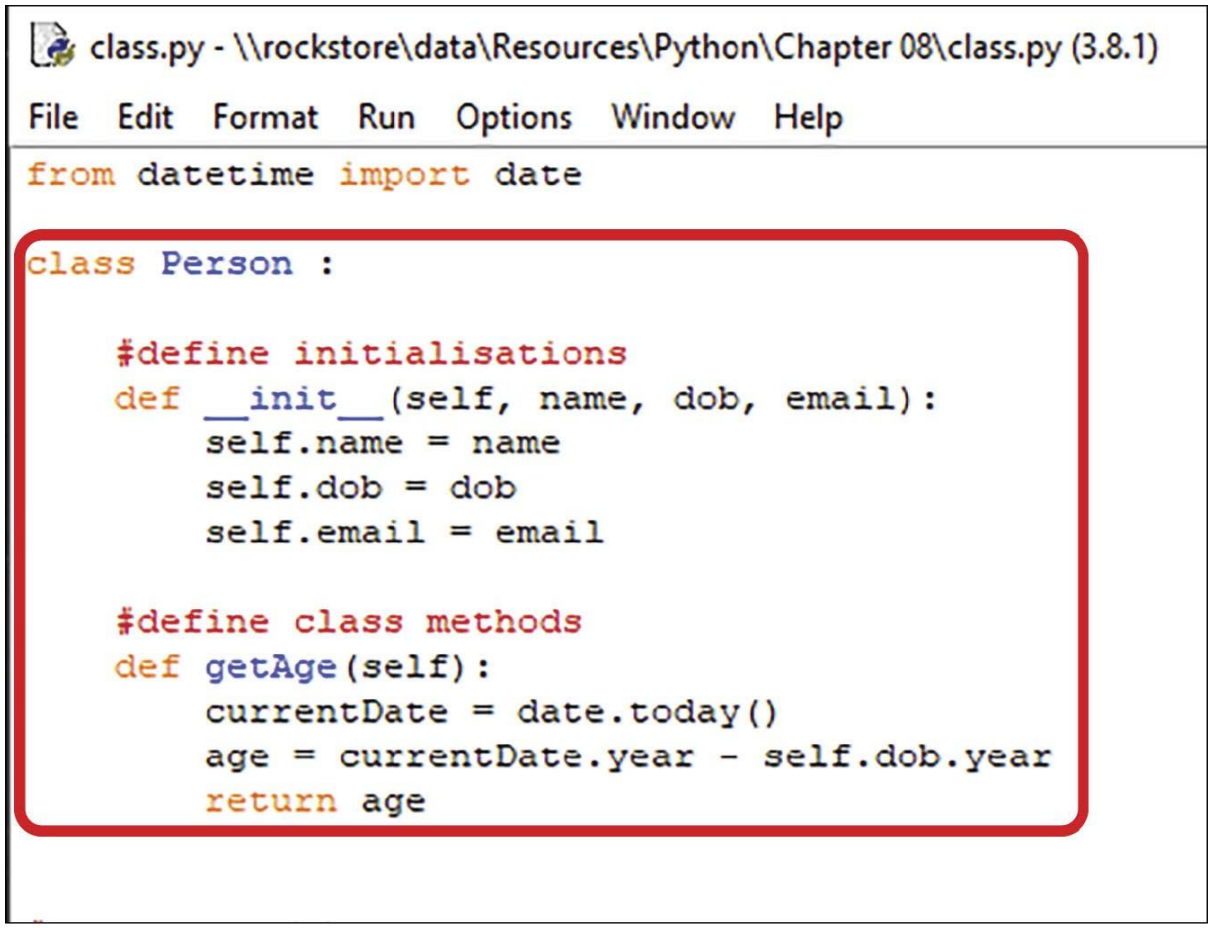
Когда вам нужно использовать какой-либо атрибут, вы используете `self`, за которым следует имя атрибута.

```
self.attribute-name
```

Так например:

```
self.email
```

Давайте рассмотрим программу. Откройте файл class.py. Здесь мы определили наш класс ‘Person’.



Чтобы создать объект из класса, вызовите класс Person(...) и передайте любые данные, используя круглые скобки ( ). Присвойте новый объект переменной, например: person.

```
#create an object
person = Person (
         "Sophie", #name
         date(1999, 4, 2), #DOB (year, month, day)
         "Sophie@mymail.com", #email
)
```

Чтобы использовать объект, используйте точечную запись.

```
classname.method()
```

или

```
classname.attribute
```

Итак, в нашем примере, чтобы использовать наши атрибуты:

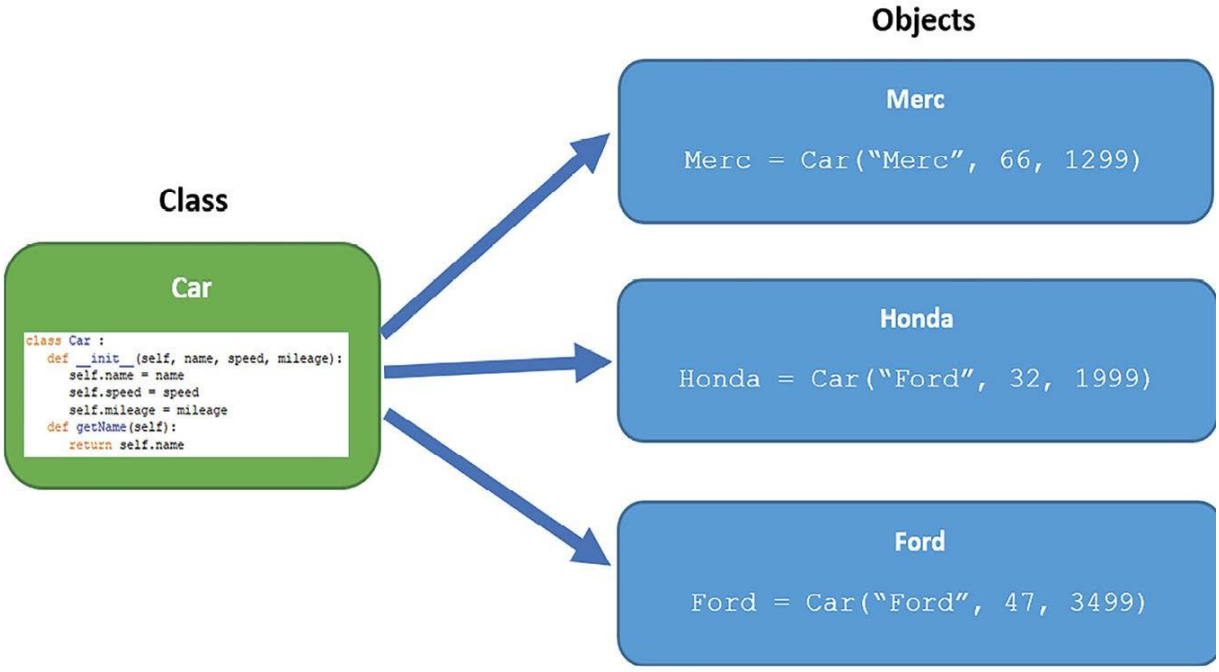
```
print(person.name)
print(person.email)
```

...и использовать наши методы:

```
print(person.getAge())
```

Давайте рассмотрим другой пример. Вы можете использовать класс Car для создания некоторых объектов, таких как Merc, Honda и Ford.

```
Merc = Car("Merc", 66, 1299)
```



При создании объекта интерпретатор Python автоматически вызывает метод `____init ()`. Этот метод называется конструктором и используется для инициализации объектов, созданных с помощью класса.

```
def  init (self, name, speed, mileage):
```

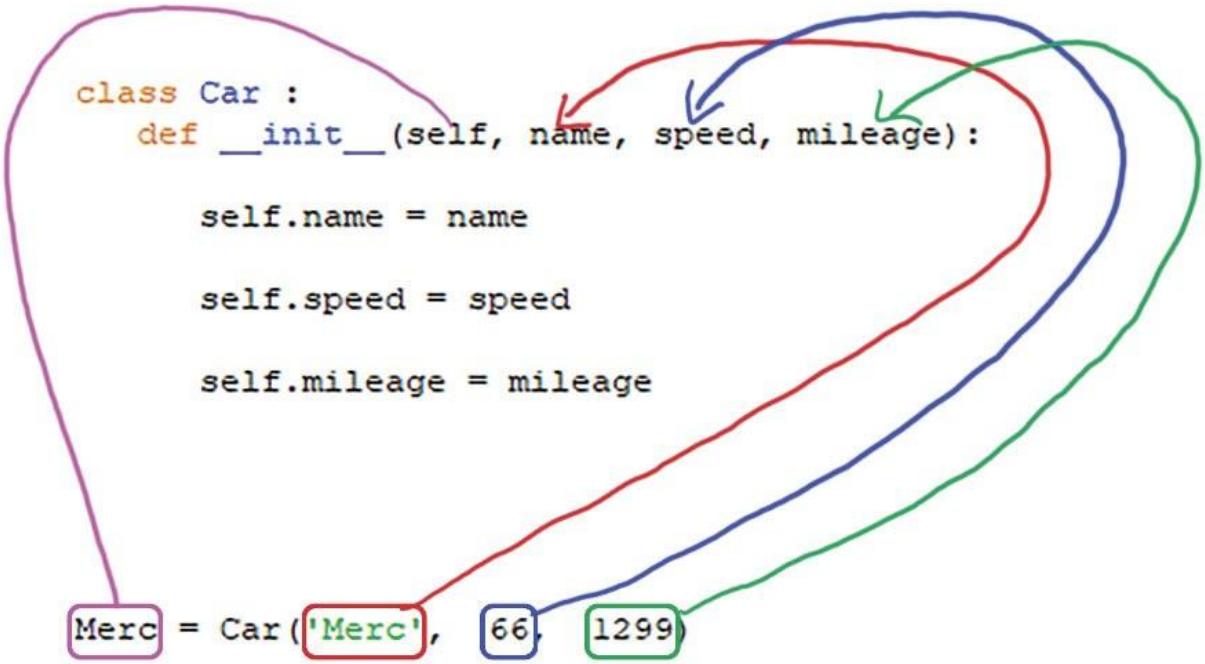
Внутри метода `init ()` мы инициализируем атрибуты.

```
self.name = name
self.speed = speed
self.mileage = mileage
```

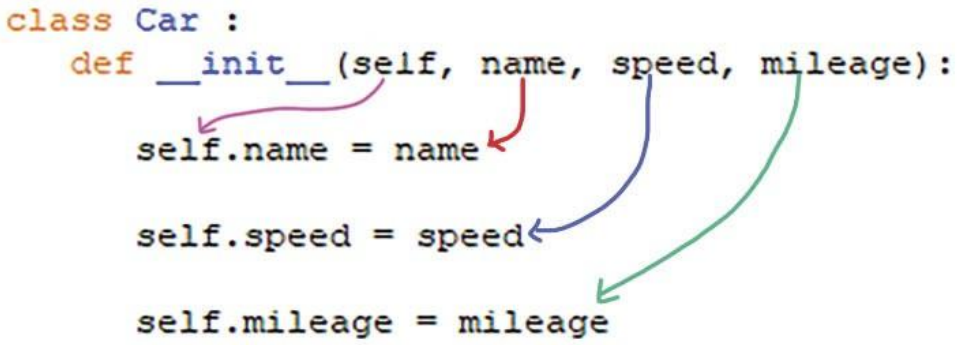
Давайте посмотрим, что происходит. В методе `init()`, self ссылается

на экземпляр самого объекта и автоматически передается.

Здесь, внизу диаграммы, представленной ниже, когда мы создаем объект (Merc) из класса Car, мы передаем некоторые данные: name (наимнование), speed (скорость), mileage (пробег).



Далее мы присваиваем атрибутам объекта атрибуты, переданные в метод `init()`.



Мы можем использовать объект класса для выполнения действий. Например:

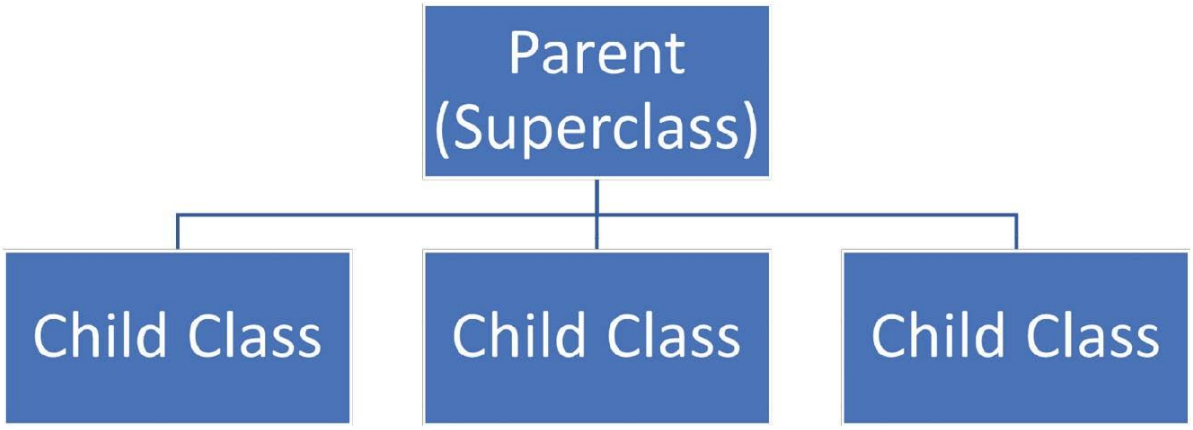
```
Merc.getName()
```

или

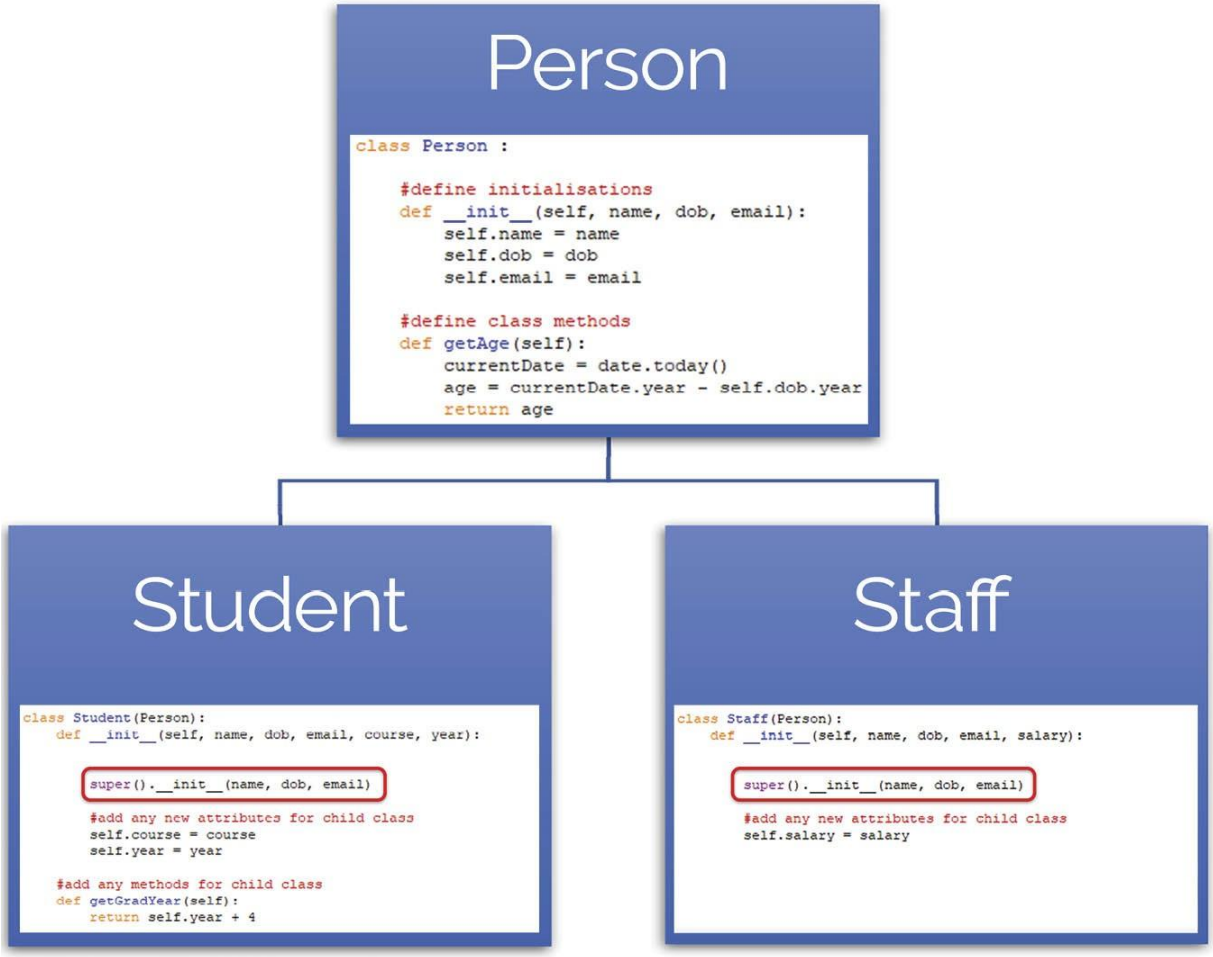
```
Merc.speed = 66
```

Наследование

Класс может наследовать все методы и атрибуты другого класса.



Если бы класс под названием ‘person’ имел name (имя), age (возраст) и dob (дату рождения). Мы могли бы определить два других дочерних класса, называемых ‘student’ и ‘staff’. Они оба могут наследовать методы и атрибуты класса ‘person’.



При определении дочерних классов вам может потребоваться доступ к методам, определенным в родительском классе. Для этого используйте функцию `super()`. Функция `super()` позволяет нам явно ссылаться на родительский класс.

Чтобы создать дочерний класс, объявите его как обычно, за исключением включения родительского класса в круглые скобки после имени класса. Так что:

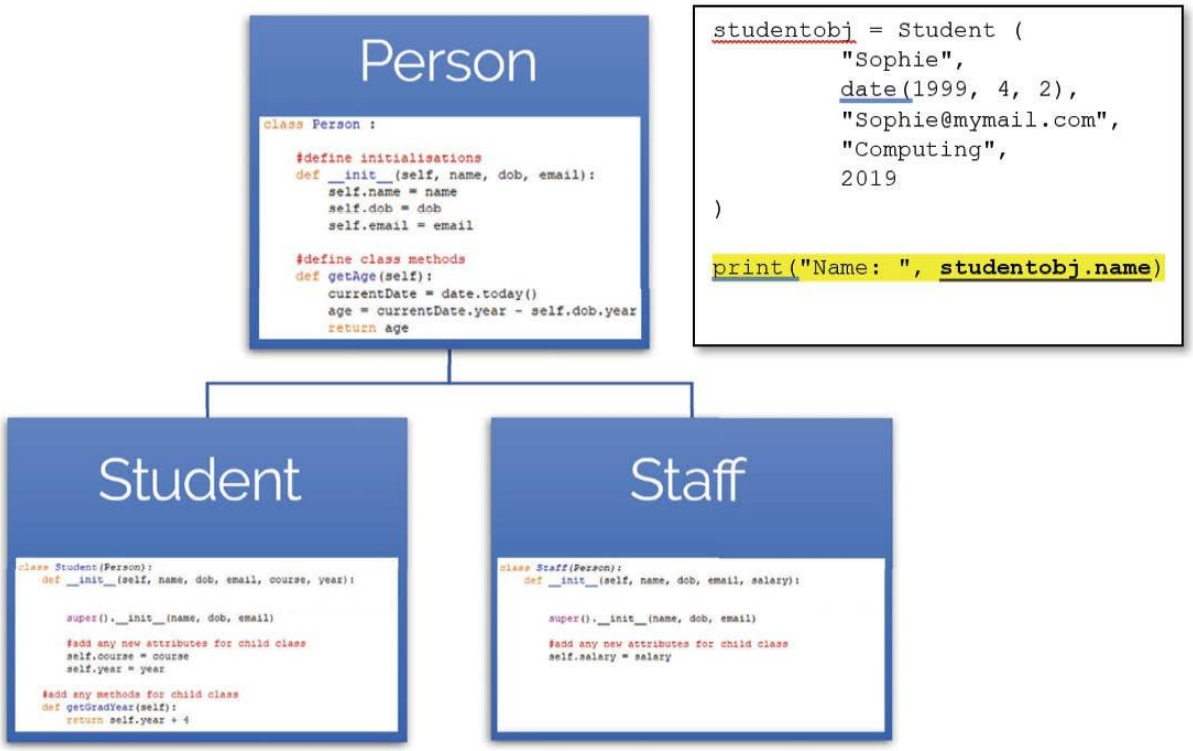
```
class child-class(parent-class):
```

Например, если вам нужен доступ к атрибутам родительского класса, вам нужно будет вызвать родительский конструктор `init()`, используя `super()` в дочернем классе.

```
super().  init  (name, dob, email)
```

В приведенном ниже примере в операторе печати мы вызываем атрибут “name”, который определен в родительском классе Person, из объекта

(studentobj), созданного с использованием класса Student.

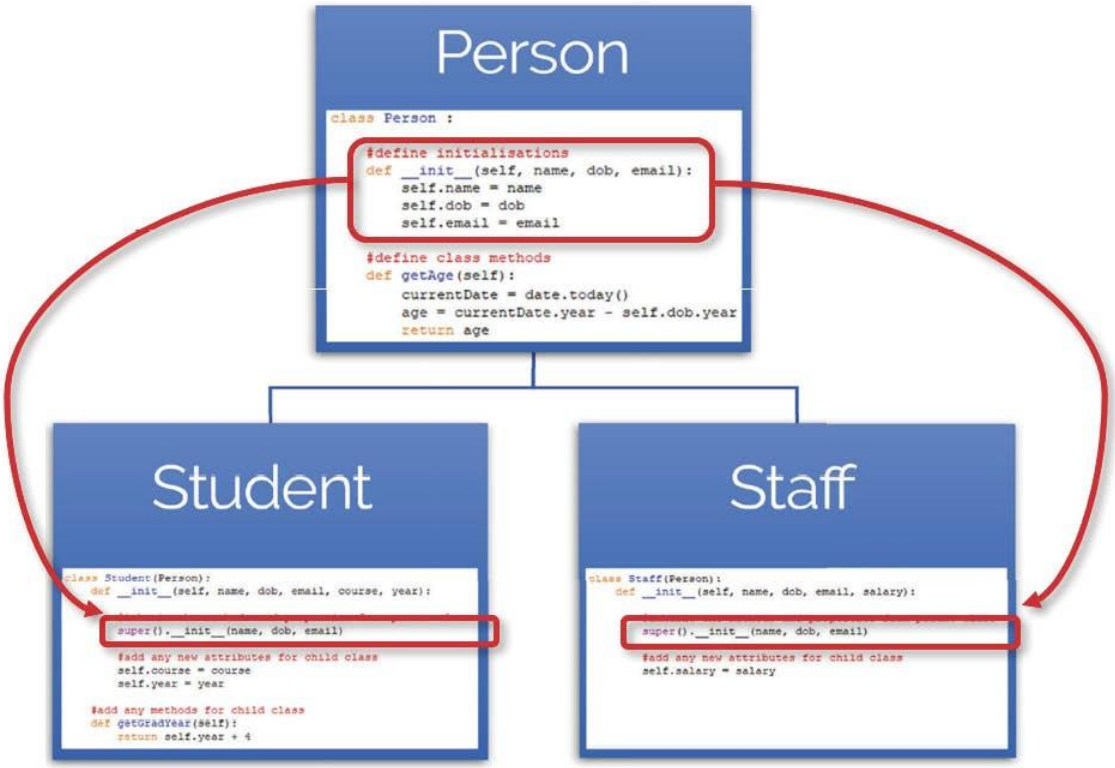


Чтобы получить к этому доступ, нам нужно использовать `super()` для вызова метода конструктора `init()`, определенного в родительском классе (Person).

```
class Student(Person):
    def __init__(self, name, dob, email, course, year):


        super().__init__(name, dob, email)

        #add any new attributes for child class
        self.course = course
        self.year = year

    #add any methods for child class
    def getGradYear(self):
        return self.year + 4
```

```
class Staff(Person):
    def __init__(self, name, dob, email, salary):


        super().__init__(name, dob, email)

        #add any new attributes for child class
        self.salary = salary
```

Здесь в двух дочерних классах (student и staff) мы вызвали функцию `super()` в методе `init()` дочернего класса.



Функция `super()` вызывает метод `init()` родительского класса, который предоставляет дочернему классу доступ ко всем атрибутам родительского класса. Если нам не нужно получать доступ к методам, определенным в родительском классе, из объекта, созданного из дочернего класса, то нам не нужна функция `super()`.

Откройте файл inherit.py. Здесь мы создали класс под названием person. Мы также создали дочерние классы под названием student и staff.

inherit.py - \\rockstore\data\Resources\Python\Chapter 08\inherit.py (3.8.1)

File Edit Format Run Options Window Help

```
class Person :

    #define initialisations
    def __init__(self, name, dob, email):
        self.name = name
        self.dob = dob
        self.email = email

    #define class methods
    def getAge(self):
        currentDate = date.today()
        age = currentDate.year - self.dob.year
        return age
```

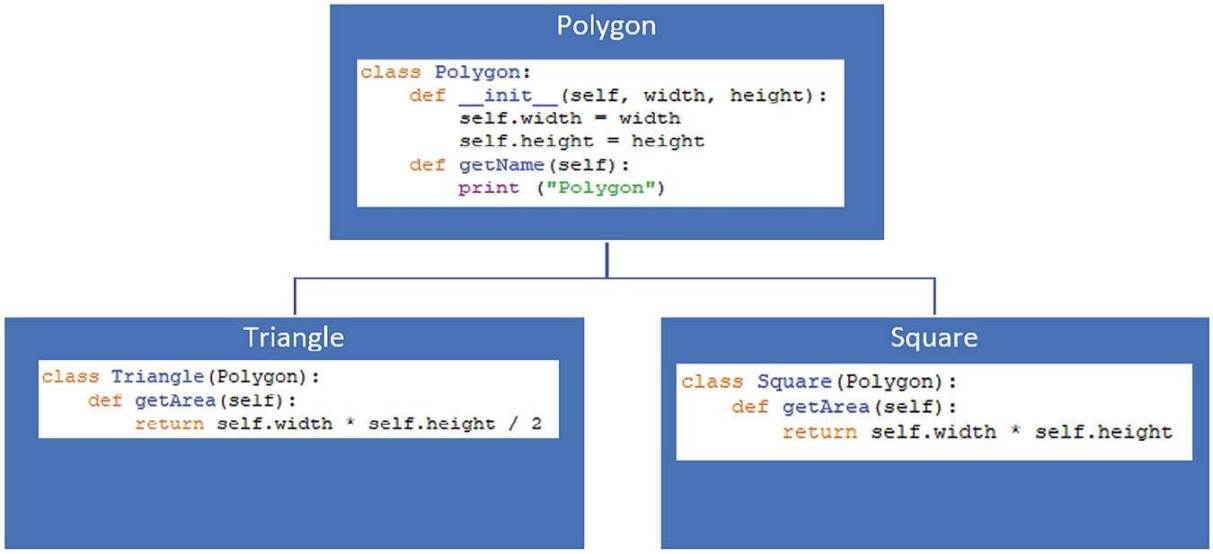
Мы можем создать объект lecturer из класса Staff

```
#create an object
lecturer = Staff (
        "John", #name
        date(1977, 4, 2), #DOB (year, month, day)
        "John@mymail.com", #email
        44000
)
```

Для ссылки на атрибуты объекта мы используем точечную запись.

```
print ("\nStaff Member: ", lecturer.name)
print ("Salary: ", "£", lecturer.salary)
```

Давайте посмотрим на другой пример: здесь мы определили родительский класс под названием Polygon и создали дочерний класс под названием Triangle.



Когда мы создаем объект из дочернего класса Triangle…

```
triobj = Triangle(3,4)
```

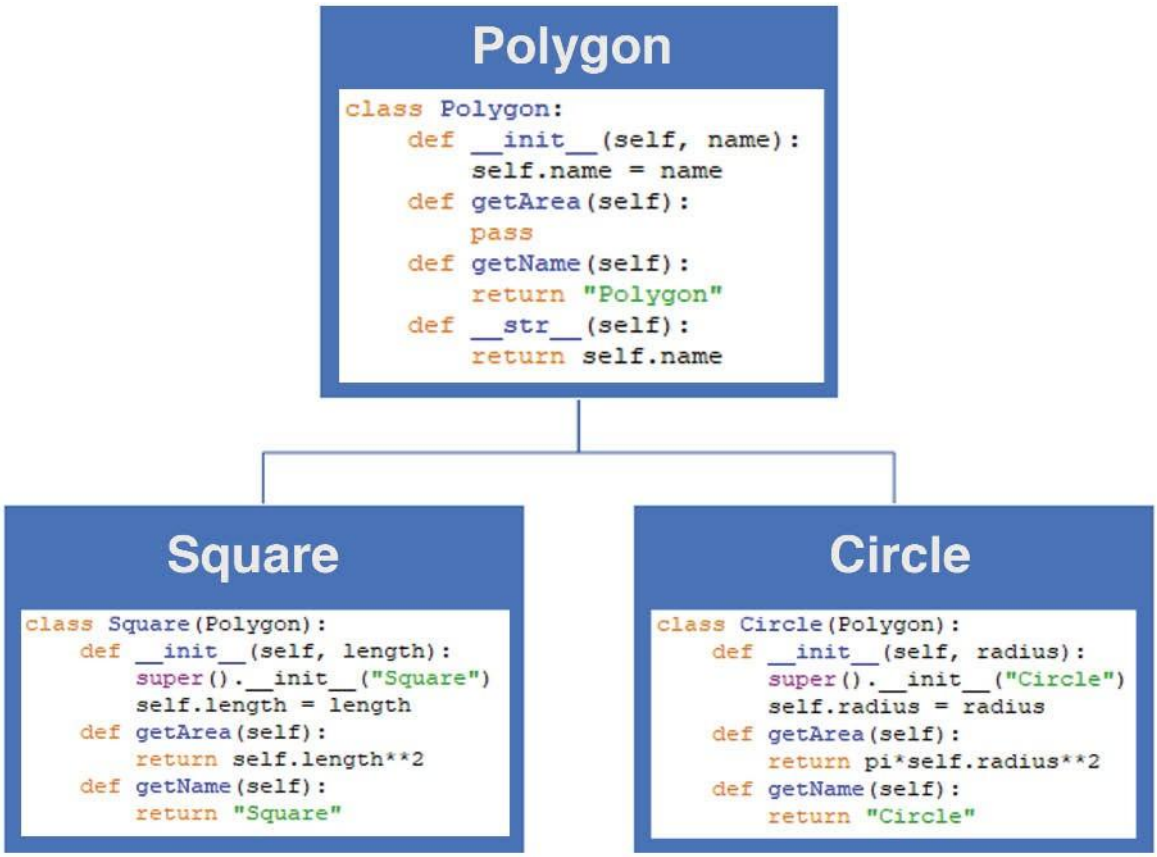
…мы не вызываем никаких методов, определенных в родительском классе (Polygon). Здесь мы вызываем метод `getArea()` из объекта треугольника (triobj), определенного в классе Triangle.

```
print(triobj.getArea())
```

Полиморфизм

Полиморфизм в сочетании с наследованием позволяет нам определять методы в дочернем классе с тем же именем, что и в родительском классе. Это действие известно как переопределение метода.

В этом примере мы собираемся создать три класса. Чтобы быть полиморфными, каждый из этих классов должен иметь общий интерфейс. Поэтому мы определяем методы для каждого класса с одинаковым именем. В этом случае мы определили метод, который вычисляет площадь в каждом классе (Square и Circle).



Откройте файл inhertit.py.

