ABSOLUTE BEGINNER'S PYTHON PROGRAMMING

The Illustrated Guide to Learning Computer Programming

FULL COLOR GUIDE WITH LAB EXERCISES

Absolute Beginner's Python Programming

Full Color Guide with Lab Exercises

Kevin Wilson

Absolute Beginner's Python Programming

iStock.com/golibo, PeopleImages, ymgerman. Photo 130859010 © Kaspars Grinvalds - Dreamstime.com. Photo 103557713 © Konstantin Kolosov - Dreamstime.com. Yuri Arcurs via Getty Images

Publisher: Elluminet Press

Director: Kevin Wilson

Lead Editor: Steven Ashmore

Technical Reviewer: Mike Taylor, Robert Ashcroft

Copy Editors: Joanne Taylor, James Marsh

Proof Reader: Steven Ashmore

Indexer: James Marsh

Cover Designer: Kevin Wilson

eBook versions and licenses are also available for most titles. Any source code or other supplementary materials referenced by the author in this text is available to readers at

www.elluminetpress.com/resources

For detailed information about how to locate your book's resources, go to

www.elluminetpress.com/resources

Table of Contents

About the Author

With over 20 years' experience in the computer industry, Kevin Wilson has made a career out of technology and showing others how to use it. After earning a master's degree in computer science, software engineering, and multimedia systems, Kevin has held various positions in the IT industry including graphic & web design, programming, building & managing corporate networks, and IT support.

He serves as senior writer and director at Elluminet Press Ltd, he periodically teaches computer science at college, and works as an IT trainer in England while researching for his PhD. His books have become a valuable resource among the students in England, South Africa, Canada, and in the United States.

Kevin's motto is clear: "If you can't explain something simply, then you haven't understood it well enough." To that end, he has created the Exploring Tech Computing series, in which he breaks down complex technological subjects into smaller, easy-to-follow steps that students and ordinary computer users can put into practice.

Acknowledgements

Thanks to all the staff at Luminescent Media & Elluminet Press for their passion, dedication and hard work in the preparation and production of this book.

To all my friends and family for their continued support and encouragement in all my writing projects.

To all my colleagues, students and testers who took the time to test procedures and offer feedback on the book

Finally thanks to you the reader for choosing this book. I hope it helps you understand Computer Hardware.

Have fun!

Intro to Computer Programming

What is a computer program? A computer program is a set of concise instructions written in a programming language that are executed in sequence in order to achieve a task.

A computer program usually takes some data such as a string, or a number and performs some kind of processing to produce results. We usually refer to the data as the program's input, and the results as the program's output.

To write computer programs we use a computer programming language. There are many different languages such as BASIC, C, C++ and Python. In this guide, we are going to concentrate on the Python programming language.

Every computer program manipulates data to produce a result, so most languages allow the programmer to choose names for each item of data.

These items are called variables. A variable is as the name suggests is an item that can contain different values as the program is being executed. Variables can store data of different types, and different types can do different things. For example, a variable could be an integer to store a whole number, a float to store numbers with decimal places, a string to store text, or a list to store multiple data items.

If we wrote a program to calculate the area of a triangle, we could have integer or float type variables for the length, the height, and one for the result as they are all numbers. The processing part of the program would use the numbers stored in the length and height variable, then assign the result to the variable for result. We'll take a look at variables and data types in more detail in chapter 3

In larger programs, we often need to make decisions based on user input, a calculated result or condition. In this case, we use an if

statement. This is called selection. Some blocks of code might also need to be repeated, in this case we use a loop. This is called repetition. We'll take a closer look at this in chapter 4.

Many modern computer programs have fancy graphical user interfaces that allow the user to interact with buttons, windows and menus. These are known as software applications or apps. We'll take a look at building a simple app with a graphical user interface in chapter 11.

The Python programming language has specific facilities to enable us to implement the concepts outlined above. Many of these will be introduced throughout this guide.

Introducing Python

Python is a high level language developed by Guido van Rossum in the late ‘80s and is used in web development, scientific applications, gaming, AI, and is well suited to education for teaching computer programming.

Python is designed to be an easily readable language. Therefore, it uses an uncluttered formatting style, and often uses meaningful English keywords and function names.

Python is an interpreted programming language, meaning Python programs are written in a text editor and then put through a Python interpreter to be executed.

Python is used in the field of artificial intelligence and can be found in many day-to-day applications. Streaming services such as Spotify use Python for data analysis, particularly user’s listening habits in order to offer suggestions on which artist to follow, other music a particular user might be interested in and so on. Python is also used within Netflix’s machine-learning algorithms for recommending relevant content to users, monitoring browsing habits, and marketing.

In the world of games development, Python is used as a companion language, meaning Python scripts are used to add customizations to the core gaming engine, script AI behaviors, or server side elements. The performance of Python isn’t fast enough for coding graphics intensive, higher end games, however you can create simple games with Python using the pygame module. We’ll take a look at this in chapter 12.

Python is used in web development and allows a web developer to develop dynamic web apps very quickly. More in chapter 13.

Python is a multi platform language and is available for Windows, MacOS, Linux and the Raspberry Pi.

Setting Up

To start coding, you'll need a computer - either Windows, MacOS or Linux, and an Integrated Development Environment (IDE) with the Python interpreter. Let's install Python.

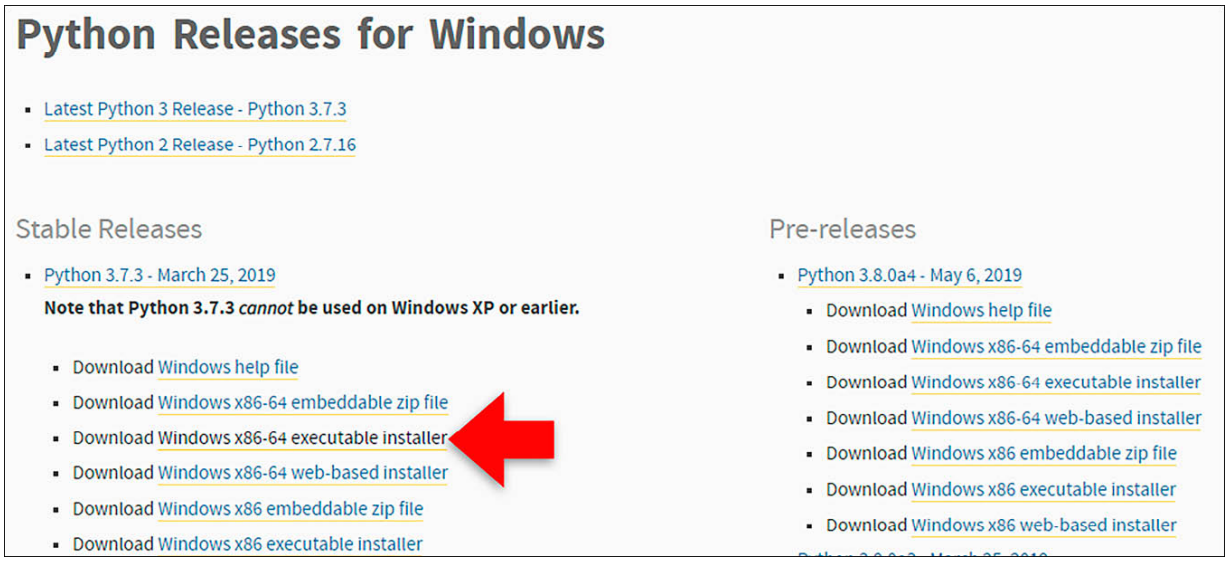
Install on Windows

In our lab, we're using windows workstations, so we'll need to install the Python Development Environment for Windows.

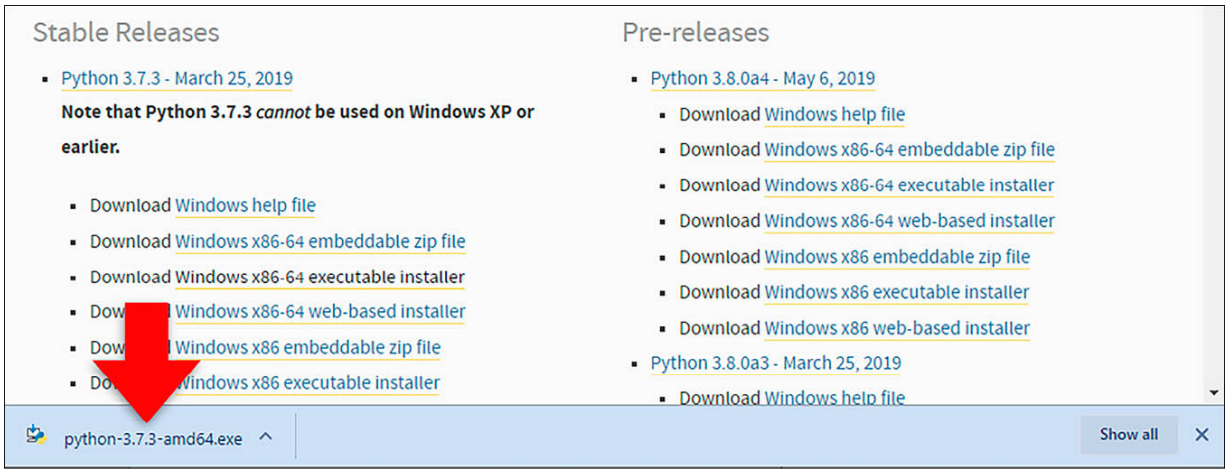
Open your web browser and navigate to the following website

`www.python.org/downloads/windows`

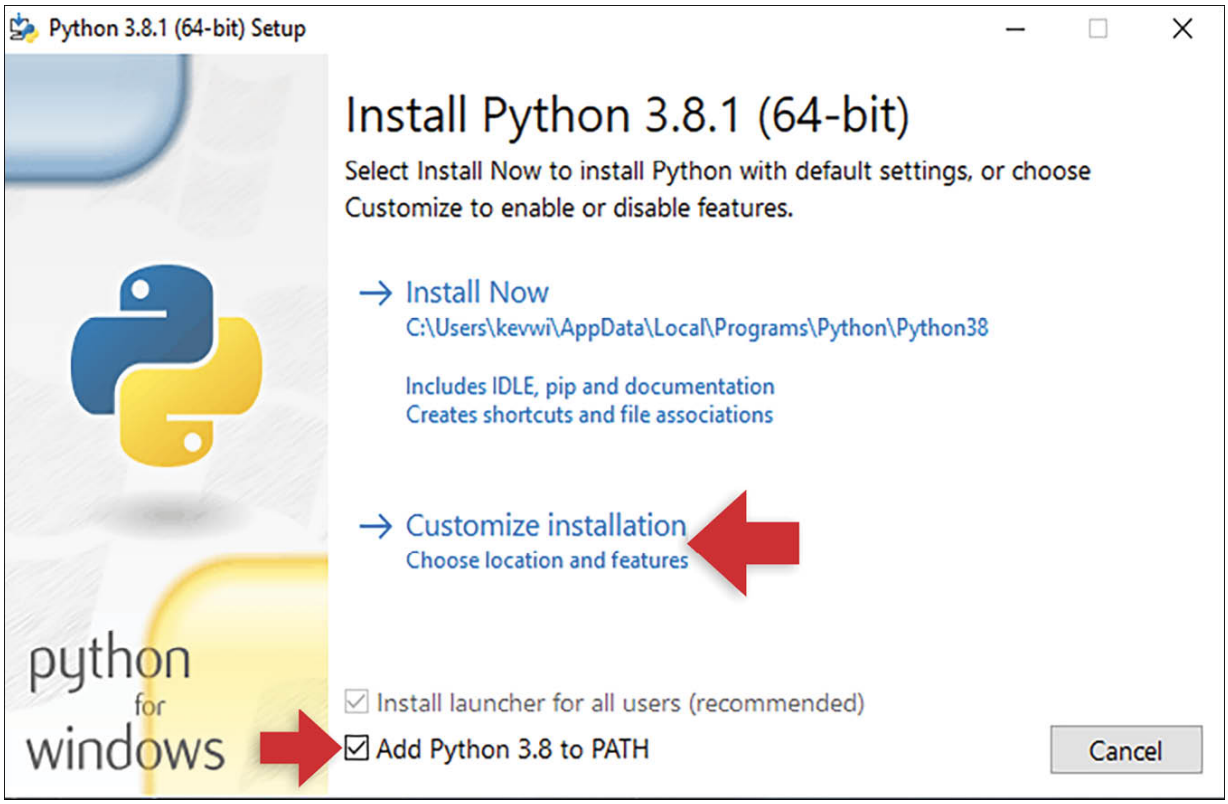
From the downloads page, select the 'executable installer' of latest stable release.



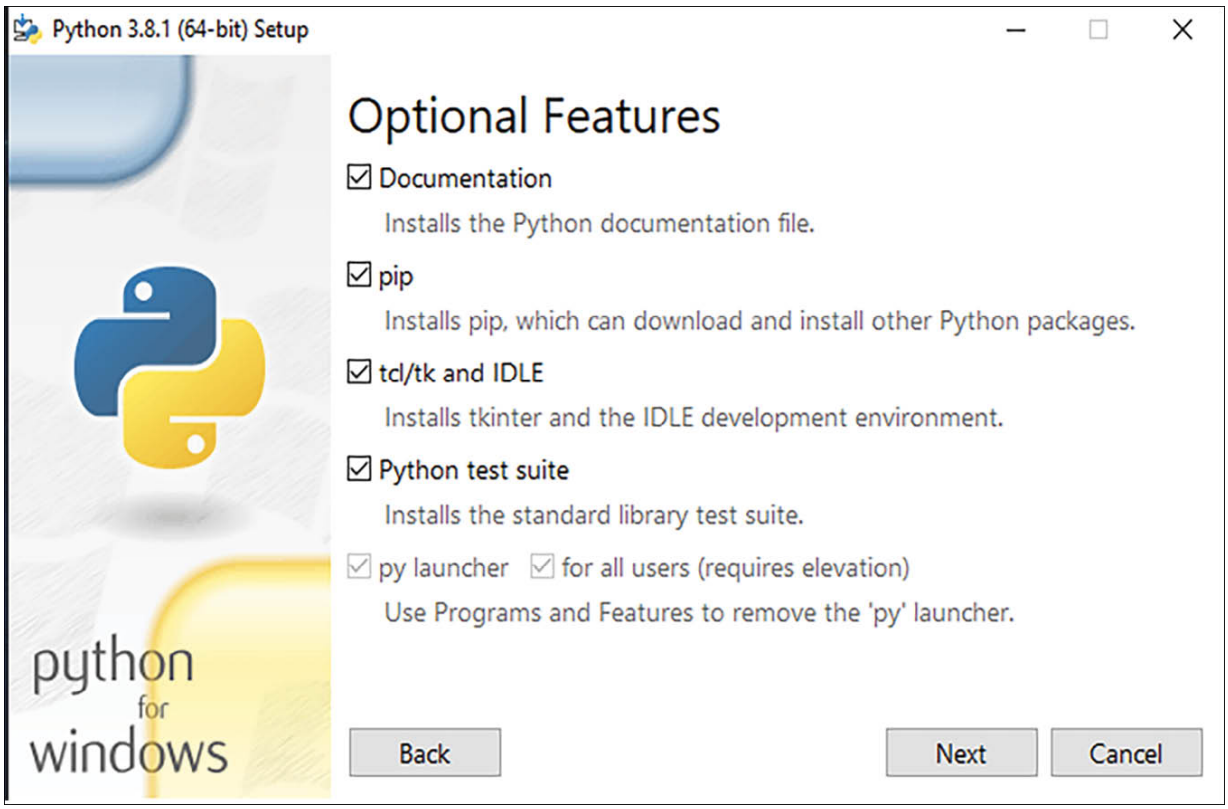
Click 'run' when prompted by your browser. Or click 'python-x.x.x-amd64.exe' if you're using Chrome.


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 x86-64 embeddable zip file
Download Windows x86-64 executable installer
Windows x86-64 web-based installer
Windows x86 embeddable zip file
Windows x86 executable installer
Pre-releases
Python 3.8.0a4 - May 6, 2019
Download Windows help file
Download Windows x86-64 embeddable zip file
Download Windows x86-64 executable installer
Download Windows x86-64 web-based installer
Download Windows x86 embeddable zip file
Download Windows x86 executable installer
Download Windows x86 web-based installer
Python 3.8.0a3 - March 25, 2019
Download Windows help file
python-3.7.3-amd64.exe
Show all

Once the installer starts, make sure ‘add python 3.x to path’ is selected, then click ‘customize installation’ to run through the steps to complete the installation.

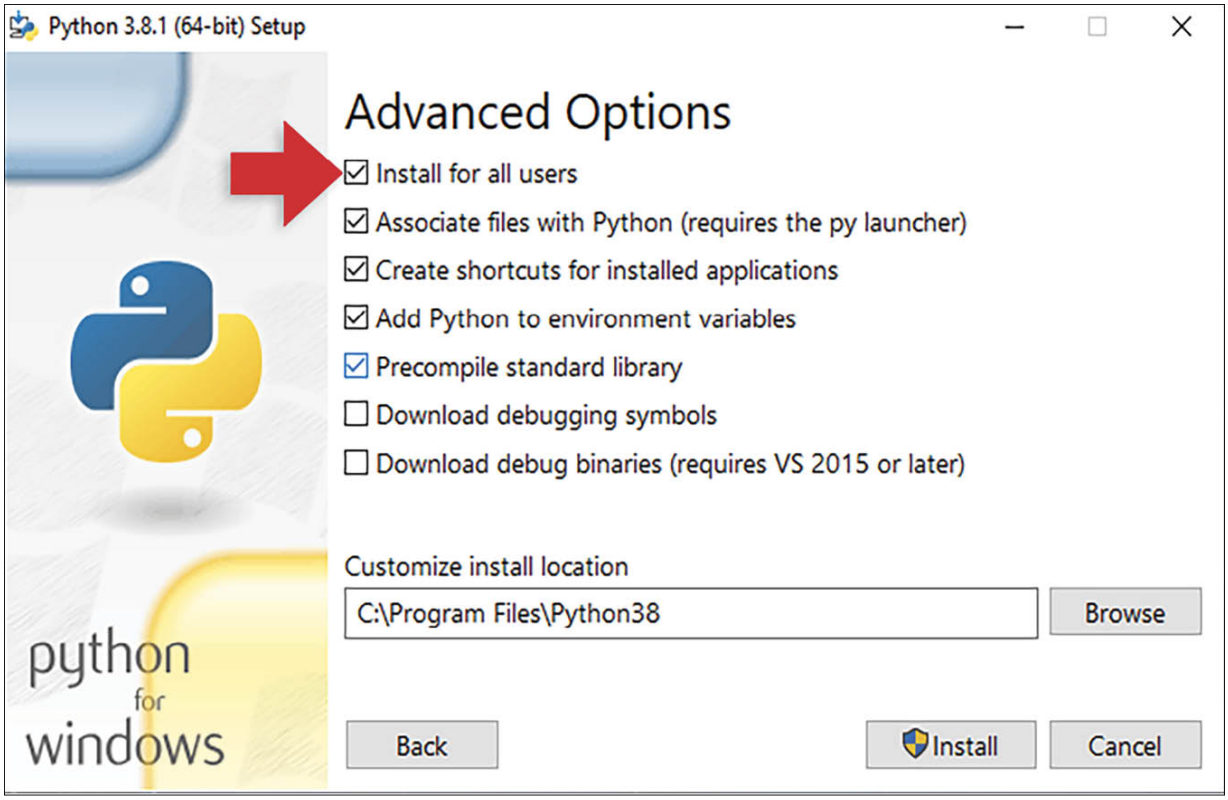


Make sure you select all the tick boxes for all the optional features.

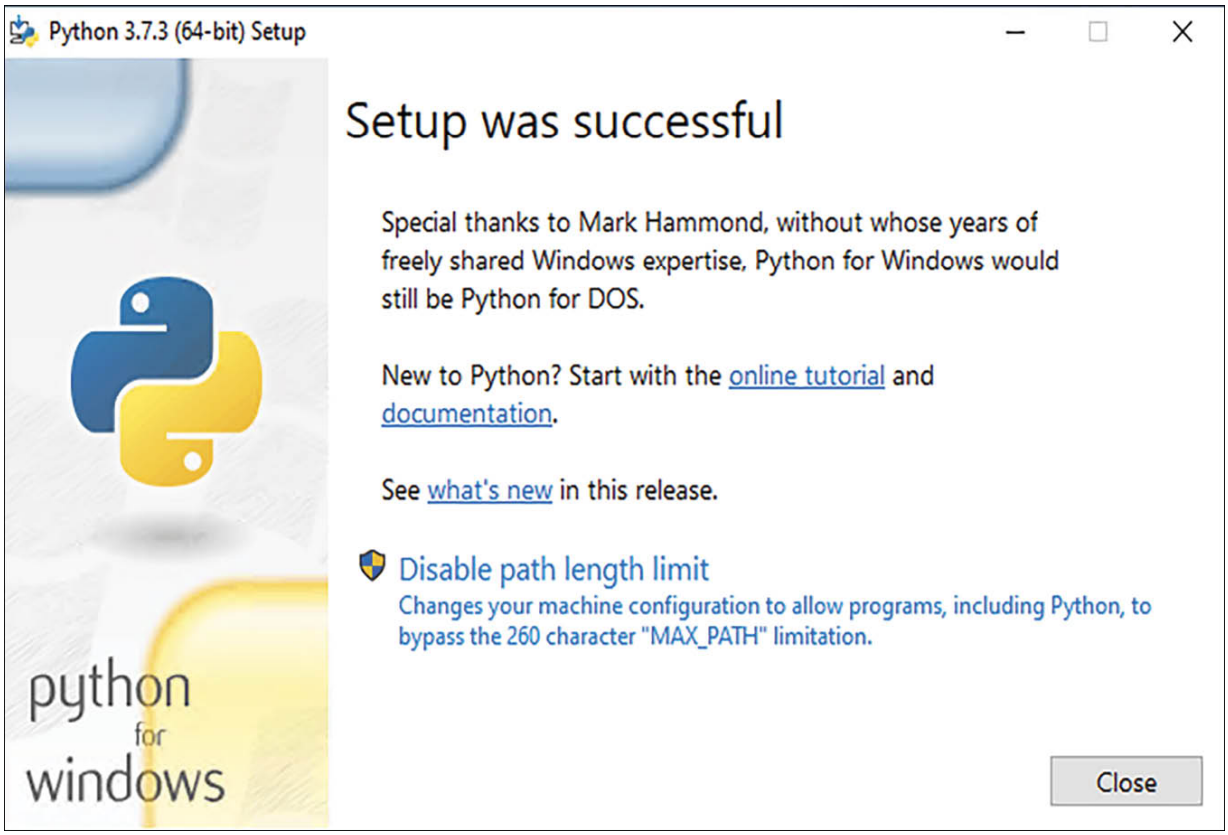

Python 3.8.1 (64-bit) Setup
Optional Features
Documentation
Installs the Python documentation file.
pip
Installs pip, which can download and install other Python packages.
tcl/tk and IDLE
Installs tkinter and the IDLE development environment.
Python test suite
Installs the standard library test suite.
py launcher
for all users (requires elevation)
Use Programs and Features to remove the 'py' launcher.
python
for
windows
Back
Next
Cancel

Click ‘next’.

Make sure ‘install for all users’ is selected at the top of the dialog box. Click ‘install’ to begin.

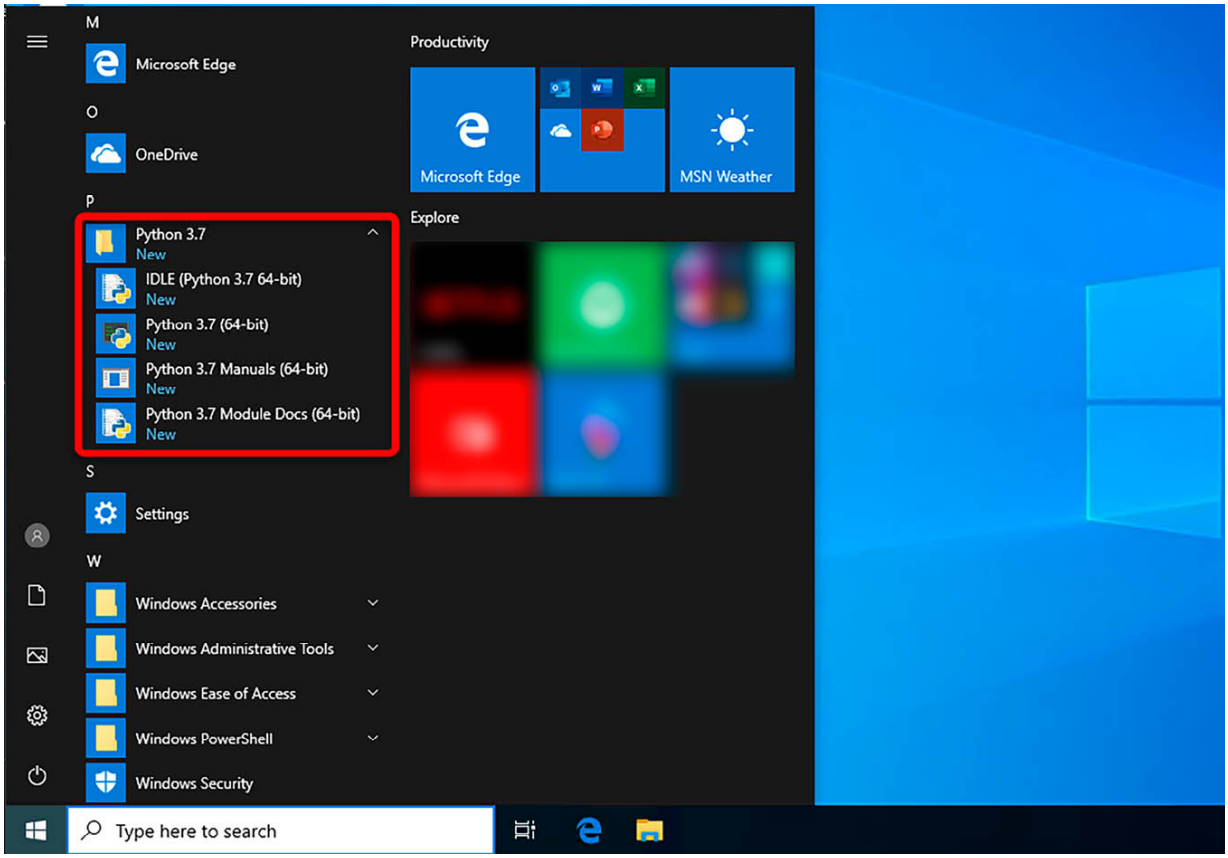


Click ‘disable path length limit’ to make sure Python runs smoothly on Windows and allow long file names.



Click ‘close’ to finish the installation.

You'll find the Python Development Environment (IDLE) and the Python interpreter, in the Python folder on your start menu.

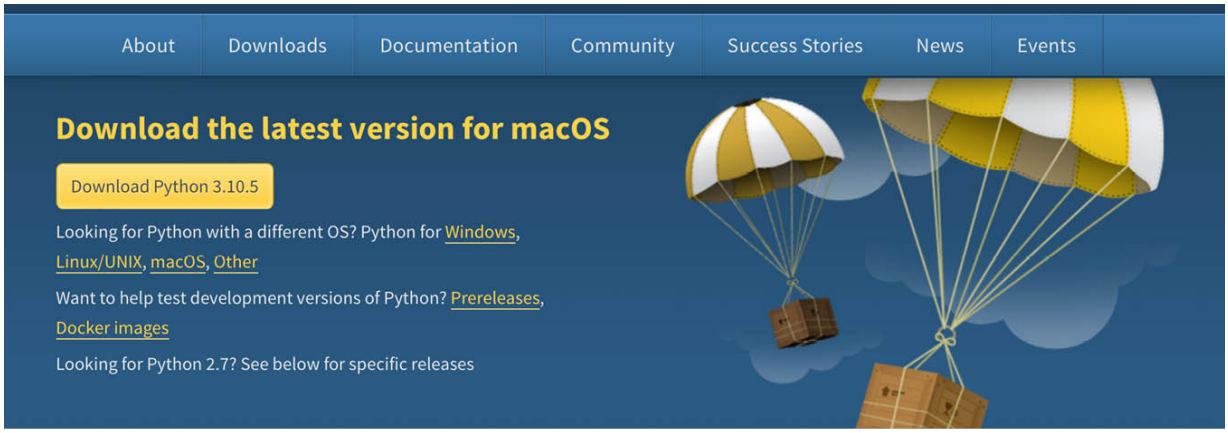


Install on MacOS

To install Python 3 with the Official Installer, open your web browser and navigate to the following website

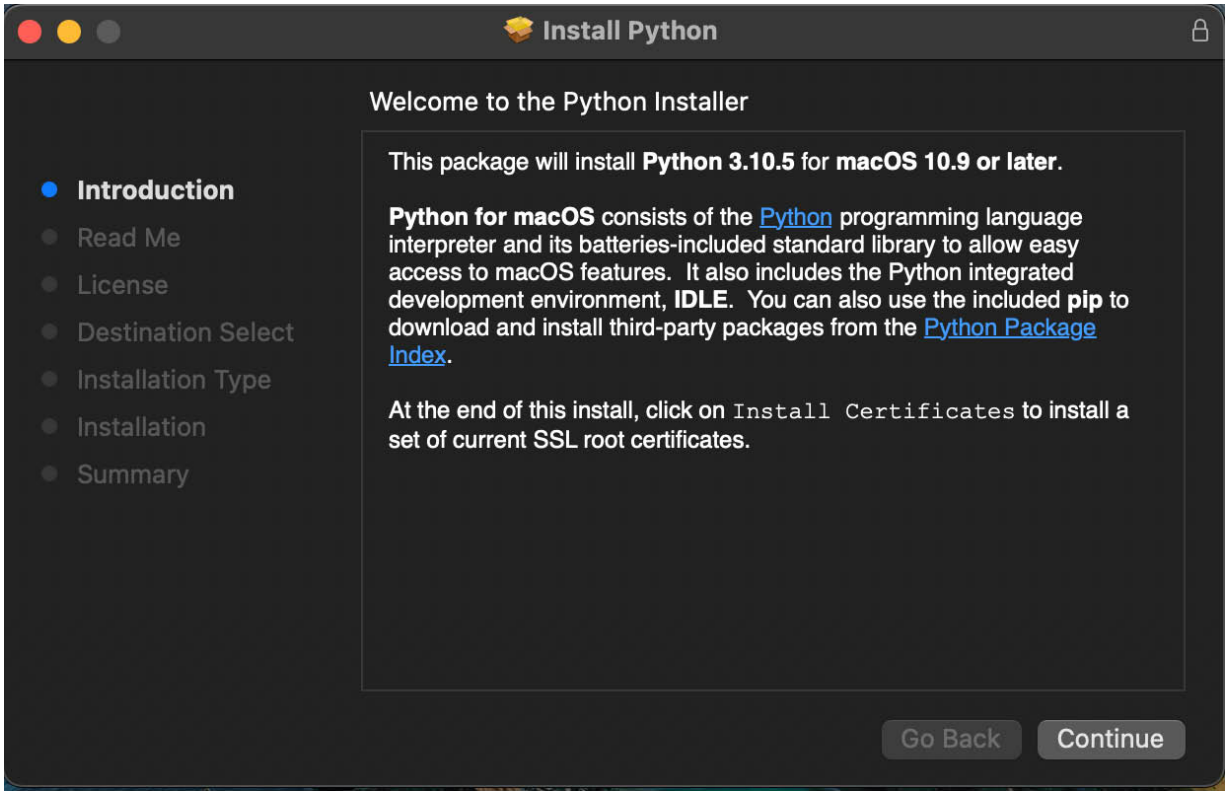
`www.python.org/downloads/macos`

Click download python.



You'll find the package in your downloads folder. Double click on the package to begin the installation

Run through the installation wizard. Click 'continue'.

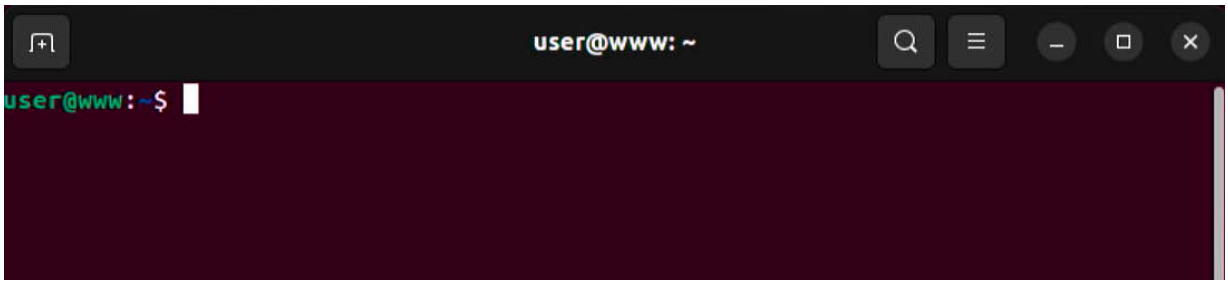


Once the installation is complete, you'll find python in the applications folder in finder, or on the launch pad.

Install on Linux

If you are running a linux distribution such as Ubuntu or have a

Raspberry Pi, you can install python using the terminal. You'll find the terminal app in your applications. You can also press Control Alt T on your keyboard.



At the terminal command prompt, type the following commands. Press enter after each line.

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

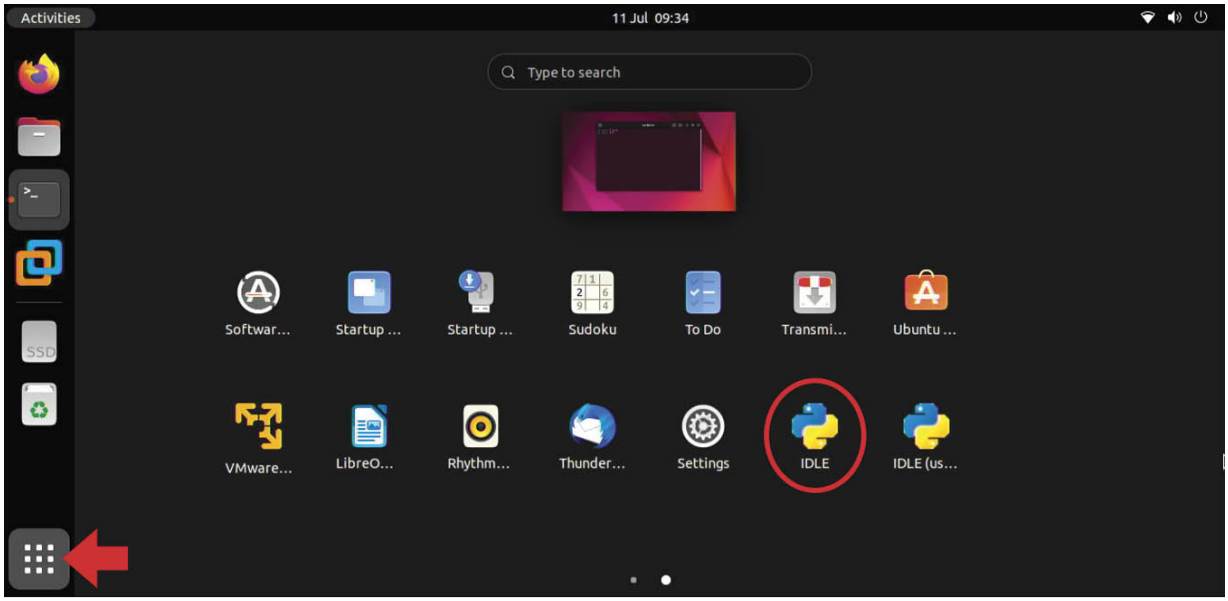
Type the following command to install Python.

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

Once the Python is installed, we need to install IDLE, the development environment. To do this, type the following command at the prompt

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

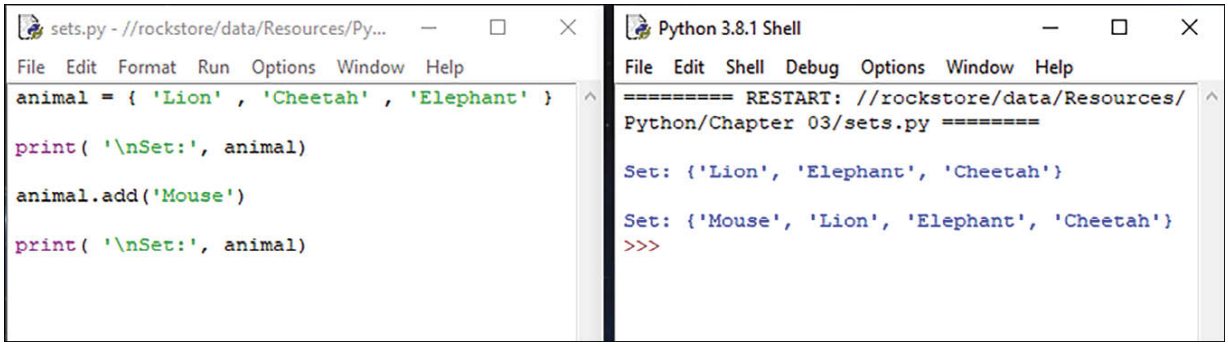
Once installed, you'll find IDLE in your applications.



Or you can type the following command at the prompt

```
idle
```

Arrange your windows so you can see your code window on the left and the shell on the right.



Here, you can write your code in the editor then execute & debug your code. You'll also notice the code editor provides syntax highlighting meaning keywords and text are highlighted in different colors, making code easier to read.

Setup a Coding Environment

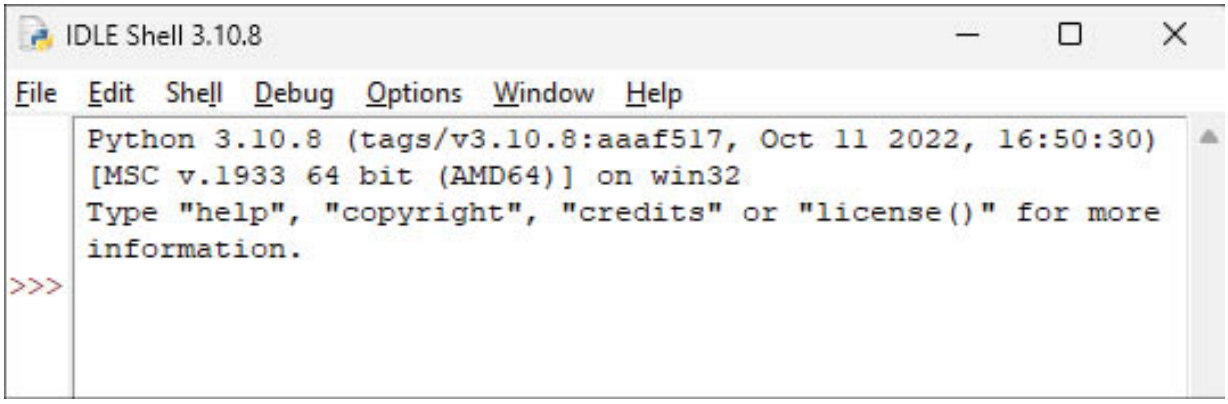
There are countless development environments and code editors out there such as

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

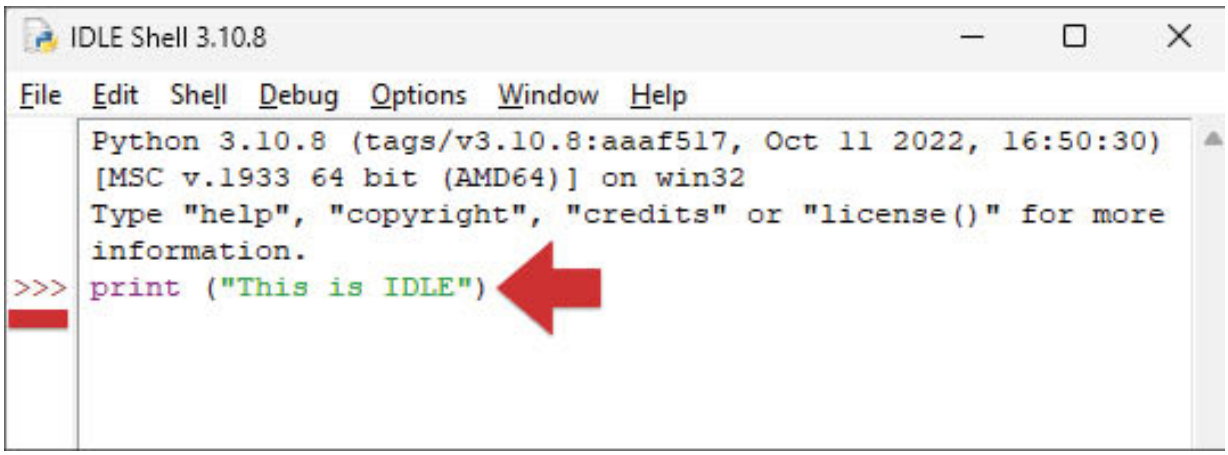
For the purpose of this guide, we'll be using IDLE, an integrated development environment (IDE) that comes with Python.