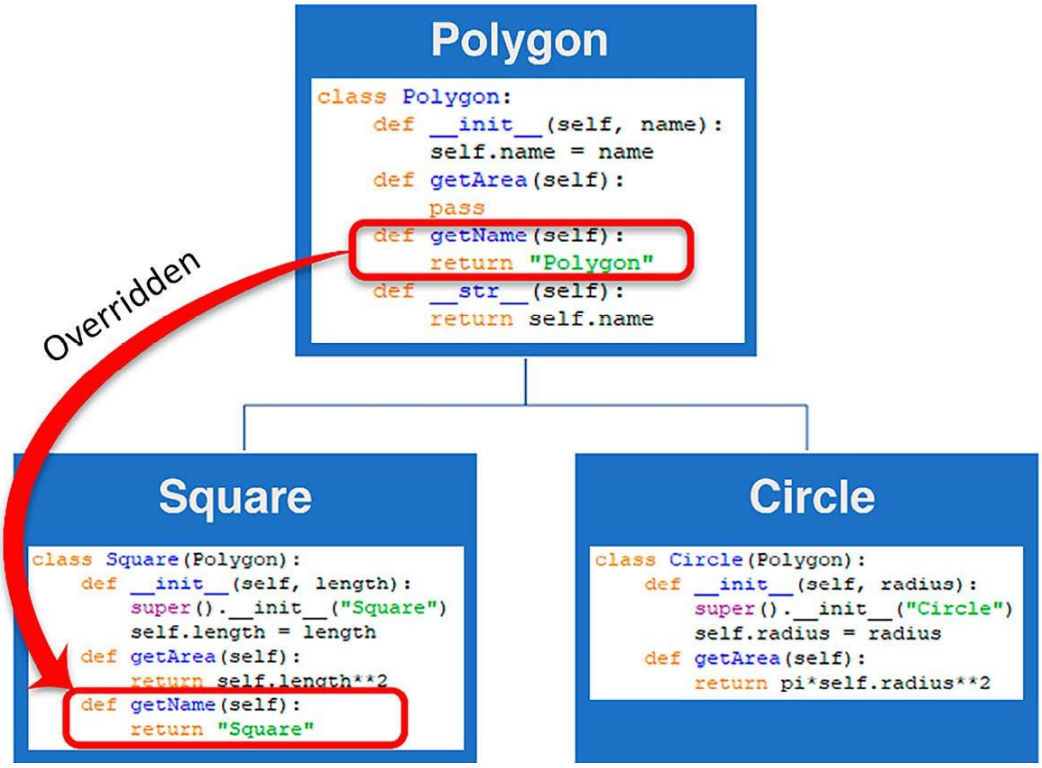
Полиморфизм позволяет нам получить доступ к этим переопределенным методам и атрибутам, имеющим то же имя, что и определенное в родительском классе. Например, если мы создадим объект с именем squareObj из класса Square:

```
square0bj = Square(7)
```

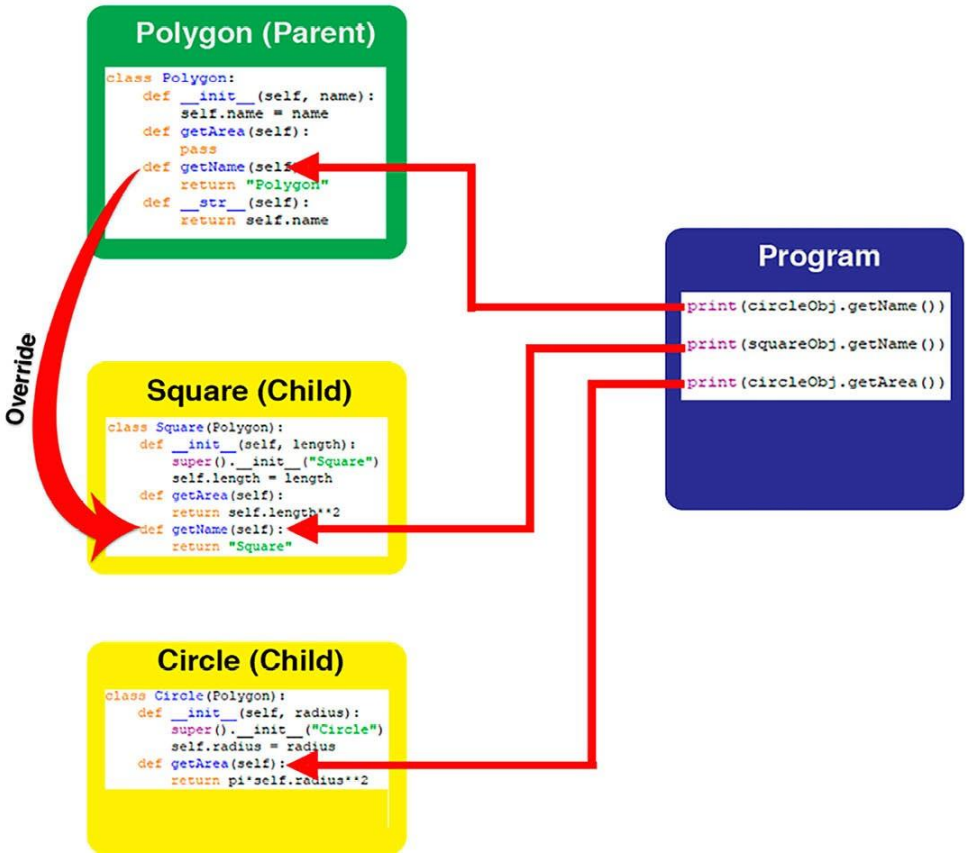
Затем вызовем метод `getName()`.

```
print(square0bj.getName())
```

Интерпретатор автоматически распознает, что метод `getName()` для squareObj переопределен.



Поэтому используется метод `getName()`, определенный в дочернем классе (Square). Метод `getName()`, определенный в классе Square, переопределяет метод `getName()`, определенный в родительском классе (Polygon). Аналогично и с `getArea()`.



Вы также заметите, что мы вызвали метод `getName()` из объекта circleObj класса Circle.

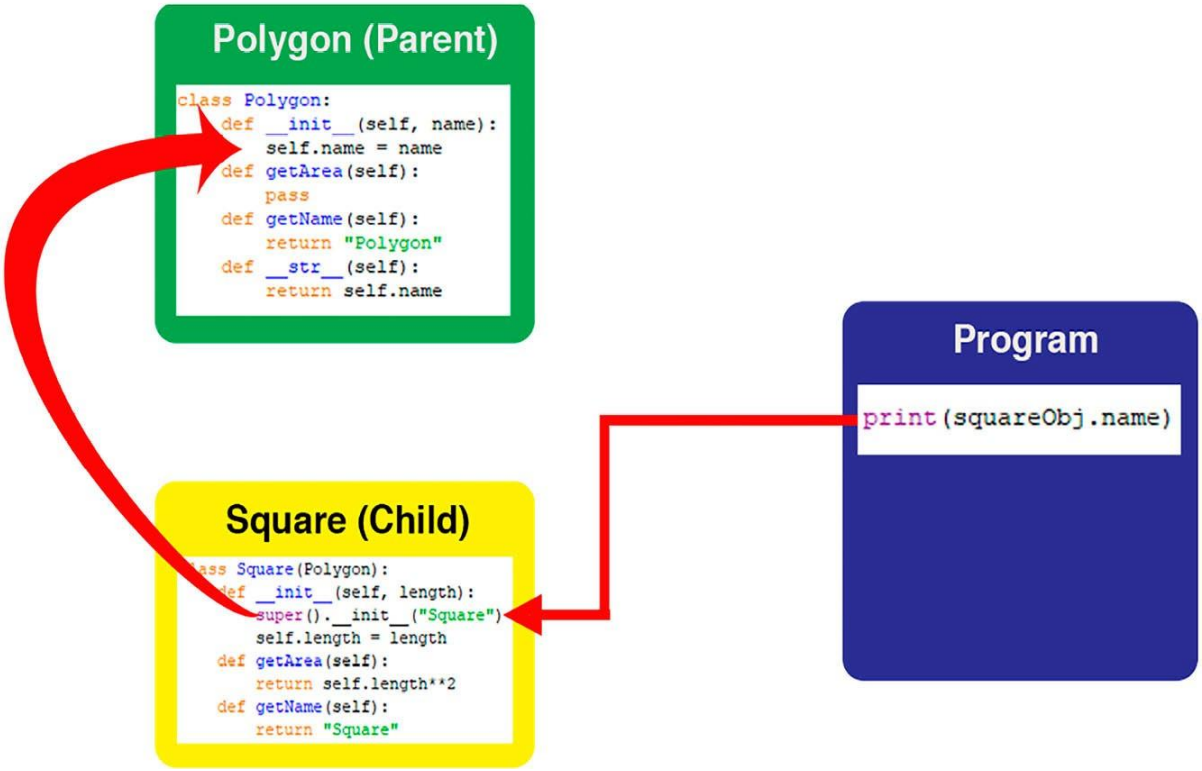
```
print (circle0bj.getName()
```

В этом случае в классе Circle не определен метод `getName()`, поэтому интерпретатор вызовет метод `getName()` из родительского класса.

Если мы ссылаемся на атрибут “name” в squareObj

```
print(square0bj.name)
```

Вы заметите, что “name” определено в родительском классе (Polygon), а не в дочернем классе (Square).



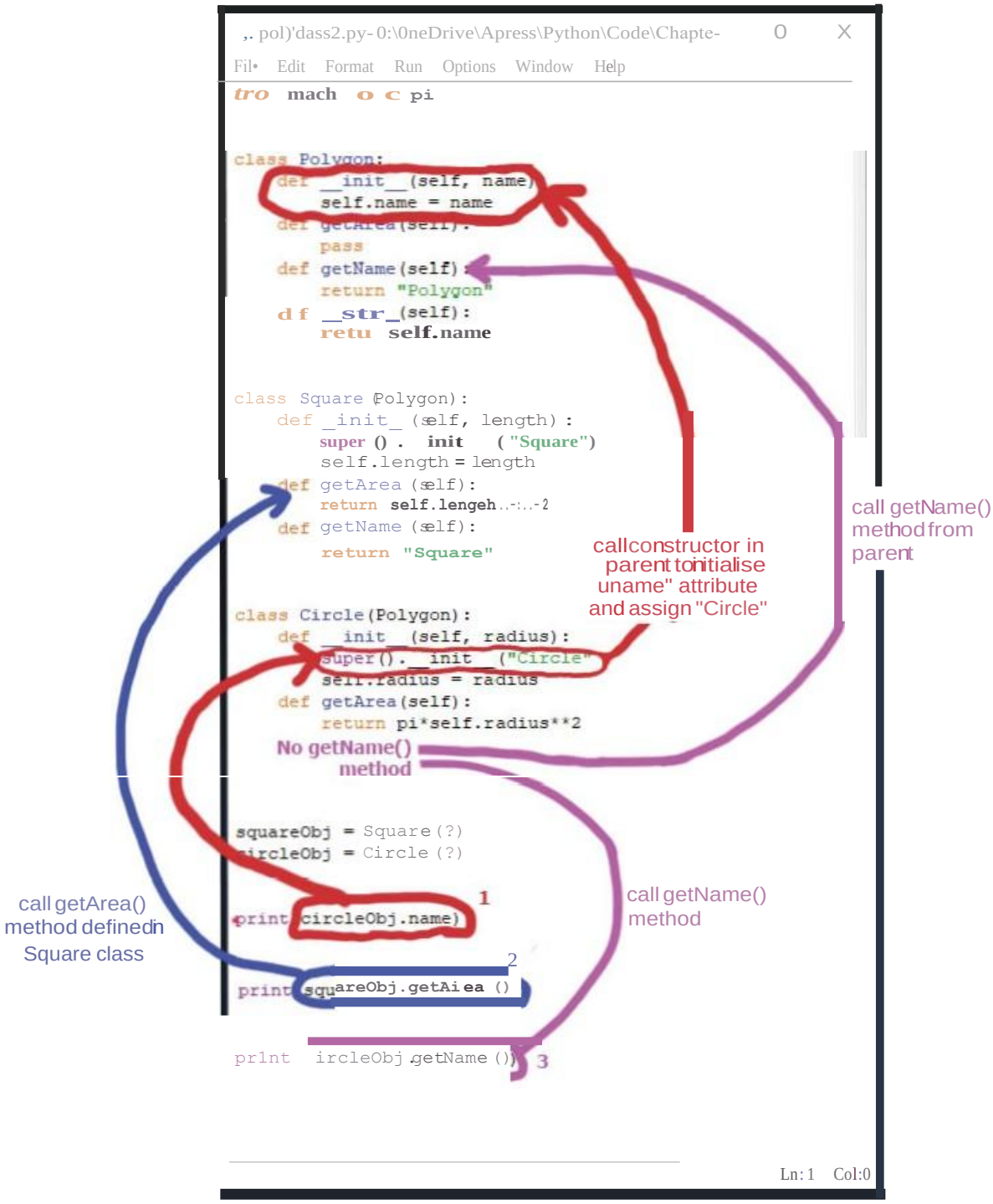
Чтобы получить доступ к этому атрибуту из дочернего элемента, нам нужна функция `super()` в дочернем классе для вызова метода конструктора `init()`, определенного в родительском классе (Polygon).

```
super().init("Square")
```

это вызывает следующий метод из класса Polygon

```
def  init  (self, name):
```

Взгляните на polyclass2.py


,. pol)'dass2.py- 0:\0neDrive\Apress\Python\Code\Chapte-
Fil• Edit Format Run Options Window Help
tro mach o c pi
class Polygon:
def __init__(self, name):
self.name = name
def getArea(self):
pass
def getName(self):
return "Polygon"
d f __str_(self):
retu self.name
class Square (Polygon):
def _init_ (self, length):
super () . init ("Square")
self.length = length
def getArea (self):
return self.lengeh..-:..-2
def getName (self):
return "Square"
class Circle(Polygon):
def __init__(self, radius):
super().__init__("Circle")
self.radius = radius
def getArea(self):
return pi*self.radius**2
squareObj = Square(?)
circleObj = Circle(?)
print circleObj.name)
print squareObj.getAiea ()
pr1nt ircleObj.getName ())
1
2
3
Ln: 1 Col:0
callconstructor in parent toinitialise uname" attribute and assign "Circle"
call getName() method from parent
No getName() method
call getArea() method defined in Square class
call getName() method

Лабораторная работа

1. Объявите новый класс Vehicle без каких-либо атрибутов и методов
   Добавьте некоторые атрибуты в класс Vehicle, например
   1. Name
   2. Speed
   3. Mileage
4. Добавьте в класс Vehicle метод, возвращающий модель автомобиля
5. Создайте дочерний класс под названием Taxi
6. Добавьте в класс Taxi метод для сбора платы за проезд. Стоимость проезда составляет 20 центов за милю.
7. Что такое класс? Покажите пример.
8. Что такое объект? Покажите пример.
9. Что такое конструктор? Покажите пример.
10. Что такое метод? Покажите пример.
11. Для чего используется `init()`? Покажите пример.
12. Для чего используется `super()`? Покажите пример.

Turtle-графика

Turtle-графика во многом похожа на чертежную доску, на которой вы можете с помощью различных команд перемещать черепаху. Мы можем использовать такие функции, как forward() и right(). Черепаха будет перемещаться по тому пути, который вы задаете с помощью этих функций, при этом оставляя за собой след.

Для этого раздела посмотрите демонстрационные видео.

`elluminetpress.com/pygraphics`

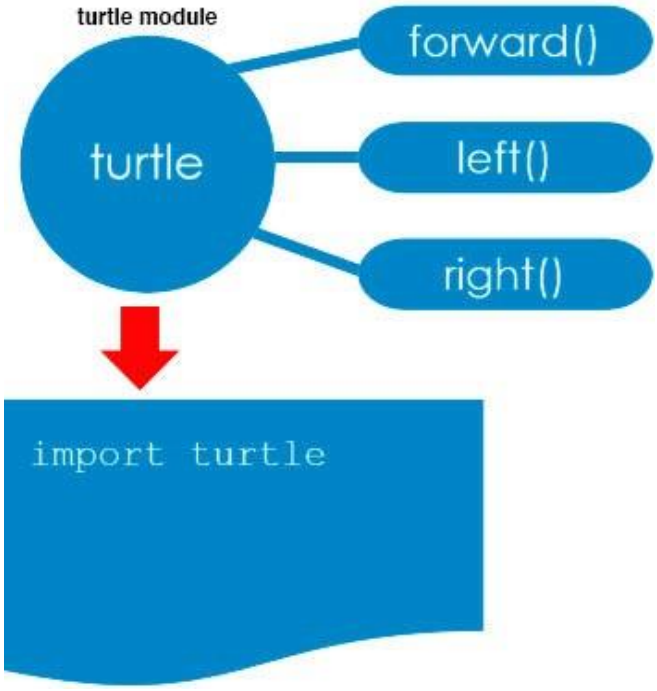
Вам также понадобятся исходные файлы из каталога Chapter 10.

Импорт графического модуля Turtle

Чтобы начать использовать turtle-графику, нам сначала нужно импортировать модули

```
import turtle
```

Этот оператор импортирует в программу все графические функции turtle.



Команды Turtle

Мы можем перемещать черепаху, используя различные команды. Доступ к ним можно получить, используя функции модуля.

```
moduleName.functionName
()
```

Например
:

```
turtle.forward(distance)

turtle.right(degrees)
```

‘turtle’ — это наименование модуля черепахи, который мы импортировали ранее, а `forward()` — это функция, определенная в графическом модуле черепахи. Аргумент — это то, насколько далеко вы хотите, чтобы черепаха переместилась, а degrees — это угол в градусах, на которы вы хотите, чтобы черепаха повернулась (например, на 90 градусов вправо).

Давайте используем это в программе. Ниже у нас есть оператор импорта для импорта всех графических модулей turtle. Под ним у нас есть оператор, который перемещает черепаху вперед, а затем оператор, завершающий программу.

Обратите внимание на то, что мы используем точечную запись для доступа к функциям модуля черепахи.

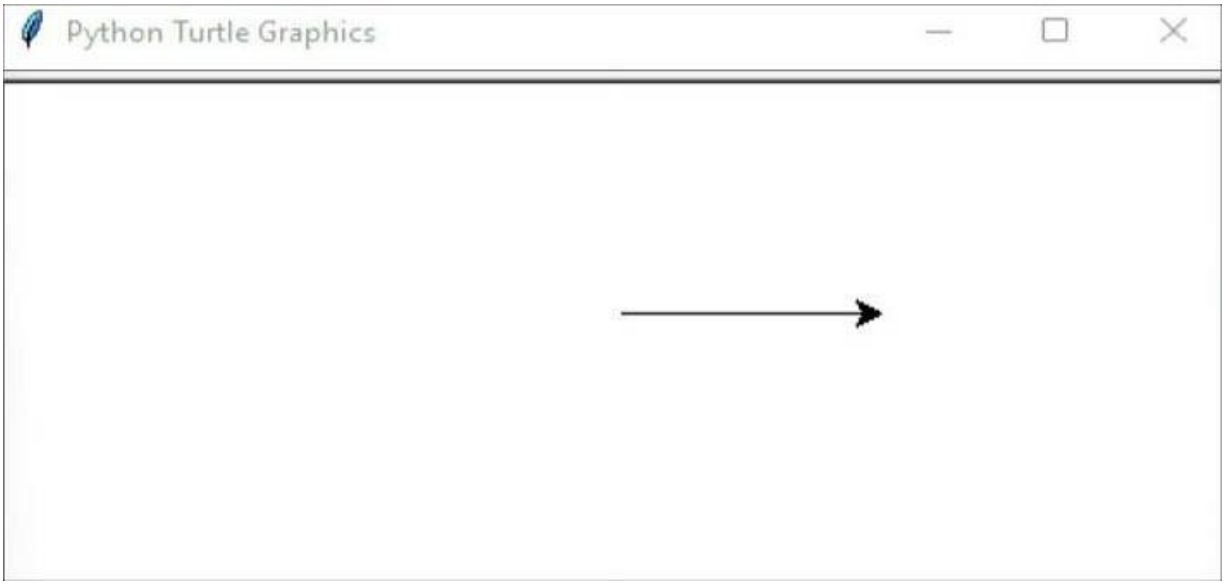
```
moduleName.functionName
()
```

Мы хотим получить доступ к функции пересылки, поэтому указываем имя модуля, в котором она находится (turtle), затем точку, а затем имя нужной функции (forward). Итак, мы получаем

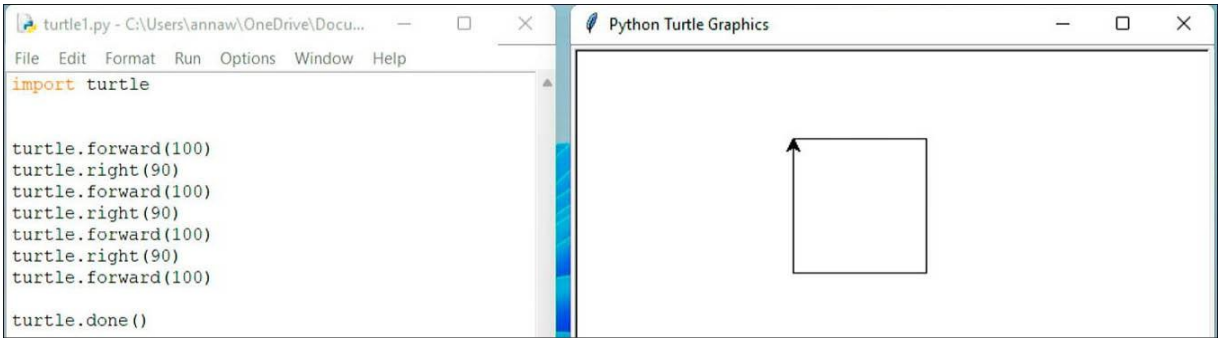
```
turtle.forward(10
0)
```

Это переместит черепаху на 100 пикселей вперед.

Здесь мы видим результат выполнения программы: черепаха переместилась на 100 пикселей вправо.



Мы можем написать программу для рисования квадрата.



Вы также можете использовать команду для перемещения назад

```
turtle.backward(distanc
e)
```

Вы можете повернуть черепаху налево или направо.

```
turtle.left(degrees)
```

```
turtle.right(degrees)
```

Вы также можете использовать другие команды, чтобы опускать и поднимать перо. Когда вы используете `pendown()`, черепаха будет рисовать. Если вы выполните `penup()`, черепаха не будет рисовать линию при движении.

```
turtle.penup()
turtle.pendown()
```

Вы также можете сменить цвет пера

```
turtle.color('color name')
```

Вы можете переместить черепаху в заданные координаты на экране

```
turtle.goto(x,y)
```

Выполните .done() в конце вашей программы

```
turtle.done()
```

Настройка окна Turtle

Мы можем настроить окно turtle. Сначала мы создаем окно turtle.

```
window = turtle.Screen()
```

Изменяем цвет фона

```
window.bgcolor("light green")
```

Создаем объект turtle

```
myTurtle = turtle.Turtle()
```

Циклические команды

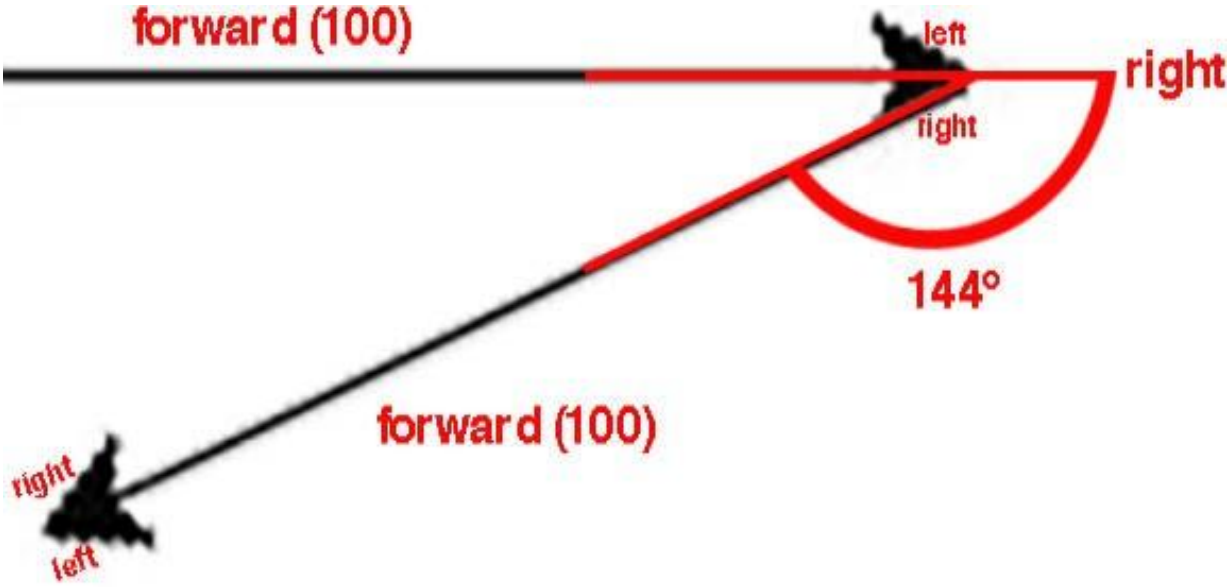
Вы можете использовать циклы для повторения команд рисования фигур. Например, если мы хотим нарисовать шестиугольник, мы можем использовать цикл for, чтобы повторить команды для каждой стороны.

```
sides = 6
angle = 360 / sides

for index in range(sides):

polygon.forward(200)

polygon.right(angle)
```

Если вы хотите нарисовать другие фигуры. Как насчет звезды?

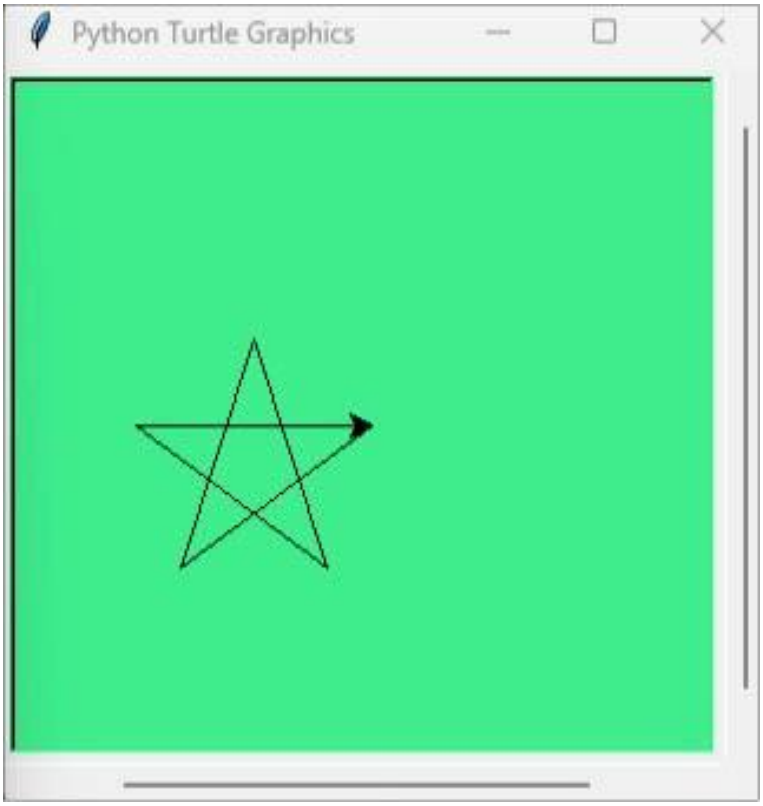
```
for counter in range(5):

myTurtle.right(144)

myTurtle.forward(100)
```

Здесь мы переместили черепаху на 100 пикселов вперед, а затем повернули ее на 144 градуса вправо.



Мы повторяем это перемещение 5 раз с помощью цикла for.

Когда вы выполняете программу...



Обратите внимание на то, что изменился цвет

фона. Попробуйте другие команды Turtle.

Команда	Описание
turtle.forward(distance)	Перемещает черепаху вперед на указанное расстояние
turtle.backward(distance)	Перемещает черепаху назад на указанное расстояние
turtle.home()	Перемещает черепаху в начало координат — координаты (0,0)
turtle.penup()	Поднимает перо
turtle.pendown()	Опускает перо
turtle.pencolor(color string)	Установите цвет пера в цветовую строку, например «red», «yellow» и т. д.
turtle.circle(radius)	Рисует круг заданного радиуса
turtle.shape("turtle")	Задает форму черепахи
turtle.undo()	Отменить (неоднократно) последние действия черепахи
turtle.clear()	Стирает все рисунки, которые в данный момент отображаются в графическом окне

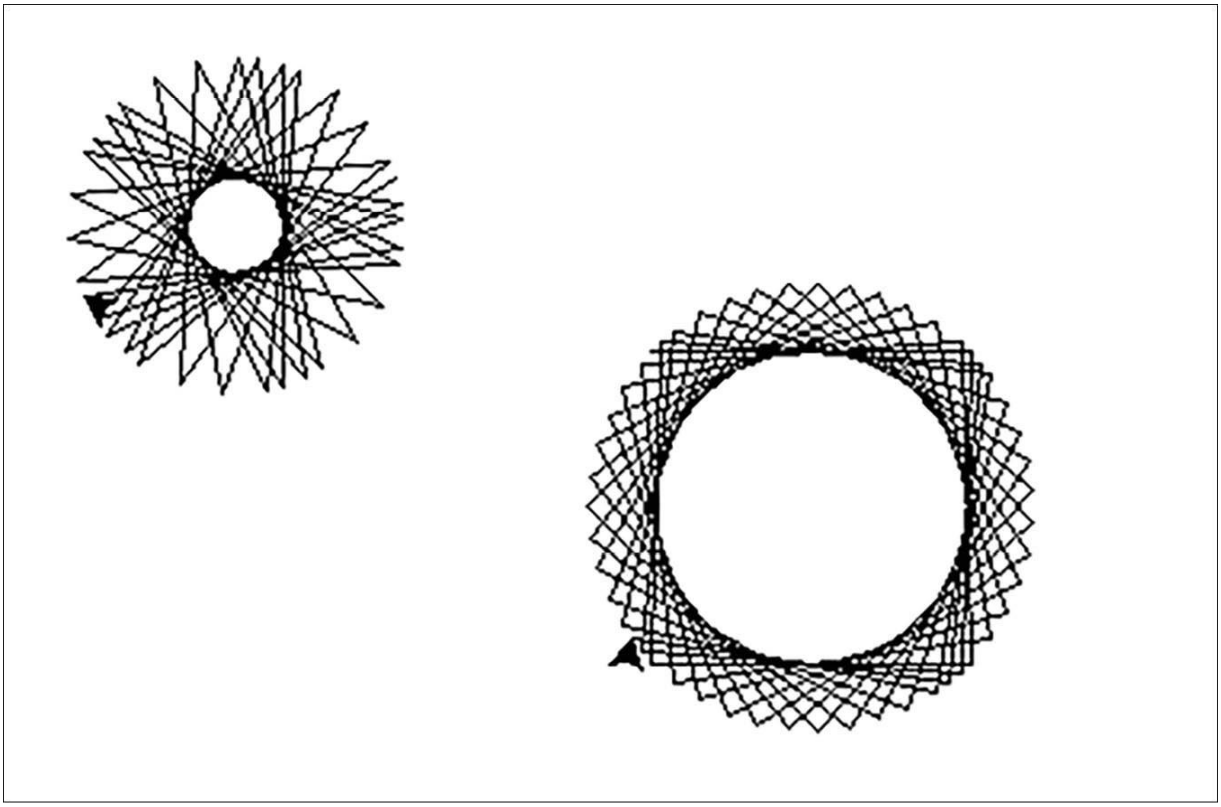
Посмотрите, какие узоры вы можете нарисовать. А как насчет спирали?

```
for i in range(50):
turtle.forward(100)
turtle.right(91 + 1)
```

Или, может быть, этот?

```
for counter in range(100):
turtle.right(147)
turtle.forward(100)
```

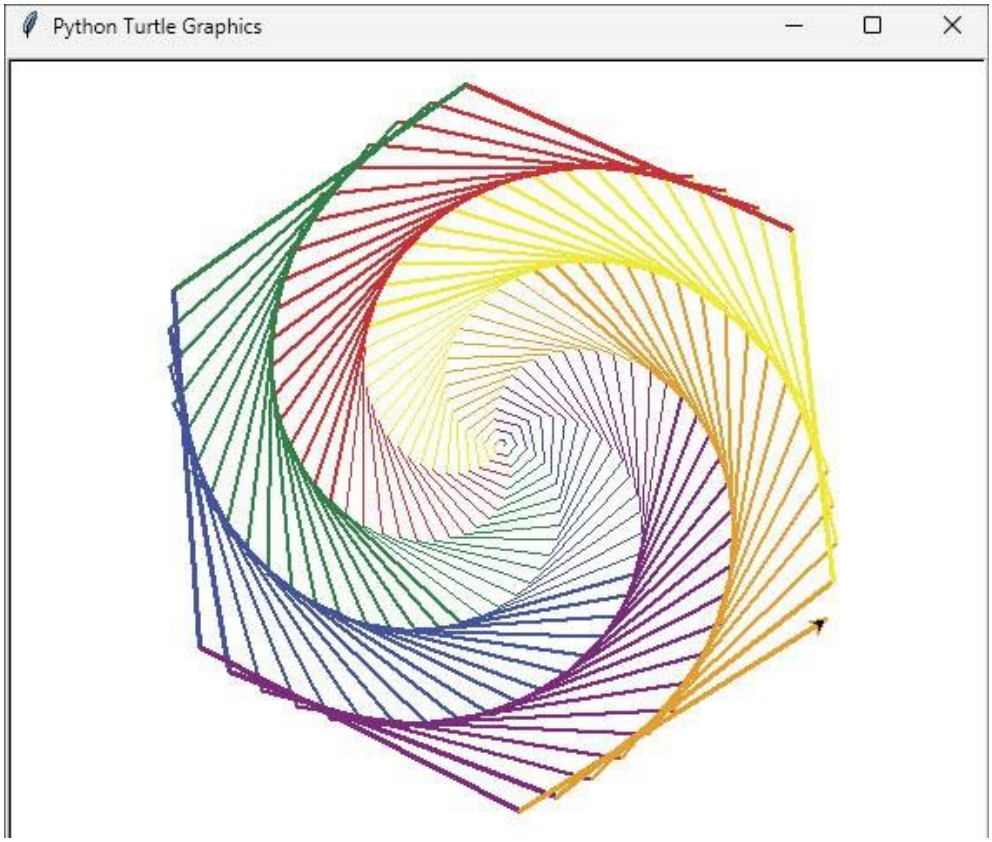
Попробуйте



Немногим более сложное. Попробуйте цветную спираль.

```
import turtle
colors=['red','green','blue','purple',
'orange','yellow']
myTurtle = turtle.Turtle()
myTurtle.speed(0)
for x in range(360):
myTurtle.pencolor(colors[x%6])
myTurtle.width(x//100+1)
myTurtle.forward(x)
myTurtle.left(59)
```

Это будет выглядеть примерно так


Python Turtle Graphics

Лабораторная работа

1. Создайте новую программу и импортируйте графический модуль turtle.
2. Поэкспериментируйте с рисованием различных фигур, используя некоторые методы графики turtle.
3. Используйте команды turtle, чтобы нарисовать несколько фигур.