You'll find IDLE on your start menu in Windows, or in Finder/Launch Pad on a Mac.

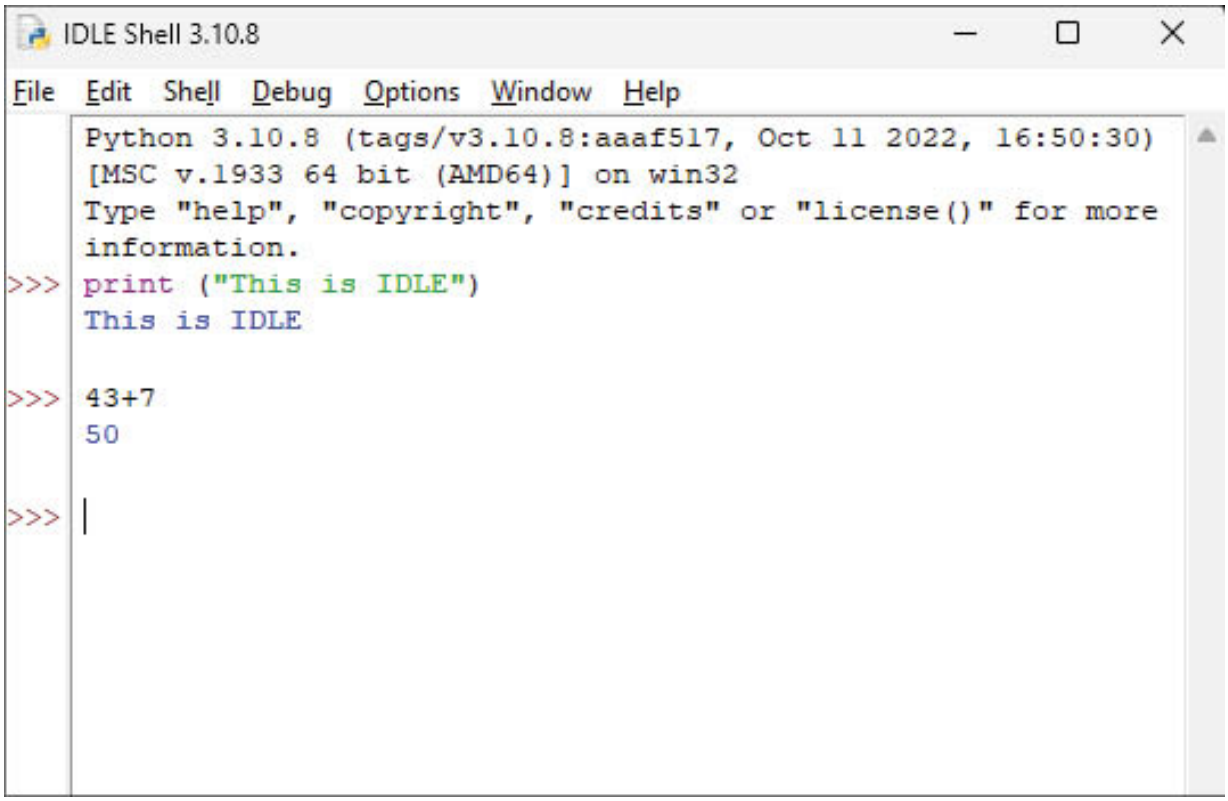
Once IDLE starts, you'll see the shell window.



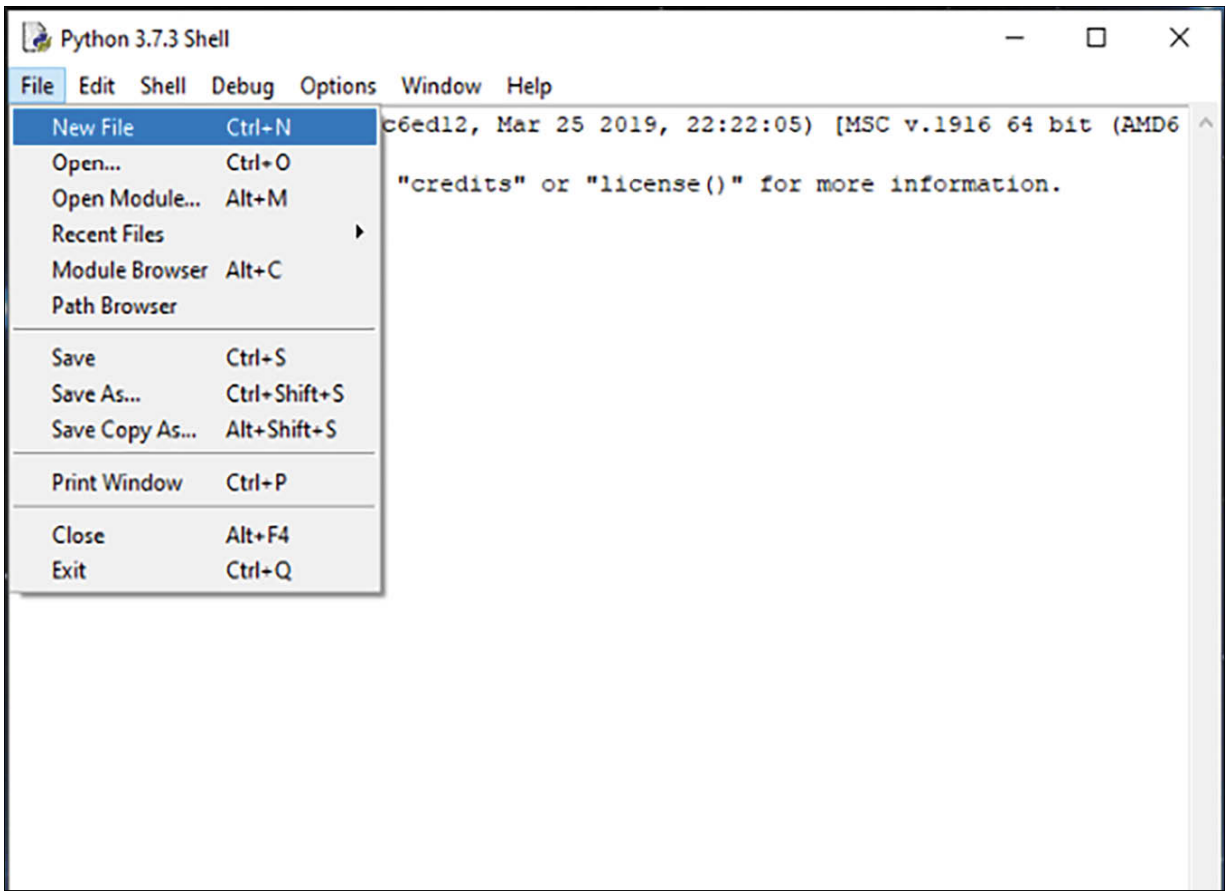
At the '>>>' prompt you can type and execute statements. You can perform mathematical operations, comparisons, take input from user, and the Python interpreter will execute them.



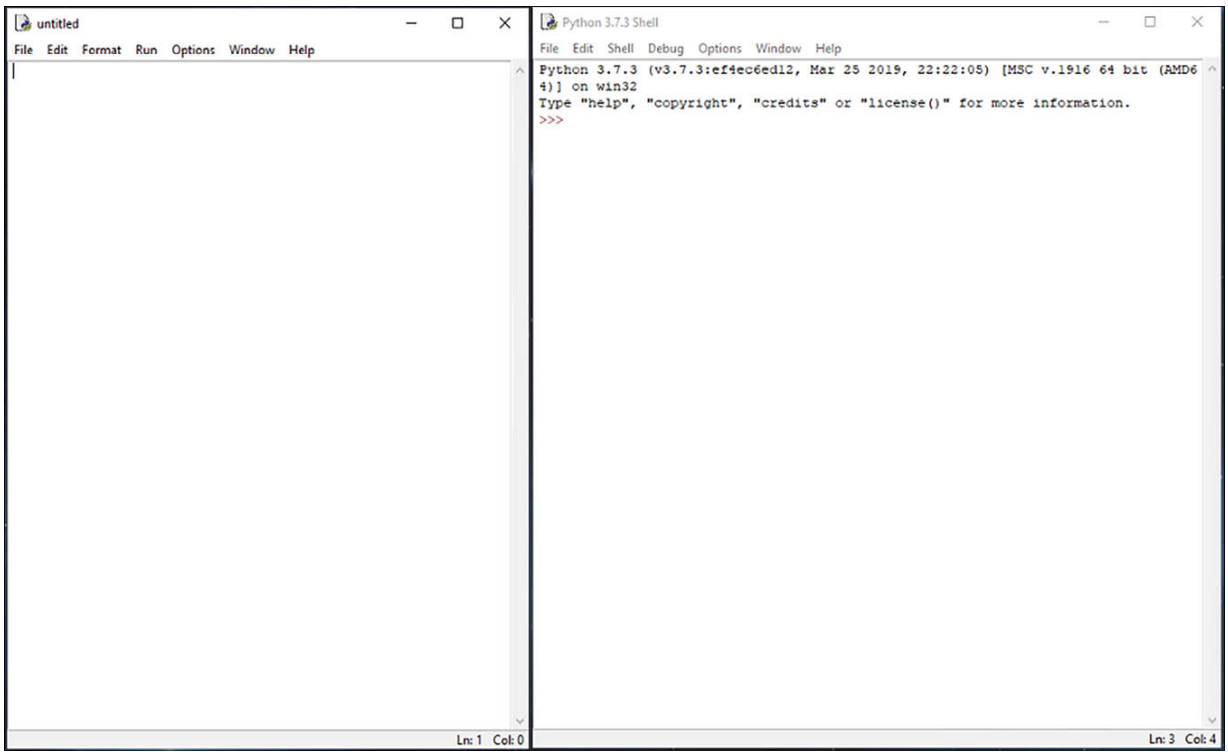
Here, we executed a print statement, and evaluated a mathematical expression. Once you hit enter, you'll see the output appear in blue underneath.



To create a python file, select 'file' menu in the upper left-hand corner of the shell window.



Align the editor window to the left of the shell window as shown below.



Now you can enter a python program in the editor window. To save the file, select the ‘file’ menu, click ‘save’. To run a program, select the ‘run’ menu on the editor window, click ‘run module’ (or press F5).

Lab Exercises

1. Set up and install Python and IDLE on your computer.
2. Create a working directory on your computer to store all your python programs. This could be in your documents folder depending on which operating system you are using.
3. Download the code source files for this book and extract the pythonfiles.zip file to the directory you created above.
   `elluminetpress.com/python`
4. What is a computer program?
5. How do you open a new file in IDLE? How do you save your program in IDLE? In which directory do you save your program? How do you run the program?
6. Where can you use the Python language?

The Basics

Python programs are written in a text editor, such as Notepad, PyCharm, or the code editor in Python's development environment (IDLE), and saved with a .py file extension.

You then use the Python interpreter to execute the code saved in the file.

For this section, take a look at the video demos

`elluminetpress.com/pybasics`

Have a look at the files in the directory Chapter 02.

Lets start at the beginning and explore what a programming language is.

A programming language is a method of expressing a set of concise instructions to be executed by a computer.

Language Classification

There are different levels of programming language: low level languages and high level languages.

Low-Level Language

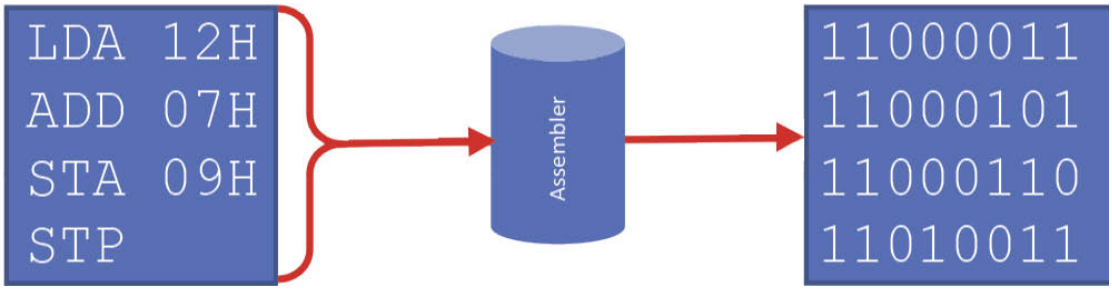
A low level language is a programming language whose functions often refer directly to the processor's instructions and is commonly written in machine code or assembly language.

Assembly language is known as a second generation programming language, machine code being the first generation.

Lets take a look at a simple program. Here, we have a little adder program written in assembly language for our processor, and might look something like this

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

Code is written in assembly language and then assembled into machine code using an assembler before it is executed.



Each assembly language instruction corresponds to a sequence of binary numbers in machine code. The numbers, characters, addresses and other data are converted into their machine code equivalents.

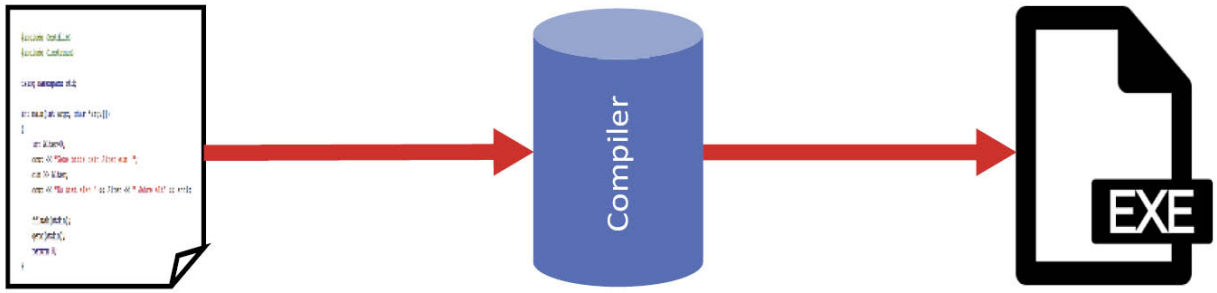
So, LDA could be represented by the binary code: 11000011, the number 12_{10} is 00001100 in binary.

The assembled machine code is then executed by the processor.

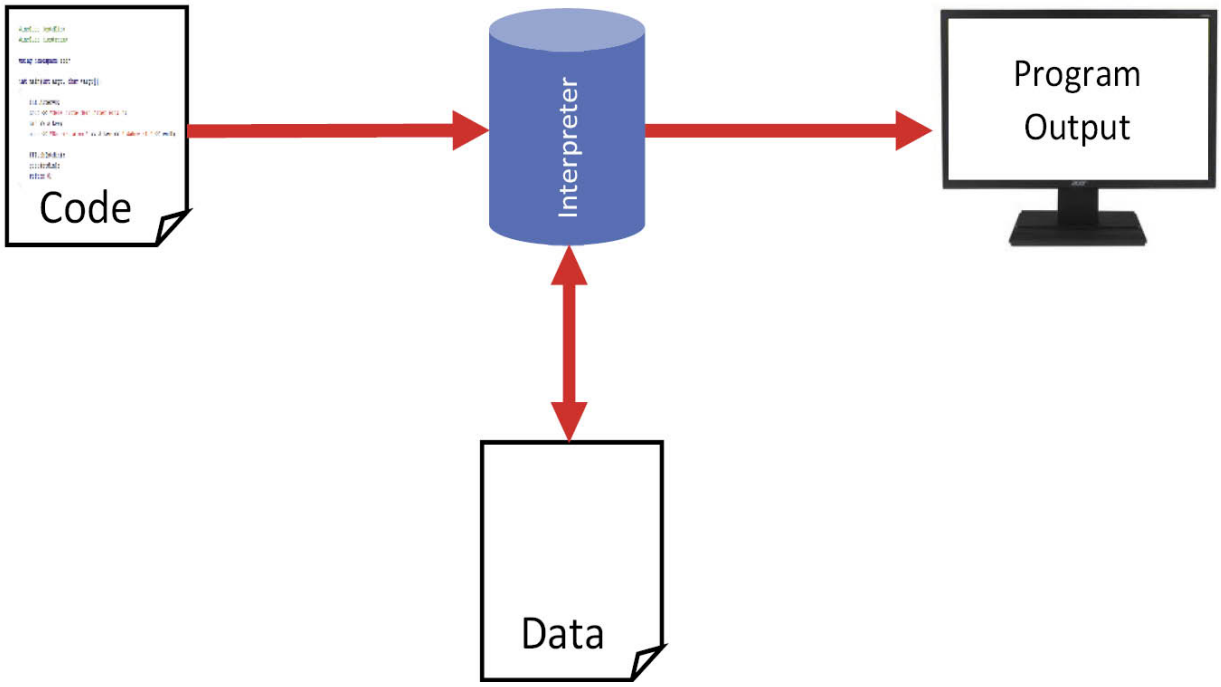
High-Level Language

Python is an example of a high-level language. Rather than dealing directly with processor registers and memory addresses, high-level languages deal with variables, human readable statements, loops, and functions.

High level language code is either compiled into a machine code executable program or interpreted. Languages such as C or C++ are often compiled, meaning the code is written and then converted into an executable file. This makes them ideal for software development to write applications such as Microsoft Word that run on a computer.



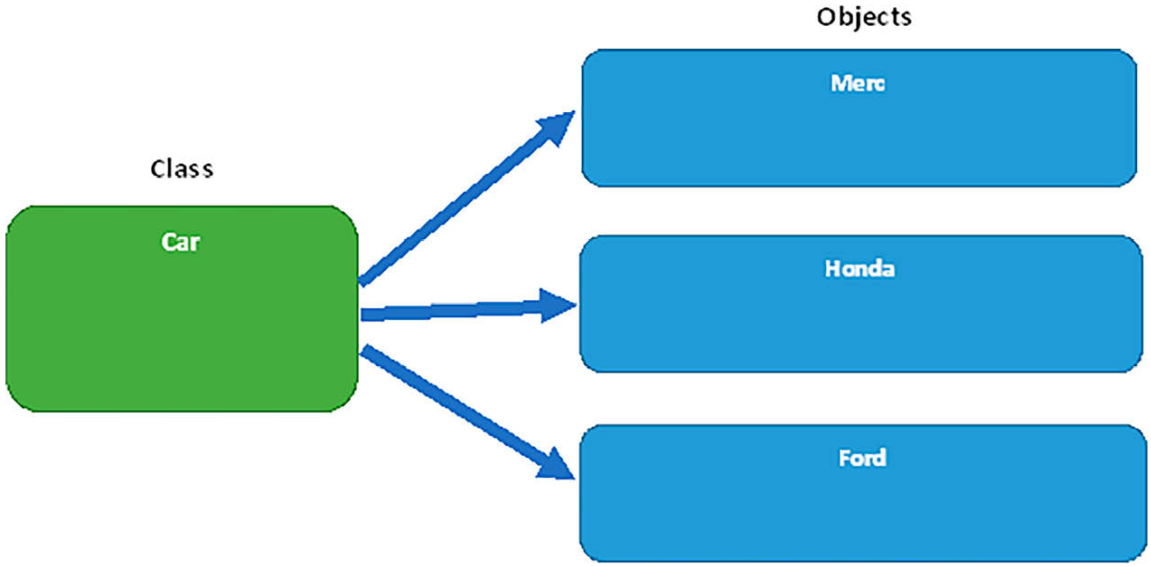
Python is an interpreted language meaning the code you write is translated into machine code directly, making it well suited to web development.



Here, you can see the interpreter executes the code line by line while accessing any data required by the program, then displays the output directly onto the screen.

Object Oriented Programming

Python is an object oriented programming language. This means that the program design is based around objects, rather than functions and logic.



We'll explore this in more detail in chapter 9 but for now, just familiarize yourself with the terminology.

Class

A class is a user-defined blueprint or template that defines the attributes (variables) and methods (functions), and contains them all in a single unit called a class.

Object

An object is an instance of a class. So, you could have a class called 'car' and use it to define an object called 'merc', 'taxi', etc

Attribute

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`

Method

A method is a function that is used by an object to perform an action. Methods are usually referenced by stating the object name dot method name. For example: `object.method()`

Python Language Syntax

The syntax defines how a program is written and interpreted and forms the basis of writing code.