Создание интерфейса

Современные компьютерные приложения создаются с учетом графических пользовательских интерфейсов. Пользователь взаимодействует с приложением, используя окна, значки, меню и указатель мыши, а не консольный ввод-вывод.

Чтобы создать графический интерфейс пользователя с помощью Python, вам понадобится Tkinter (Tk interface). Этот модуль входит в состав стандартных дистрибутивов Python для всех платформ.

Для этого раздела посмотрите демонстрационные видео

`elluminetpress.com/pygraphics`

Вам также понадобятся исходные файлы из каталога Chapter 11.

Создание окна

Первое, что вам нужно сделать, это импортировать модуль tkinter в вашу программу. Для этого используйте

```
from tkinter import *
```

Чтобы создать окно, используйте метод Tk()

```
window = Tk()
```

Добавьте заголовок

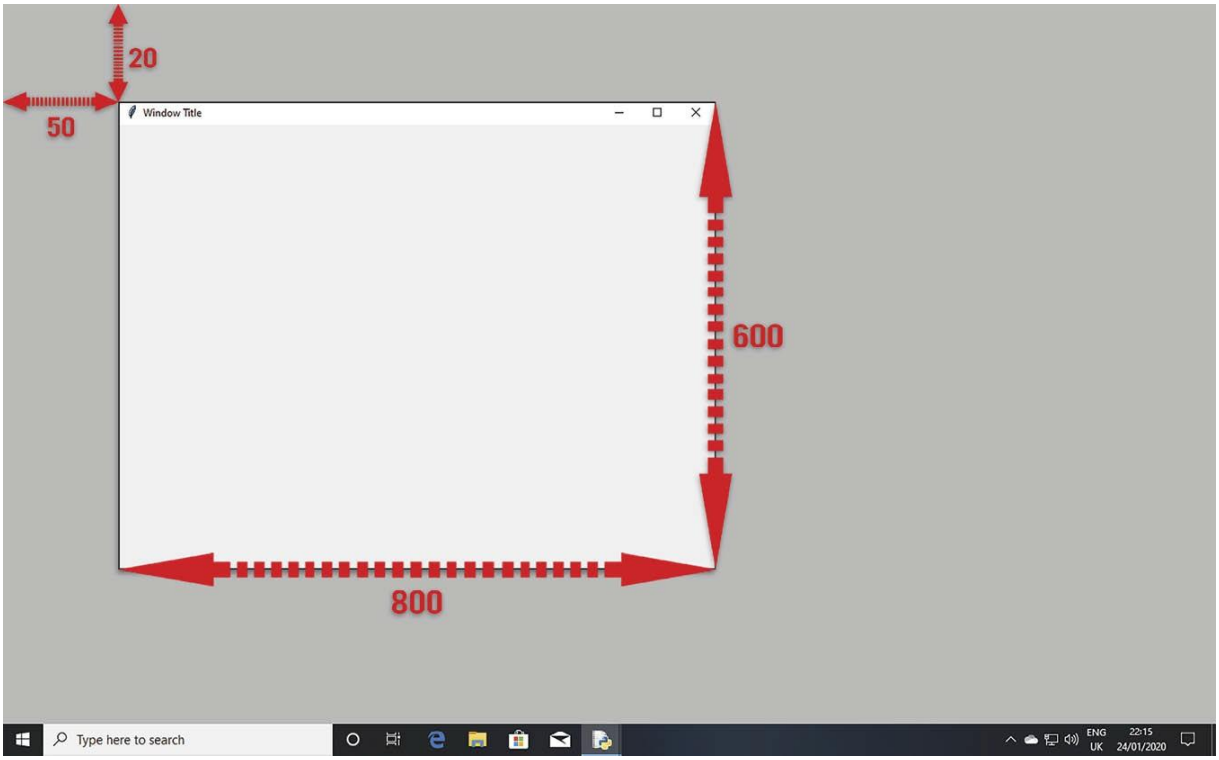
```
window.title('Window Title')
```

Установите начальный размер и положение окна. Используйте для этого метод .geometry().

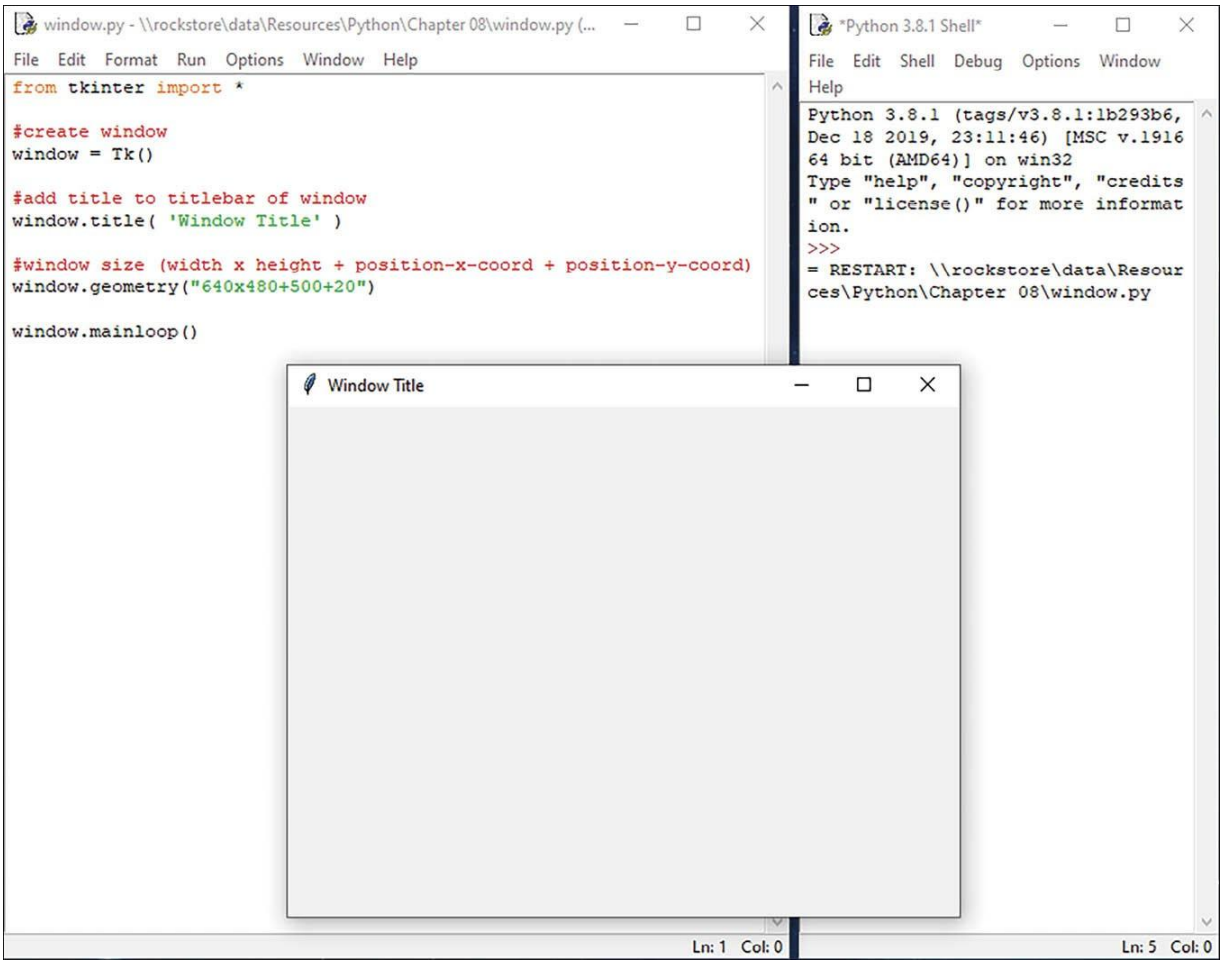
```
window.geometry("800x600+50+20")
```

Первые два числа, в данном случае '800x600', задают размер окна. Установите желаемый размер окна в пикселях. Это может быть 1280x720, 1920x1080 и т. д.

Вторые два числа, в данном случае '50+20', задают начальное положение окна на экране с использованием координат *X* и *Y*



Давайте посмотрим на программу. Откройте window.py. Здесь мы создали окно. Вы можете сделать это с помощью функции `Tk()` и назначить ее объекту window.



Мы установили размер окна таким образом, чтобы он составлял 640 пикселей в ширину и 480 пикселей в высоту. Мы также расположили окно на 500 пикселей по горизонтали от верхнего левого угла и на 20 пикселей вниз. Вы можете сделать это, используя метод `.geometry()`. Это первоначальный размер и положение окна на экране.

Мы также добавили заголовок окна. Вы можете сделать это, используя метод `.title()`. Это помогает идентифицировать ваше приложение.

Наконец, чтобы окно появилось, нам нужно войти в цикл событий Tkinter. Вы можете сделать это с помощью метода `.mainloop()`.

```
window.mainloop()
```

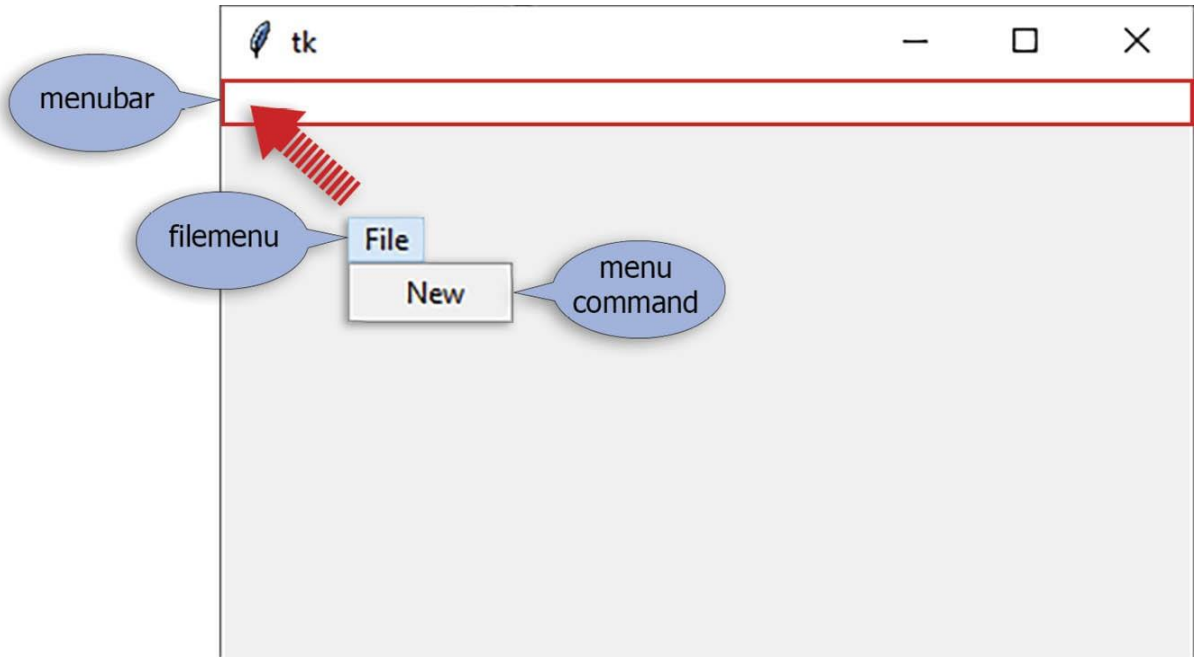
Это бесконечный цикл, используемый для запуска приложения, и он называется циклом событий. Метод `.mainloop()` ожидает события, такого как нажатие клавиши или щелчка мыши, в системе окона, и отправляет их виджетам приложения (фреймам, кнопкам, меню и т. д.).

Добавление виджетов

Виджеты — это стандартные элементы для создания графического пользовательского интерфейса с помощью Tkinter.

Меню

Добавим меню. Вы можете создать панель меню, используя функцию `Menu()`. Присвойте его объекту (например, панели меню) и закрепите на главном окне.



Теперь вам нужно создать отдельные меню (например, ‘file’, ‘edit’ и т.д.) на панели меню.

```
filemenu = Menu(menubar-to-add-to, menu-style)
```

Добавьте меню на панель меню

```
menubar.add_cascade(menu-label, menu-to-add-to)
```

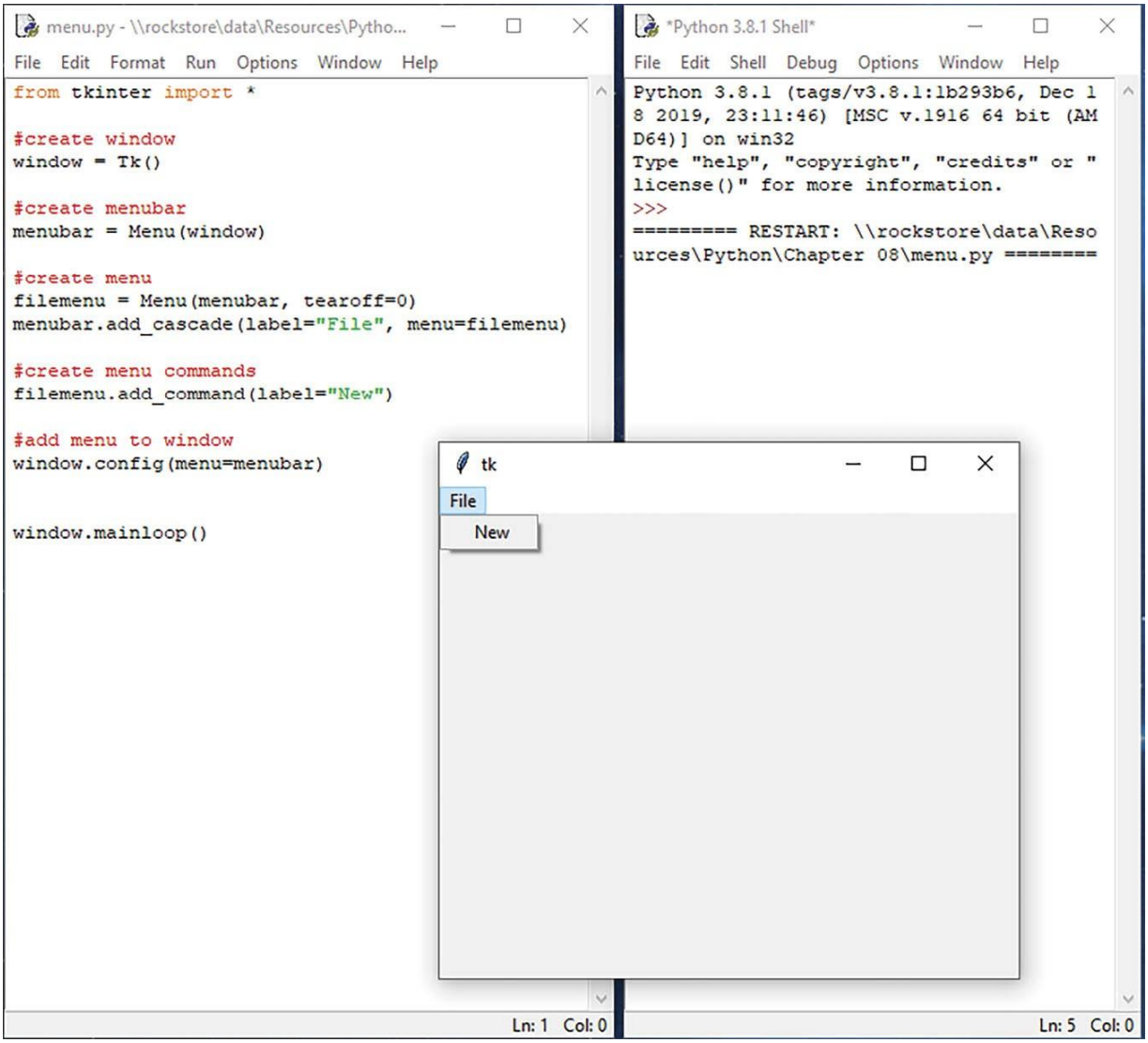
Для каждого создаваемого вами меню (например, filemenu), вам необходимо создать каждую команду меню (например, ‘new’, ‘save’, ‘exit’ и т.д.)

```
filemenu.add_command(command-label, function)
```

Наконец, добавьте созданную вами панель меню на главное окно.

```
window.config(menu-to-add)
```

Давайте рассмотрим программу. Откройте файл menu.py. Здесь мы написали программу, которая описана выше.



Рабочая область

Рабочая область используется для рисования, создания графики и макетов. Здесь вы размещаете графику, текст, кнопки и другие виджеты для создания своего интерфейса.

Чтобы создать рабочую область, используйте:

```
myCanvas = Tkinter.Canvas (parent-window,
bg="sky blue", height=300, width=300)
```

Используйте `parent-window`, чтобы заерепить рабочую область в окне. Используйте `height` и `width` для определения размера рабочей области.

Используйте `bg`, чтобы установить цвет фона. Откройте `colorchart.py` и запустите программу. Это сформирует цветовую диаграмму, показанную ниже.

Выберите название цвета из таблицы для использования в параметре `bg`.

Давайте нарисуем фигуру на рабочей области

```
rect = myCanvas.create_rectangle
(100, 100, 25, 50, fill="yellow")
```

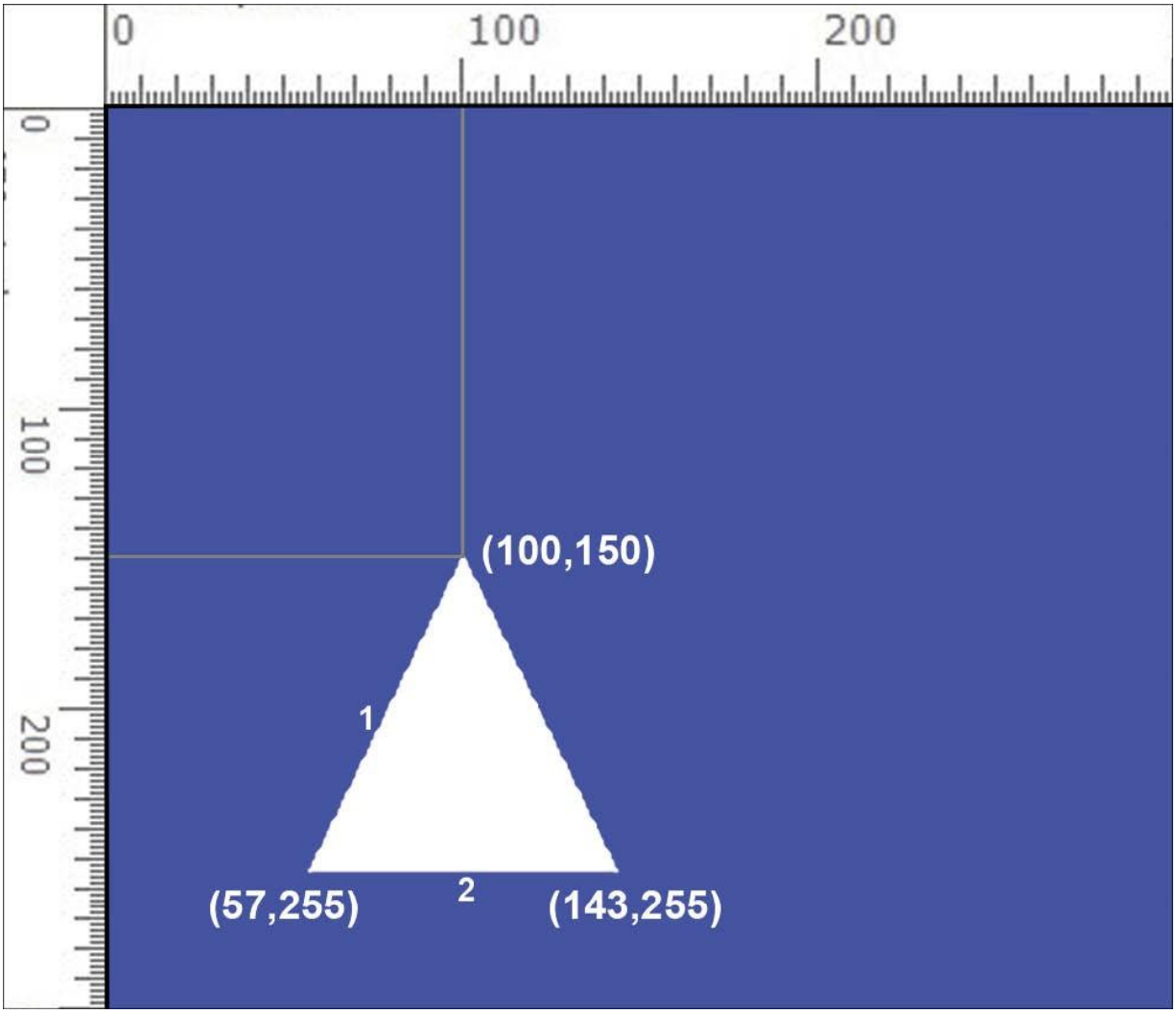
Первые два числа — это координаты x и y на рабочей области. Вторые две цифры — длина и ширина.

Вы также можете нарисовать многоугольник

```
tri = myCanvas.create_polygon
      (100,150, 57,225, 143,225, fill="green")
```

1 2

В этом примере мы создаем треугольник. У треугольника три стороны, поэтому нам нужно провести три линии. Первые две цифры обозначают начальную точку первой линии; вторые две цифры обозначают конечную точку первой линии и так далее. Давайте посмотрим:

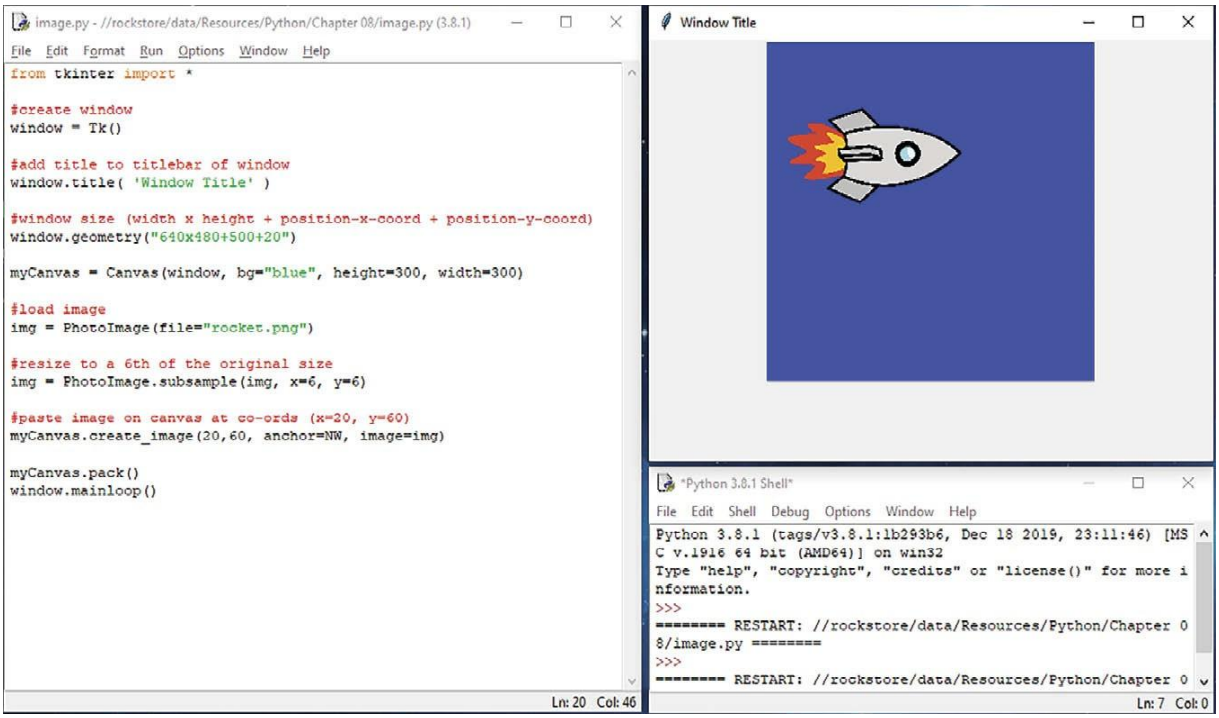


Попробуйте нарисовать пятиугольник. У пятиугольника пять сторон, поэтому нужно провести пять линий.

```
pent = myCanvas.create_polygon
(100,150, 52,185, 71,240, 129,240, 148,185,
fill="lime green")
```

Изображения

Вы можете добавлять изображения на рабочую область. Посмотрите images.py. Для загрузки изображения используйте функцию `PhotoImage()`.



Чтобы вставить изображение на рабочую область, используйте метод `.create_image()`.

Кнопки

Вы можете добавить командные кнопки на холст. Для этого используйте функцию `Button()`. Взгляните на button.py.

```
myButton = Button(window, text="label", command)
```

Используйте `window`, чтобы указать наименование окна, в котором должна работать кнопка.

Используйте `command`, чтобы указать функцию, которую вы хотите вызвать для обработки того, что делает кнопка. Для этого вы можете вызвать существующие функции или определить свои собственные функции.

Используйте метод `.pack`, чтобы добавить кнопку в окно.

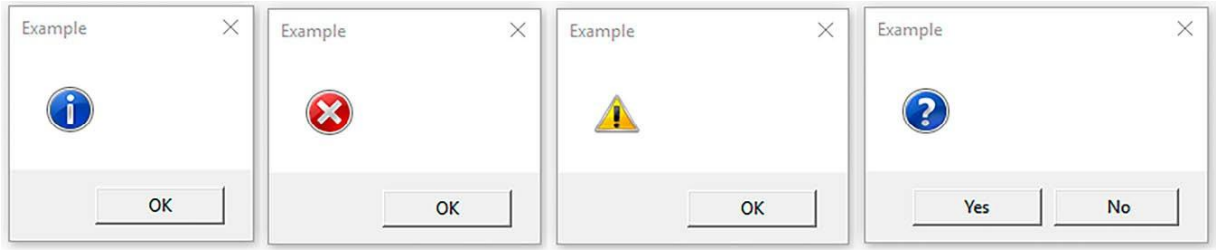
```
myButton.pack()
```

Окна сообщений

Вы можете добавлять в свои программы окна сообщений. Для этого вам нужно будет импортировать функции окна сообщений из модуля tkinter. Вы можете сделать это с помощью команды импорта.

```
from tkinter import messagebox
```

Вы можете создавать различные типы окон сообщений. Информационное окно, окно предупреждения, окно ошибки и окно с запросом ответа типа yes/no (да/нет).



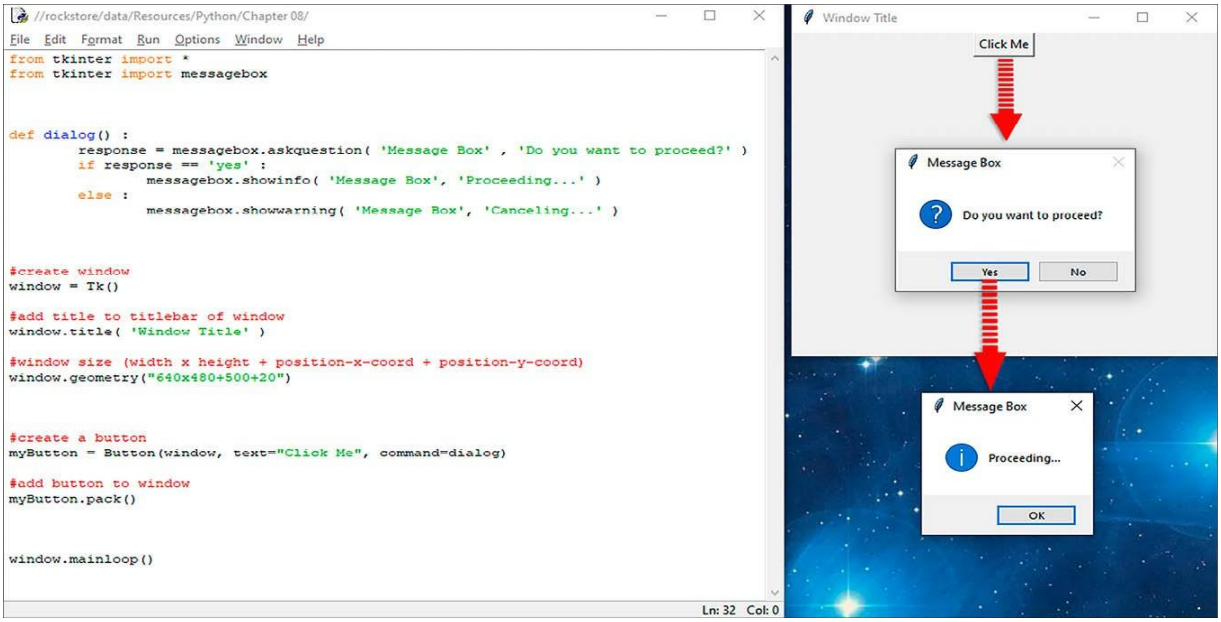
messagebox.showinfo() messagebox.showerror() messagebox.showwarning() messagebox.askquestion()

```
messagebox.showinfo('Message Title', 'Message')
```

Если вы просите пользователя ответить да/нет, то вам необходимо обработать его ответ.

```
response = messagebox.askquestion
('Message Box' , 'Question...')

if response == 'yes' :
executed if user clicks 'yes'

else :

executed if user clicks 'no'
```

Давайте рассмотрим messagebox.py



Текстовое поле

Используйте функцию `Entry()` для создания текстового поля.

```
userInput = Entry( window )
```

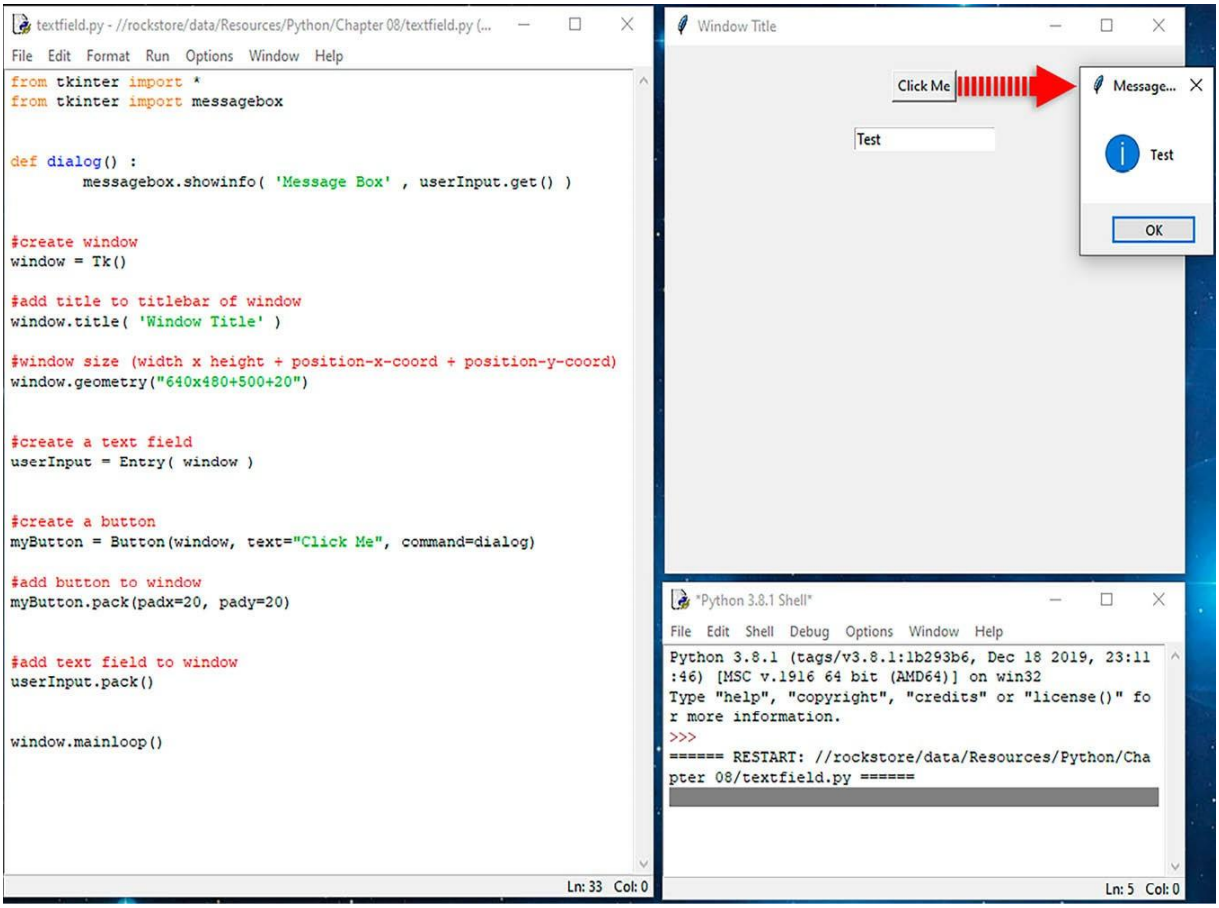
Используйте метод `.pack()`, чтобы добавить поле в ваше окно.

```
userInput.pack()
```

Чтобы получить данные из текстового поля, используйте метод `.get()`

```
userInput.get()
```

Давайте добавим это в программу. Взгляните на textfield.py. Здесь мы добавили текстовое поле на рабочую область под командной кнопкой.