Reserved Words

These are words reserved by the programming language that define the syntax and structure. Here are some of the most common ones:

and	A logical operator commonly used in if statements
as	Creates an alias
assert	For testing conditions and used in debugging
break	Breaks out of a loop
class	Creates a class
continue	Continues to the next iteration of a loop
def	Defines a function
del	Deletes an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Defines code to run when error occurs (an exception)
false	Boolean value, result of comparison operations
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
for	Creates a for loop
from	Used to import only a specified section from a module.
global	Declares a global variable
if	Create a conditional statement
import	Imports a module
in	Used to check if a value is present in a sequence
is	Used to test if two variables refer to the same object
lambda	Used to create small anonymous functions
None	Represents a null value
nonlocal	Used to work with variables inside nested functions,
not	A logical operator
or	A logical operator
pass	Does nothing, used as a placeholder
raise	Used to raise an exception.
return	To exit a function and return a value
true	Boolean value, result of comparison operations
try	To make a try...except statement
while	Creates a while loop
with	Used to simplify exception handling
yield	Used to end a function, returns a generator

For example, the word ‘while’ indicates a while loop. The word ‘if’ defines an ‘if statement’. You can’t use a reserved word as a variable name or function name.

Identifiers

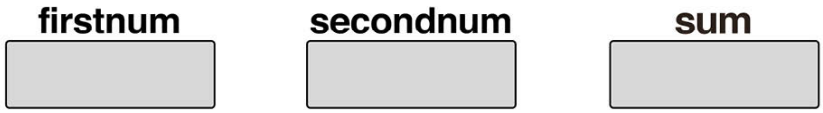
An identifier is a name given to a class, function, or a variable. Identifiers can be a combination of uppercase or lowercase letters,

numbers or an underscore _. Identifiers cannot be a keyword, and you can't use any special symbols such as *, ?, !, @, #, or $. Identifiers are case-sensitive, cannot contain spaces, nor can the first character be a digit. Try to keep the identifiers meaningful, that they describe what they're used for.

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

Variables

A variable is a labeled location in memory that is used to store values within a computer program. There are two types of variables: local and global.



Variables defined within a function are called local variables, as they are local to that particular function. These variables can only be seen by the function in which they are defined. These variables have local scope.

Here, 'sum', 'firstnum' and 'secondnum' are local variables to the function 'addsum'.

```
def addsum(firstnum, secondnum):

    sum = firstnum + secondnum

    return sum
```

Global variables are defined in the main body of the program outside any particular functions. These variables can be seen by any function and are said to have global scope.

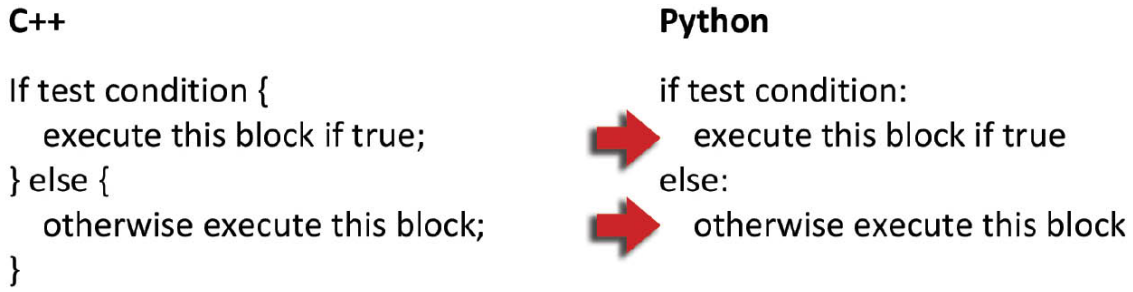
Here in this example, 'a' and 'b' are global variables.

```
a = int(2)

b = int(3)

def addsum(firstnum, secondnum):

sum = ffirstnum + secondnum
```

```
return sum
```

Indentation

Most other programming languages such as C and C++ use braces { } to define a block of code. Python uses indentation. Use the tab key.



This will make more sense when we start using if/else statements and loops in chapter 4, and functions in chapter 6.

Comments

Comments are very important while writing a program. You should clearly document all your code using comments, so other developers working on a project can better understand what your code is doing.

Use the hash character (#) to write single line comments.

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

If you need to write a block describing the functionality then use a triple quote before and after the comment block.

For example:

```
""" Prompt user for two numbers
one after the other using a text input """
a = input ('Enter first number: ')
b = input ('Enter second number: ')
```

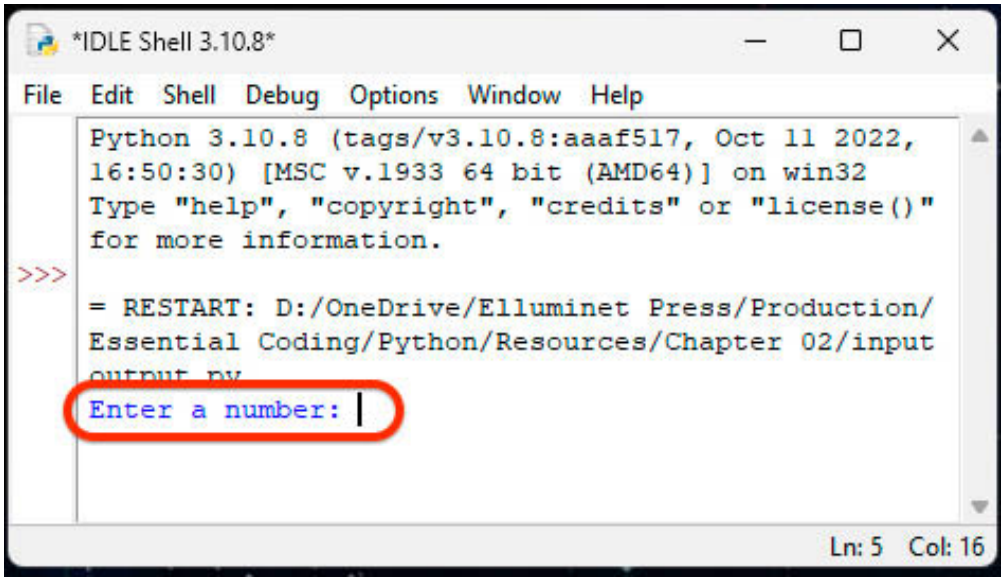
Input

You can obtain input from the user using the `input()` function.

This function prompts the user to type in some data and assigns it to the variable ‘number’.

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

When we execute this line, the input function displays a prompt in the shell console. The data entered will be assigned to a variable.



Output

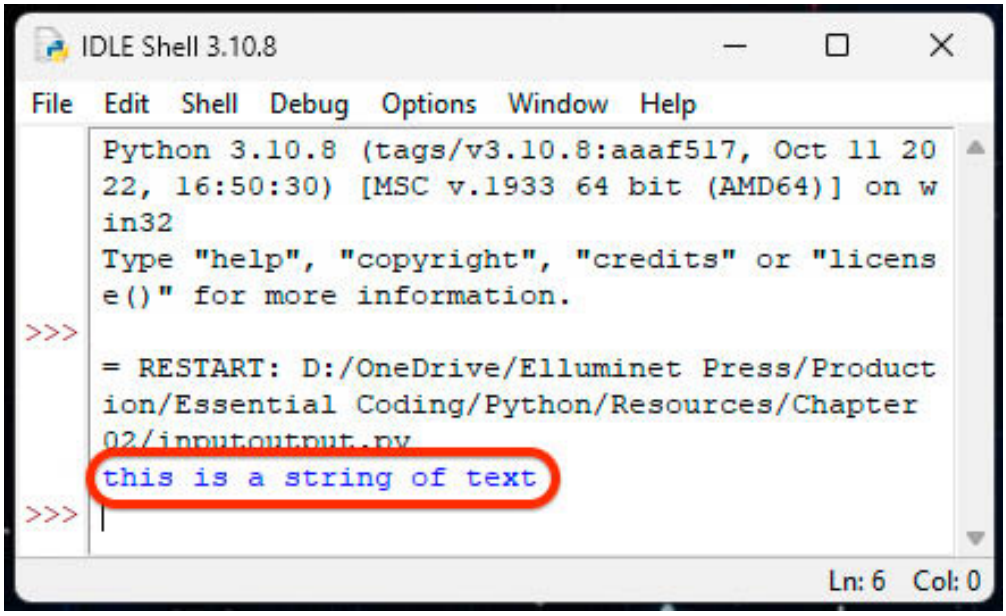
You can display information on the screen with the `print()` function. You can print the contents of a variable, or enclose a string within the parameters of the `print()` function. For example:

```
print (number)
```

...or...

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

When we execute this line, the text is printed to the shell console.



Functions

A function is a block of code that is only executed when it is called. Python comes with various built-in functions such as

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

A function takes parameters or arguments enclosed in parenthesis and returns a result. Most people seem to use both these terms interchangeably but they do have a difference.

- A parameter is a variable in the function definition.
- An argument is a value passed during a function call.

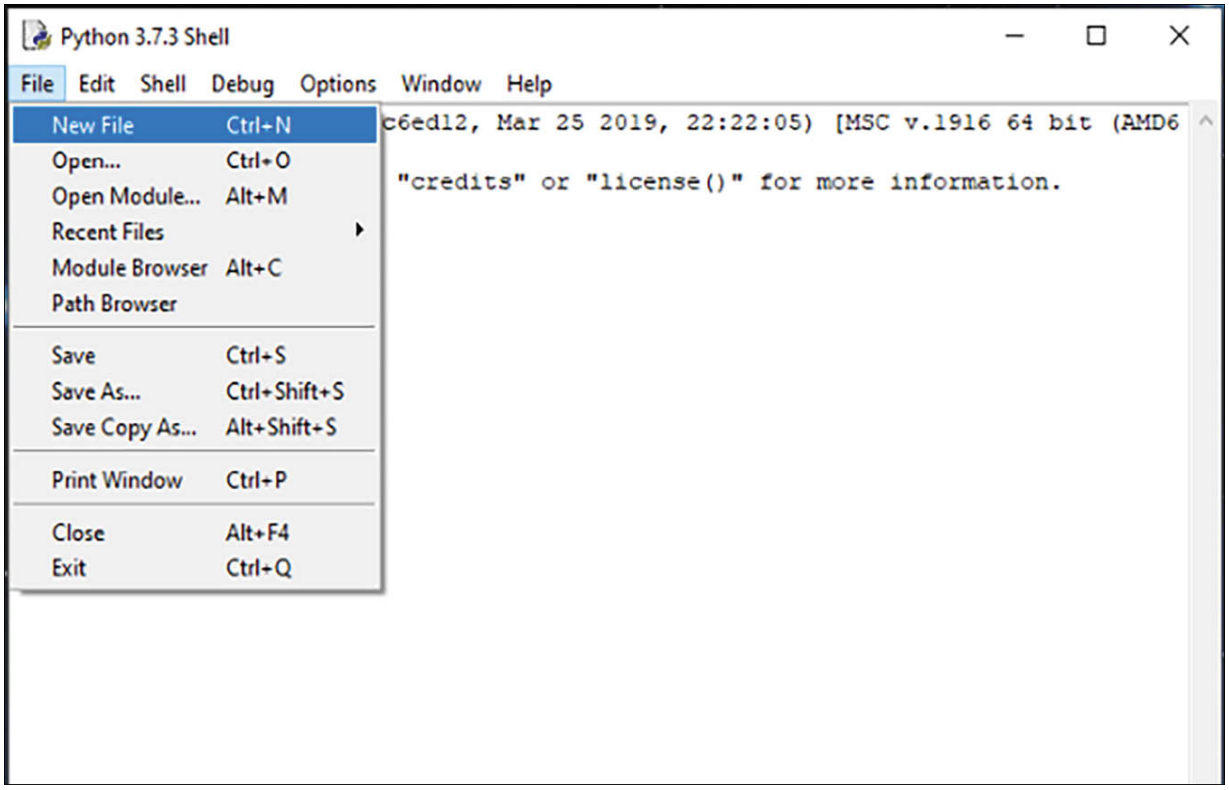
You can also create your own functions. These functions are called user-defined functions. For example:

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

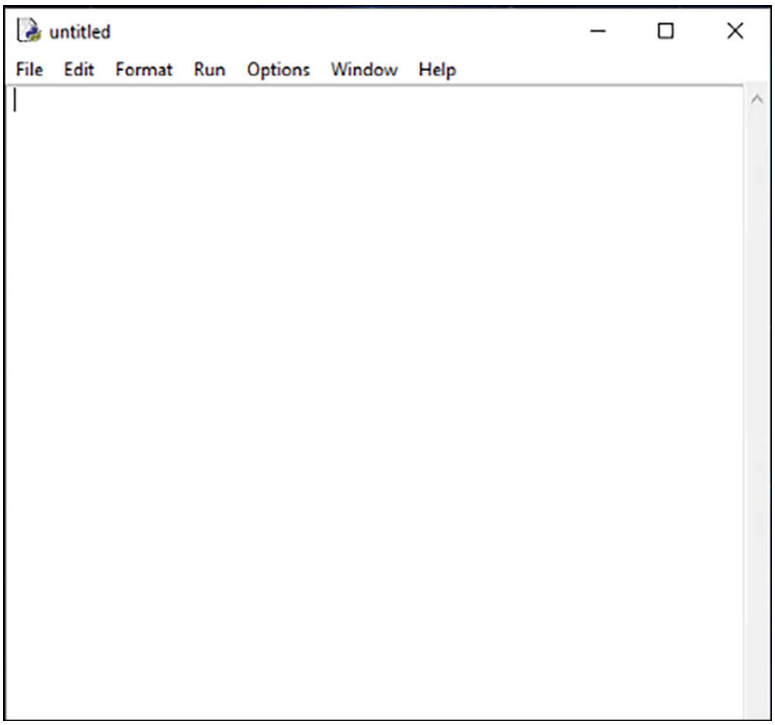
We'll cover functions in chapter 6.

Writing a Program

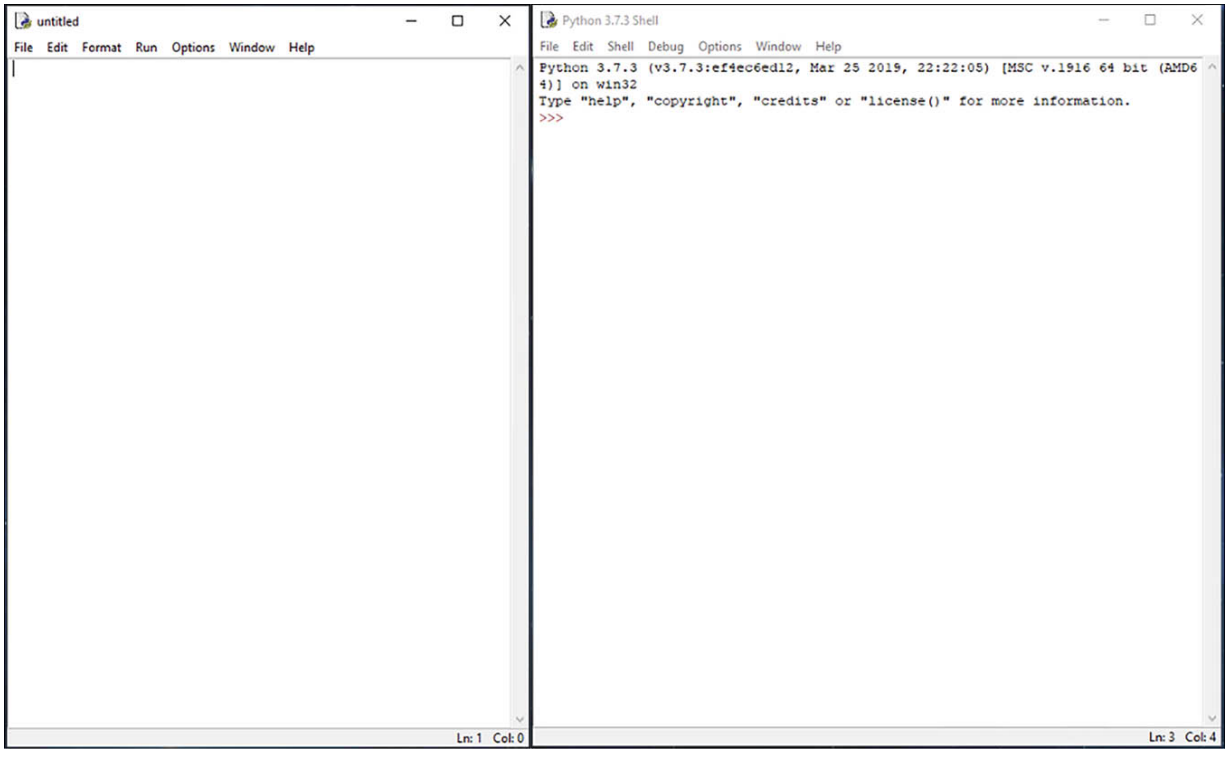
To write a program, open IDLE Python from the start menu. Select the file menu then click ‘new file’.



A new blank window will appear. This is the code editor. Here, you can write all your Python code.

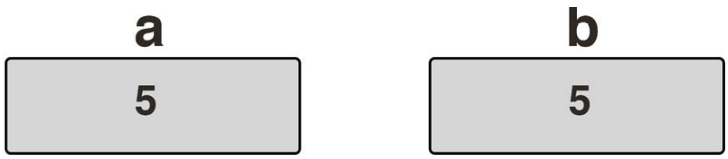


Arrange your windows as shown below, with the Python Shell on the right hand side - this is where you'll see the results of your programs. Put the code editor window next to the Python Shell window on the left.



For our first program, we're going to write something that adds two numbers together, then displays the result.

First, we need two variables to store the numbers.

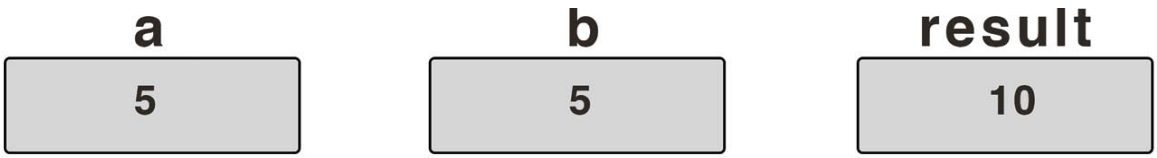


We'll use 'a' and 'b'. We'll assign the number 5 to each variable.

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

Next we need a piece of code that will add the two numbers together and store the result.

In this case the values assigned to the variables 'a' and 'b' will be added together and stored in the variable 'result'.