Мы также добавили код в функцию `dialog()`, чтобы получать данные из текстового поля и отображать их в окне сообщения.

Функция `dialog()` вызывается при нажатии кнопки ‘click me’.

Запустите программу и посмотрите, что она делает.

Поле списка

Используйте для создания поля списка функцию `Listbox()`.

```
list = Listbox(window)
```

Используйте метод `.insert()` для добавления элементов в список.

```
list.insert(1, 'Item One')
```

Используйте метод `.pack()`, чтобы добавить поле списка в ваше окно. Используйте параметры Padx и Pady, чтобы добавить отступы для определения положения поля списка в окне.

```
list.pack(padx=20, pady=20)
```

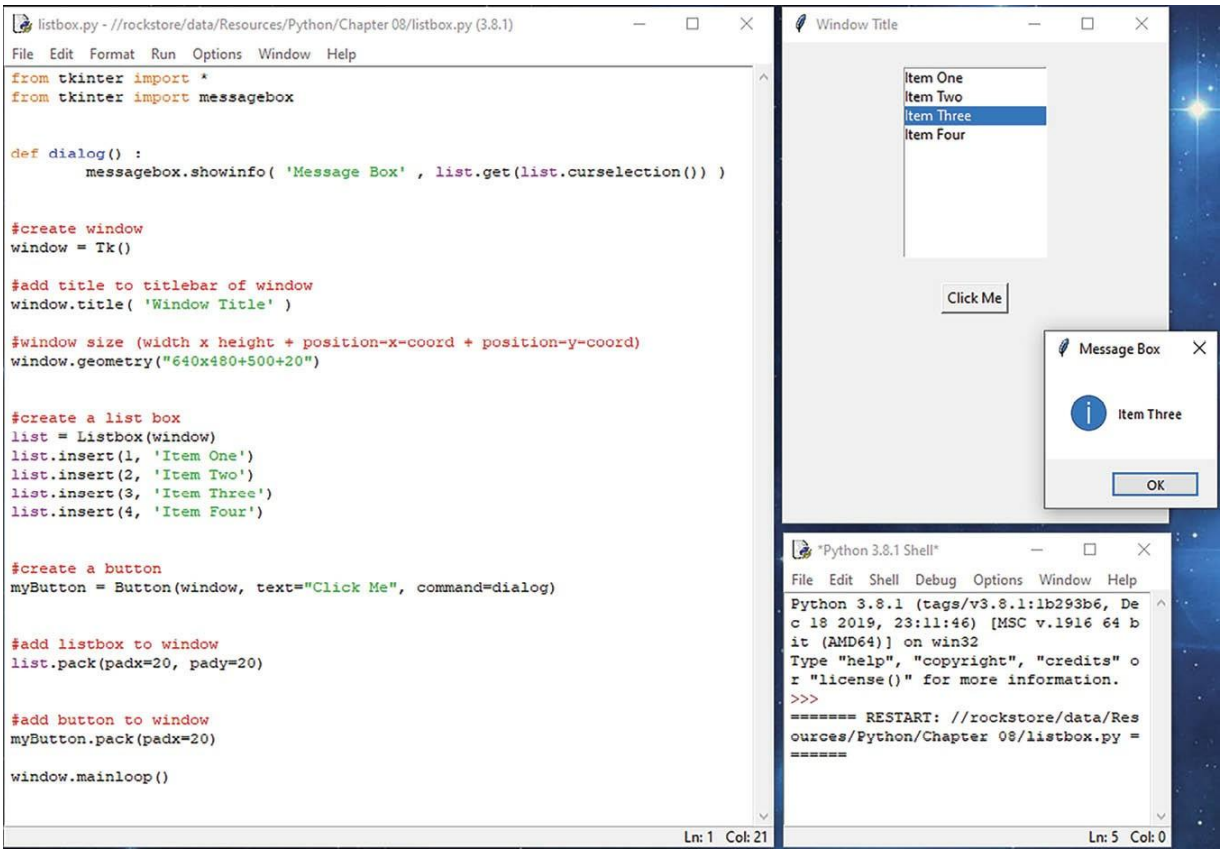
Используйте метод `.curselection()`, чтобы получить индекс элемента, выбранного пользователем. Помните, индекс первого элемента равен 0.

```
selectedItem = list.curselection()
```

Используйте метод `.get()` для возврата элемента

```
list.get (selectedItem)
```

Давайте рассмотрим программу. Откройте listbox.py.



Флажок

Используйте для создания каждого флажка функцию `Checkbutton()`.

```
box1 = Checkbutton(window, text="Red",
variable=box1Checked, onvalue=1)
```

Вам нужно будет создать переменную для каждого флажка, чтобы назначить ее ‘onvalue’ (значение), если пользователь нажмет на флажок.

```
box1Checked = IntVar()
```

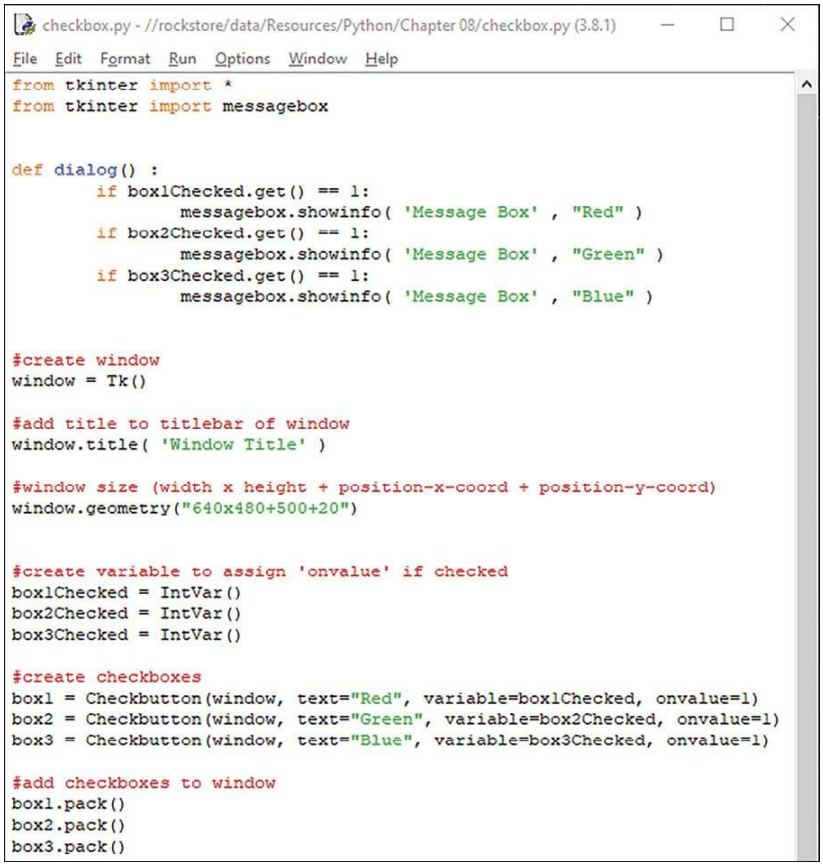
Переменные, созданные вами выше, будут иметь значение 1 или 0. Для параметра Onvalue установлено значение 1, поэтому переменной будет присвоено значение 1, когда пользователь нажмет флажок. Используйте метод `.get()`, чтобы получить значение.

```
if box1Checked.get() == 1:
messagebox.showinfo( 'Msg' , "Red" )
```

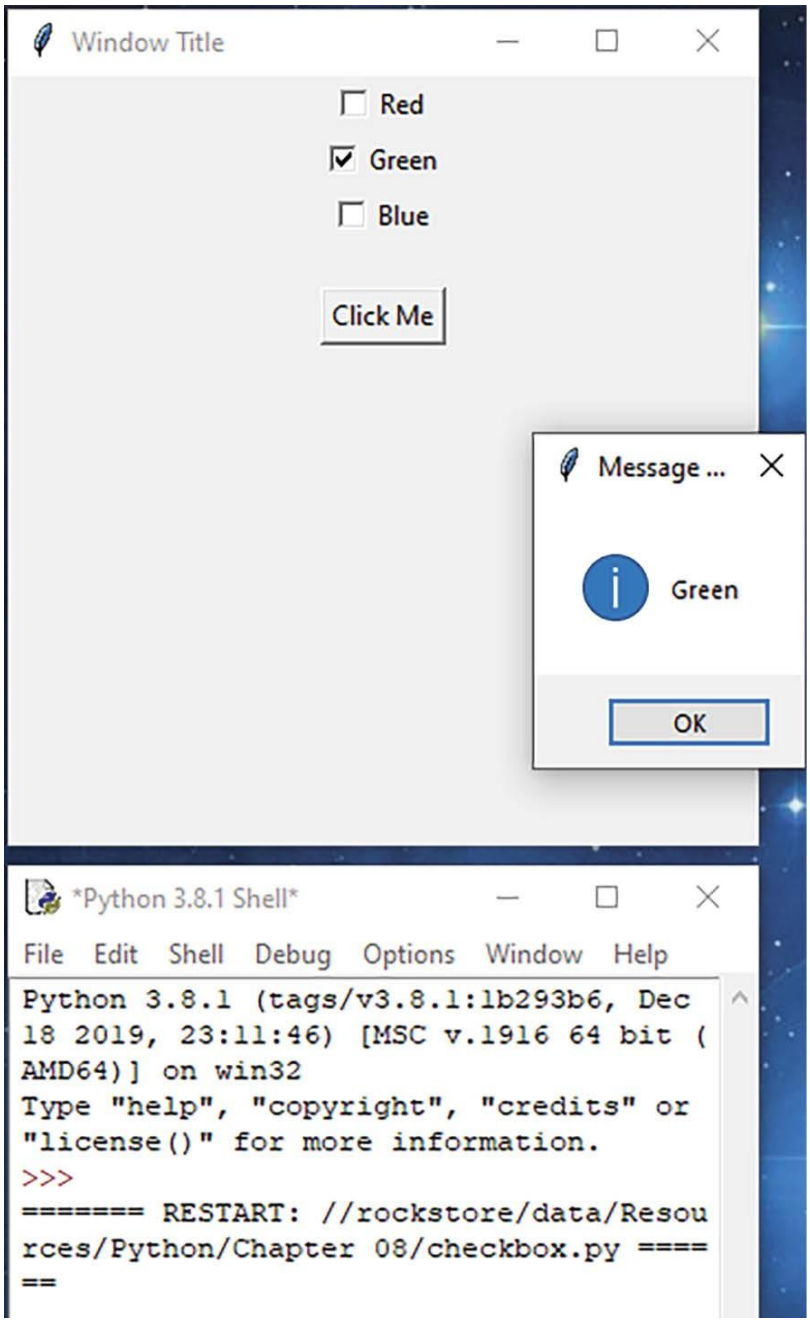
Используйте метод `.pack()`, чтобы добавить каждый из ваших флажков в окно.

```
box1.pack()
```

Давайте рассмотрим программу. Откройте checkbox.py.



При запуске программы вы можете установить любой из флажков. Когда вы нажимаете кнопку флажка, функция считывает, какой флажок установлен, и возвращает значение.



Метки

Вы можете создавать метки для обозначения текстовых полей и других элементов вашего интерфейса. Используйте для этого Label()

```
textLabel = Label(window, text="Enter Name:")
```

Используйте pack(), чтобы добавить метку в ваше окно.

```
textLabel.pack()
```

LabelFrame

LabelFrame используется для группировки связанных виджетов, таких как флажки, переключатели или текстовые поля.

Сначала вам нужно создать рамку для группы меток. Вы можете сделать это с помощью `LabelFrame()` следующим образом

```
group1 = LabelFrame(window, text="label",
padx=5, pady=5)
```

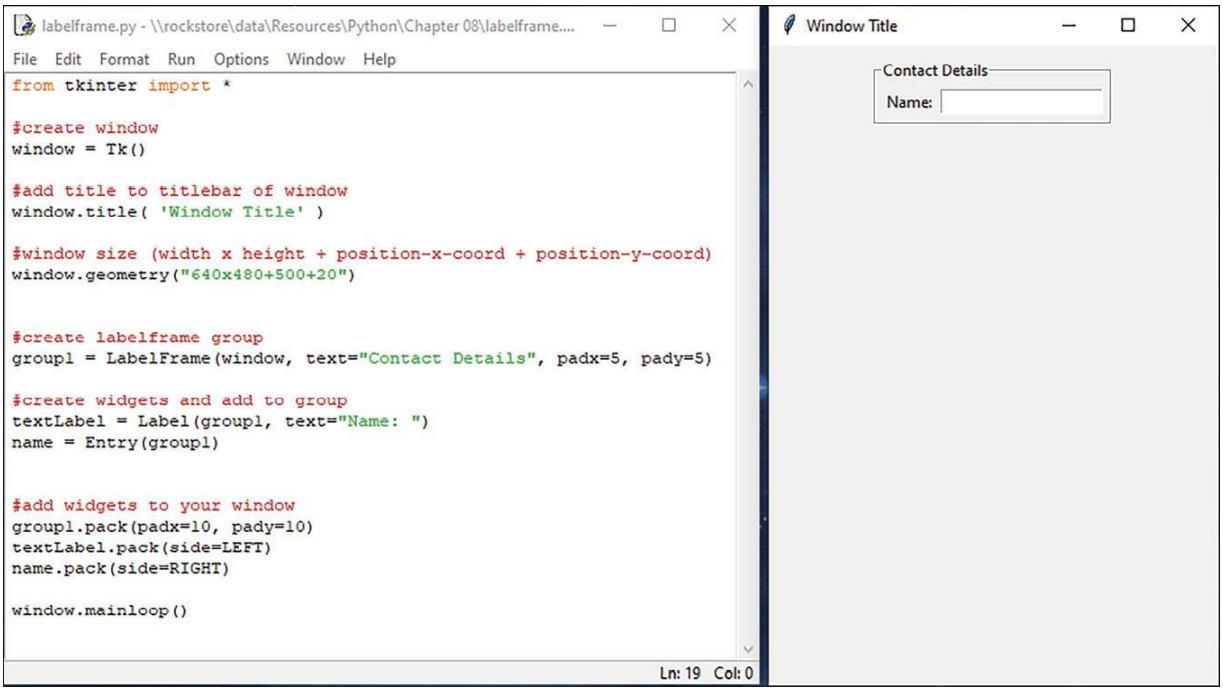
Используйте первое окно параметров, чтобы закрепить группу на главном окне. Далее вам нужно добавить в группу виджеты. Сделать это можно обычным способом, только в функциях виджета нужно указать, к какому виджету подключиться. Итак, чтобы добавить нашу текстовую метку, укажите виджет, к которому нужно подключиться, используя первый параметр (наша рамка метки, определенная выше, называется group1, поэтому используйте group1, подчеркнутую ниже).

```
textLabel = Label(group1, text="Name: ")
```

Добавьте свои виджеты в окно обычным способом.

```
textLabel.pack(side=LEFT)
```

Давайте рассмотрим программу. Откройте файл labelframe.py.



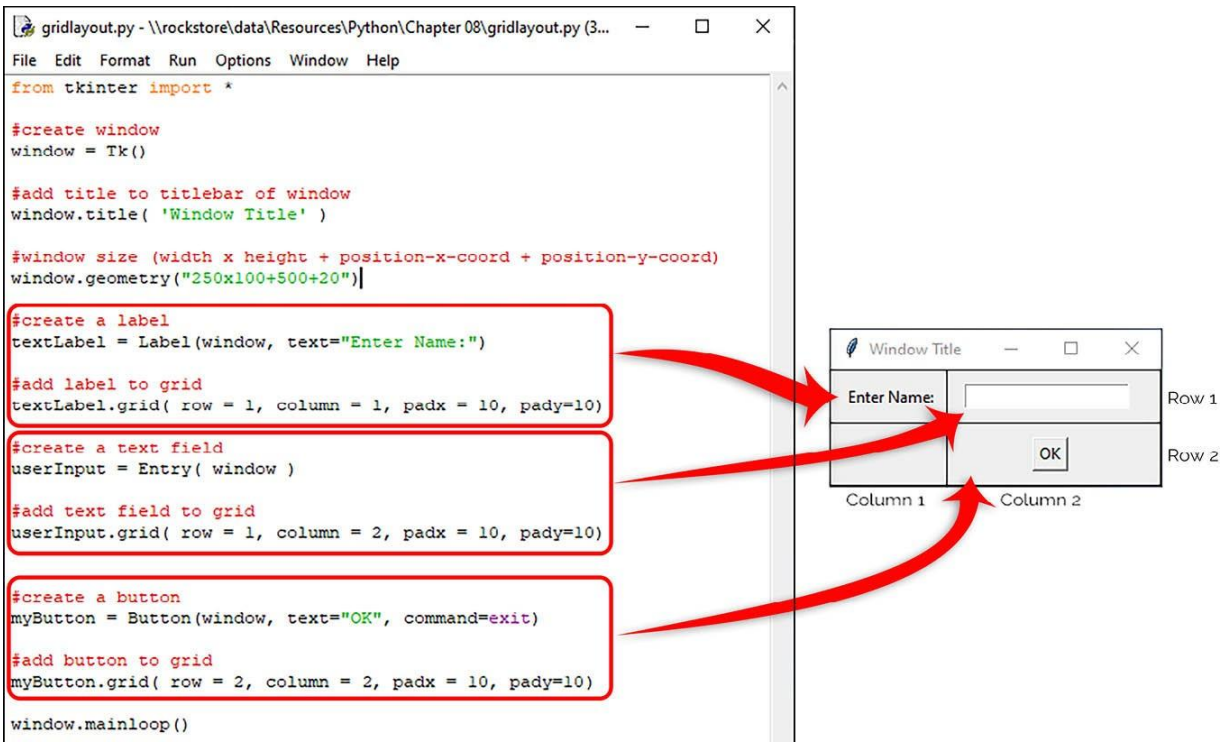
Здесь мы создали текстовую метку и текстовое поле внутри группы меток (group1).

Дизайн интерфейса

Теперь, когда мы знаем, как создавать окна, меню и добавлять различные типы виджетов, мы рассмотрим, как расположить их в окне, чтобы создать удобный интерфейс.

Вы можете сделать это с помощью менеджера сетки макета. Давайте посмотрим на пример. Откройте файл gridlayout.py.

Используйте метод `.grid()` для размещения виджетов в окне в соответствии с макетом сетки. Используйте параметры `row` и `column`, чтобы указать, в какую ячейку сетки поместить виджет. Используйте параметры `padx` и `pady`, чтобы добавить пространство вокруг виджетов в сетке.

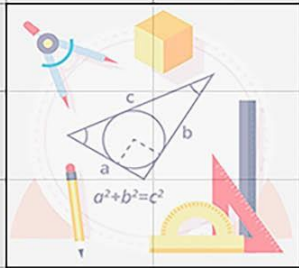


Здесь мы разместили текстовую метку в строке 1 и столбце 1. В строке 1, столбец 2 расположено текстовое поле. Командную кнопку мы разместили в строке 2 и столбце 2.

Запустив программу, вы увидите результат, показанный ниже.

Enter Name:	
	OK

Давайте разработаем простой интерфейс для приложения-конвертера единиц измерения. Чтобы создать этот интерфейс, мы разделим окно на 3 строки и 5 столбцов.

	1	2	3	4	5
1			convert:	drop down	
2				text field	button
3				label	

Теперь мы разместим логотип слева, а разместим мы его на 2 столбца вправо и на 3 строки вниз.

```
img = PhotoImage(file="logo.png")

imgLbl = Label(window, image=img)

imgLbl.grid( row = 1, column = 1, padx = 10,
pady=10, columnspan=2, rowspan=3)
```

Мы также разместим метку в строке 1 и столбце 3.

```
textLabel = Label(window, text="Convert:")

textLabel.grid( row = 1, column = 3,
padx = 10, pady=10)
```

Раскрывающийся список в строке 1 и столбце 4,

```
conversions.grid( row = 1, column = 4,
padx = 10, pady=10)
```

Текстовое поле в строке 2 и столбце 4, с кнопкой в строке 2 и столбце 5.

```
userInput.grid( row = 2, column = 4,
padx = 10, pady=10)
```

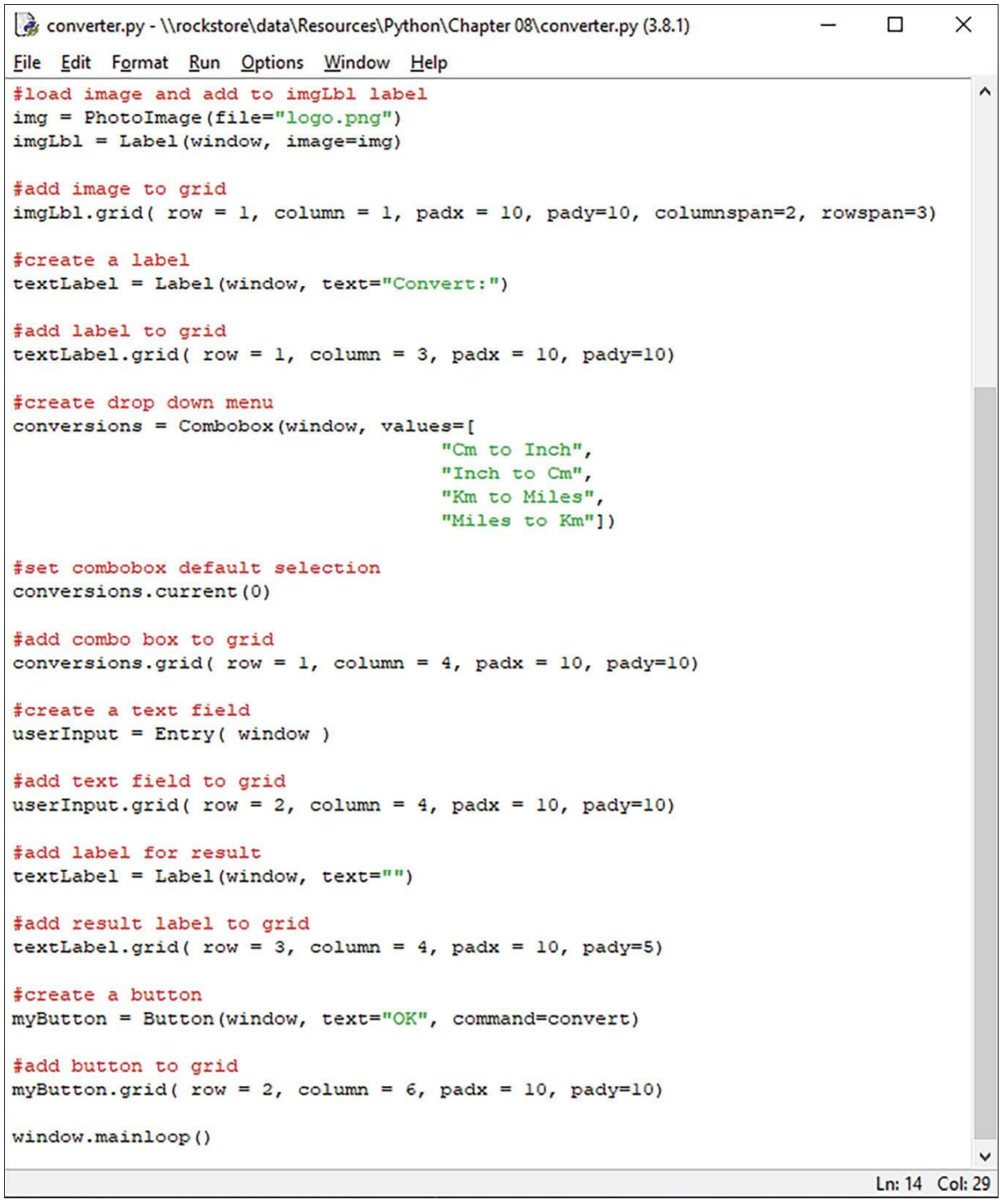
Метка внизу в строке 3 и столбце 4, чтобы обозначить результат.

```
textLabel.grid( row = 3, column = 4,
padx = 10, pady=5)
```

Добавьте командную кнопку в строку 2 и столбец 6

```
myButton.grid( row = 2, column = 6,
padx = 10, pady=10)
```

Давайте рассмотрим программу. Откройте файл Converter.py. Здесь мы добавляем виджеты в сетку с помощью метода `.grid()`.



Мы использовали параметры padx и pady для распределения виджетов на сетке макета.

Вот мы и разобрались с построением интерфейса. В нынешнем виде программа ничего не сделает, если вы нажмете кнопку или введете число в текстовое поле.

Нам нужно написать функцию, которая позаботится об этом и которая будет вызываться при нажатии кнопки.

Объявим функцию обычным способом. Мы назовем ее `convert()`.

```
def convert():
    if conversions.current() == 0:
        n = float(userInput.get()) * 0.39
        textLabel = Label(window, text=n)
        textLabel.grid( row = 3, column = 4, padx = 10, pady=5)
    elif conversions.current() == 1:
        n = float(userInput.get()) * 2.54
        textLabel = Label(window, text=n)
        textLabel.grid( row = 3, column = 4, padx = 10, pady=5)
    elif conversions.current() == 2:
        n = float(userInput.get()) * 0.62
        textLabel = Label(window, text=n)
        textLabel.grid( row = 3, column = 4, padx = 10, pady=5)
    elif conversions.current() == 3:
        n = float(userInput.get()) * 1.60
        textLabel = Label(window, text=n)
        textLabel.grid( row = 3, column = 4, padx = 10, pady=5)
```

Вам нужно будет прочитать, что выбрано в поле со списком. Вы можете сделать это с помощью метода `.current()`. Первый элемент в поле со списком имеет индекс 0, второй — 1 и так далее. Используйте оператор if, чтобы отделить вычисления для каждого выбора в поле со списком.

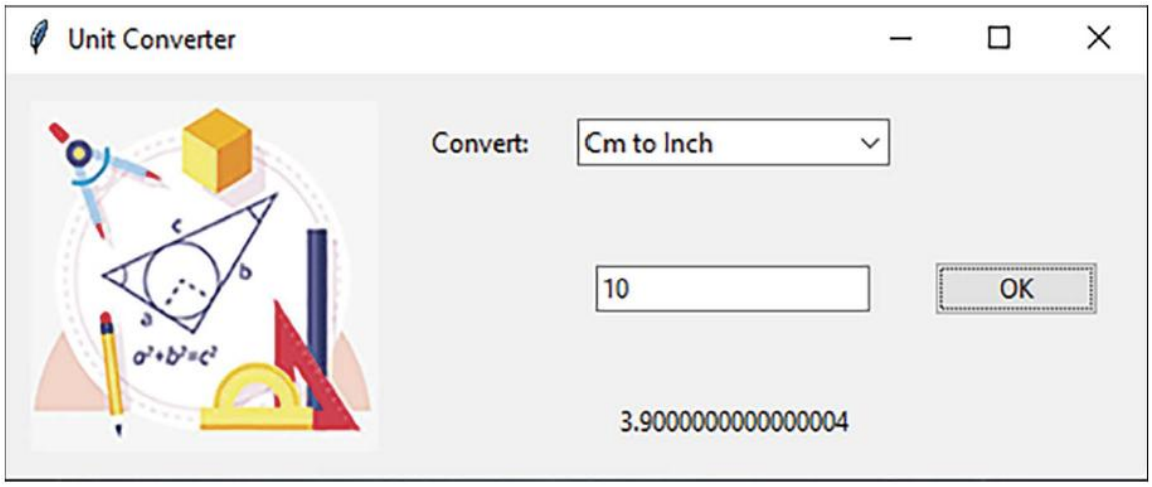
```
if conversions.current() == 0:
```

Далее вам нужно будет получить данные из текстового поля. Вы можете сделать это с помощью метода `.get()` для текстового поля. Не забудьте привести тип данных к числу с плавающей запятой, поскольку данные, получаемые из текстового поля представляют собой строку.

```
n = float (userInput.get())
```

Выполните расчет и верните результат в пустую текстовую метку в строке 3, столбце 4 сетки.

Теперь, когда вы запустите программу, вы получите красиво оформленный интерфейс


Unit Converter
Convert:
Cm to Inch
10
OK
3.9000000000000004

Разработка игр

Чтобы начать создавать свои собственные игры с использованием Python, вам понадобится модуль PyGame. Этот модуль не входит в состав стандартных дистрибутивов Python, поэтому вам необходимо установить его перед началом работы.

Для этого раздела посмотрите демонстрационные видео

`elluminetpress.com/pygraphics`

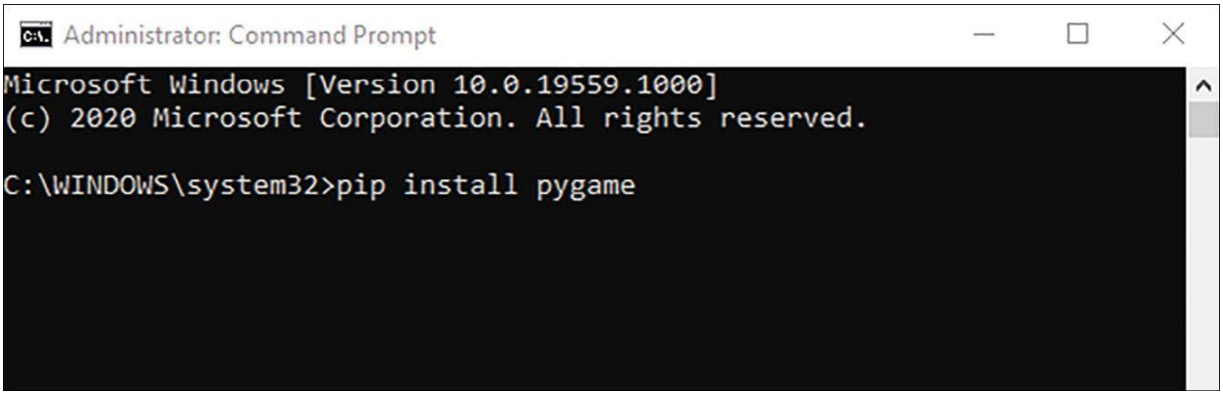
Вам также понадобятся исходные файлы из каталога Chapter 12.

Установка PyGame

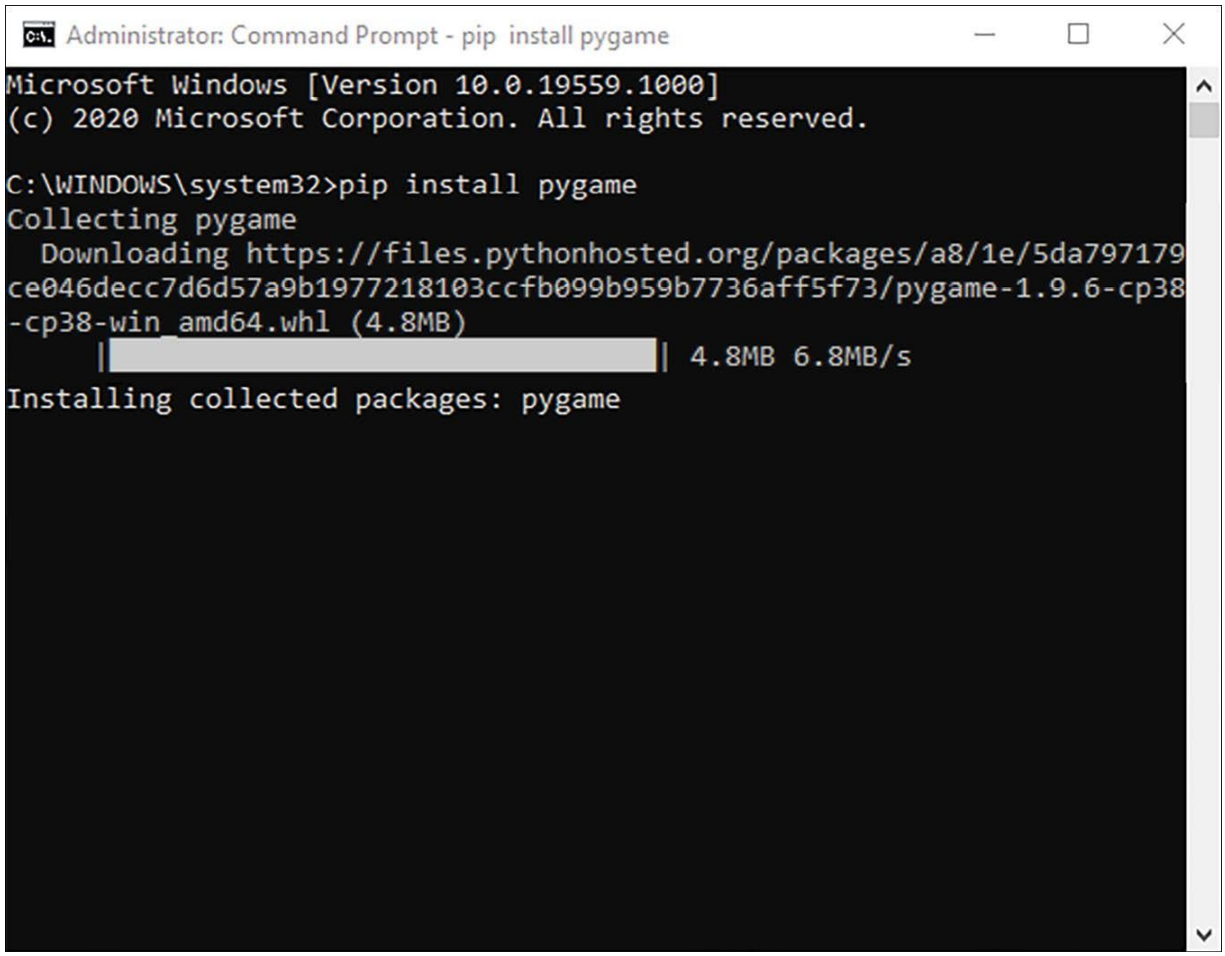
Чтобы установить модуль, откройте командную строку. Убедитесь, что вы запускаете ее от имени администратора. В командной строке введите

```
pip install pygame
```

Как только вы нажмете Enter, начнется процесс установки.



Разрешите утилите pip загрузить и установить модуль.



После завершения процесса вы можете начать использовать pygame.

Открытие окна

Первое, что вам нужно сделать, это импортировать модуль module.

```
import pygame
```

pygame использует метод `.init()`

```
pygame.init()
```

Откройте окно. Установите размер окна 640 пикселей в ширину и 480 пикселей в высоту. Вы также можете установить 800x600, 1920x1080 и т. д.

```
gamewindow = pygame.display.set_mode((640,480))
```

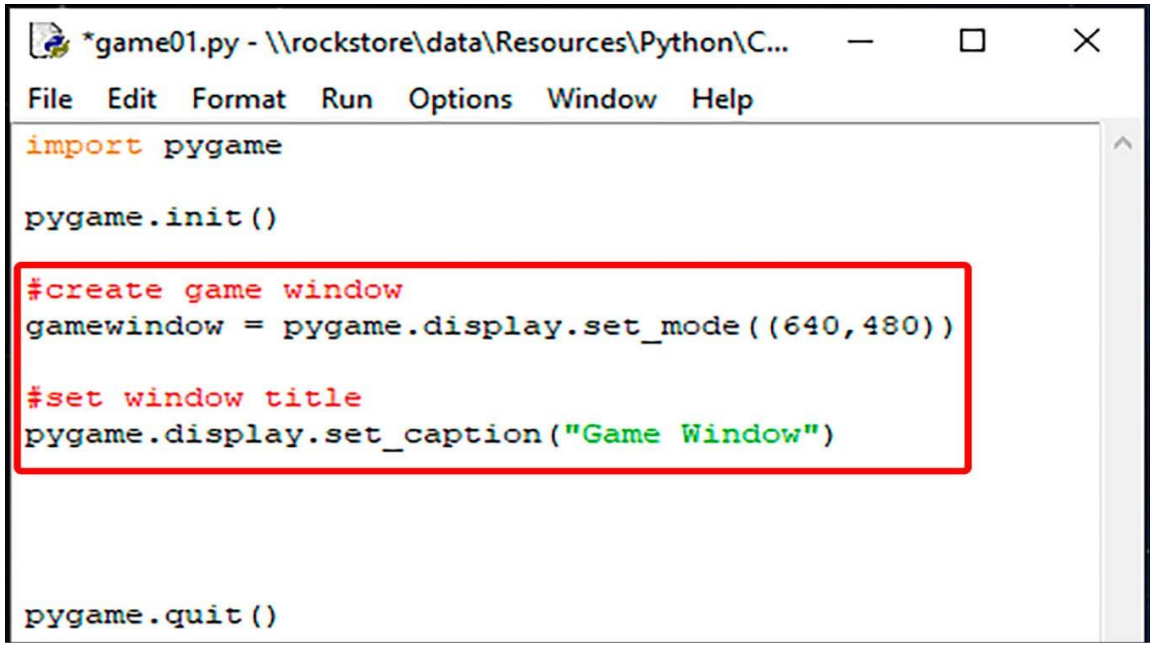
Задайте заголовок окна. Этот заголовок будет отображаться в строке заголовка окна.

```
pygame.display.set_caption("Game Window")
```

Вы всегда должны завершать свои сценарии pygame методом `.quit()`.

```
pygame.quit()
```

Давайте рассмотрим программу поближе.



Если запустить эту программу, окно игры инициализируется, откроется и тут же закроется. Это нормально, поскольку другого кода для выполнения нет.

Добавление изображения

Добавим изображение в окно игры. Это будет наш персонаж игры. В данном случае мы собираемся использовать космическую ракету. Мы можем добавить оператор загрузки изображения в нашу программу.

```
sprite = pygame.image.load('rocket.png')
```

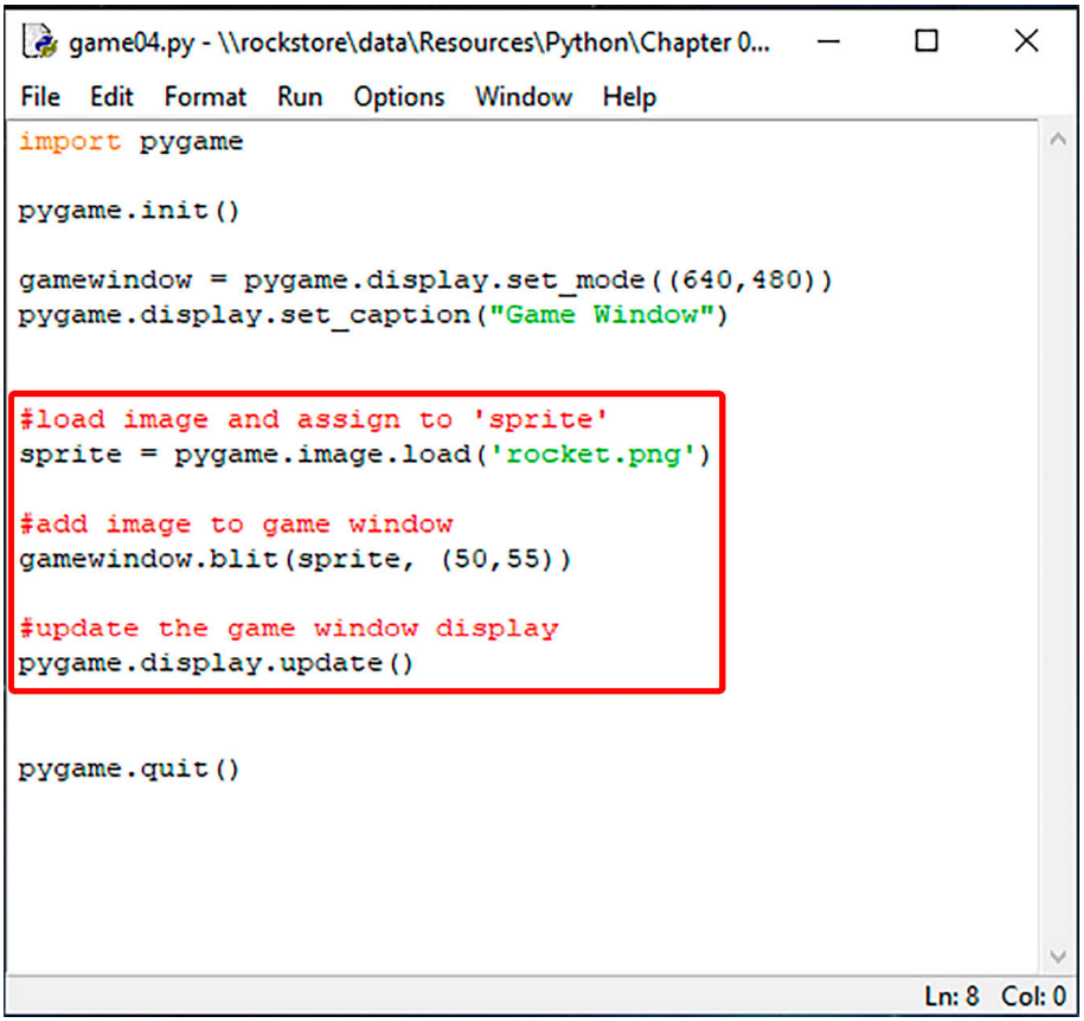
Вставьте изображение (спрайт) в окно игры с помощью метода `.blit()` и назначьте его начальную позицию на экране (x, y)

```
gamewindow.blit(sprite, (x,y))
```

Обновите состояние дисплея, чтобы отображалось изображение

```
pygame.display.update()
```

Давайте рассмотрим программу.



Игровой цикл

Теперь давайте заставим нашу ракету что-нибудь сделать. Для этого нам нужно создать игровой цикл для рисования наших спрайтов, обновления экрана и поддержания работы программы.

Мы можем взять следующие два оператора и добавить их в наш игровой цикл. Для этого мы будем использовать цикл while.

```
gamewindow.blit(sprite, (x,y))
```

Обновите дисплей, чтобы отображалось изображение

```
pygame.display.update()
```

Поместите все это внутри цикла while.

```
while running == 1:
gamewindow.blit(sprite, (x,y))
pygame.display.update()
```

Нам также потребуется инициализировать некоторые переменные: x и y — начальное положение на экране и running — чтобы указать, запущена программа или нет.

```
running = 1
x = 250
y = 280
```

Давайте рассмотрим программу.

```
#initialize our variables
running = 1
x=250
y=280

while running:
    #add image to game window
    gamewindow.blit(sprite, (x,y))

    #update the game window display
    pygame.display.update()


pygame.quit()
```

Цикл событий

В этой простой игре мы хотим, чтобы ракета перемещалась влево и вправо, когда пользователь нажимает на клавиатуре клавиши со стрелками влево и вправо. Для этого нам нужно что-то для обработки этих событий.

Для этого мы можем поместить цикл for внутри нашего цикла while (игрового цикла). Мы отслеживаем каждое происходящее событие. Вы можете использовать метод `.get()` для чтения каждого события.

```
for event in pygame.event.get():
```

Теперь внутри цикла for нам нужно реализовать выбор в зависимости от того, какая клавиша нажата. Для этого мы можем использовать оператор `if`.

Сначала нам нужно проверить, была ли нажата клавиша.

```
if event.type == pygame.KEYDOWN:
```

Внутри этого оператора if нам нужно определить, какая клавиша была нажата. Вы можете использовать другой оператор if.

```
if event.key == pygame.K_LEFT:
```

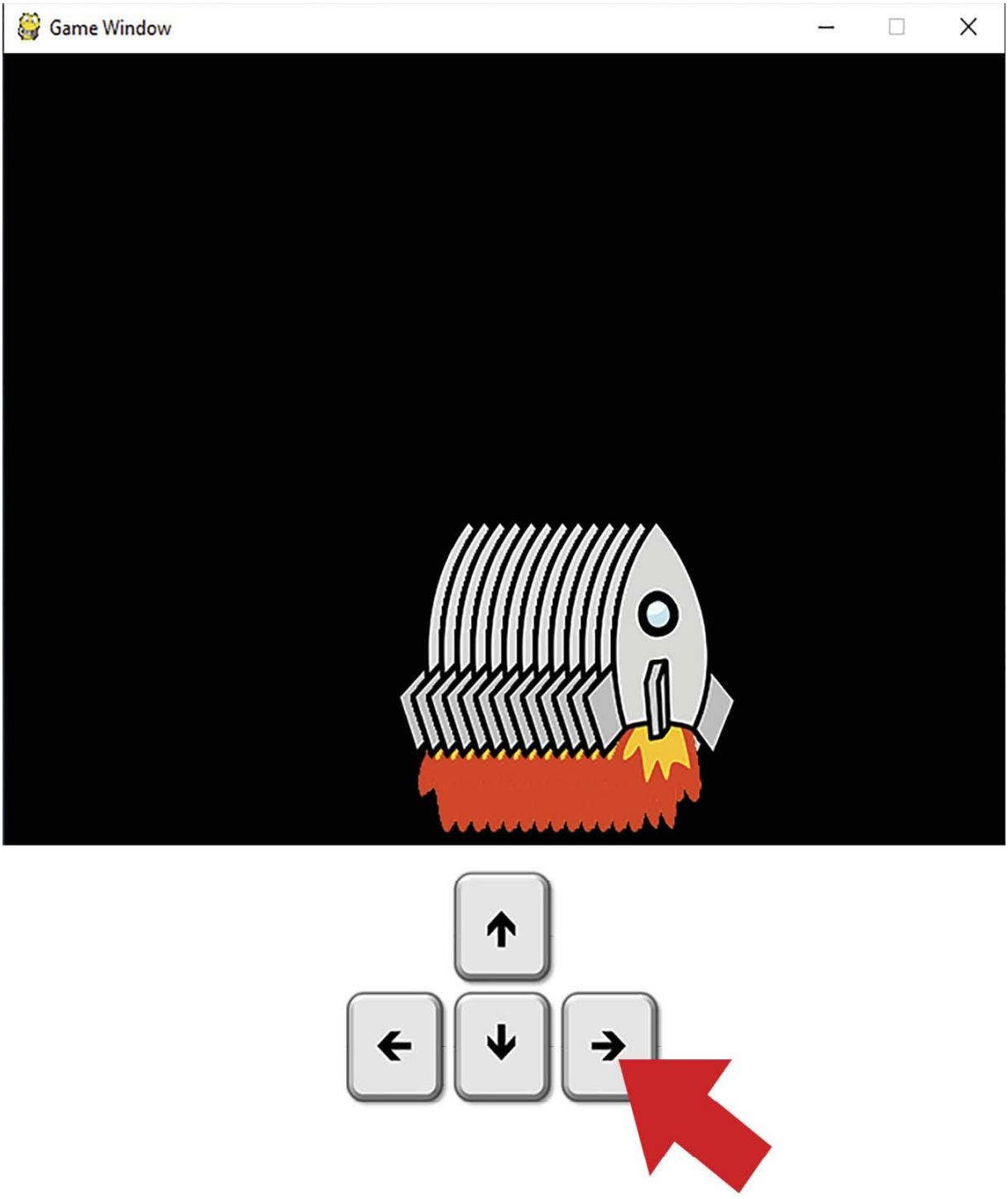
Мы делаем то же самое для всех остальных кнопок, которые собираемся использовать. Просто добавьте операторы `elif` в оператор if .

Давайте взглянем

```
while running:

    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                x = x - 10 #shift image left 10 pixels
            elif event.key == pygame.K_RIGHT:
                x = x + 10 #shift image right 10 pixels

    gamewindow.blit(sprite, (x,y))
    pygame.display.update()


pygame.quit()
```

Ln: 18 Col: 0

Теперь, когда мы запустим программу, вы увидите, как ваша ракета движется влево, когда вы нажимаете клавишу влево, и вправо, когда вы нажимаете клавишу вправо.



Вы также заметите кое-что еще. Изображение повторяется на экране. Чтобы исправить это, вам необходимо очищать экран (обновлять) каждый раз при перемещении объекта. Вы можете использовать:

```
gamewindow.fill((0,0,0))
```

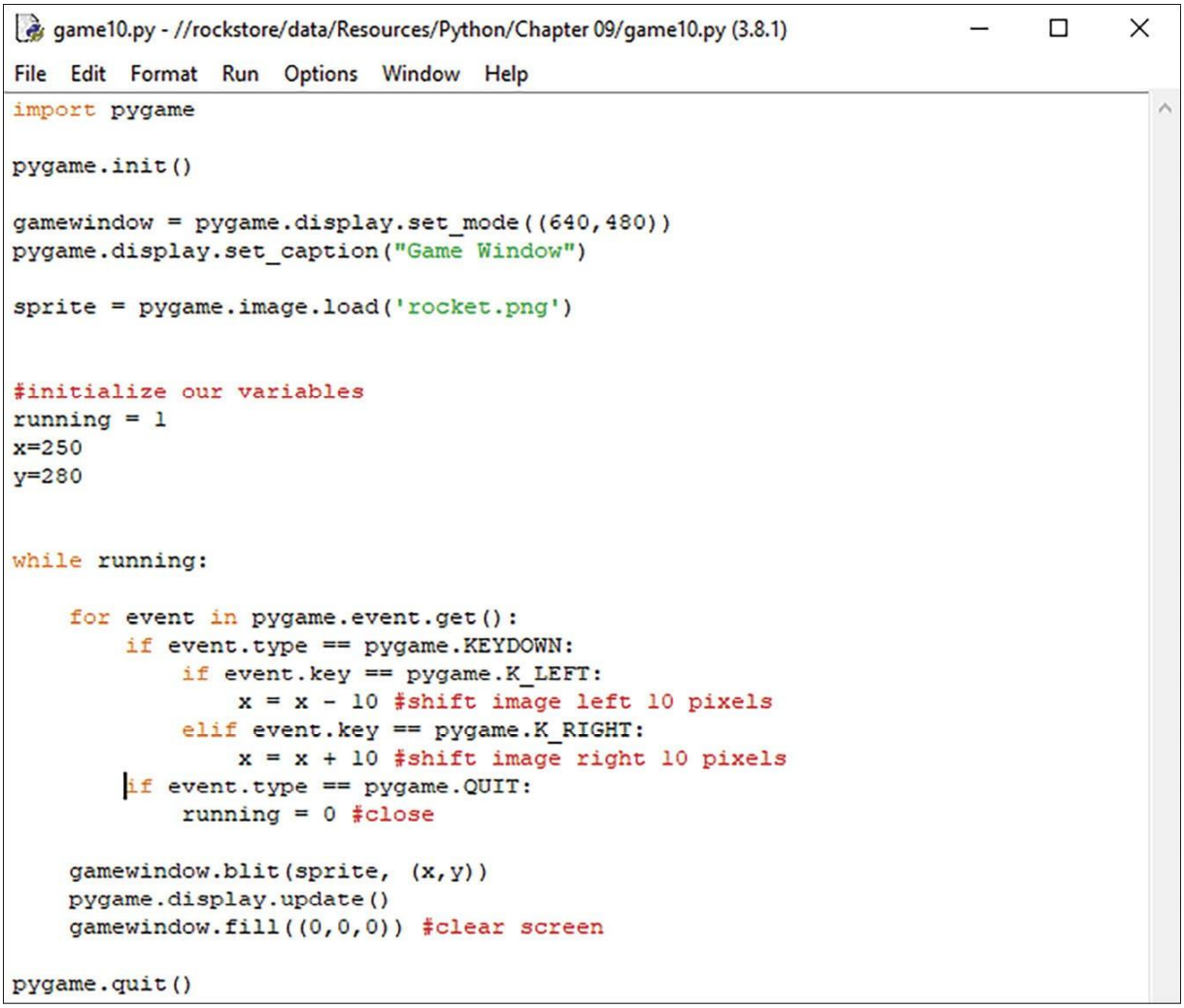
Это возвращает экрану черный цвет в конце каждой итерации игрового цикла (цикла while).

Также рекомендуется включать событие выхода в цикл событий, чтобы программа корректно завершала работу.

```
if event.type == pygame.QUIT:
running = 0
```

Это установит нашу рабочую переменную в 0, что означает, что игровой цикл завершится и программа закроется. Это событие произойдет, когда вы щелкните значок закрытия в правом верхнем углу окна.

Давайте рассмотрим программу поближе.



Вы также заметите, что ракета не движется, когда вы удерживаете кнопку нажатой. Это связано с тем, что функция повтора нажатия клавиш отключена. Чтобы включить ее, добавьте следующую строку

перед игровым циклом.

```
pygame.key.set_repeat(1, 25)
```

Первый параметр — это задержка перед повторением ключевого события. Второй параметр — интервал между повторами.

Фигуры

Вы можете добавлять такие формы, как круги, эллипсы, прямоугольники и другие многоугольники.

Чтобы нарисовать прямоугольник, используйте метод `.rect()`. Укажите поверхность или окно, на котором вы хотите рисовать, цвет, затем укажите положение x и y, а затем ширину и длину прямоугольника.

```
pygame.draw.rect(gamewindow, colour,
(x, y, width, length), thickness)
```

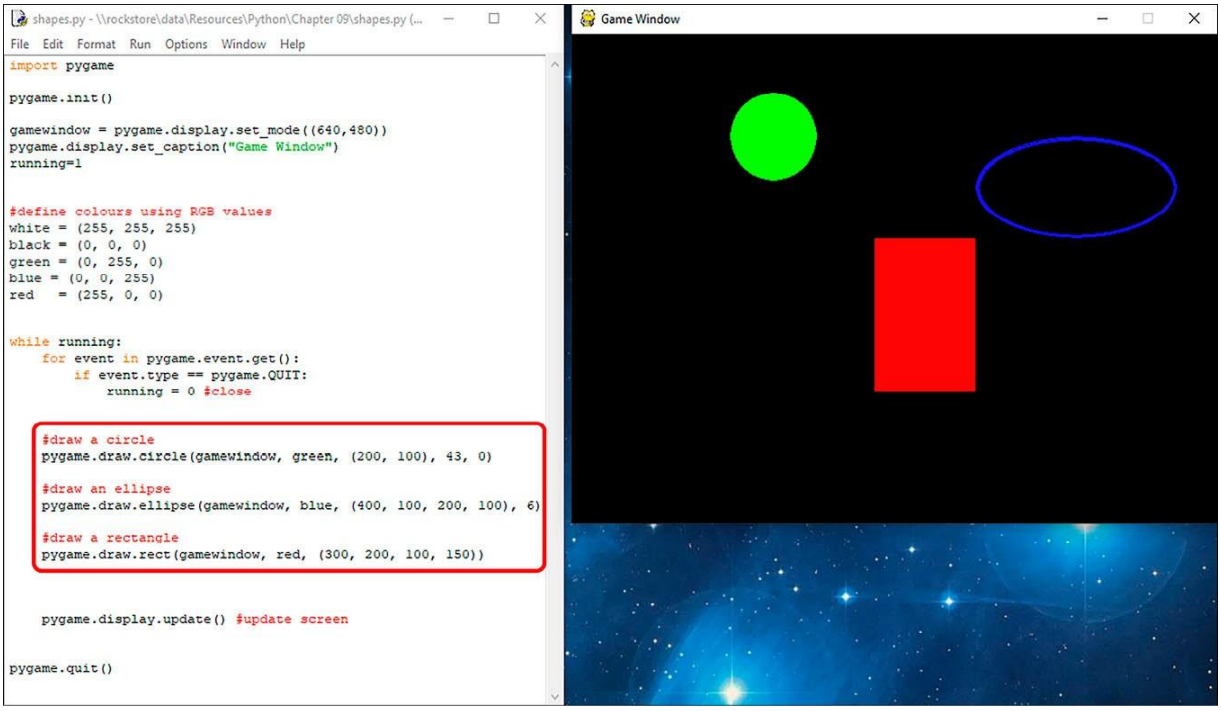
Чтобы нарисовать эллипс, используйте метод `.ellipse()`. Рисуя эллипс, вы фактически рисуете его внутри невидимого прямоугольника. Вот почему вы указываете ширину и длину при рисовании эллипса.

```
pygame.draw.ellipse(gamewindow, colour,
(x, y, width, length), thickness)
```

Чтобы нарисовать круг, используйте метод `.circle()`. Укажите поверхность или окно, на котором вы хотите рисовать, цвет, затем укажите положение x и y, а затем радиус круга.

```
pygame.draw.circle(gamewindow, colour,
(x, y), radius, thickness)
```

Взгляните на shapes.py


shapes.py - \\rockstore\data\Resources\Python\Chapter 09\shapes.py (...
File Edit Format Run Options Window Help
import pygame
pygame.init()
gamewindow = pygame.display.set_mode((640,480))
pygame.display.set_caption("Game Window")
running=1
#define colours using RGB values
white = (255, 255, 255)
black = (0, 0, 0)
green = (0, 255, 0)
blue = (0, 0, 255)
red = (255, 0, 0)
while running:
for event in pygame.event.get():
if event.type == pygame.QUIT:
running = 0 #close
#draw a circle
pygame.draw.circle(gamewindow, green, (200, 100), 43, 0)
#draw an ellipse
pygame.draw.ellipse(gamewindow, blue, (400, 100, 200, 100), 6)
#draw a rectangle
pygame.draw.rect(gamewindow, red, (300, 200, 100, 150))
pygame.display.update() #update screen
pygame.quit()
Game Window

Базовая анимация

Чтобы продемонстрировать базовую анимацию, мы собираемся переместить наш объект НЛО по экрану.

Сначала нам нужно загрузить наше изображение. Вы можете сделать это, используя метод `.load()`, как мы делали это раньше.

```
ufo = pygame.image.load('ufo.png')
```

Теперь, поскольку изображение по умолчанию загружается на объект на поверхности, мы не можем перемещать его или манипулировать им. Чтобы обойти эту проблему, мы привязывам изображение к прямоугольнику. Вы можете сделать это, используя метод `.rect()`.

```
ufo_rect = ufo.get_rect()
```

Нам также необходимо определить некоторые переменные скорости и направления. Мы можем сделать это со списком. Это список, содержащий координаты `[x, y]` на экране *(speed[0] - это x, speed[1] - это y)*.

```
speed = [10, 0]
```

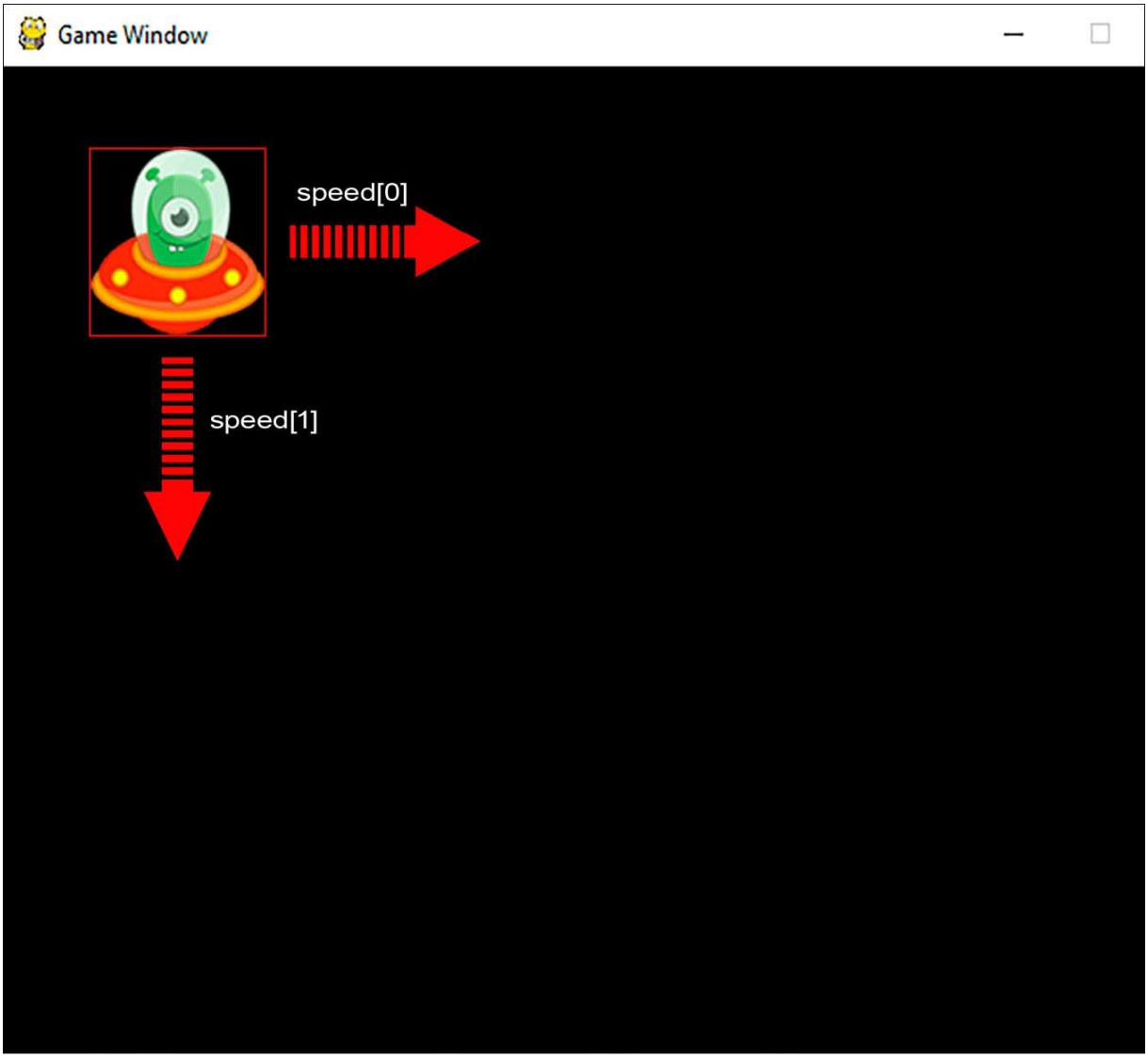
Для перемещения объекта используйте метод `.move_ip()`

```
ufo_rect.move_ip(speed)
```

Давайте рассмотрим программу. Посмотрите на anim02.py

```
#set horizontal by vertical list
speed = [10, 0]

#load ufo image
ufo = pygame.image.load('ufo.png')

#assign image to rectangle so we can manipulate its position
ufo_rect = ufo.get_rect()


#game loop
while running:

    #execute loop at 25 frames per second
    clock.tick(25)

    # move ufo
    ufo_rect.move_ip(speed)
```

Теперь, когда вы запустите эту программу, НЛО будет летать по экрану слева направо.