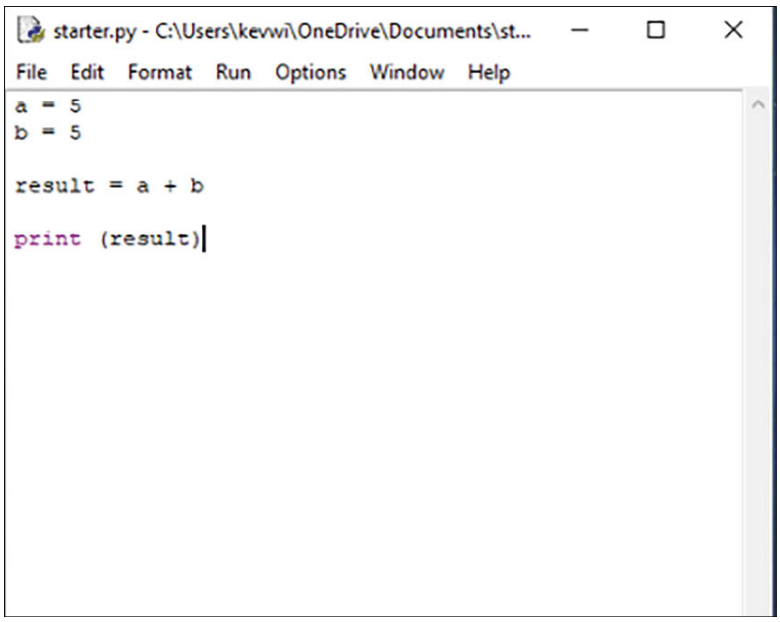
So...

```
result = a + b
```

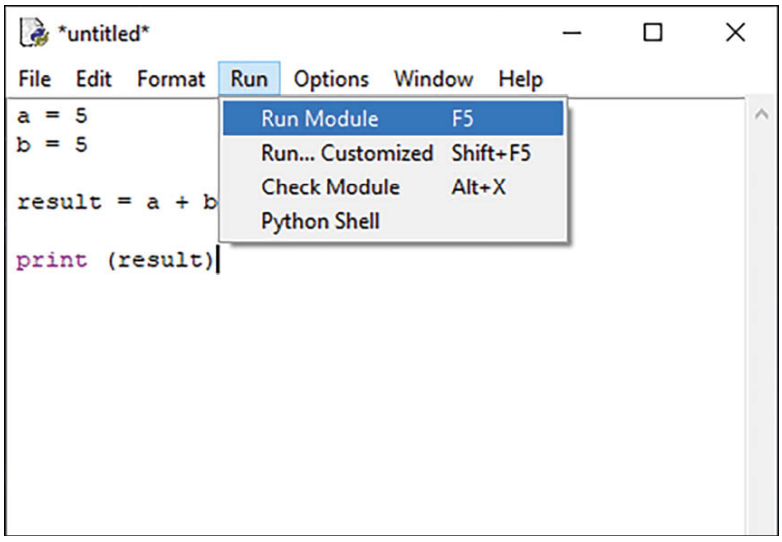
Next we'll need a function to print the result on the screen.

```
print (result)
```

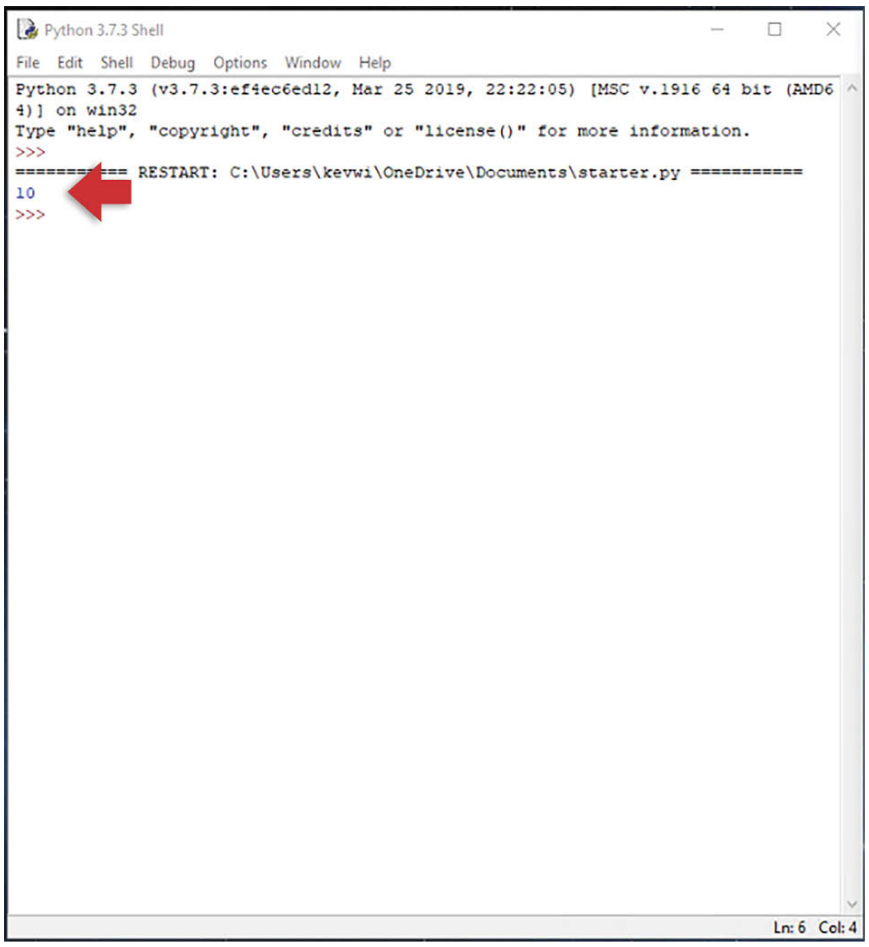
Lets put it all together in a program.



To run the program, press F5, or go to the ‘run’ menu in your code editor and click ‘run module’.



You can see in the image below, the output of the program. In this case ‘10’.



This particular program isn't very useful. It would be much better if we could allow the user to enter the numbers they want to add together. To do this, we'll need to add a function that will prompt the user for a value.

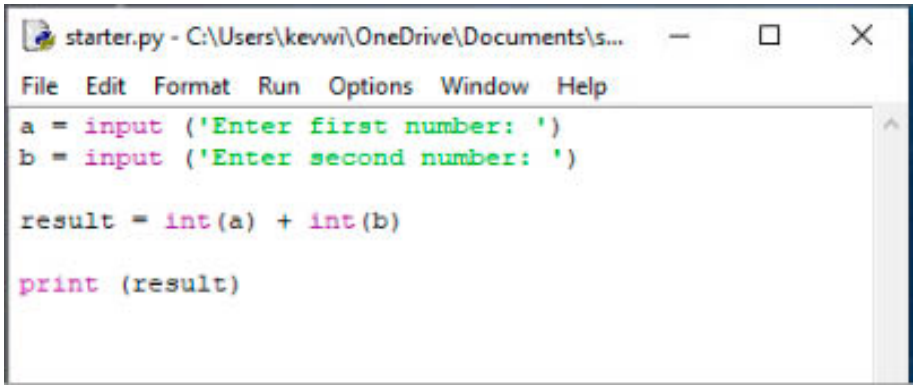
We'll use the `input()` function. We can replace the variables 'a' and 'b' from the previous program with the input function.

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

Now, because the input function reads the values entered as text (called a string), we need to convert these to numbers. So we need to modify the code that adds the two numbers together. We can use the `int()` function - this converts the text to an integer which is a fancy name for a whole number.

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

Lets put it all together in a program.



You can see in image below, the output of the program. The program prompted the user for two numbers, added them together, then displayed the result underneath.

Lab Exercises

1. What is the output produced by the following code fragment?

```
num1 = 2

num2 = 3

print (num1 + num2)
```

2. What is the output produced by the following code fragment?

```
num1 = 2

num2 = 3

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

3. Find the errors in the following program

```
Num1 = 2

num2 := 3

Sum = num1 + num2;

printf(sum)
```

4. What is an identifier?
5. Which of the identifiers below is valid and which are invalid? Why?

```
Num1

time-of-day

tax_rate

x5

int

7th_Rec

yield
```

6. How do you write comments in your code? Explain with an example.
7. Why should you include comments?
8. What is a reserved word?
9. What is a low level and a high level language?

10. What is object oriented programming?

Working with Data

You can store and manipulate all different types of data: a number, a string, list, and so on. To store these we use a variable which is a temporary container for storing data values.

With Python, you don't need to declare all your variables before you use them as they are automatically declared when you first assign some data.

Variables can contain different types of data such as a string, number, list of items and so on.

For this section, take a look at the video demos

`elluminetpress.com/pybasics`

You'll also need the source files in the directory Chapter 03.

Basic Data Types

A variable can store various types of data, called a data type. Lets introduce some of the basic types we'll be looking at.

Integers

An integer (int) is a whole number, and can be positive or negative. Integers can usually be of unlimited length.

```
score = 45
```

Floating Point Numbers

A floating point number (float), sometimes called a real number, is a number that has a decimal point.

```
temperature = 44.7
```

Strings

In Python code, a string (str) must be enclosed in quotes “...” or ‘...’, and you can assign a string to a variable using the '=' sign.

```
name = “John Myers”
```

If you want to assign a multiline string, enclose in triple quotes"""

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

Concatenation

Concatenation means to combine two strings. To do this, can use the + operator.

```
wordOne = "Hi! "

wordTwo = "My name is..."

sentence = wordOne + wordTwo
```

The sentence would be the result of the two strings joined together:

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

Slice

You can slice a string, meaning you can return a range of characters

from a string. For example, if we wanted to slice the following

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

We specify the start index and the end index separated by a colon.

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

Here we're slicing 4 to 8 (the last position is not included)

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

This will return

```
MINE
```

String Methods

Python provides numerous methods to operate on strings. For example if we wanted to count how many times 'L' appears in the string

```
Str = 'ELLUMINET'
```

You can use the count() method

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

Here are a few other common string methods. Try them out.

Function	Description	Example
count()	Returns the number of times a specified value occurs in a string	`Str.count("string to count")`
find()	Searches the string for a specified value and returns the position of where it was found	`Str.find("chars to find")`
format()	Formats specified values in a string	`Str.format()`
join()	returns a new string that is a concatenation of strings.	`Str.join("string 1", "string 2"…)`
lower()	Convert a string to lowercase.	`Str.lower()`
replace()	Creates a new string by replacing some parts of another string.	`Str.replace("value to replace", "new value")`
split()	Splits a string into a list of strings based on a delimiter such as a comma.	`Str.split()`
upper()	Convert a string to uppercase.	`Str.upper()`

Escape Characters

An escape character tells the interpreter to perform a specific operation such as a line break or tab or a reserved character such as a quote mark or apostrophe. Escape characters start with the a backslash (\) and are used to format a string.

Escape Character	Function
\n	Line break
\t	Tab (horizontal indentation)
\	New line in a multi-line string
\\	Backslash
\'	Apostrophe or single quote
\"	Double quote

For example, you could use the tab escape and break line character to format some text.

```
print("John \t 45 \nJoanne \t 15")
```

The output to this line would look something like this:

```
John 45
Joanne 15
```

Formatting Strings

Formatting output using `.format()` method uses { } to mark the substitution of a variable within an output string within the `print()` function.

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

These are known as format fields and are replaced with the objects that are passed into `.format()` method. We can use number in the brackets to refer the position of the object passed into `format()` method.

```
print('Time is: {0}, day is {1}'.format(time, day))
```

String Characters

We can access individual characters using an index. Remember, the index starts from 0.

If we declared

```
Str = 'ELLUMINET'
```

We would get something like this:

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

To index any of the characters the string name with the index in square brackets.

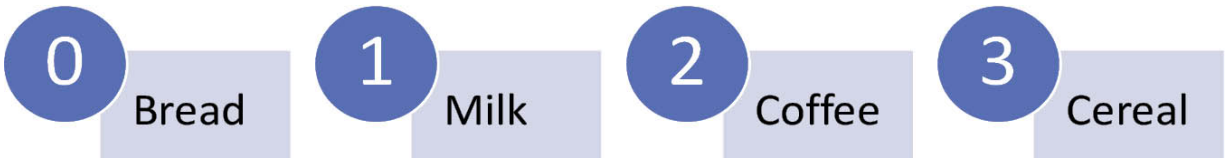
```
Str[index]
```

So in the example below, Chr would = 'L'

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

Lists

A list is an ordered sequence of data items usually of the same type, each identified by an index (shown below in the circles). This is known as a one dimensional list.



Lists are known as arrays in other programming languages and you can create one like this - list elements are enclosed in square brackets [ ]:

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

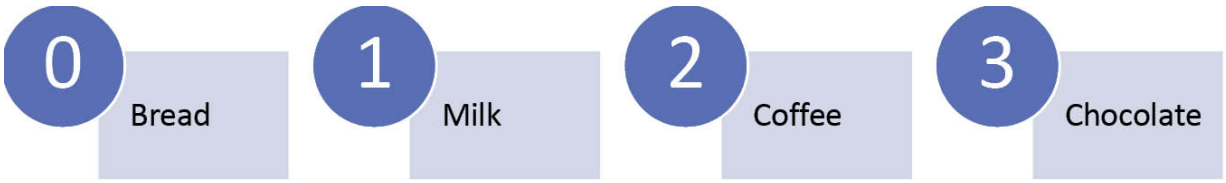
To reference an item in a list, put the reference in square brackets:

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

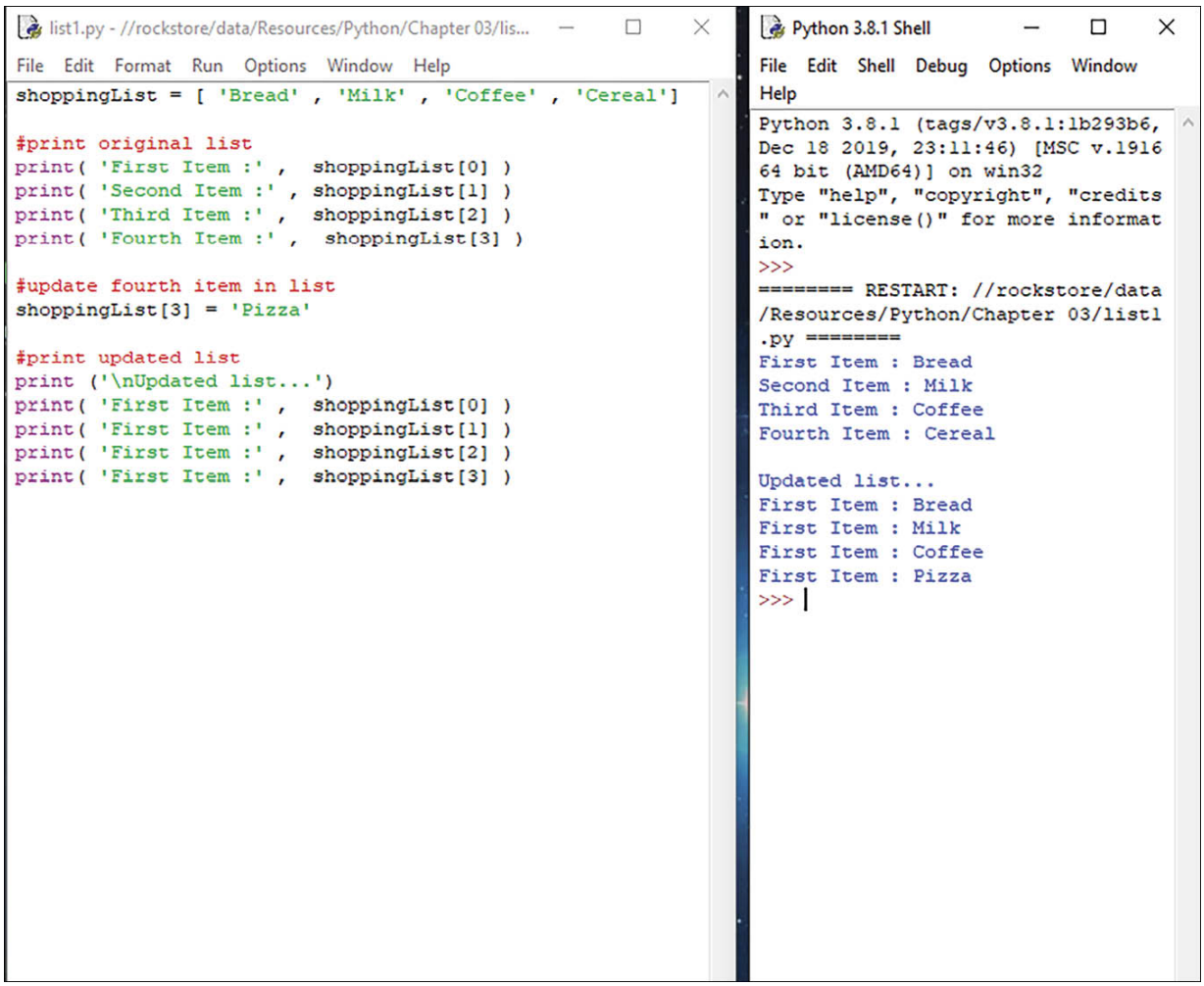
You can assign another value to an item in the list (eg change cereal)

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

You would end up with something like this



Let's look at a program. Open the file list1.py. Here, we've created a list and initialized it with some data.



We can output data from the list using a print statement and a

reference to the item in the shopping list.

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

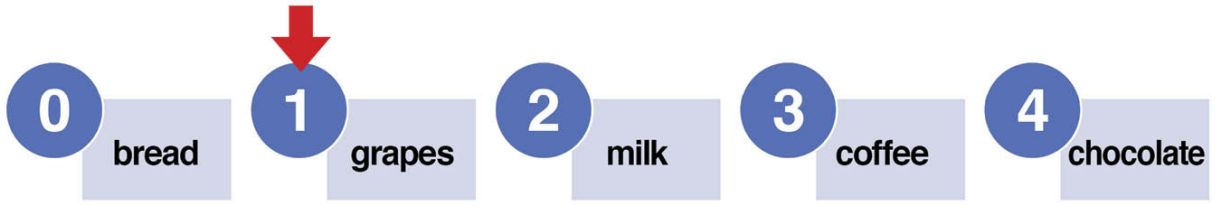
We can also update an item in the list using the reference.

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

To insert an item – specify index to insert item at. For example to insert 'grapes' at index 1

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

This inserts grape at index[1] and shifts items after insert point to the right.



To append an item, use `append()`. This adds the item to the end of the list.

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

To remove an item by value

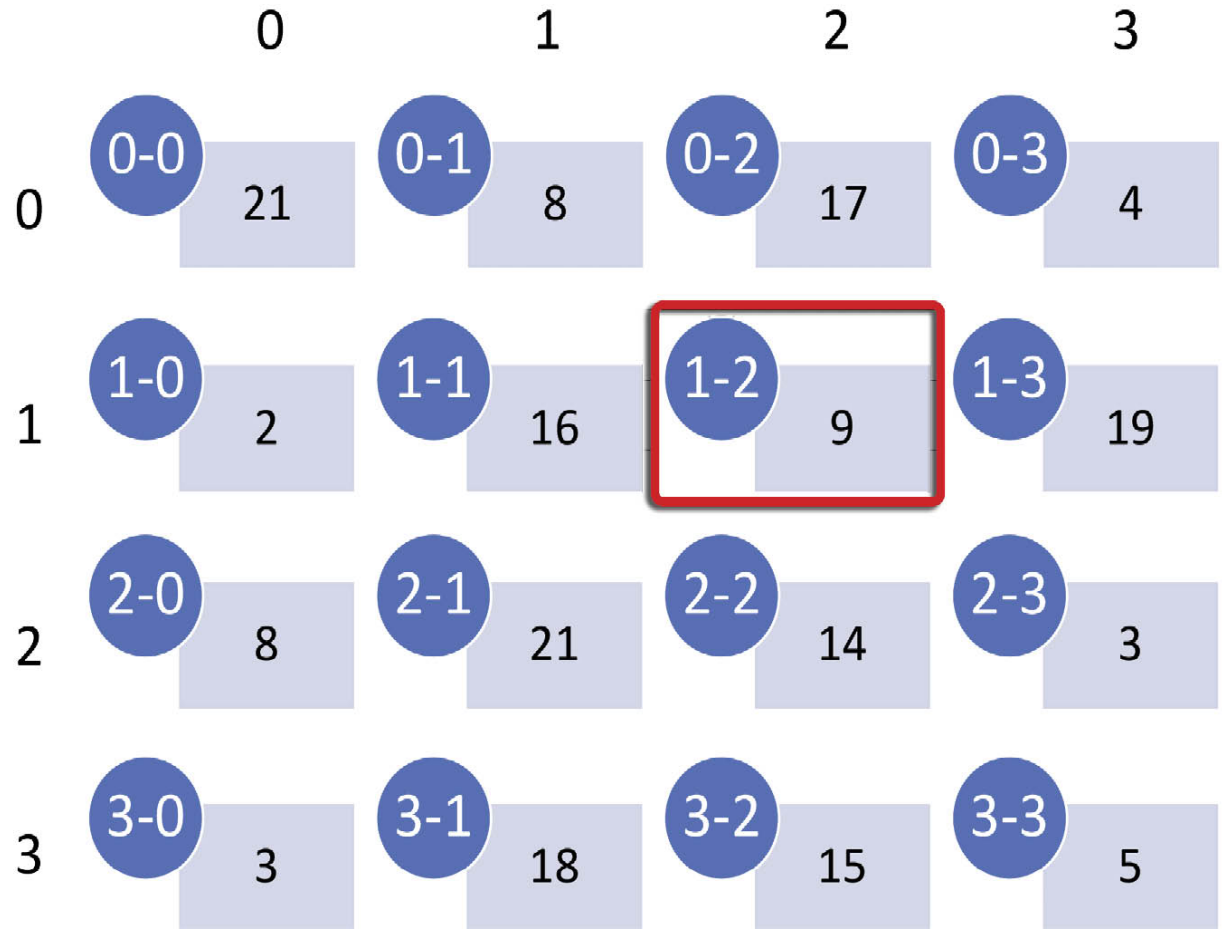
```
shoppingList.remove('blueberry')
```

Remove item at specified index

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

Two Dimensional Lists

Two dimensional lists are visualized as a grid similar to a table with rows and columns. Each column in the grid is numbered starting with 0. Similarly, each row is numbered starting with 0. Each cell in the grid is identified by an index - the row index followed by the column index (shown below in the circles).



You could declare the above list as

```
scoreSheet = [
[ 21, 8, 17, 4 ],
[ 2, 16, 9, 19 ],
```

```
[ 8, 21, 14, 3 ],
[ 3, 18, 15, 5 ]
]
```

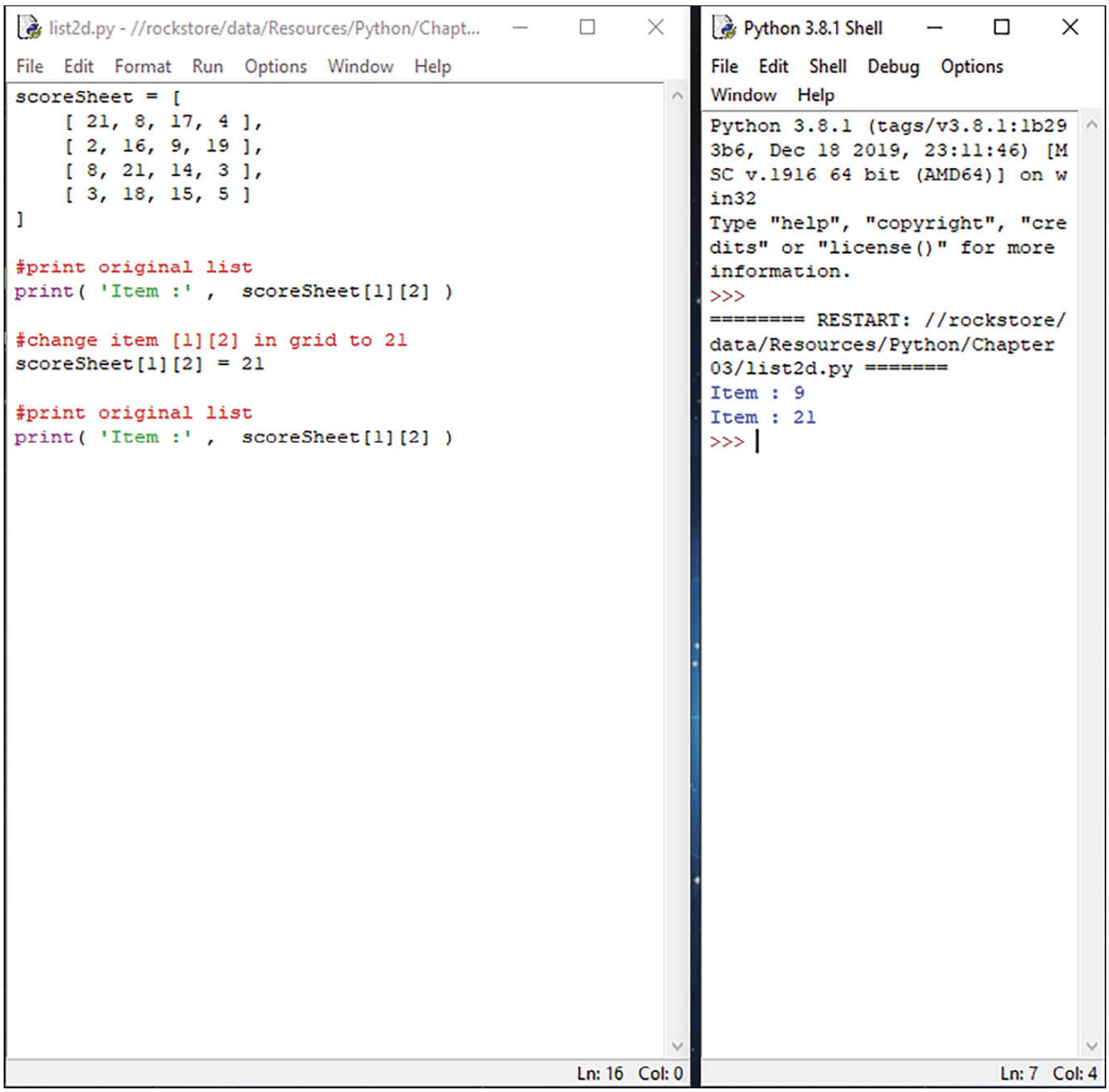
To reference an item in a two dimensional list, put both the references in square brackets (first the row index, then the column index):

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

You can change items in the list, put both the references in square brackets (first the row index, then the column index), then assign the value:

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

Lets take a look at a program. Open the file list2d.py. Here, we've declared our shoreSheet list and initialized it with some data.



We can add an item to a particular location in the list

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

We can also output data stored at a particular location

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

Sets

A set is an unordered collection of unique items enclosed in curly braces { }. Sets can contain different types.

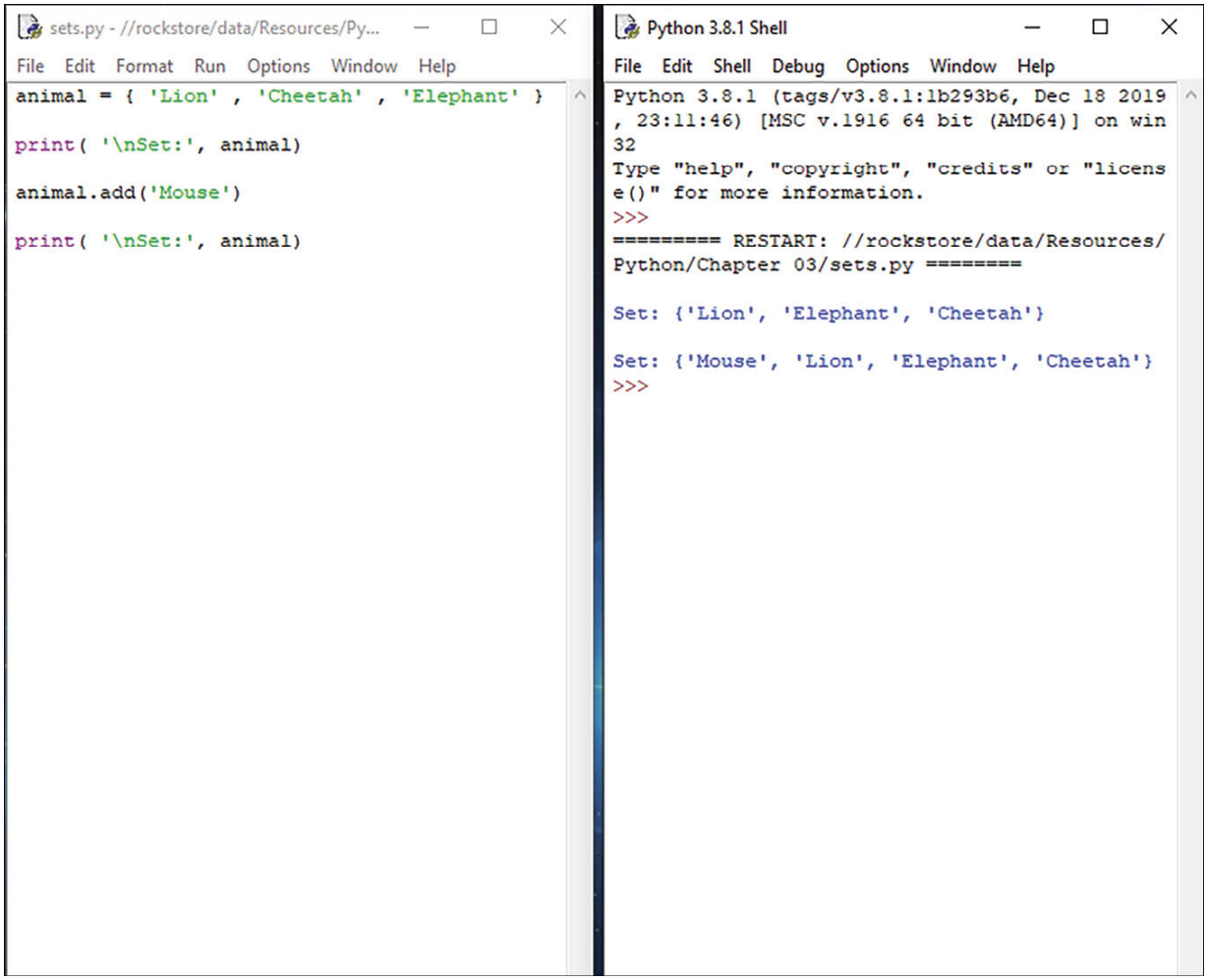
You can create a set like this:

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

Two things to note about sets. Firstly, you can't index individual elements in the set as it is an unordered data type. Secondly, you can't change individual values in the set like you can with lists. However, you can add or remove items. Use the `.add()` method, type the data to add, in the parentheses.

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

Lets take a look at a program. Open the file sets.py. Here, we've created a set with some animal names. We can output out the data in the set.



We can also add an item to the set using the `.add()` method.

Tuples

A tuple is similar to a list and is a sequence of items each identified by an index. As with lists, the index starts with 0 not 1.

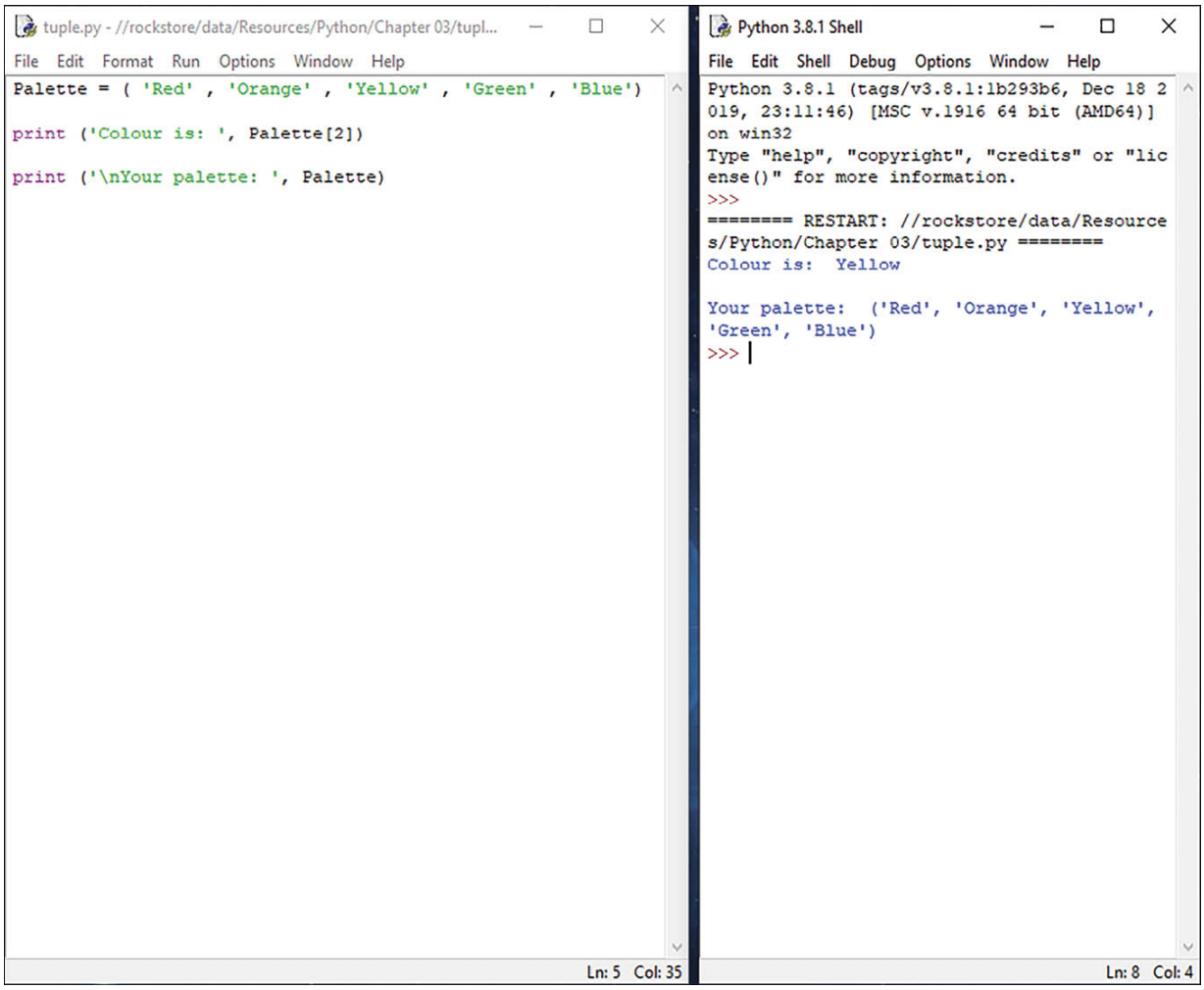
In contrast to lists, items in a tuple can't be changed once assigned and tuples can contain different types of data.

To create a tuple enclose the items inside parentheses ( ).

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

Use a tuple when you want to store a data of a different type, such a sign in details for your website.

Lets take a look at a program. Open the file tuple.py. Here, we've created a tuple with some colors.



We can output out the data in the tuple using a print statement.

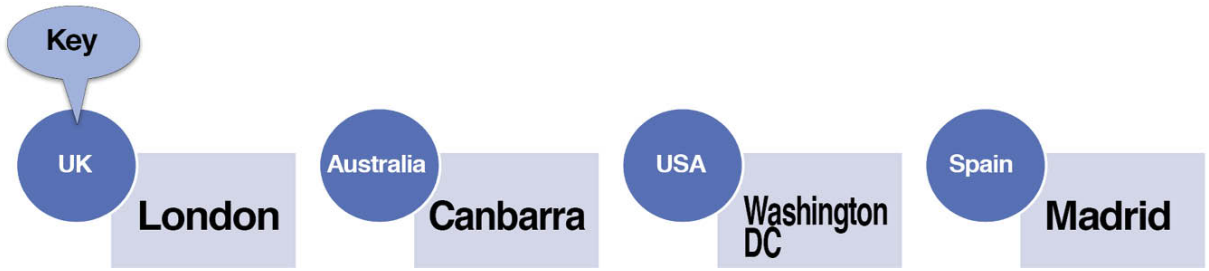
```
print (Palette[2])
```

Dictionaries

A dictionary is an unordered collection of items, each identified by a key. To create a dictionary, enclose the items inside braces { }. Identify each item with a key using a colon.

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

When we declare the above, we can visualize it like this:

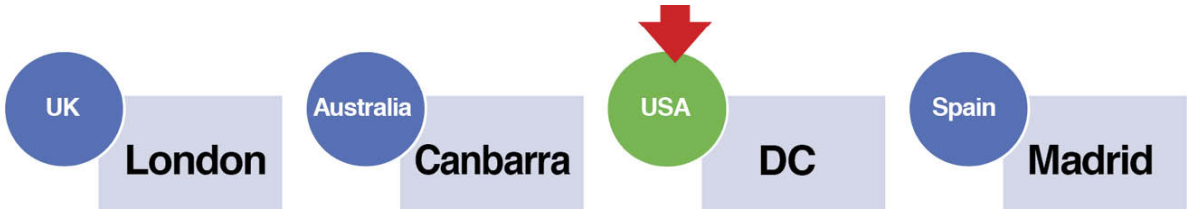


To read a value, state the dictionary name, then put the key in square brackets.

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

To change a value, state the dictionary name, put the key in square brackets. Then assign the value using '='.

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



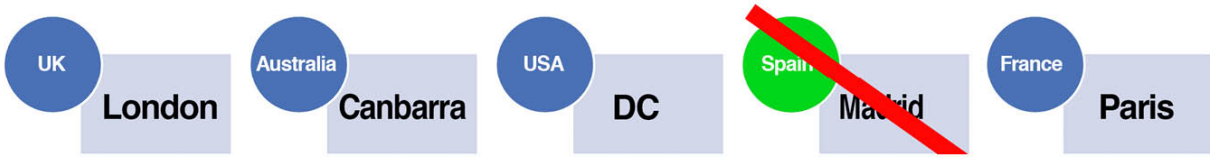
To add an item, use the .update() method. Enclose the data to add in braces `{'Key' : 'Data'}`. If the key doesn't

exist, it will add the item to the end.

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

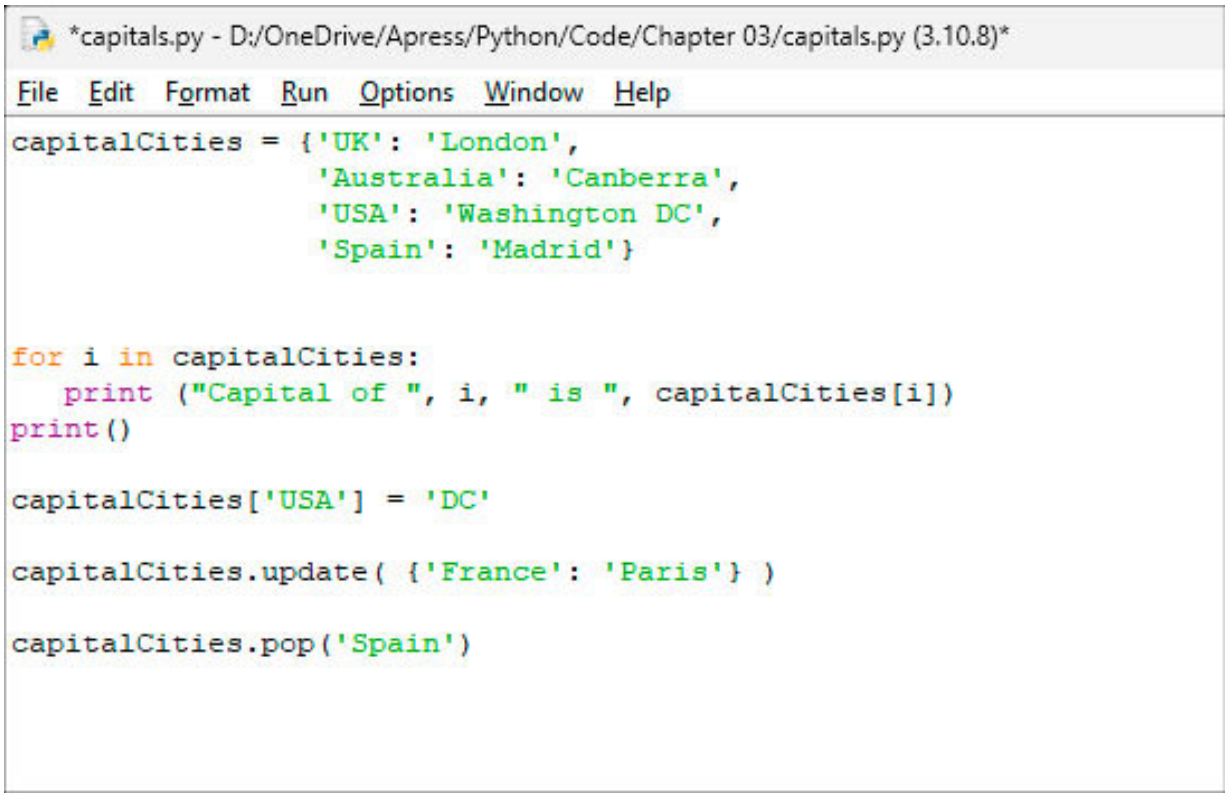
To remove item, specify key of the item to remove 'spain'.

```
capitalCities.pop('Spain')
```



We'll end up with this:

Lets take a look at a program. Open the file capitals.py. Here, we've created a dictionary with some data. We've looped through the dictionary using a for loop to print out each item in the dictionary.



We updated 'USA' to 'DC' by assigning 'DC' to the dictionary at key 'USA'.

We've also used the `.update()` method to add 'france'.

Finally we've used the `.pop()` method to delete 'spain'.

Casting Data Types

Variables can contain various types of data such as text (called a string), a whole number (called an integer), or a floating point number (numbers with decimal points).

With Python, you don't have to declare all your variables before you use them. However, you might need to convert variables to different types. This is known as type casting.

Python has two types of type conversion: implicit & explicit.

With implicit type conversion, Python automatically converts one data type to another.

With explicit type conversion, the programmer converts the data type to the required data type using a specific function. You can the following functions to cast your data types:

- `int( )` converts data to an integer
- `long( )` converts data to a long integer
- `float( )` converts data to a floating point number
- `str( )` converts data to a string

For example, you could use the **input( )** function to prompt the user for some data

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

This example would prompt the user for some data, then store the data in the 'a' variable as a string.

This might sound ok, but what if we wanted to perform some arithmetic on the data? We can't do that if the data is stored as a string. We'd have to type cast the data in the variable as an integer

or a float.