Поскольку мы устанавливаем `speed = [10, 0]`, это означает, что мы перемещаем наш НЛО на 10 пикселей вдоль оси X каждый раз, когда выполняем `ufo_rect.move_ip(speed)` в игровом цикле.

Если мы установим `speed = [0, 10]`, это означает, что мы перемещаем наш НЛО на 10 пикселей вдоль оси Y каждый раз, когда выполняем `ufo_rect.move_ip(speed)` в игровом цикле.

Попробуйте изменить значения в программе anim02.py и посмотрите, что произойдет.

```
speed = [??, ??]
```

Попробуйте бОльшие значения.

Давайте продвинем нашу программу еще дальше. Давайте заставим НЛО прыгать по экрану.

Для этого нам нужно проверить, выходит ли левый край, правый край, верхний край и нижний край ufo_rect за края экрана. Мы можем использовать для этого операторы if.

Если левый край НЛО заходит за левый край x меняется на обратное направление

Чтобы изменить направление, все, что вам нужно сделать, это изменить speed[0] на отрицательное число.

```
if ufo_rect.left < 0:
speed[0] = -speed[0]
```

Если правый край НЛО пересекает правый край, меняется направление x

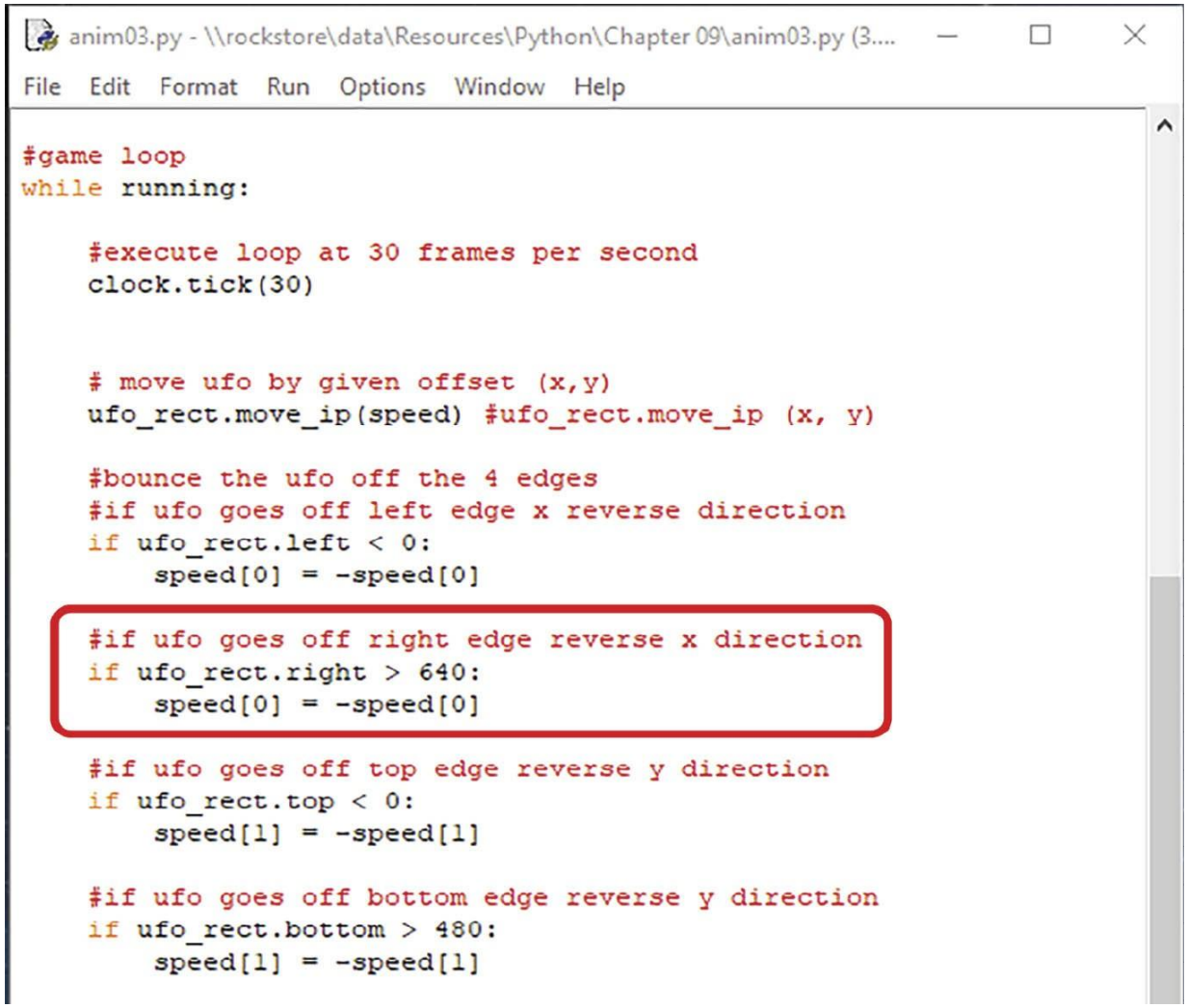
Снова speed[0] изменяется на отрицательное число.

```
if ufo_rect.right >
640: speed[0] = -
```

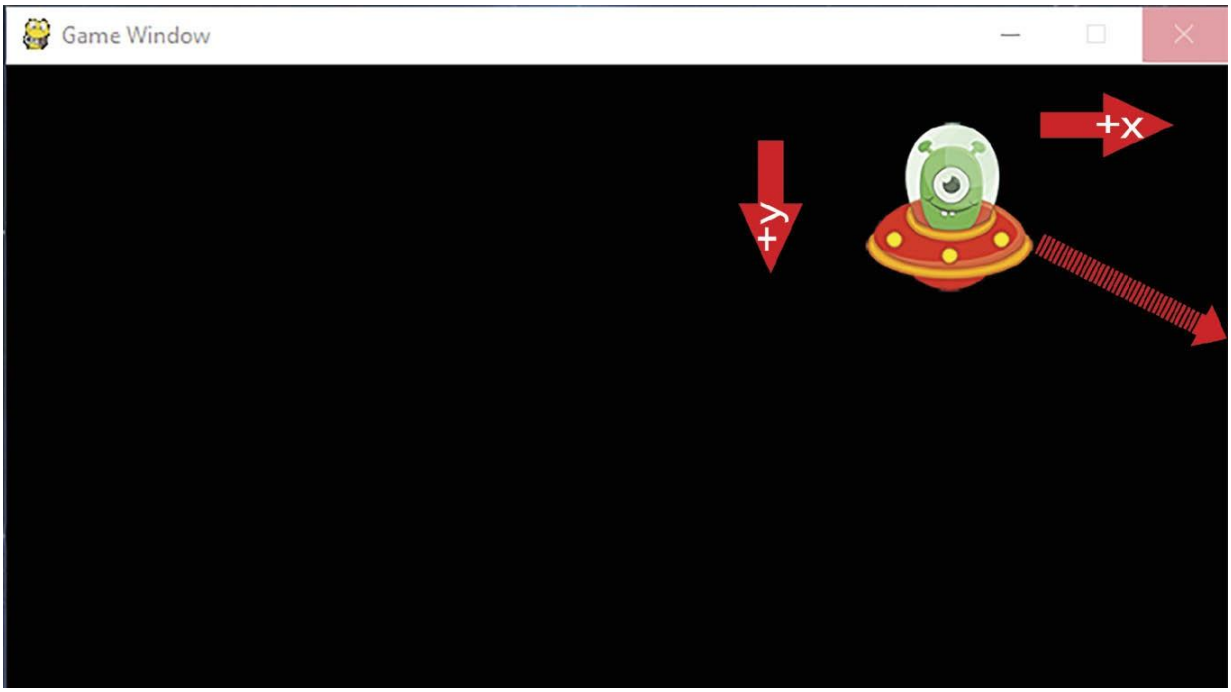
speed[0]

То же самое проделайте с верхом и низом. Попробуйте.

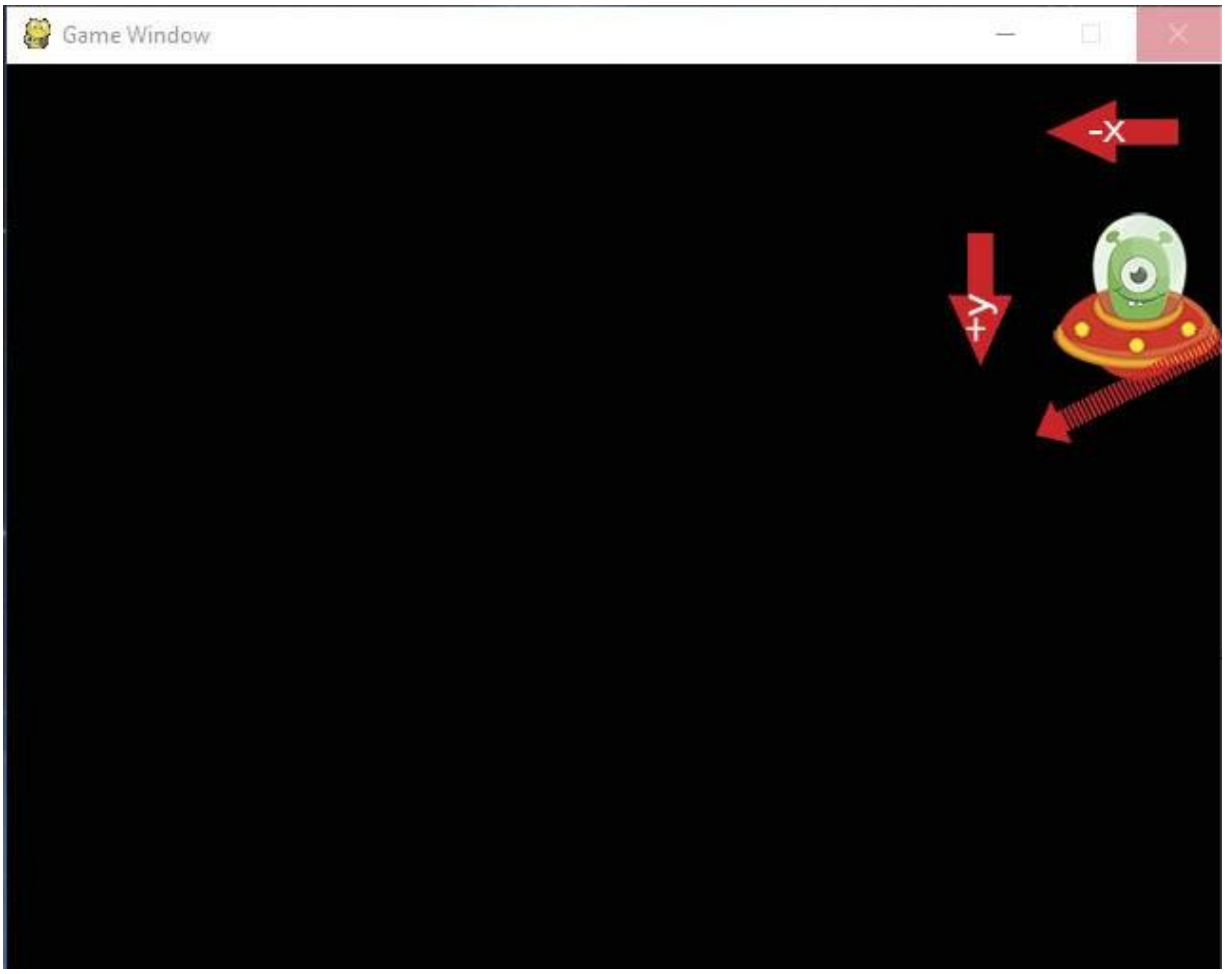
Давайте рассмотрим программу. Откройте файл anim03.py. Здесь вы увидите, как НЛО прыгает по экрану.



Когда НЛО движется к правой границе, увеличивается x (speed[0]) и (speed[1]).



Когда НЛО врезается в правую границу, мы меняем направление x (speed[0]), но не y (speed[1]).



Теперь x (speed[0]) уменьшается, но y (speed[1]) все еще увеличивается. Аналогично для остальных трех направлений.
Что произойдет, если мы изменим переменные `speed[ ]`?

```
speed = [??, ??]
```

Как бы мы добавили еще один НЛО?

Как бы мы добавили нашу ракету из предыдущего раздела?

To animate a character, you need to load your frames into a list.

```
frames = [pygame.image.load('frame1.png'),
pygame.image.load('frame2.png'),
pygame.image.load('frame3.png')]
```

Теперь внутри основного игрового цикла вы можете нарисовать кадр, используя метод `lit()`, чтобы нарисовать кадр из списка кадров.

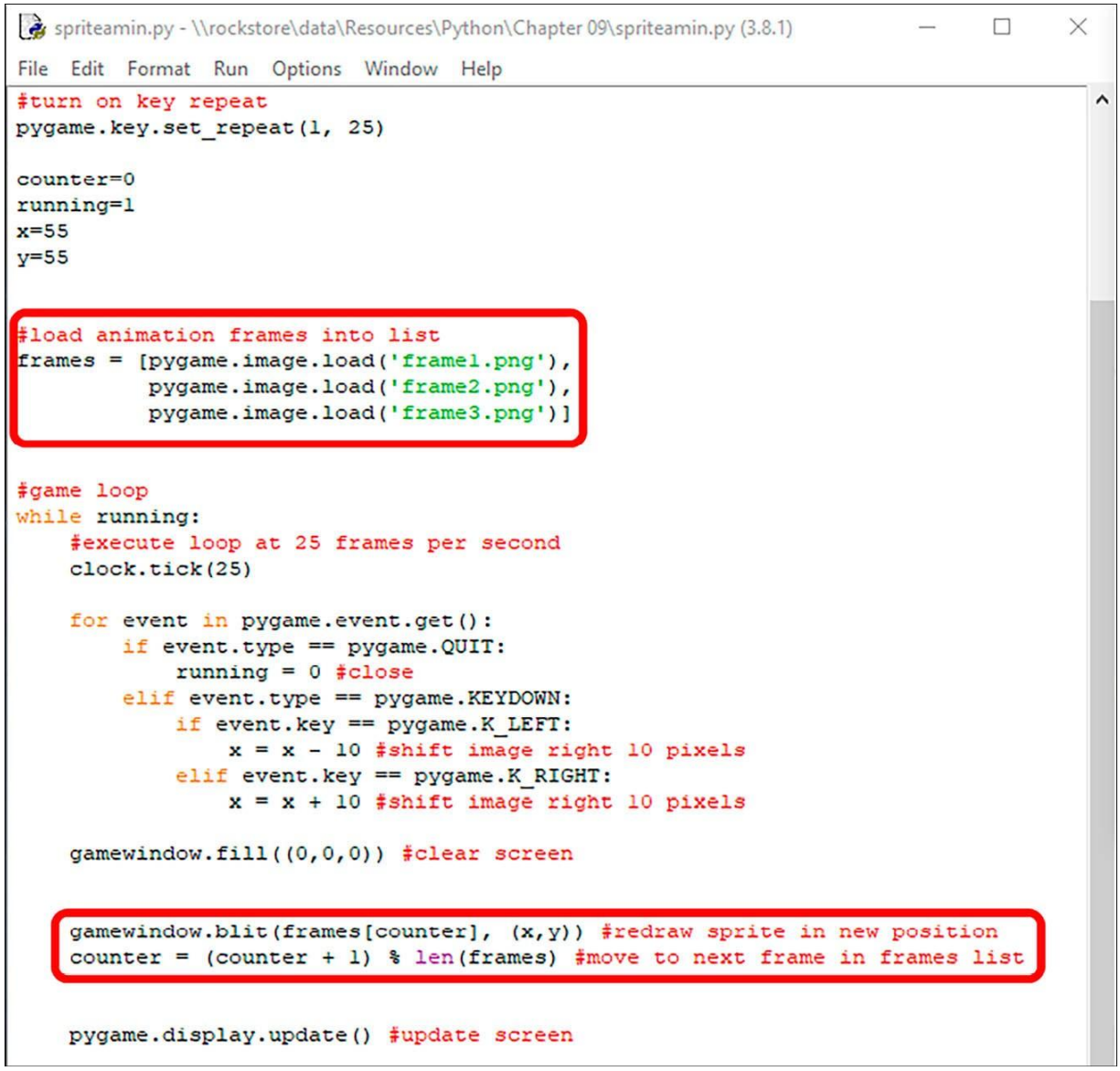
```
gamewindow.blit(frames[counter], (x,y))
```

Выберите следующий кадр из списка и вернитесь к первому кадру в

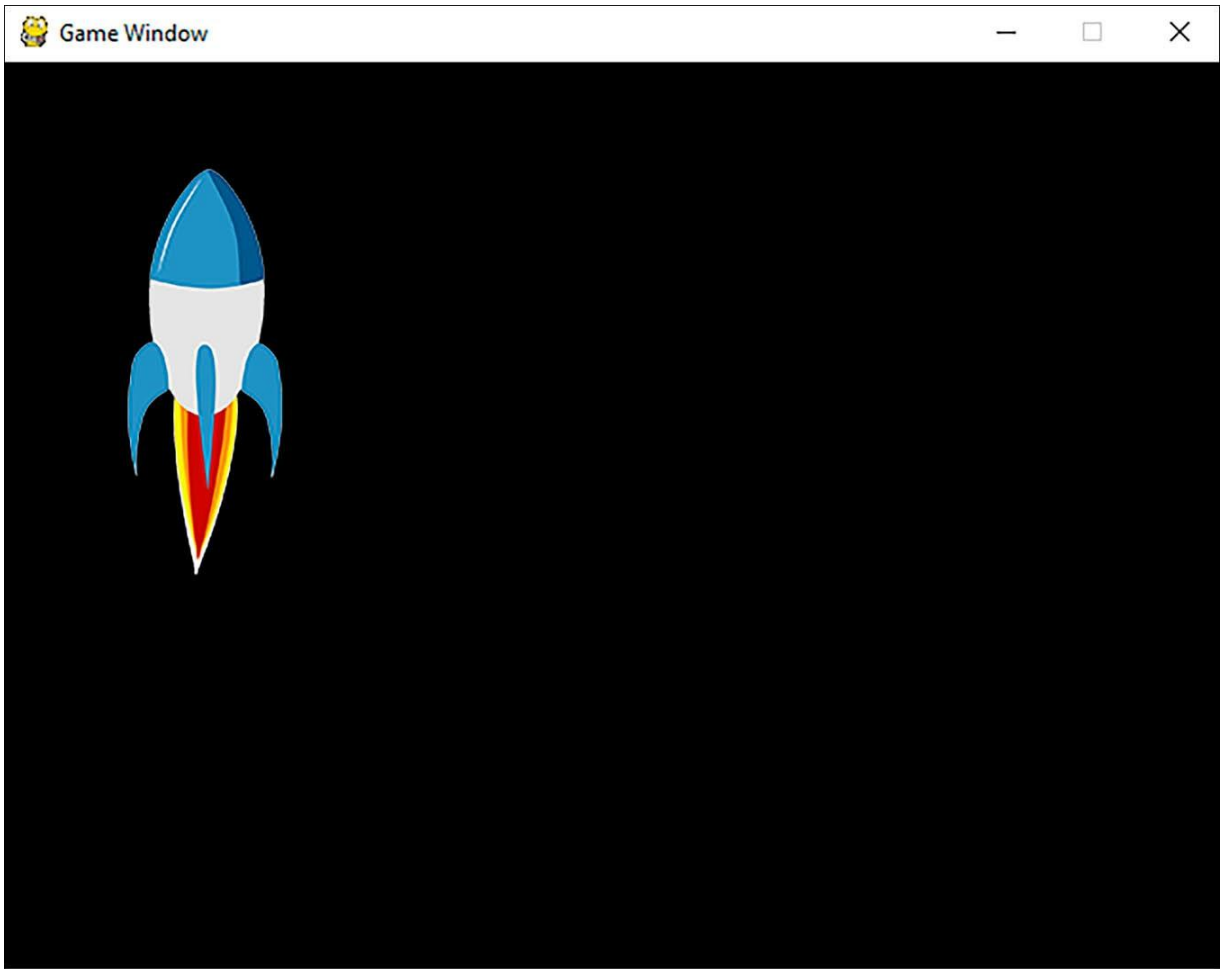
конце списка. Мы можем сделать это с помощью функции `len()`, чтобы вернуть количество кадров в списке и модуль деления.

```
counter = (counter + 1) % len(frames)
```

Давайте рассмотрим программу. Откройте spriteanim.py



В этой конкретной анимации у нас есть три кадра для анимации эффекта пламени ракеты.


Game Window

Собираем все это вместе

Теперь, когда мы изучили некоторые основы использования PyGame, давайте рассмотрим, как мы можем объединить все эти навыки и создать простую игру. В этой конкретной версии мы не будем использовать какие-либо методы объектно-ориентированного программирования, мы просто будем использовать процедуры.

Взгляните на файл Final.py

Чтобы завершить игру, нам нужно добавить несколько пуль для ракеты, которая будет стрелять по НЛО. Мы можем использовать изображение пули PNG.

```
bulletImage=pygame.image.load('bullet.png')
```

Нам также нужны некоторые переменные для положения пули на экране

```
bullet_X = 0

bullet_Y = y #the y position of the rocket
```

На сколько пикселей перемещать пулю за один раз

```
bullet_Xchange = 0

bullet_Ychange = 5
```

Также необходимо условие того, выпущена ли пуля или нет

```
bullet_state = "nofire"
```

Далее нам нужно обновить обработчик событий, чтобы мы могли стрелять с помощью пробела. Мы перемещаем положение изображения пули в то же положение, что и ракета, плюс примерно 30 пикселей, чтобы выглядело так, будто оно выходит из передней части ракеты. Нарисуйте пулю на экране и установите состояние пули на “fire”. Так программа узнает, выпущена пуля или нет.

```
for event in pygame.event.get():
if event.type == pygame.QUIT:
running = 0
elif event.type == pygame.KEYDOWN:
if event.key == pygame.K_LEFT:
x = x - 10 pixels
elif event.key == pygame.K_RIGHT:
x = x + 10 pixels
elif event.key == pygame.K_SPACE:
if bullet_state == “nofire”:
bullet_X = x + 30
gamewindow.blit(bulletImage,
(bullet_X, bullet_Y))
bullet_state = “fire”
```

Далее нам нужно перемещать пулю вверх, пока она не выйдет за верхнюю границу экрана, которая равна 0 по оси Y

```
if bullet_Y <= 0:
bullet_Y = y
bullet_state = “nofire”
if bullet_state == “fire”:
gamewindow.blit(bulletImage,
(bullet_X,
bullet_Y))
```

```
bullet_state = “fire”
bullet_Y -= bullet_Ychange
```

А что происходит, когда пуля попадает в НЛО? Это называется столкновением. Во-первых, нам нужно добавить несколько переменных, которые будут содержать фактическое положение НЛО. Здесь мы взяли центр прямоугольника, содержащего изображение НЛО.

```
ufo_X_pos = ufo_rect.centerx
ufo_Y_pos = ufo_rect.centery
```

Нам нужно определить функцию. Мы берем координаты x и y НЛО, а также координаты x и y пули и используем формулу для расчета расстояния между ними.

$$distance = \sqrt{(\text{ufo\_X\_pos} - \text{bullet\_X})^2 + (\text{ufo\_Y\_pos} - \text{bullet\_Y})^2}$$

Если расстояние меньше 35 пикселей, мы считаем, что произошло попадание, и возвращаем true.

```
def isCollision(ufo_X_pos, ufo_Y_pos, bullet_X, bullet_Y):
distance = math.sqrt(math.pow(ufo_X_pos - bullet_X, 2) +
(math.pow(ufo_Y_pos - bullet_Y, 2)))
if distance < 35:
return True
else:
```

```
return False
```

После того, как мы определили функцию обнаружения столкновений, мы можем вызвать эту функцию, чтобы проверить расстояние между пулей и НЛО.

```
collision = isCollision(ufo_X_pos, ufo_Y_pos, bullet_X, bullet_Y)
```

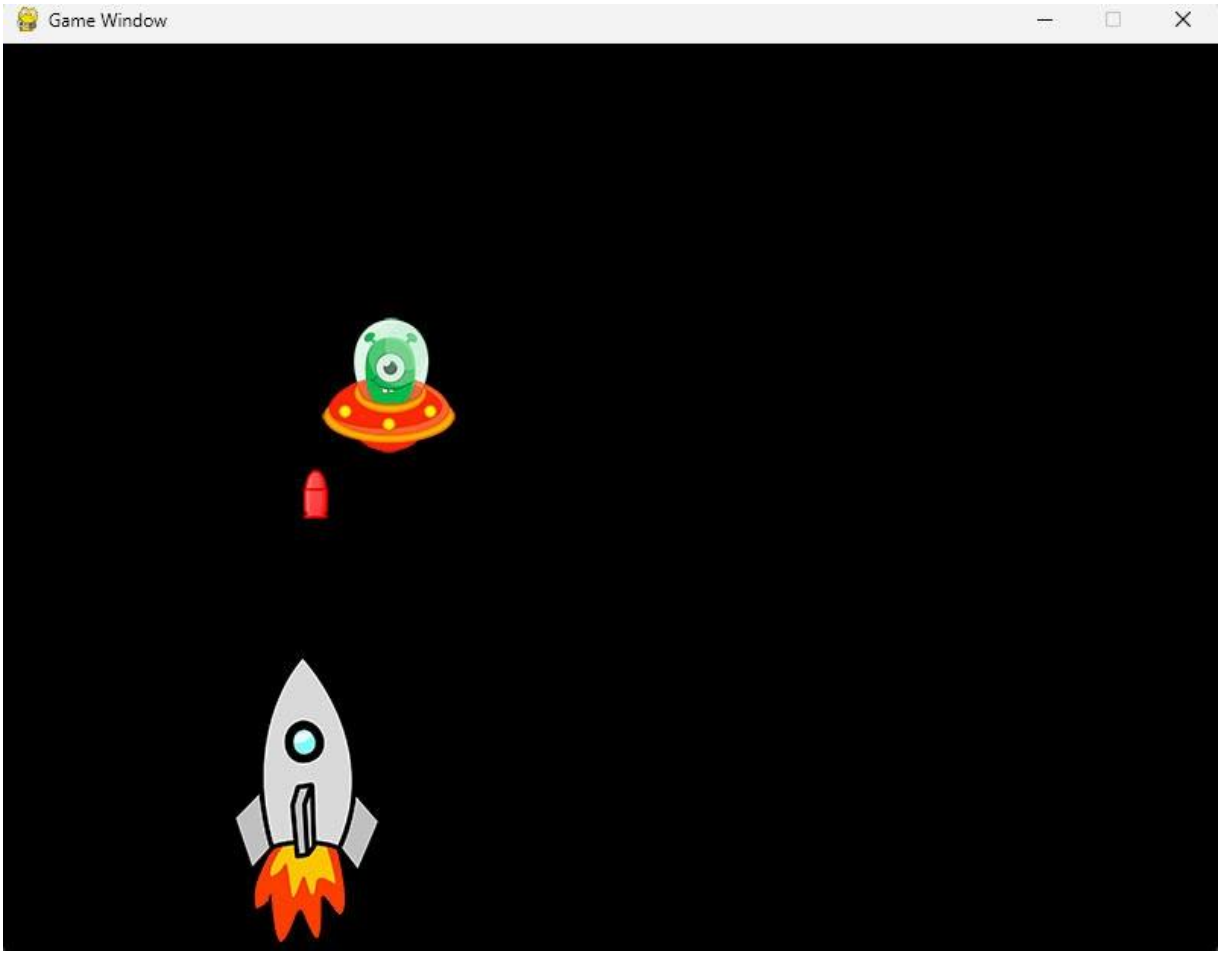
Как только НЛО поражен, мы можем сбросить положение пули, а затем в месте положения пули отобразить изображение взрыва, чтобы показать игроку, что НЛО был поражен

```
if collision:
bullet_Y = y
bullet_state = “nofire”
gamewindow.blit(exp, (ufo_X_pos, ufo_Y_pos))
```

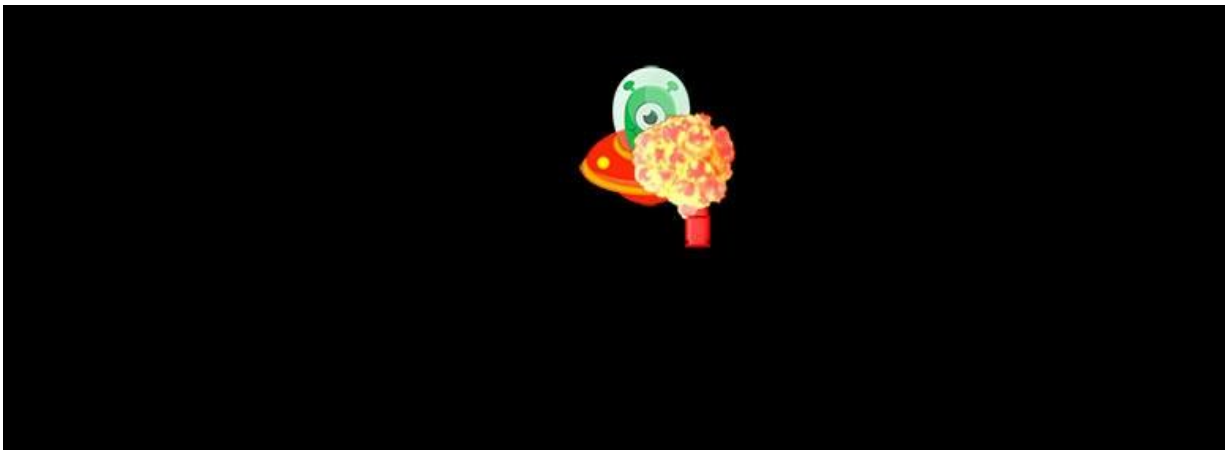
Как только НЛО будет сбит, мы можем перерисовать НЛО в случайном месте на экране.

```
ufo_rect.centerx = random.randint(0, 700)
ufo_rect.centery = random.randint(0, 500)
```

Выполнив программу, вы сможете перемещать ракету влево и вправо, а также стрелять пулями.



НЛО будет прыгать по экрану. Когда вы попадете в НЛО пулей, вы увидите взрыв, и игра перезапустится, разместив НЛО в случайном месте на экране.



Лабораторная работа

1. В коде есть несколько ошибок, попробуйте и посмотрите, сможете ли вы их найти и улучшить программу.
2. Как бы вы улучшили программу?
3. Не могли бы вы добавить систему подсчета очков? Посмотрите text.py. Какой текст вам понадобится (укажите шрифт и размер):

   ```
   gameFont = pygame.font.Font('fontfilename', 32)
   ```

   Чтобы указать текст для отображения и цвет имени в кавычках:

   ```
   gameText = gameFont.render('text…', True, 'color')
   ```

   Чтобы отобразить текст в окне по координатам x,y:

   ```
   gamewindow.blit(gameText,(x,y))
   ```

4. Не могли бы вы добавить еще один НЛО?
5. А как насчет добавления фонового изображения?
6. Попробуйте добавить звуковые эффекты. Изучите audio.py. Вам нужно будет загрузить звуки:

   ```
   bulletSound = pygame.mixer.Sound('bullet.wav')
   hitSound = pygame.mixer.Sound('hit.wav')
   ```

   Чтобы воспроизвести звук:

   ```
   bulletSound.play()
   hitSound.play()
   ```

Мини-проект

Большую часть времени подобные более крупные программы будут написаны с использованием объектно-ориентированного программирования. Давайте рассмотрим объектно-ориентированную версию. Перейти на следующий сайт

`elluminetpress.com/python`

Загрузите и разархивируйте проект My Invaders CH12 OOP. Посмотрите My Invaders Notes.pdf вместе с файлами кода.