```
int(a)
```

or

```
float(a)
```

Arithmetic Operators

Within the Python language there are some arithmetic operators you can use.

Operator	Description
**	Power, indices
/	Divide
*	Multiply
+	Add
-	Subtract

Operator Precedence

BIDMAS (sometimes called BODMAS) is an acronym commonly used to remember mathematical operator precedence - ie the order in which you evaluate each operator.

1. Brackets **( )**
2. Indices (sqrt, power, squared^2, or cubed^3, etc) ******
3. Divide **/**
4. Multiply *****
5. Add **+**
6. Subtract **-**

Performing Arithmetic

If you wanted to add 20% sales tax to a price of $12.95, you could do something like this...

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

According to the precedence list above, you would first evaluate the ‘divide’ operator:

```
20 / 100 = 0.2
```

Next is multiply

```
12.95 * 0.2 = 2.59
```

Finally addition

12.95 + 2.59 = 15.54

Comparison Operators

These are used to compare values and are commonly used in conditional statements or constructing loops.

Operator	Description
==	Equal to
!=	Not equal to
>	Greater that
<	Less that
>=	Greater than or equal to
<=	Less than or equal to

For example, comparing two values in an ‘if’ statement, you could write something like this:

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

Boolean Operators

Also known as logical operators and are commonly used in conditional statements (if...) or constructing loops (while... for...). We'll look at if statements, and loops in chapter 4.

Operator	Description
and	Returns true if both the operands are true
or	Returns true if either of the operands is true
not	Returns true if operand is false

For example, you could join to two comparisons in an ‘if’ statement using ‘and’, like this:

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

Using the ‘and’ operator would mean both conditions (a >= 0) and (a <= 10) must be true.

Bitwise Operators

Bitwise operators are used to compare binary numbers:

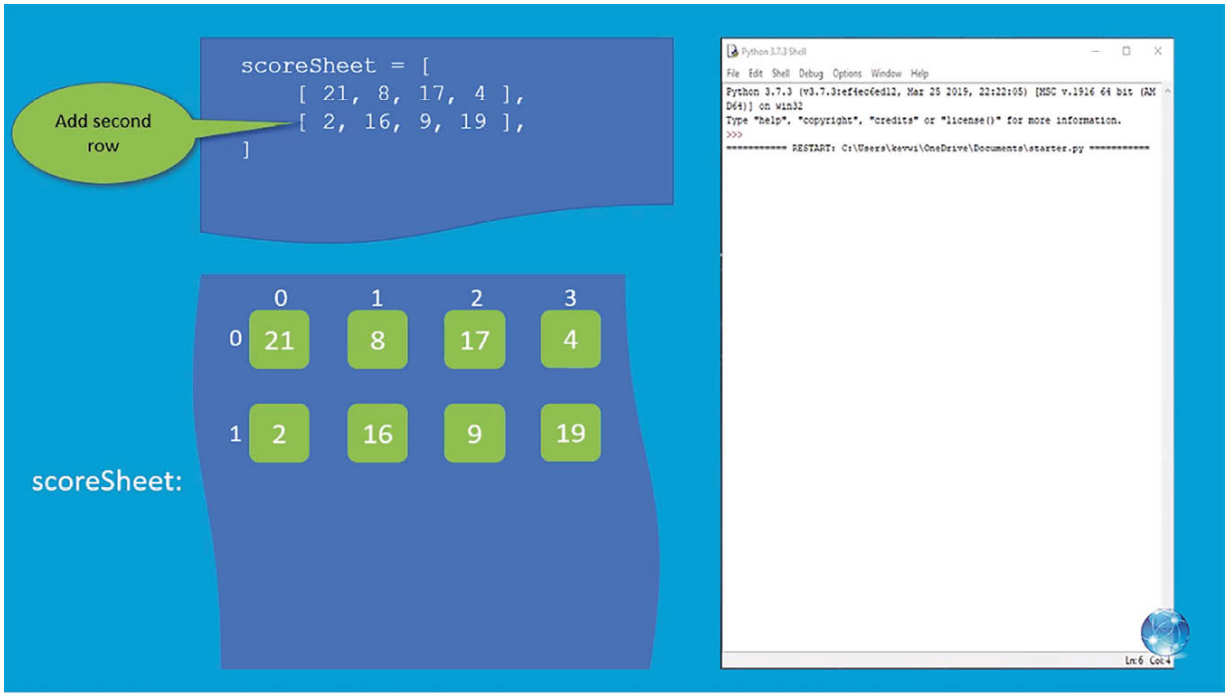
Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Left shift	Shift bits to the left
>>	Right shift	Shift bits to the right

You can apply the bitwise operators

```
a >> 2 #shift bits of 'a' left by 2 units
a << 2 #shift bits of 'a' right by 2 units
a & b #perform AND operation on bits
```

Have a look at the video demos and familiarize yourself with assigning variables, data types, lists, tuples, and dictionaries.

`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\kevwi\OneDrive\Documents\starter.py ==========
Ln: 6 Col: 4


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

Lab Exercises

1. Write a program that accepts a length in inches and prints the length in centimeters (1 inch = 2.54cm).
2. Write a program that accepts your forename, surname and year of birth and adds them to an array.
3. Write a program that adds some employee data to a dictionary. Use an employee number as the key.
4. Write a program that converts temperatures from Celsius to Fahrenheit.
   $F = C \times 9/5 + 32$
5. Write a program that calculates the volume of a sphere
   $V = 4/3\ \pi r^3$
6. Write a program to calculate and display and employee's gross and net pay. In this scenario, tax is deducted from the gross pay at a rate of 20% to give the net pay.
7. Write a program that stores a shopping list of 10 items. Print the whole list to the screen, then print items 2 and 8.
8. Extend the previous program, to insert an item into the list.
9. What is a Boolean operator? Write a program to demonstrate.
10. What is a comparison operator? Write a program to demonstrate.
11. What is data type casting? Why do we need it? Write a program to demonstrate.

Flow Control

Flow control is controlling the order in which statements or function calls of a program are executed.

There are three control structures: Sequence, Selection and Iteration

Python has various control structures such as while loops, for loops, and if statements, which are used to determine which section of code is executed according to certain conditions.

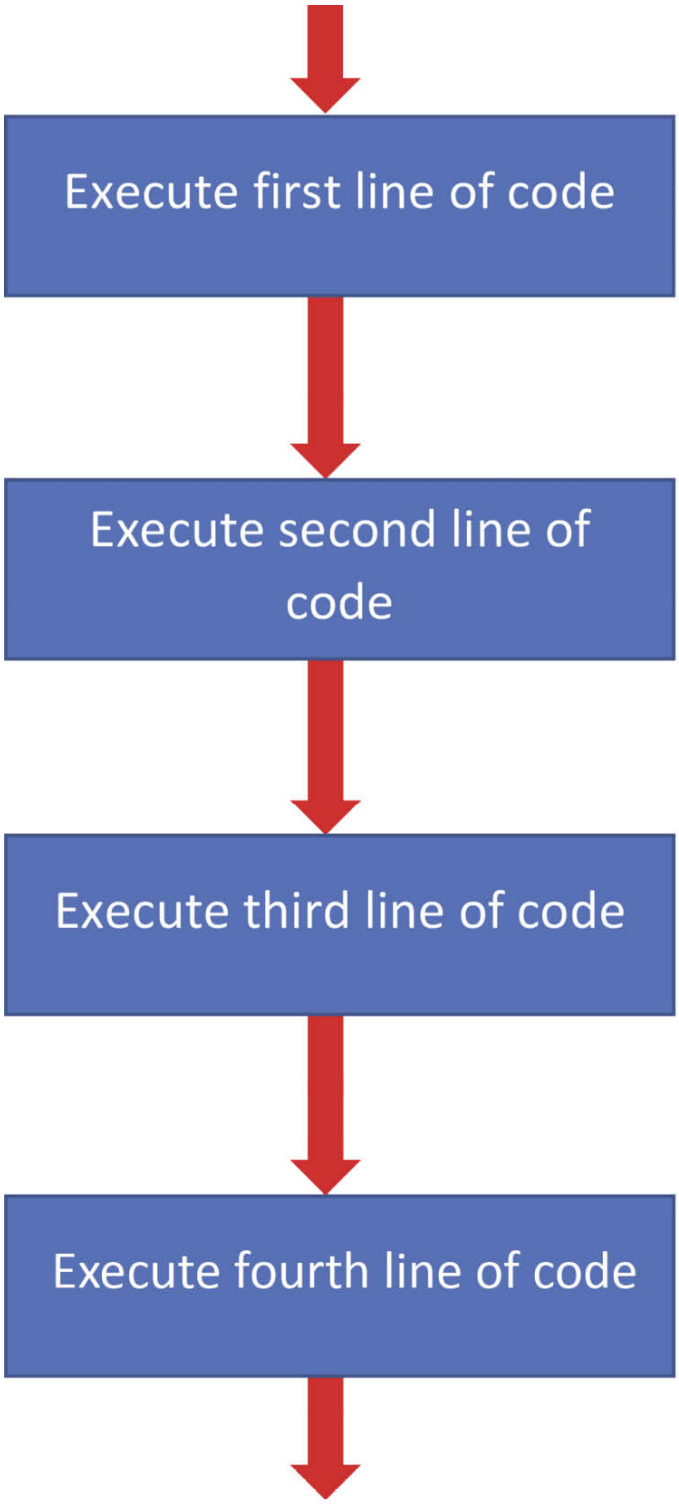
For this section, take a look at the video demos

`elluminetpress.com/pythonflow`

You'll also need the source files in the directory Chapter 04.

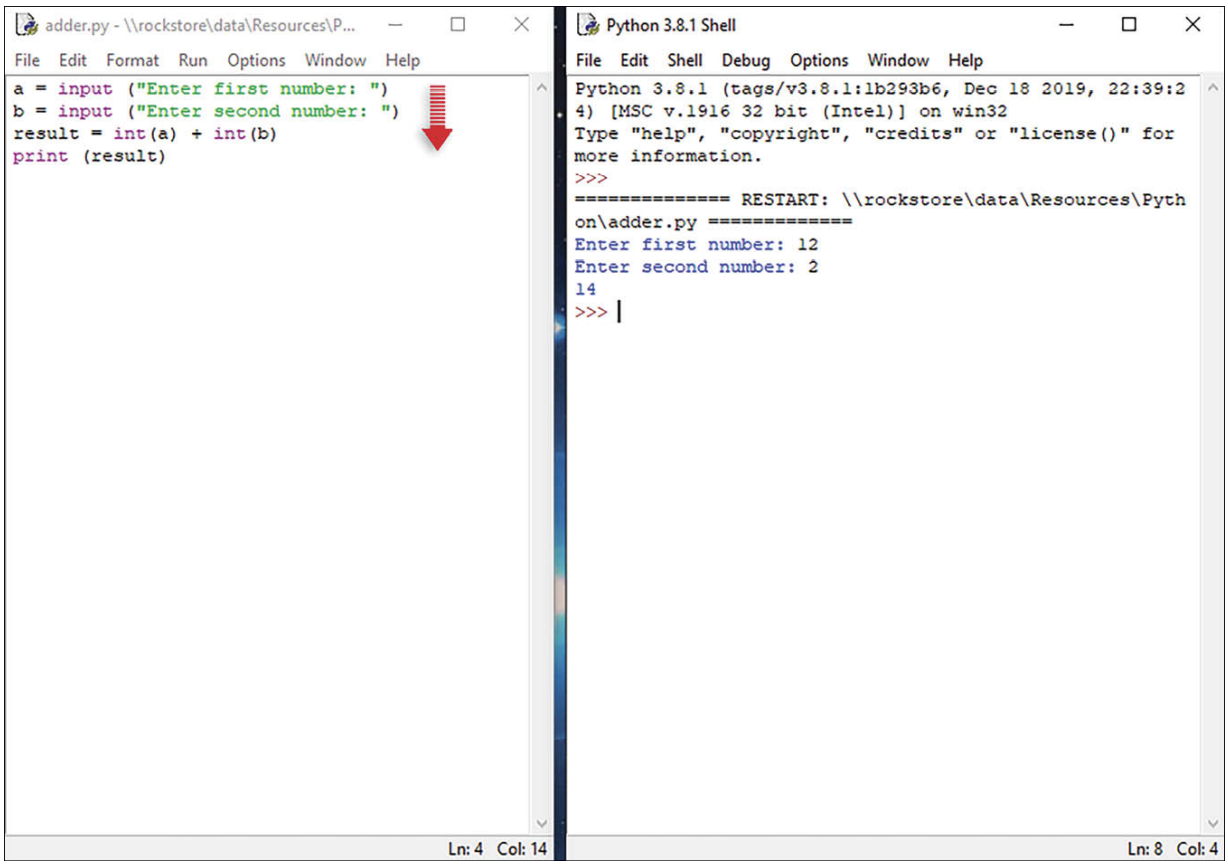
Sequence

A computer program is a set of step-by-step instructions that are carried out in sequence to achieve a task or solve a problem. The sequence can contain any number of instructions, but no instruction can be skipped in the sequence.



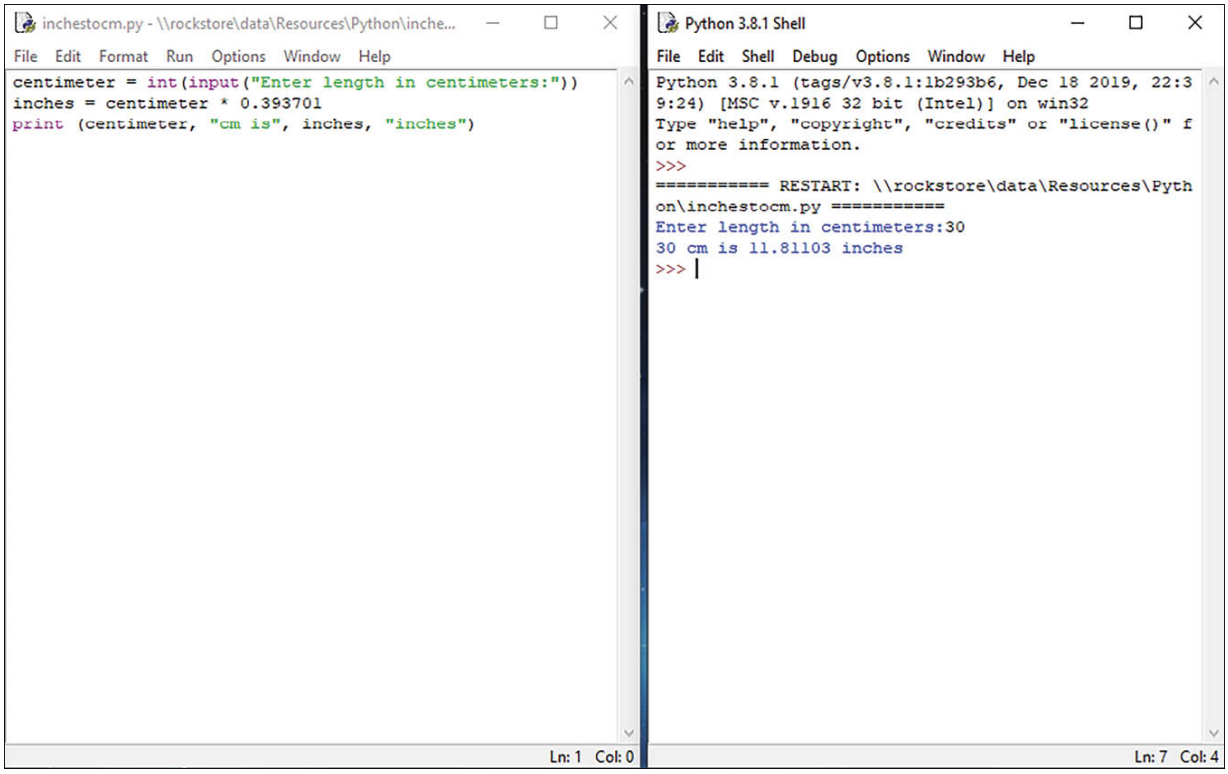
The interpreter will follow and execute each line of code in sequence until the end of the program.

Let’s have a look at a program. Open adder.py. This program has four statements.



Once you execute the program, the instructions are carried out in sequence.

Let’s try another example. Open inchestocm.py

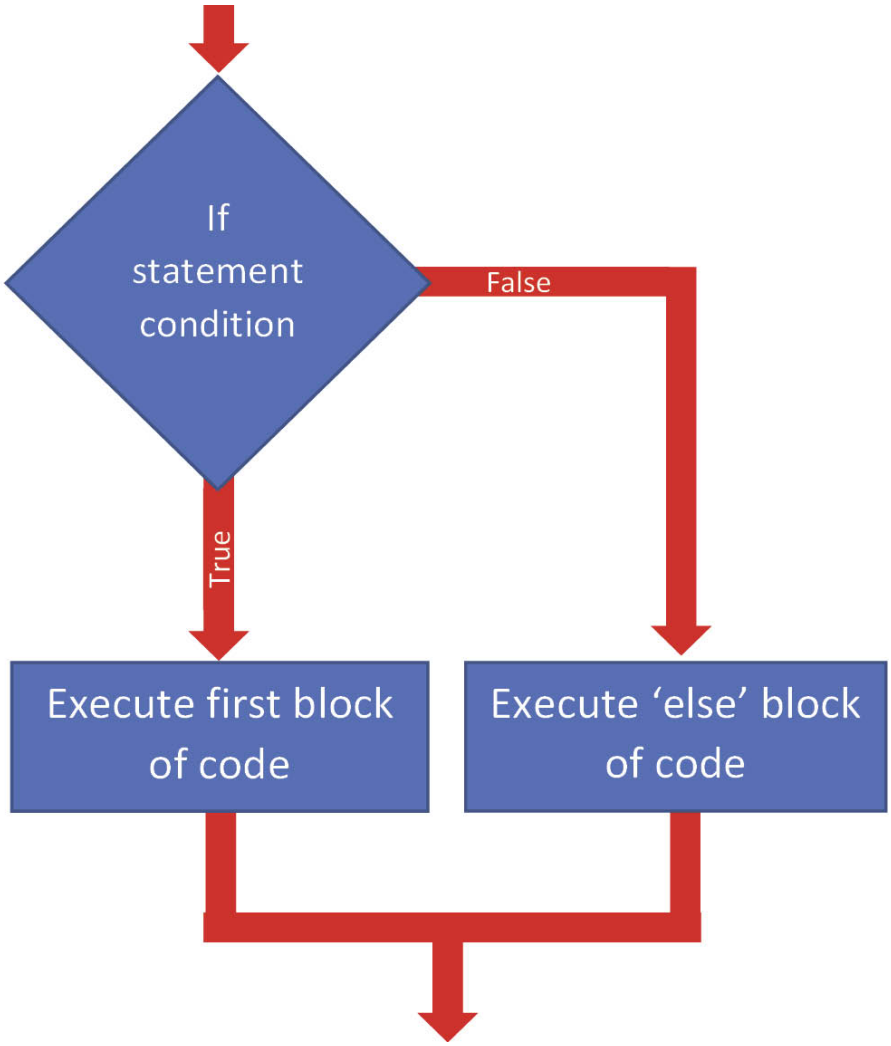

inchestocm.py - \\rockstore\data\Resources\Python\inche...
File Edit Format Run Options Window Help
centimeter = int(input("Enter length in centimeters:"))
inches = centimeter * 0.393701
print (centimeter, "cm is", inches, "inches")
Ln: 1 Col: 0
Python 3.8.1 Shell
File Edit Shell Debug Options Window 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\Python\inchestocm.py ==========
Enter length in centimeters:30
30 cm is 11.81103 inches
>>>
Ln: 7 Col: 4

Selection

In most computer programs, there are certain points where a decision must be made. This decision is based on a condition, and that condition can be either true or false.

if...else

If statements are used if a decision is to be made. If the condition is true, then the if statement will execute the first block of code, if the condition is false, the if statement will execute the 'else' block of code if included.

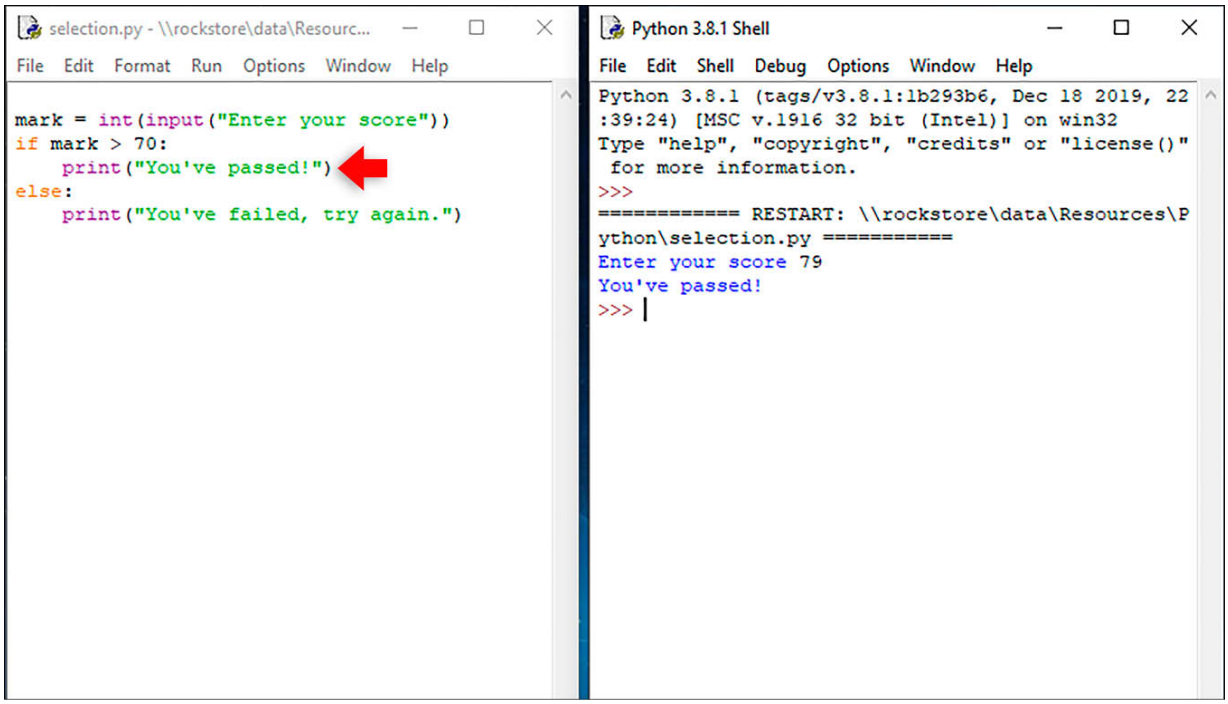


So for example:

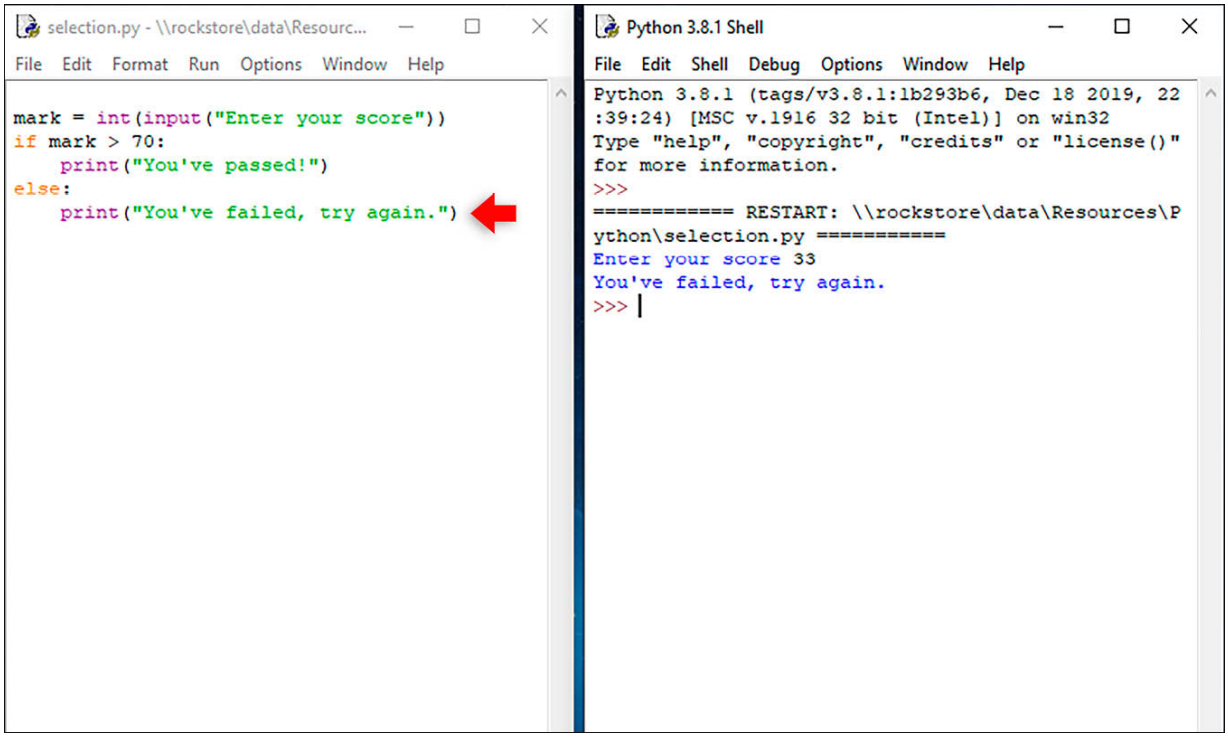
```
if num >= 0: #condition
print("Positive or Zero") #first block
else:
print("Negative number") #else block
```

Let's have a look at a program. Open selection.py. Here, we can see a very simple if statement to determine whether a test score is a pass or fail. The pass mark is 70, so we need an if statement to return a pass message if the value entered is greater than 70. Remember that we also need to cast the variable 'mark' as an integer (int).

If you enter a value greater than 70, the python interpreter will execute the first block of the 'if statement'.

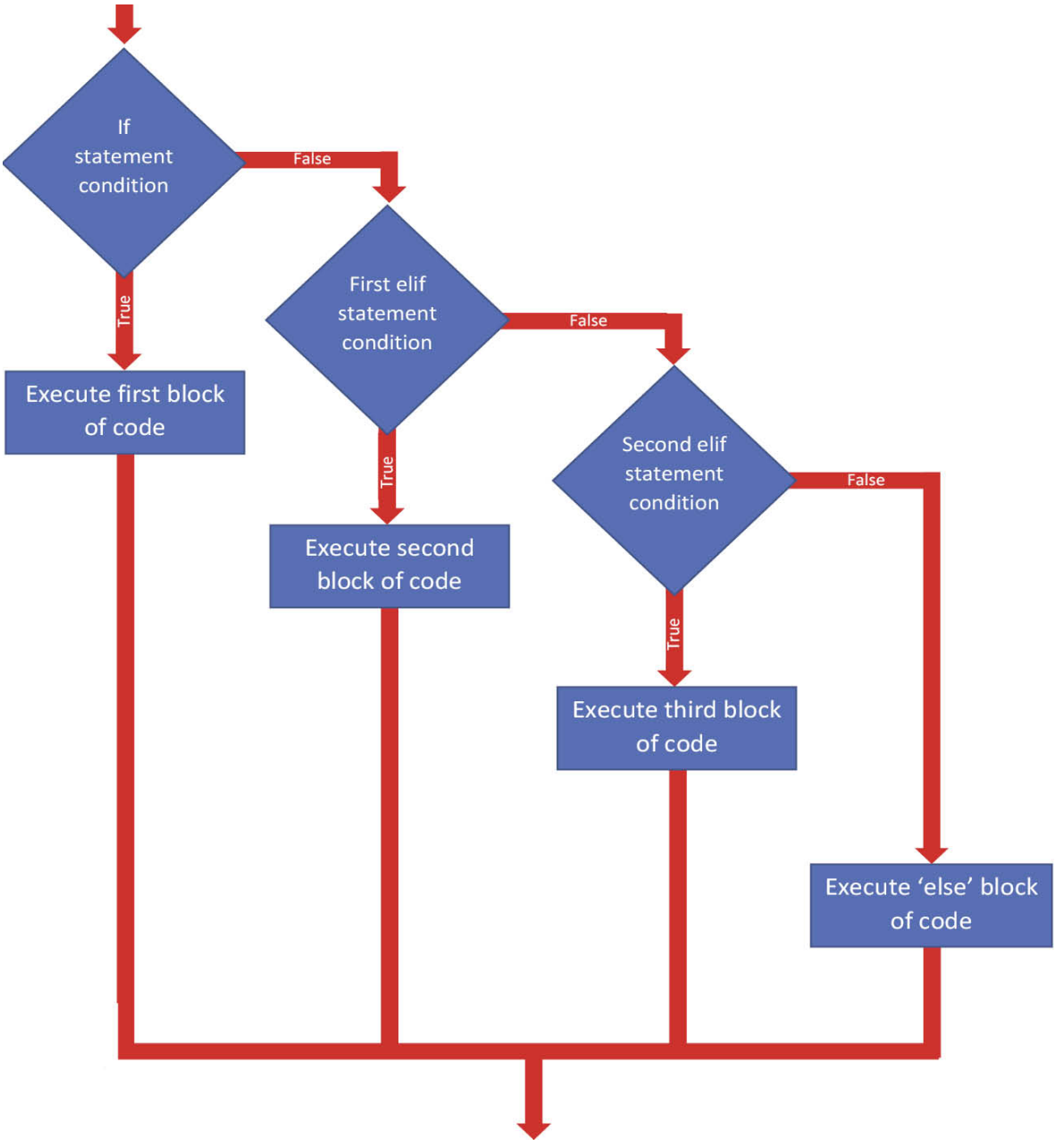


If you enter a value below 70, the python interpreter will execute the 'else block' of the 'if statement'.


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
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\selection.py ==========
Enter your score 33
You've failed, try again.
>>>

elif

Use the `elif` statement if multiple decisions are to be made. Each decision (or condition) will have a set of instructions to be executed.

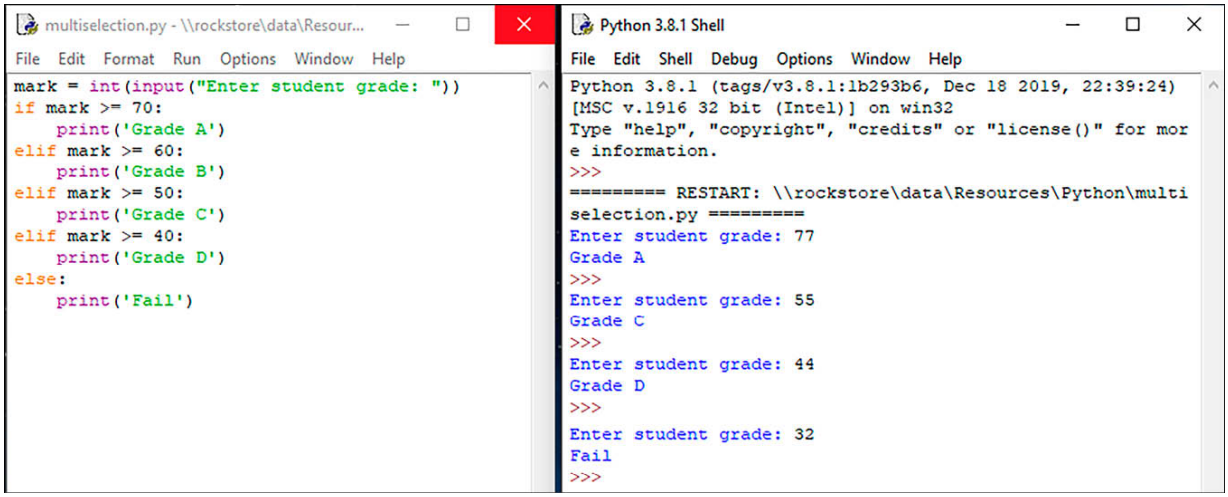


So for example:

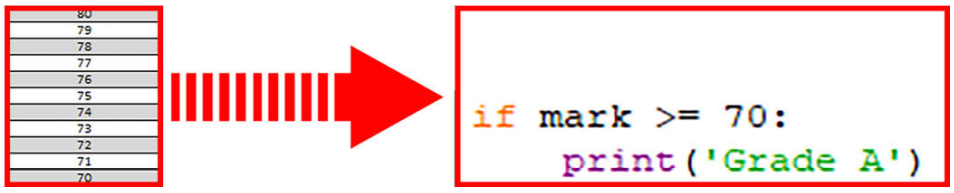
```
if condition: #if condition
[statements] #first block of code
```