Веб-разработка на Python

Python широко используется для разработки крупномасштабных веб-приложений, которые невозможно создать с использованием .NET и PHP.

Python поддерживает функции, которые выполняются с помощью различных платформ, таких как Django, Flask, Pyramid и CherryPy, которые обычно используются на таких сайтах, как Spotify и Mozilla.

Веб-серверы

Большинство веб-приложений Python выполняются на веб-сервере через интерфейс WSGI (интерфейс шлюза веб-сервера). Другие сценарии Python выполняются через CGI (интерфейс общего шлюза).

Здесь мы установили интерпретатор Python и включили модуль адаптера WSGI для веб-сервера Apache.

Для этого раздела вам понадобится доступ к веб-серверу с поддержкой Python.

Загрузите свои сценарии в каталог public_html на сервере, затем на своем компьютере откройте веб-браузер и введите URL-адрес своего сценария.

```
http://server-name/script-name.py
```

Например:

```
http://titan/script.py
```

Чтобы начать писать сценарии Python, вам нужно сообщить веб-серверу, где найти интерпретатор Python. Обычно это:

```
#!/python/python
```

или на Linux-сервере:

```
#!/usr/bin/python
```

Это первая строка вашего скрипта

Установка веб-сервера

В Aprelium доступен бесплатный веб-сервер под названием Abyss Web Server X1, который вы можете установить на свой компьютер для разработки и тестирования веб-сайтов

```
aprelium.com/abyssws/download.php
```

Выберите свою версию: Windows, Mac или Linux.

Откройте каталог, в котором вы сохранили пакет программного обеспечения. Дважды щелкните значок программного обеспечения (abwsx1.exe) и следуйте инструкциям на экране.

Отмените выбор компонентов, которые вы не хотите устанавливать. Auto Start включает автоматический запуск веб-сервера Abyss при запуске сеанса Windows — снимите этот флажок. Ярлыки меню «Пуск» позволяют добавлять ярлыки веб-сервера Abyss в меню «Пуск». Документация устанавливает файлы справки. Нажмите Next.

Выберите каталог, в который вы хотите установить файлы веб-сервера Abyss. Отныне <Abyss Web Server directory> будет ссылаться на этот каталог.

```
c:\Abyss Web Server
```

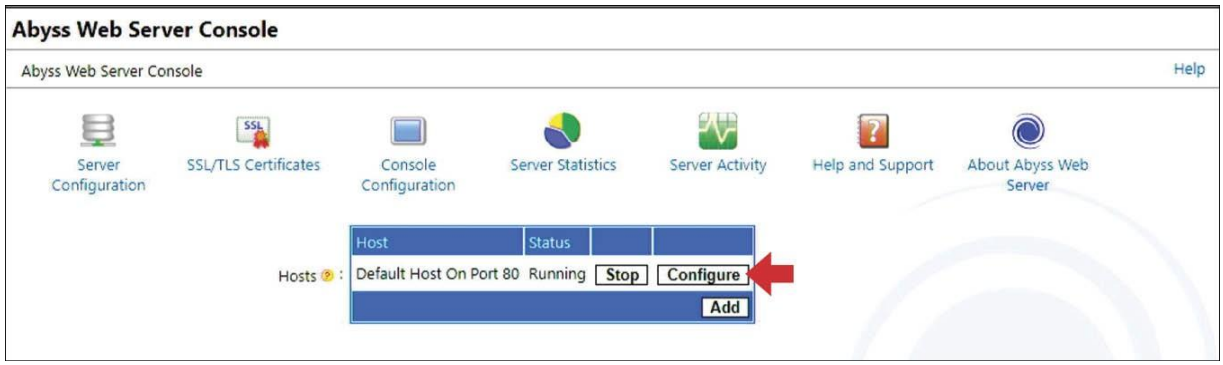
Нажмите Install.

Настройка поддержки Python

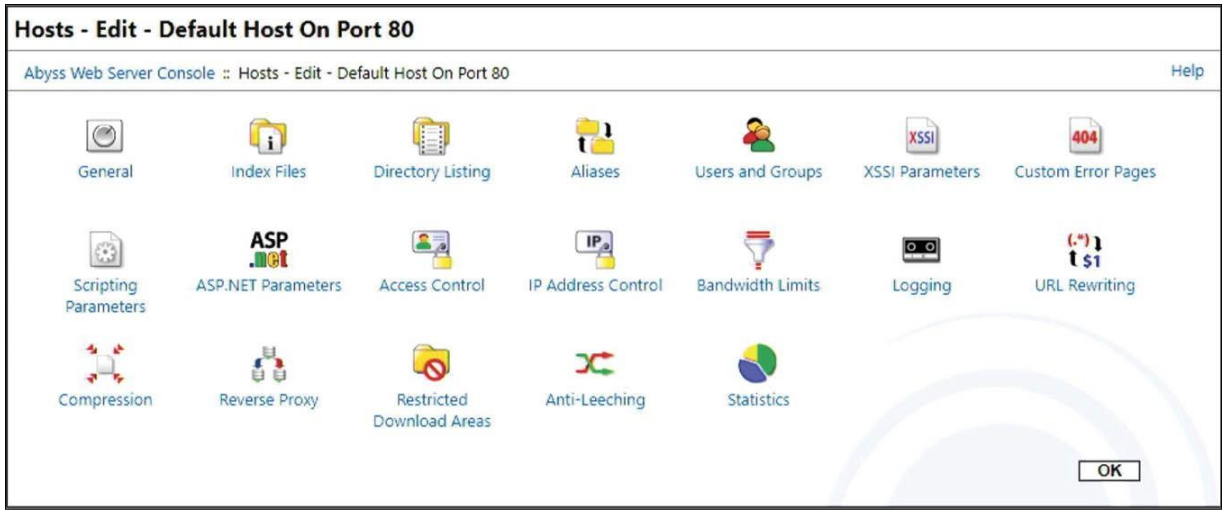
Запустите веб-сервер (выберите Abyss Webserver X1 в меню «Пуск»), затем откройте консоль Abyss Web Server. Введите следующее в адресной строке своего веб-браузера

```
127.0.0.1:9999
```

В таблице Hosts нажмите Configure в строке, соответствующей хосту, к которому вы хотите добавить поддержку Python.

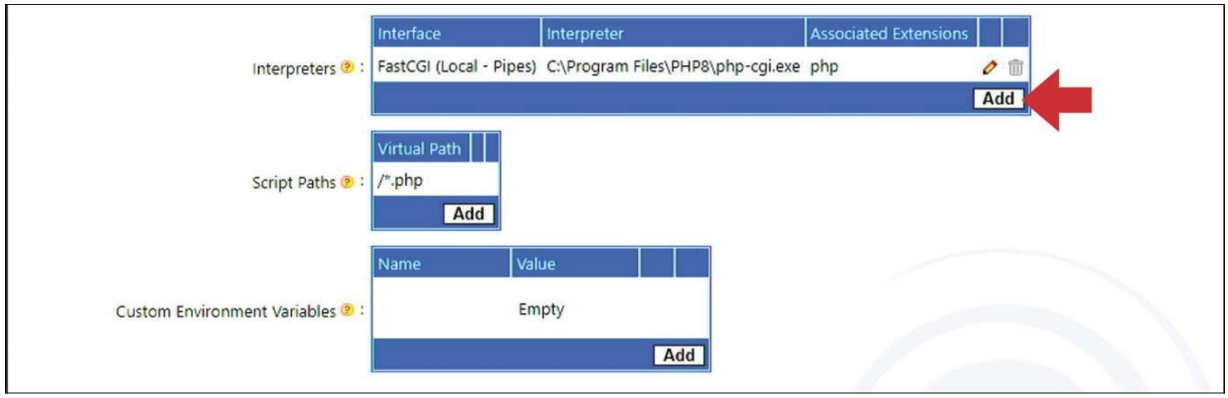


Выберите параметры сценария:

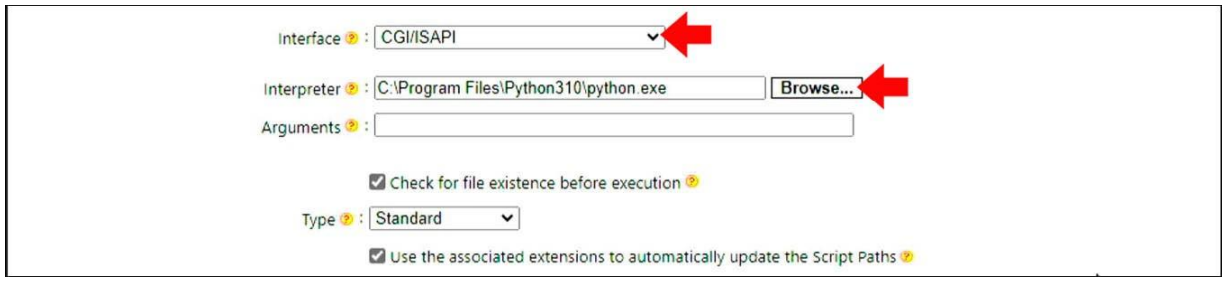


Установите флажок Enable Scripts Execution.

Нажмите Add в таблице Interpreters.



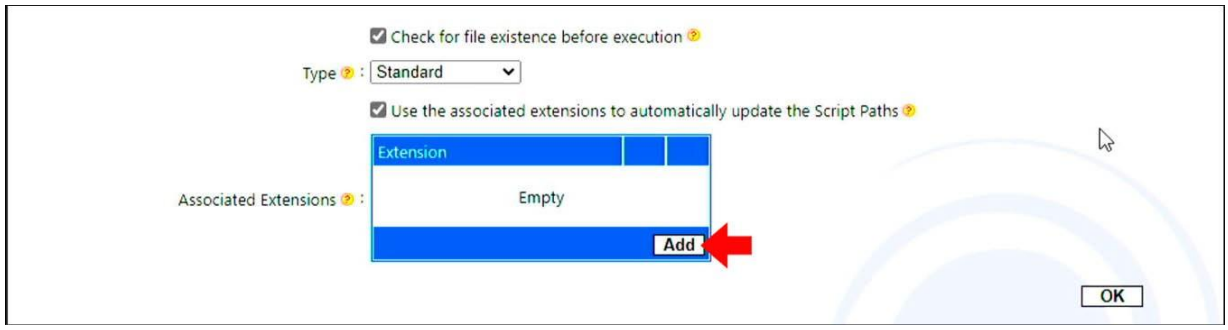
Выберите интерфейс CGI/ISAPI.



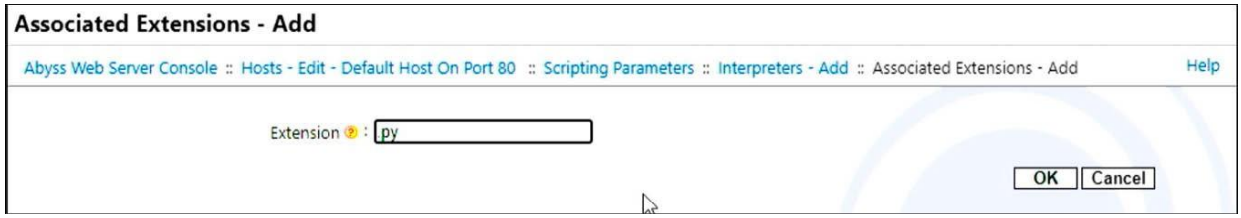
В поле Interpreter нажмите Browse... Перейдите в каталог, в который вы установили Python, и нажмите python.exe.

Установите флажок Use the associated extensions to automatically update the Script Paths.

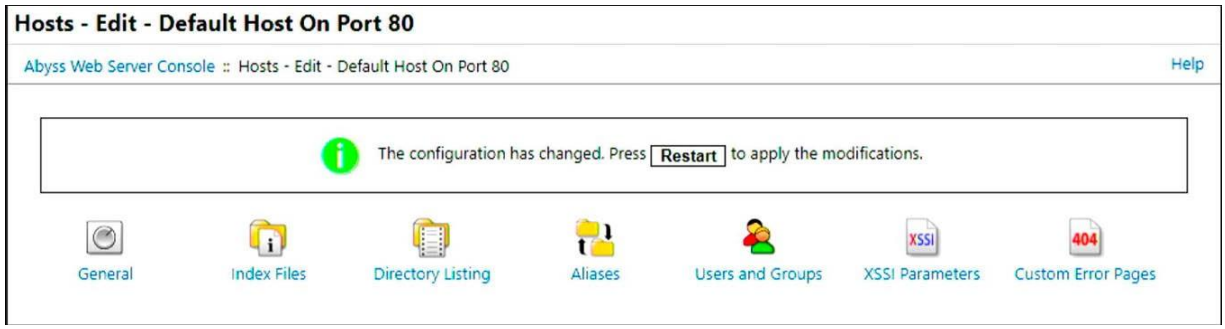
Нажмите Add в таблице Associated Extensions.



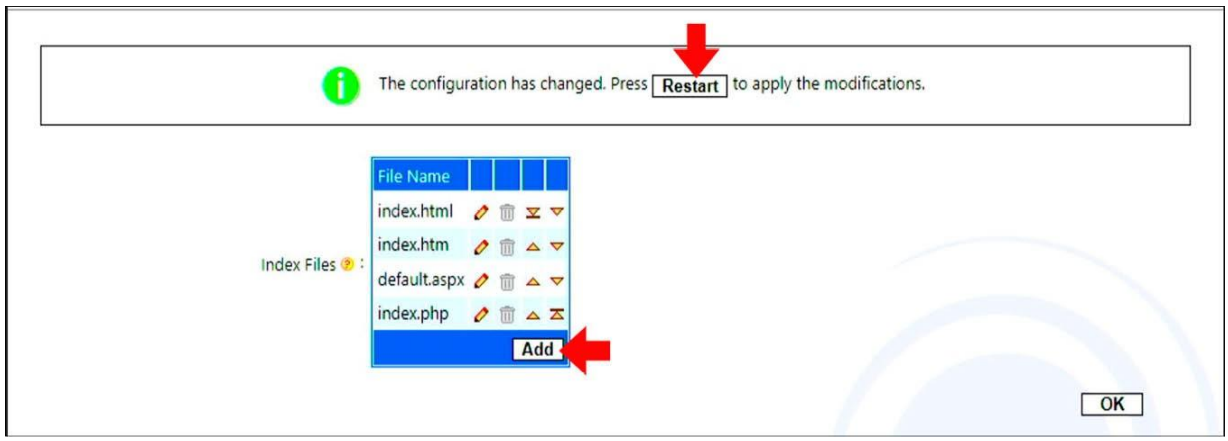
Введите py в поле Extension и нажмите OK. Затем снова нажмите OK.



Нажмите OK в диалоговом окне Scripting Parameters, чтобы вернуться на экран хостов. Выберите Index Files.



Нажмите Add в таблице Index Files.

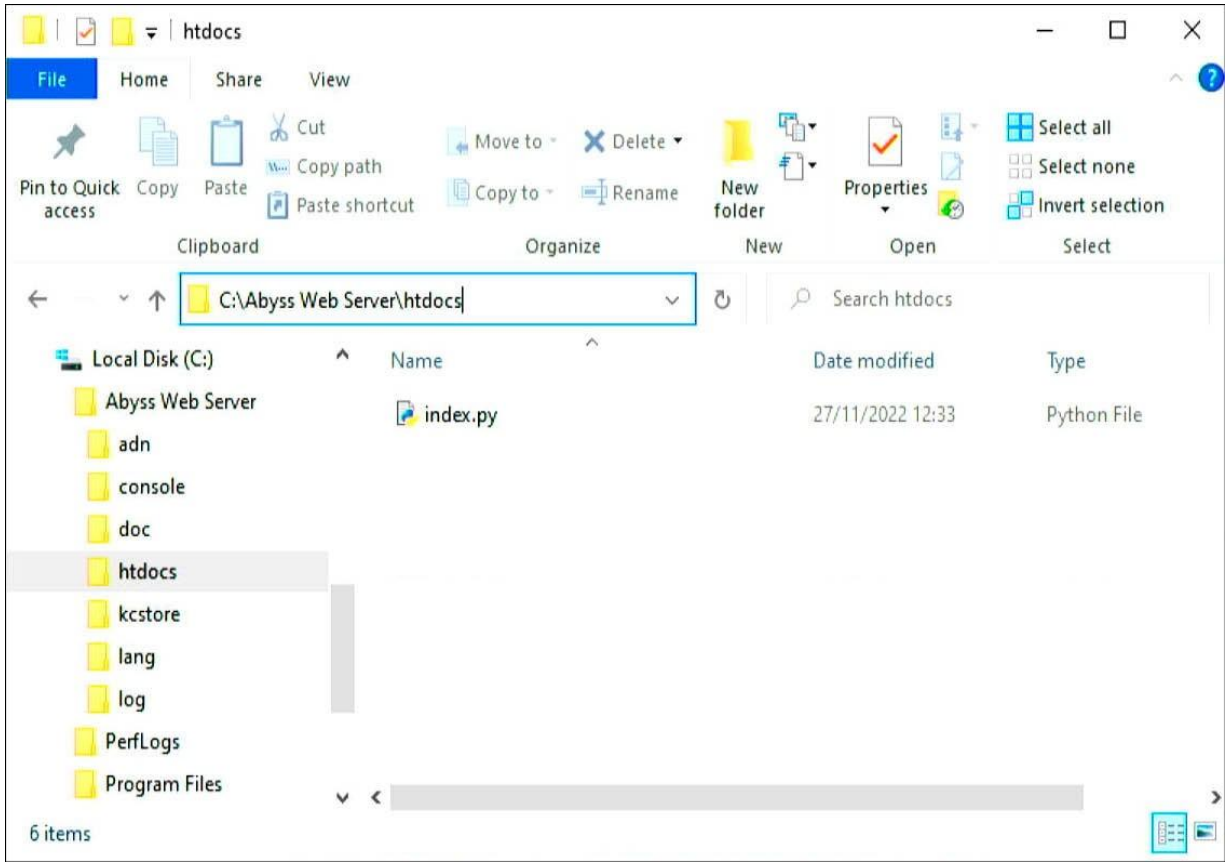


Введите index.py в поле File Name и нажмите OK.

Нажмите Restart вверху экрана, чтобы перезагрузить сервер.

Где сохранять скрипты Python

Загрузите свои сценарии в каталог public_html или htdocs на сервере. Если вы используете Aprelium, это будет htdocs.



На веб-сервере Aprelium это будет

```
c:\Abyss Web Server\htdocs
```

Выполнение сценария

Чтобы начать писать сценарии Python, вам нужно сообщить веб-серверу, где найти интерпретатор Python. Это первая строка вашего скрипта.

В среде Windows/Mac используйте

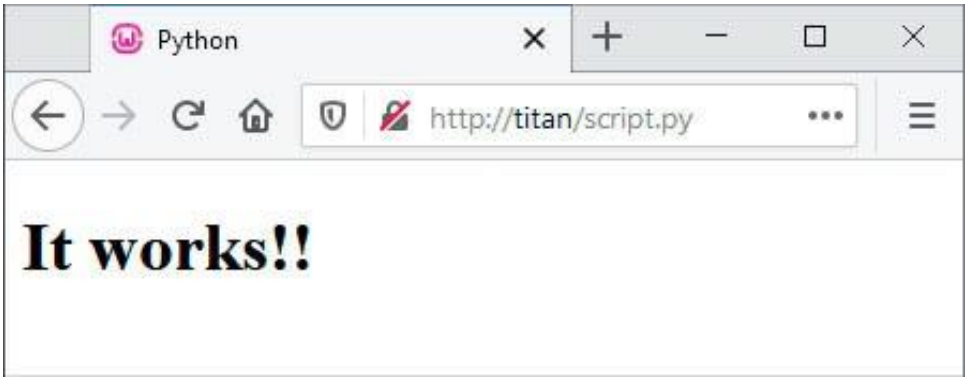
```
#!/python/python
```

Или в среде сервера Linux:

```
#!/usr/bin/python
```

Давайте рассмотрим пример. Взгляните на script.py. Здесь мы написали скрипт для вывода простой HTML-страницы.

На этой странице просто выводится заголовок ‘It works!!’. Загрузите сценарий в каталог public_html на своем веб-сервере, затем перейдите по URL-адресу сценария с помощью веб-браузера на своем компьютере.



В URL-адресе нашей лаборатории это будет:

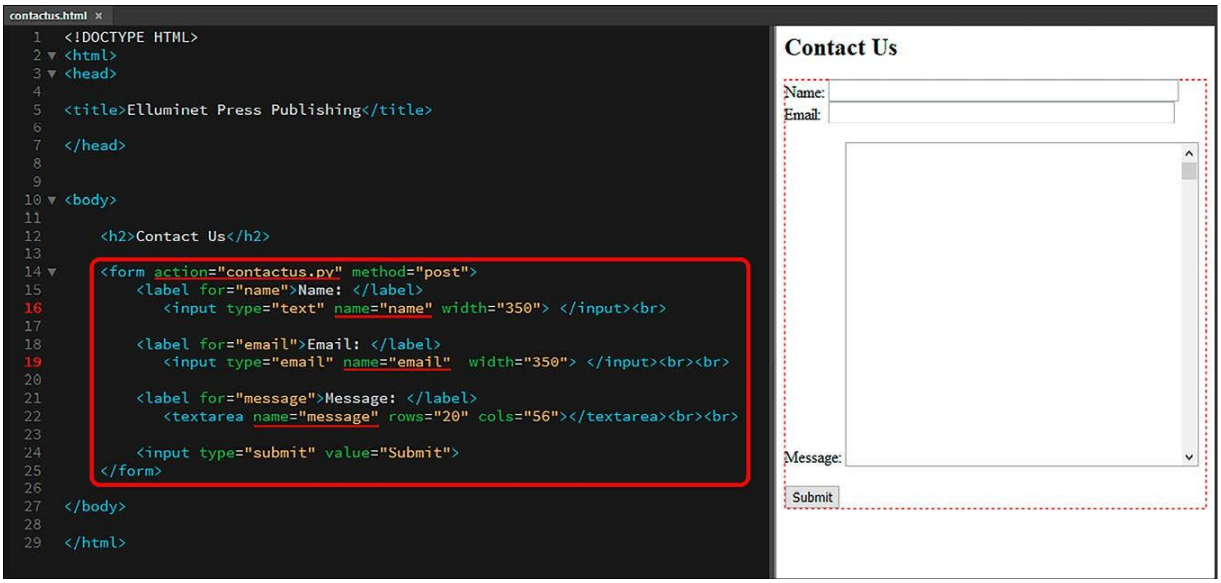
```
http://titan/scriptfilename.py
```

Если вы используете персональный сервер Aprelium на своем компьютере, вы можете использовать

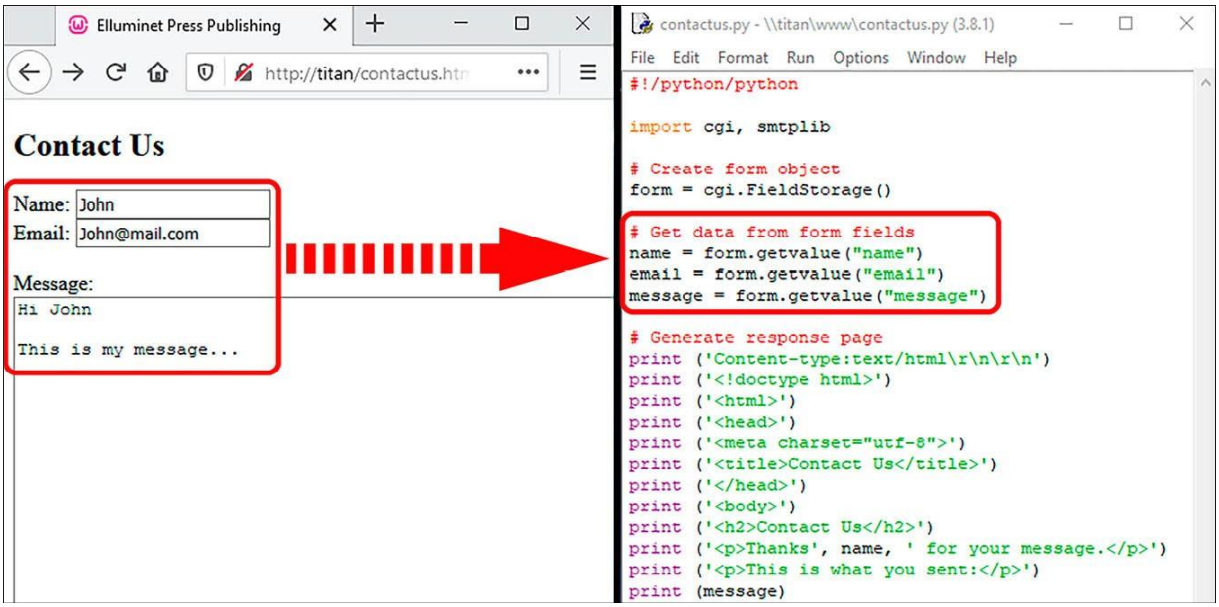
```
http://localhost/scriptfilename.py
```

Замените ‘scriptfilename.py’ именем файла вашего скрипта.

Давайте рассмотрим практический пример. Здесь мы создаем простую контактную форму. Пользователю предоставляется HTML-форма, в которой запрашивается имя, адрес электронной почты и сообщение.



Когда пользователь нажимает кнопку ‘submit’, HTML-форма вызывает скрипт Python под названием contactus.py



Скрипт Python обрабатывает данные, переданные из формы HTML, и сохраняет их в объекте формы.

Затем мы можем получить значения, переданные из формы HTML, и сохранить их в этом объекте.

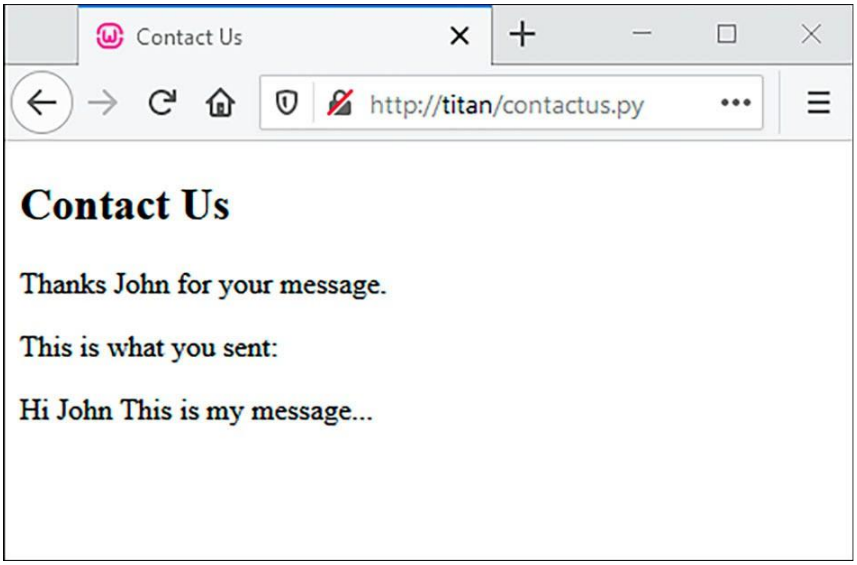
Скрипт Python генерирует еще одну HTML-страницу, используя операторы печати для ответа пользователю.

contactus.py - \\titan\www\contactus.py (3.8.1)

File Edit Format Run Options Window Help

```
#!/python/python

import cgi, smtplib

# Create form object
form = cgi.FieldStorage()

# Get data from form fields
name = form.getvalue("name")
email = form.getvalue("email")
message = form.getvalue("message")

# Generate response page
print ('Content-type:text/html\r\n\r\n')
print ('<!doctype html>')
print ('<html>')
print ('<head>')
print ('<meta charset="utf-8">')
print ('<title>Contact Us</title>')
print ('</head>')
print ('<body>')
print ('<h2>Contact Us</h2>')
print ('<p>Thanks', name, ' for your message.</p>')
print ('<p>This is what you sent:</p>')
print (message)
print ('</body>')
print ('</html>')
```

Ниже вы можете увидеть результат в браузере.

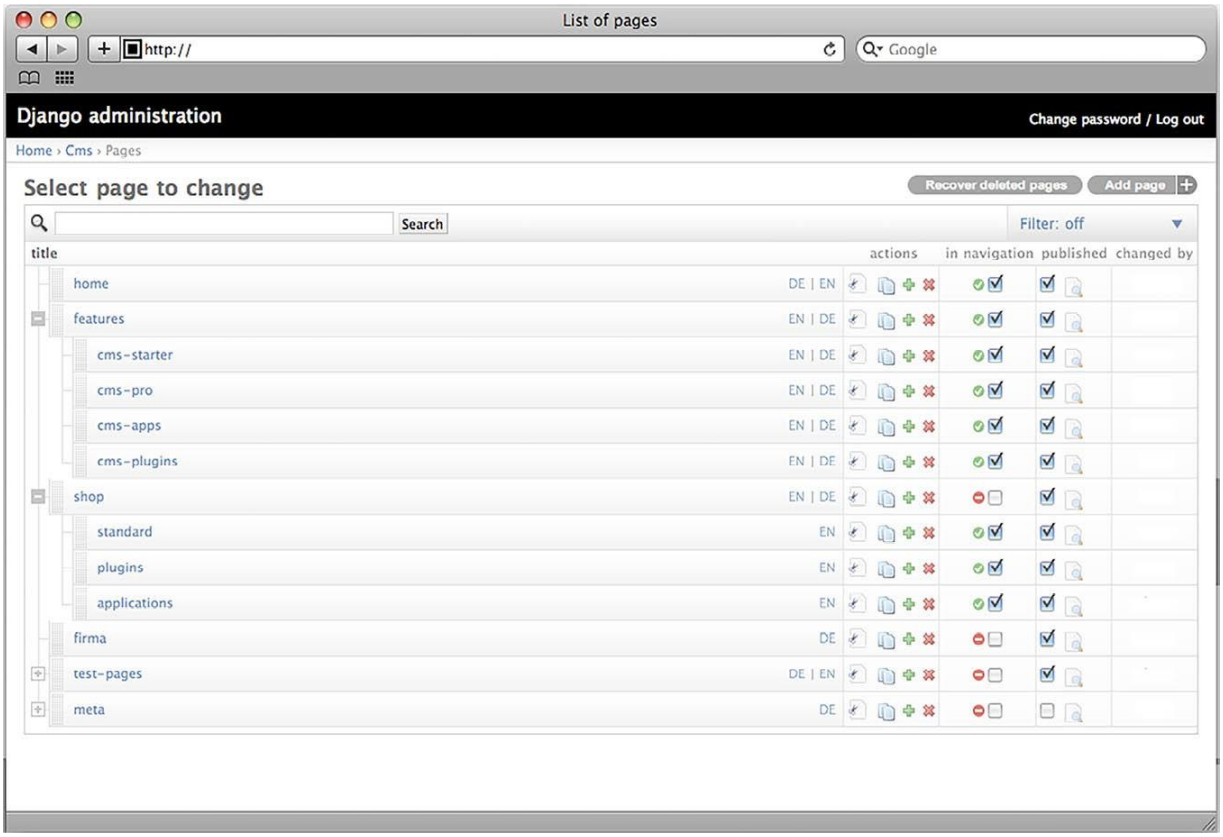


Веб-фреймворки Python

Если вы используете Python в веб-разработке, вы, скорее всего, будете использовать веб-фреймворк Python, а не старый CGI, который мы рассматривали в предыдущем разделе.

Веб-фреймворк Python — это набор инструментов, библиотек и технологий, которые позволяют создавать веб-приложения.

Одним из примеров веб-фреймворка Python является Django (произносится «Джанго»).



Другой пример — Flask. Давайте посмотрим, как создать простое веб-приложение с использованием этой платформы.

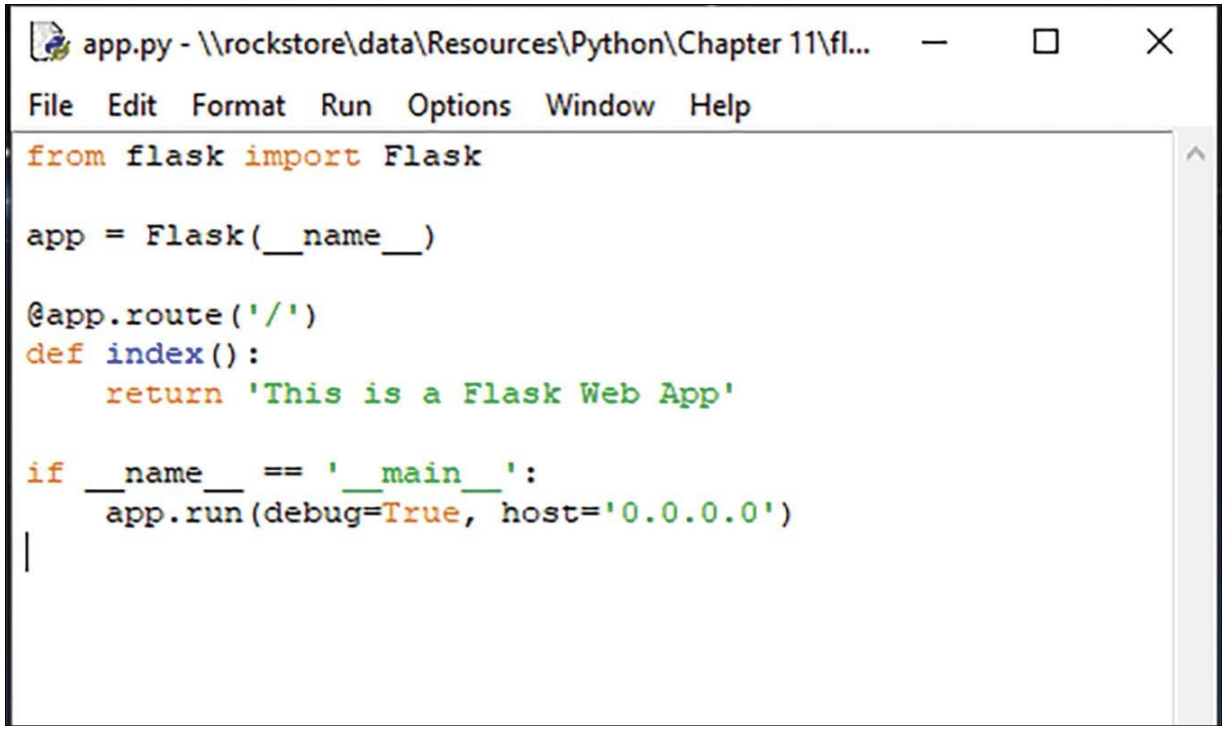
Первое, что вам нужно сделать, это установить модуль Flask, если вы еще этого не сделали. Используйте следующую команду в командной строке администратора

```
pip install Flask
```

Используйте показанную ниже команду, если вы используете компьютер под управлением Linux

```
sudo pip install Flask
```

Давайте создадим приложение. Первое, что нам нужно сделать, это создать нашу основную программу. Для этого мы создаем новый файл с именем app.py. Мы включили все эти файлы в каталог Flask в файлах ресурсов.



Здесь мы импортировали наш модуль Flask.

Современные веб-приложения используют метод маршрутизации. Это означает, что вместо URL-адреса страницы

```
localhost/resources.php
```

...мы используем маршрут

```
localhost/resources/
```

Итак, наш первый маршрут — это корень нашего веб-сайта и обычно это индексная страница. Мы используем `@app.route('/')`, чтобы определить это.

'/' означает корень сайта:

```
http://localhost:5000/
```

`def index()` — это наименование, которое вы даете маршруту,

определенному выше. Это называется индексом, потому что это индекс (или домашняя страница) веб-сайта.

`host='0.0.0.0'` означает, что приложение доступно с любого компьютера в сети.

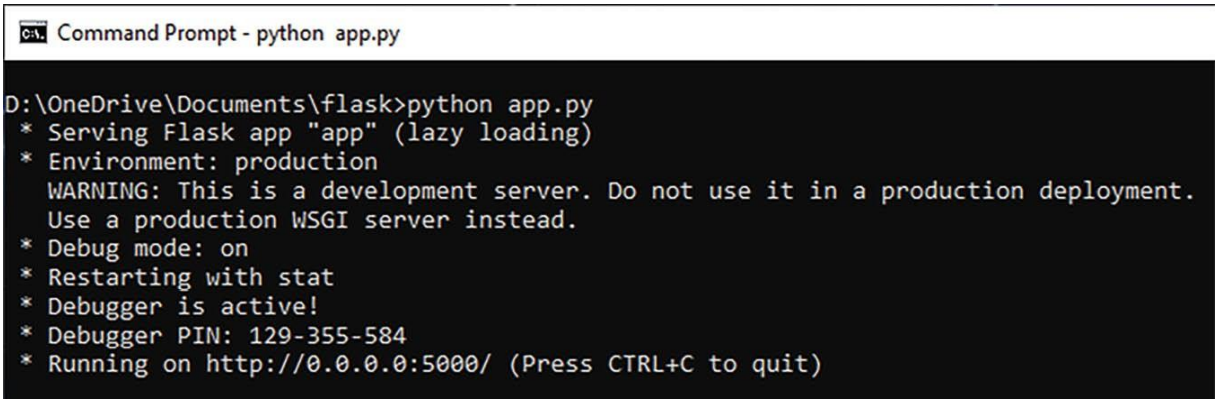
Чтобы запустить и протестировать приложение, нам нужно открыть его с помощью среды разработки. Это простой веб-сервер, который позволяет вам открыть приложение в веб-браузере для тестирования.

Для этого откройте каталог вашего приложения в командной строке. В данном конкретном примере файлы приложения находятся в OneDrive/Documents/Flask

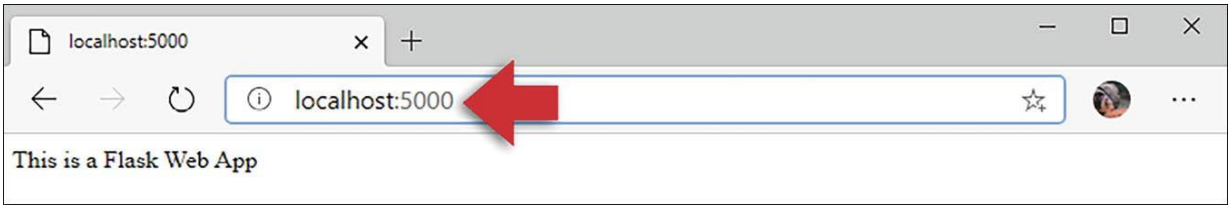
Чтобы запустить приложение, введите

```
python app.py
```

Как только вы нажмете Enter, сервер запустится.



Вы можете открыть приложение в веб-браузере. На вашей рабочей станции вы можете использовать `localhost:5000`



Чтобы добавить еще одну страницу, добавьте еще один маршрут.

```
@app.route('/shop')
def shop():
    return 'This is the shop page'
```

Теперь в веб-браузере вы можете использовать

`localhost:5000/shop`

Теперь, когда у нас есть базовое приложение, мы можем разработать веб-страницы для вызова приложения. Эти веб-страницы называются шаблонами, и мы храним их в каталоге шаблонов. Вот простая HTML-страница, которую мы создали и сохранили в каталоге шаблонов.

index.html - D:\OneDrive\Documents\flask\templates\in...

File Edit Format Run Options Window Help

```
<html>
<head>
<meta charset="utf-8">
<title>Elluminet Press</title>
</head>

<body>

<h1>Welcome to Elluminet Press Ltd</h1>

<p>Elluminet Press, a member of the Independent Book
Publisher's Association, is a publisher with more
than 100 books in print, beautifully bound in either
hardback or paperback, as well as in electronic formats,
covering a variety of genres</p>

</body>
</html>
```

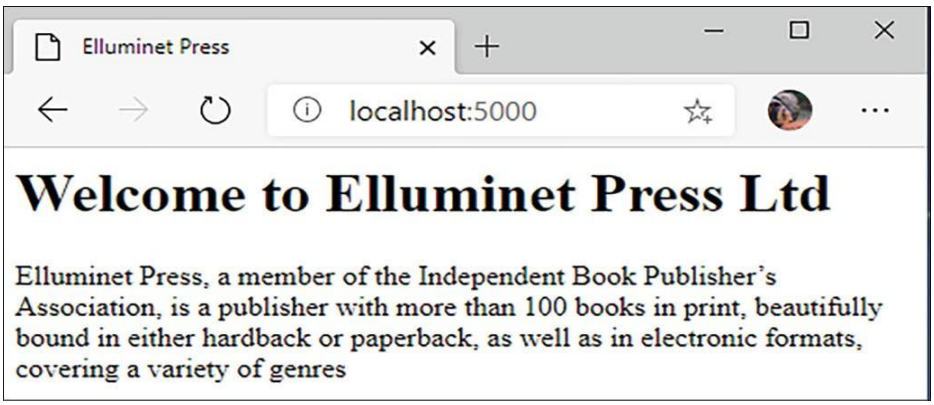
Давайте вызовем нашу HTML-страницу из нашего приложения. Мы можем использовать функцию `render_template()` function.

app02.py - D:\OneDrive\Documents\flask\app02.py (3.8.1)

File Edit Format Run Options Window Help

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')


if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

Когда мы просматриваем страницу в браузере, мы видим визуализированную версию HTML-страницы.

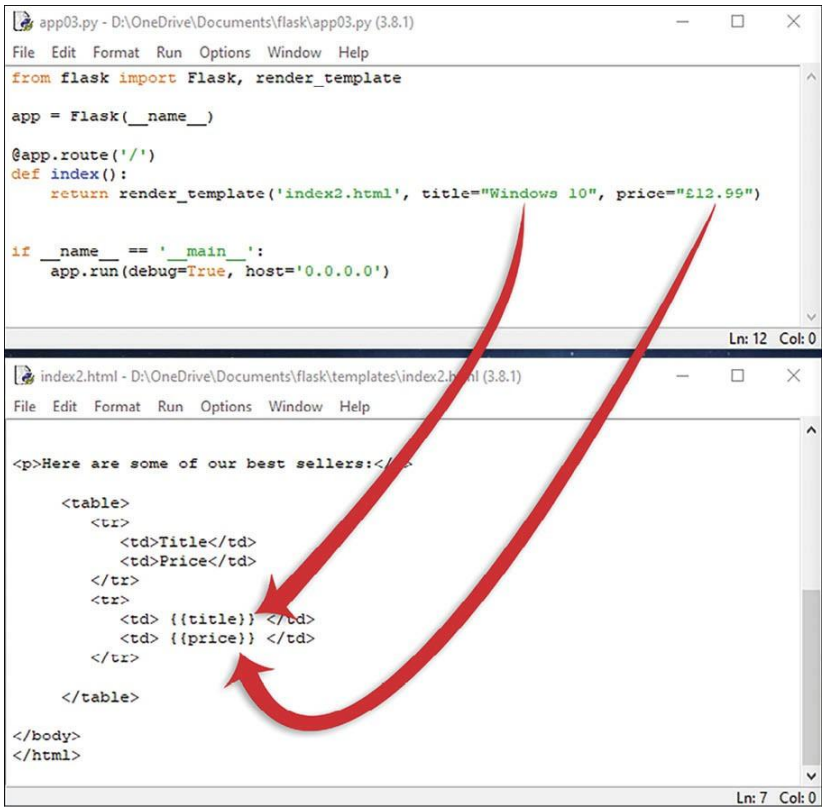


Вы можете передавать переменные в свои HTML-шаблоны. Для этого вставьте переменную в свой HTML, используя `{{variable-name}}`.

Затем добавьте переменную в качестве параметра в функцию `render_template()`.

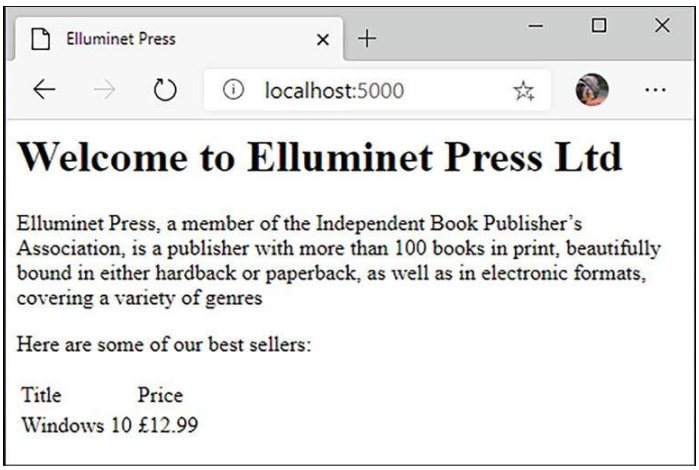
```
render_template('index.html', variable-name = "...")
```

Давайте рассмотрим пример. Откройте файлы app3.py и index2.html



Здесь мы передали заголовок и цену в качестве переменных в HTML-шаблон.

Вы можете добавлять изображения в свои шаблоны, используя код HTML и CSS. Вы также можете встроить код Python.



Ресурсы

Чтобы помочь вам понять процедуры и концепции, рассмотренные в этой книге, мы разработали несколько видеоресурсов и демонстраций приложений, которые вы сможете использовать при работе с книгой.

Чтобы найти ресурсы, откройте веб-браузер и перейдите на следующий веб-сайт

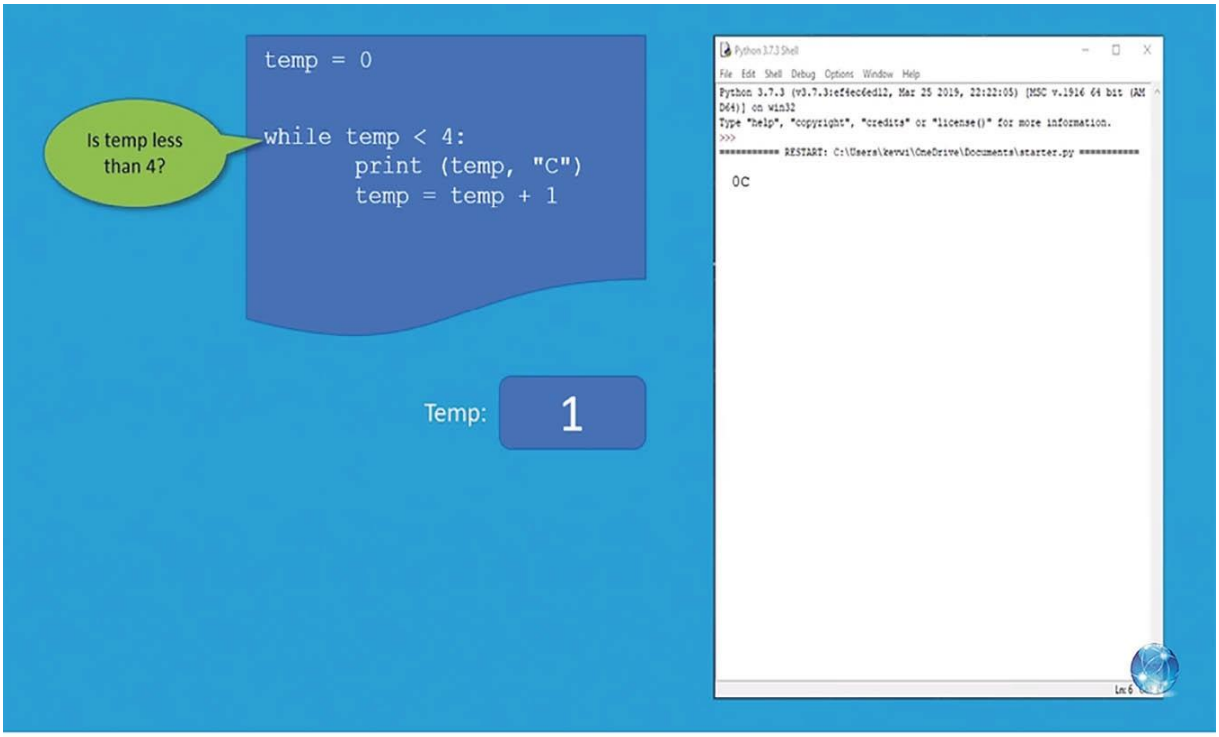
`elluminetpress.com/python`

В начале каждой главы вы найдете ссылку на веб-сайт, содержащий ресурсы для этой главы.

Использование видео

К каждому разделу книги вы найдете видеоресурсы. Просто нажмите на ссылку, чтобы открыть разделы.

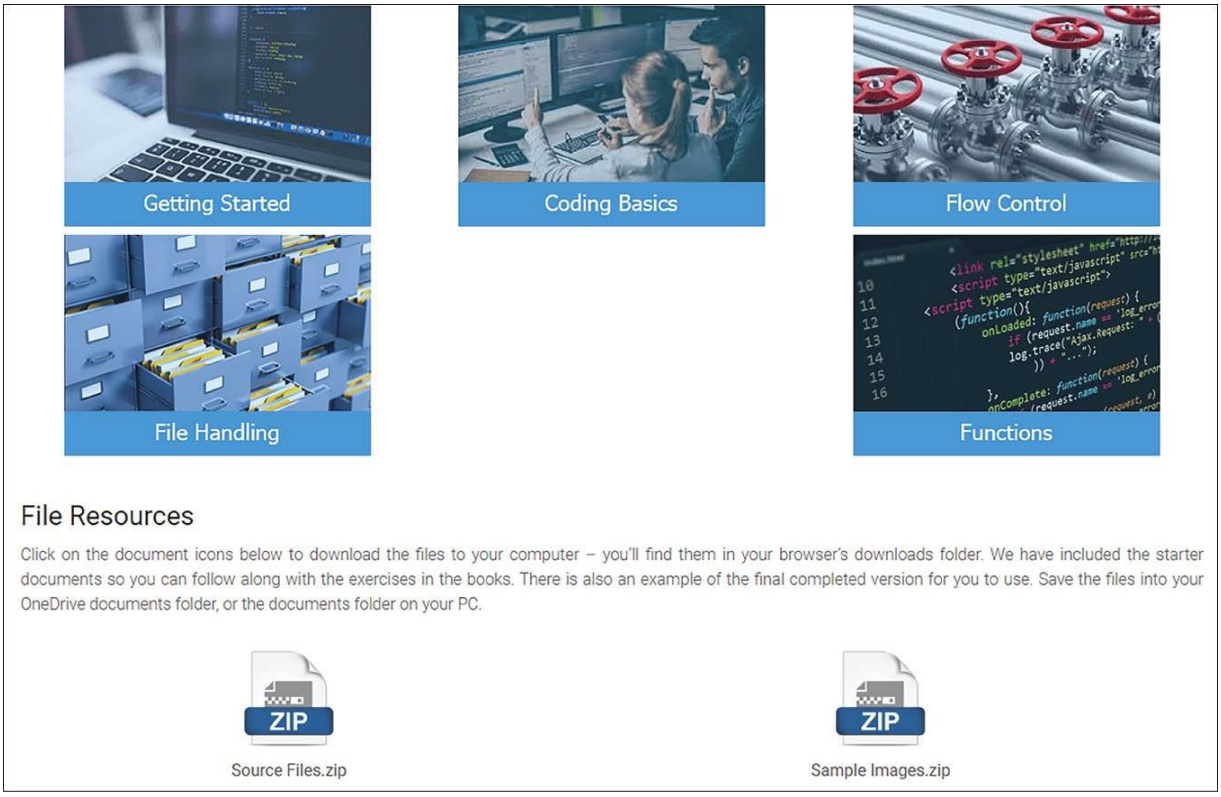
Открыв ссылку на видеоресурсы, вы увидите внизу список миниатюр.



Нажмите на миниатюру конкретного видео, которое хотите посмотреть. Большинство видеороликов с описанием процедуры длятся от 30 до 60 секунд, другие немного длиннее.

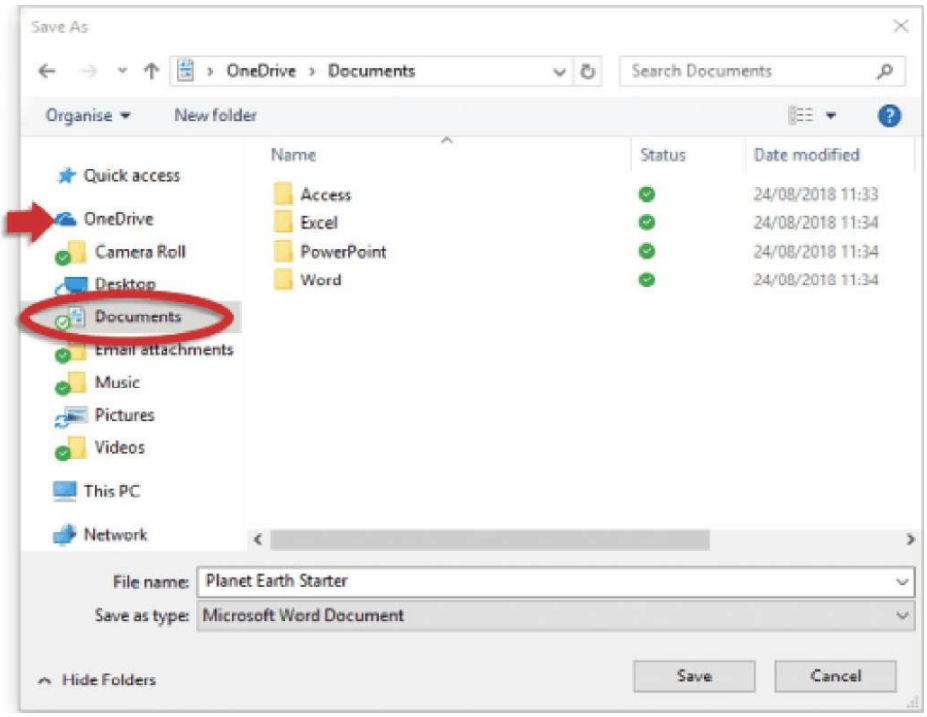
Загрузка кода примера

Вы также найдете файлы исходного кода внизу главной страницы ресурсов книги в разделе ‘file resources’.

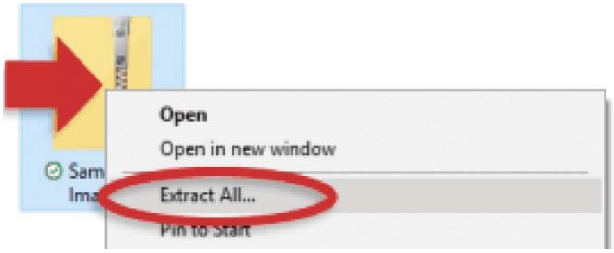


Чтобы загрузить zip-файл, щелкните правой кнопкой мыши значок zip на странице выше: ‘pythonfiles.zip’.

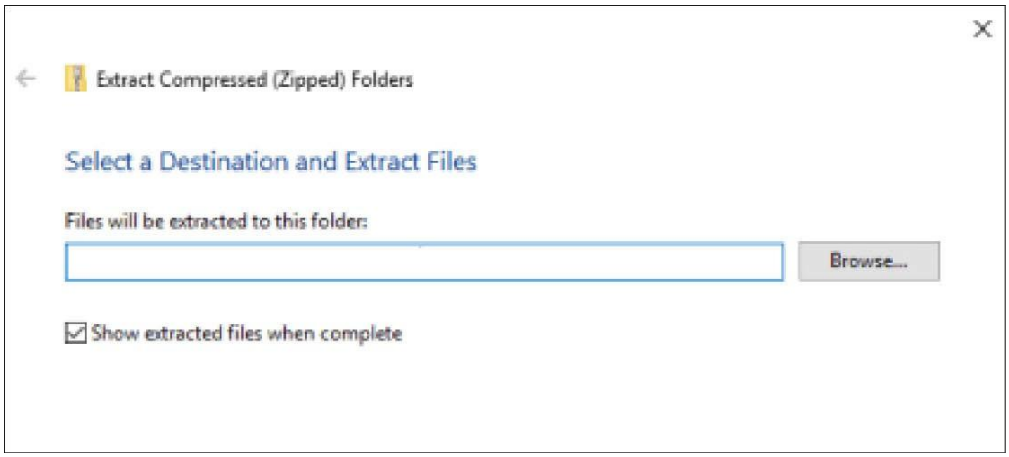
Выберите «Сохранить объект как» (или «Сохранить ссылку как» в некоторых браузерах) и сохраните его в «Документы» в папке OneDrive.



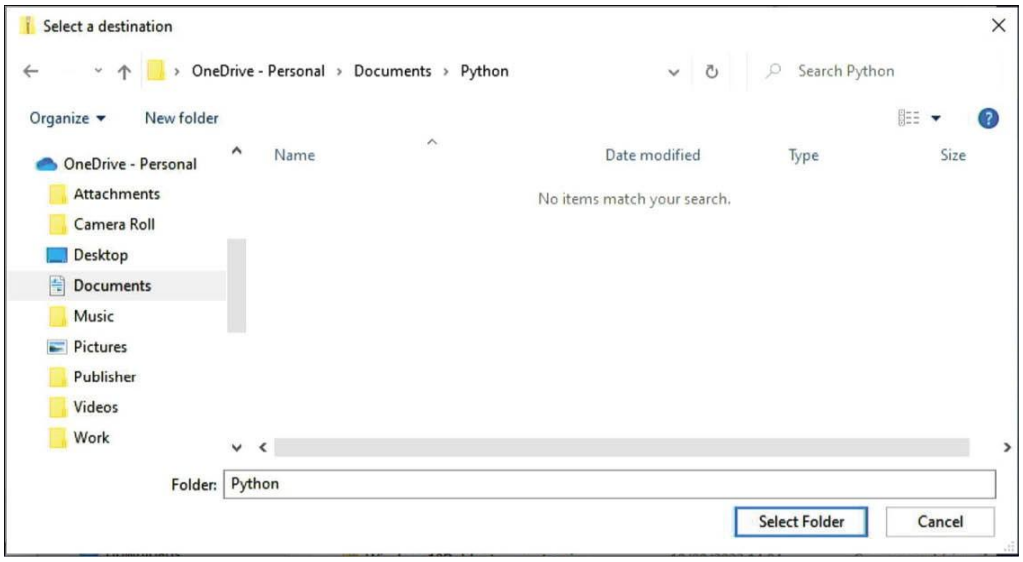
Загрузив zip-файл, перейдите в папку ‘documents’ в OneDrive, щелкните правой кнопкой мыши zip-файл и выберите в меню ‘extract all’.



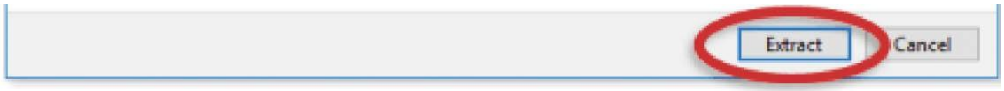
В диалоговом окне нажмите ‘browse’.



Перейдите в папку, в которую вы хотите извлечь файлы. Например: OneDrive -> папка Documents. Если вы хотите создать новую папку, нажмите ‘click ‘new folder’ - дайте ей имя. Нажмите ‘select folder’



Нажмите ‘extract’.

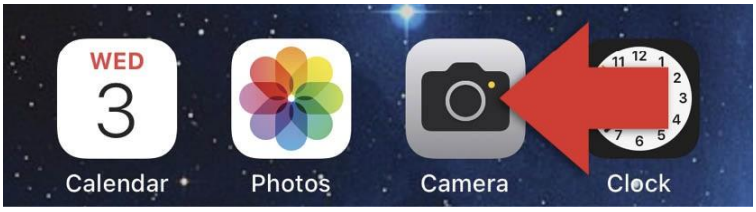


Сканирование кодов

В начале каждой главы вы увидите QR-код, который можно отсканировать с помощью телефона, чтобы получить доступ к дополнительным ресурсам, файлам и видео.

iPhone

Чтобы отсканировать код с помощью iPhone/iPad, откройте приложение камеры.

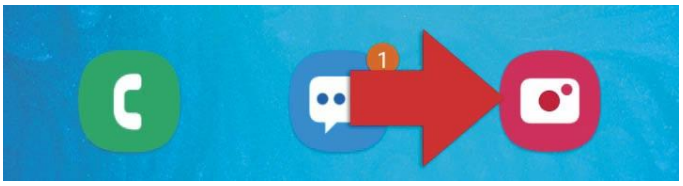


Разместите код в центре экрана. Нажмите на всплывающее окно веб-сайта вверху.

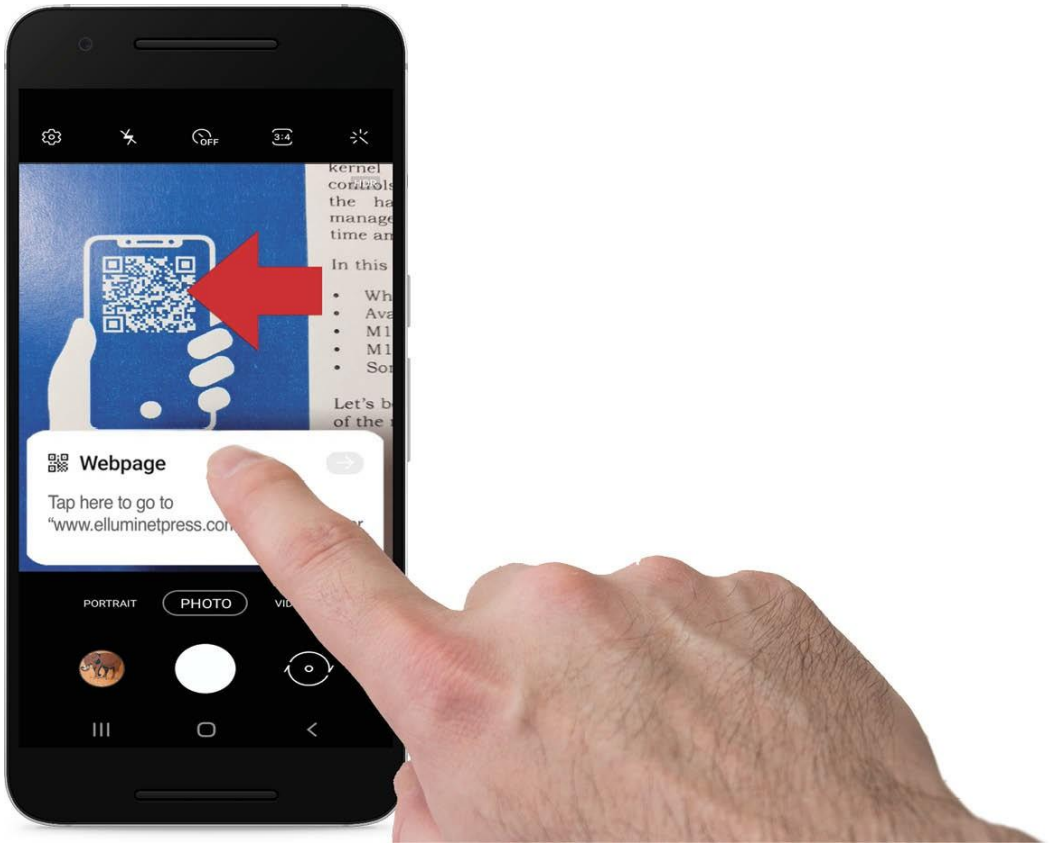

Website QR Code
Open "elluminetpress.com"
in Safari
SLO-MO
VIDEO
PHOTO
PORTRAIT

Android

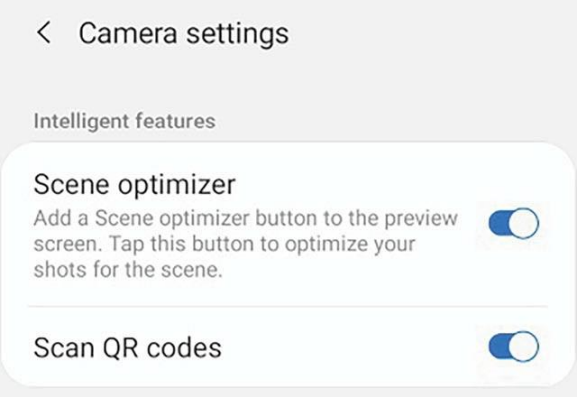
Чтобы отсканировать код с помощью телефона или планшета, откройте приложение камеры.



Разместите код в центре экрана. Нажмите на всплывающее окно веб-сайта вверху.



Если он не сканирует, включите ‘Scan QR codes’. Для этого коснитесь значка настроек в левом верхнем углу. Включите ‘scan QR codes’.



Если настройка отсутствует, вам необходимо загрузить сканер QR-кода. Откройте Google Play Store, затем найдите “QR Code Scanner”.