```
elif condition: #first elif statement
[statements] #second block of code
elif condition: #second elif statement
[statements] #third block of code
else:
[statements] #else block of code
```

Let's have a look at a program. Open multiselection.py.



If we analyse the eilf statement, we can see how it works. For the first condition, any number entered above 70 will execute the first block.



Any number between 60 and 69, the interpreter will execute the second block.

Similarly for the other conditions. 50-59 and 40-49.

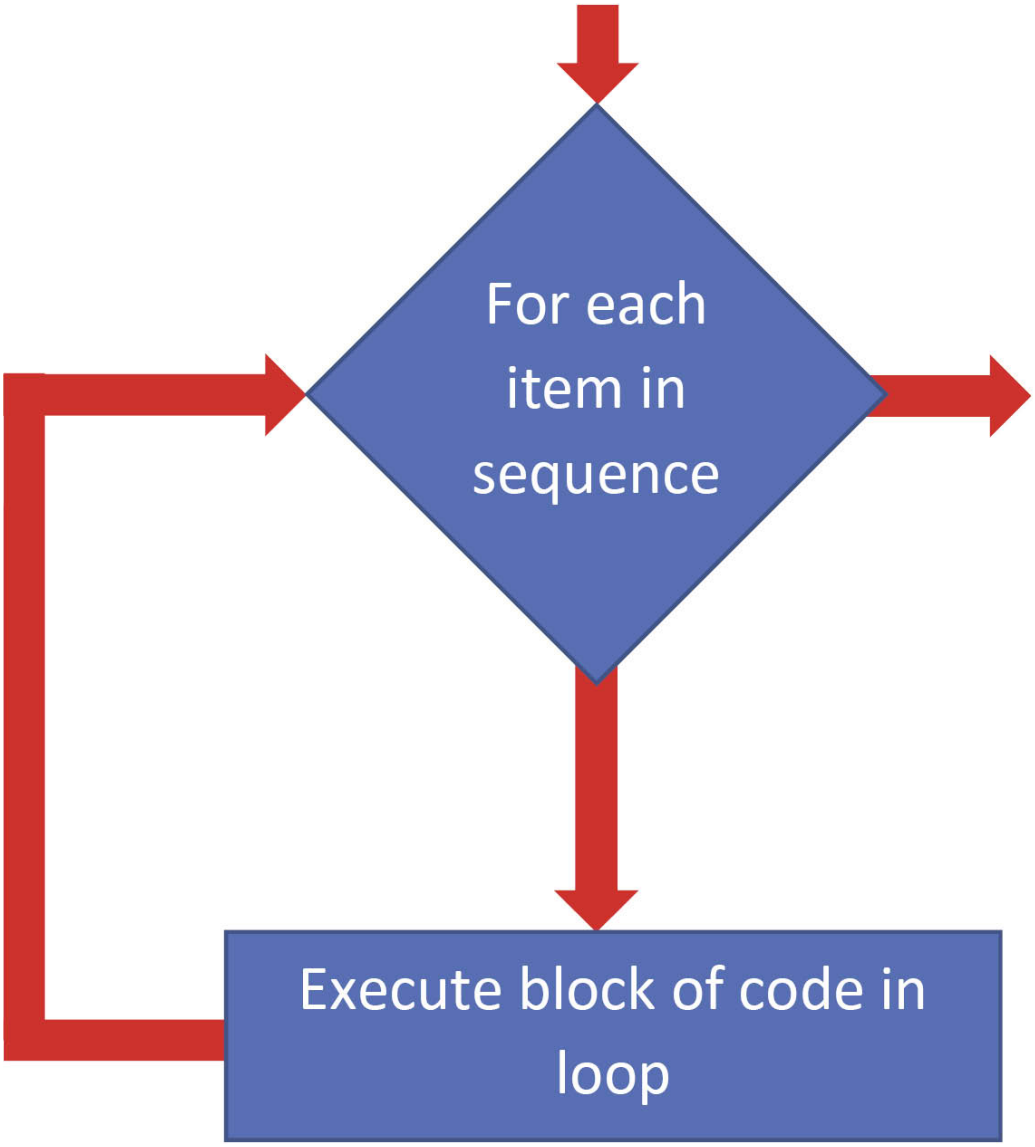
Any condition not met by the above ‘elif’ statements, the interpreter will execute the ‘else block’ at the end.

Iteration (Loops)

A loop is a set of statements that are repeated until a specific condition is met. We will look at two types of loops: the for loop, and the while loop.

For loop

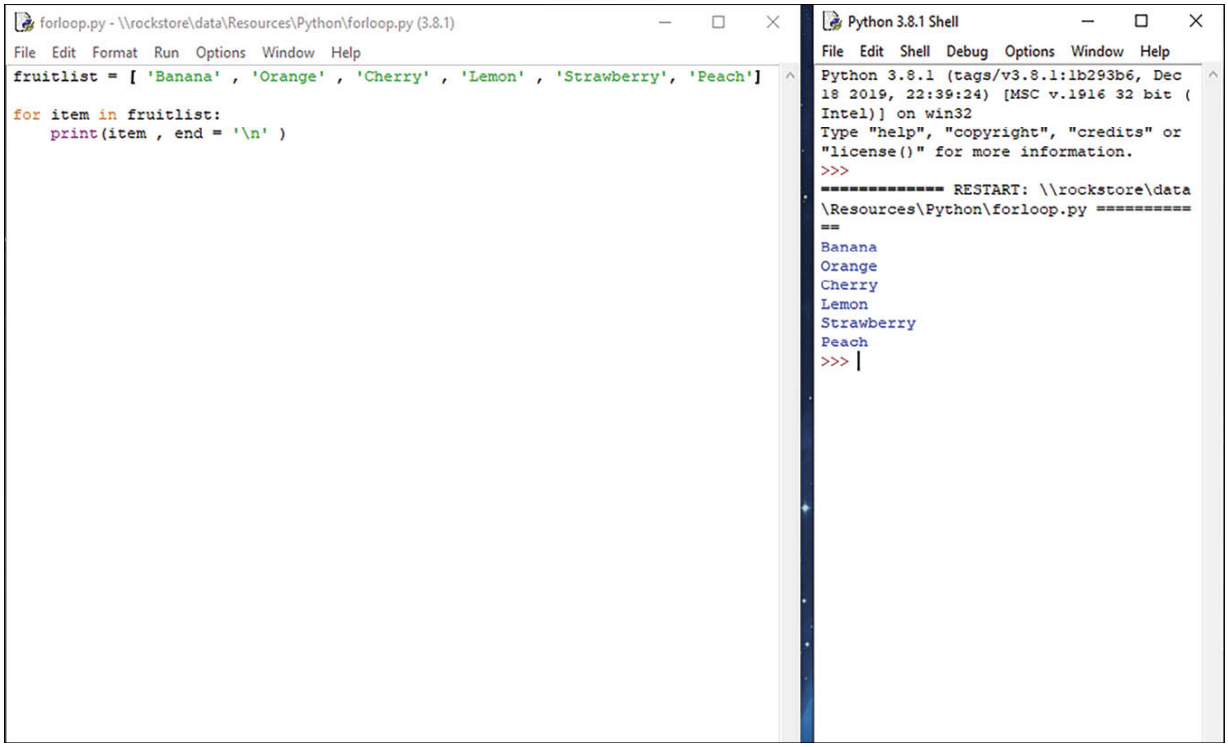
A for loop executes a set of statements for each item in a sequence such as a list or string or a range. It allows a code block to be repeated specific number of times.



This particular loop will print out each name in the list on a new line

```
myList = [‘john’, ‘lucy’, ‘kate’, ‘mike’]

for val in myList:

print (val) #block of code in loop
```

Let’s have a look at a program. Open forloop.py. The for loop contains a loop condition. In this example, the loop will execute for each item in the fruitlist (sequence).



The ‘item’ variable in the ‘for loop’ statement is a pointer or counter to the current value or item in the sequence.

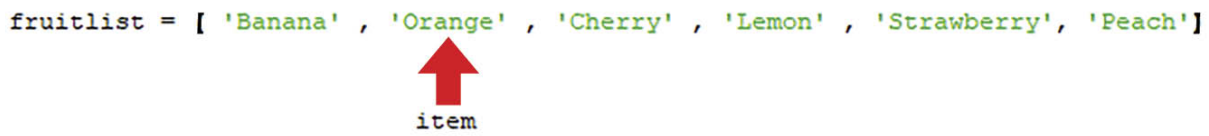
```
fruitlist = [ 'Banana' , 'Orange' , 'Cherry' , 'Lemon' , 'Strawberry', 'Peach']
              item
```

For each of these ‘items’, the interpreter will execute everything inside the loop. In this example the ‘print’ statement.

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

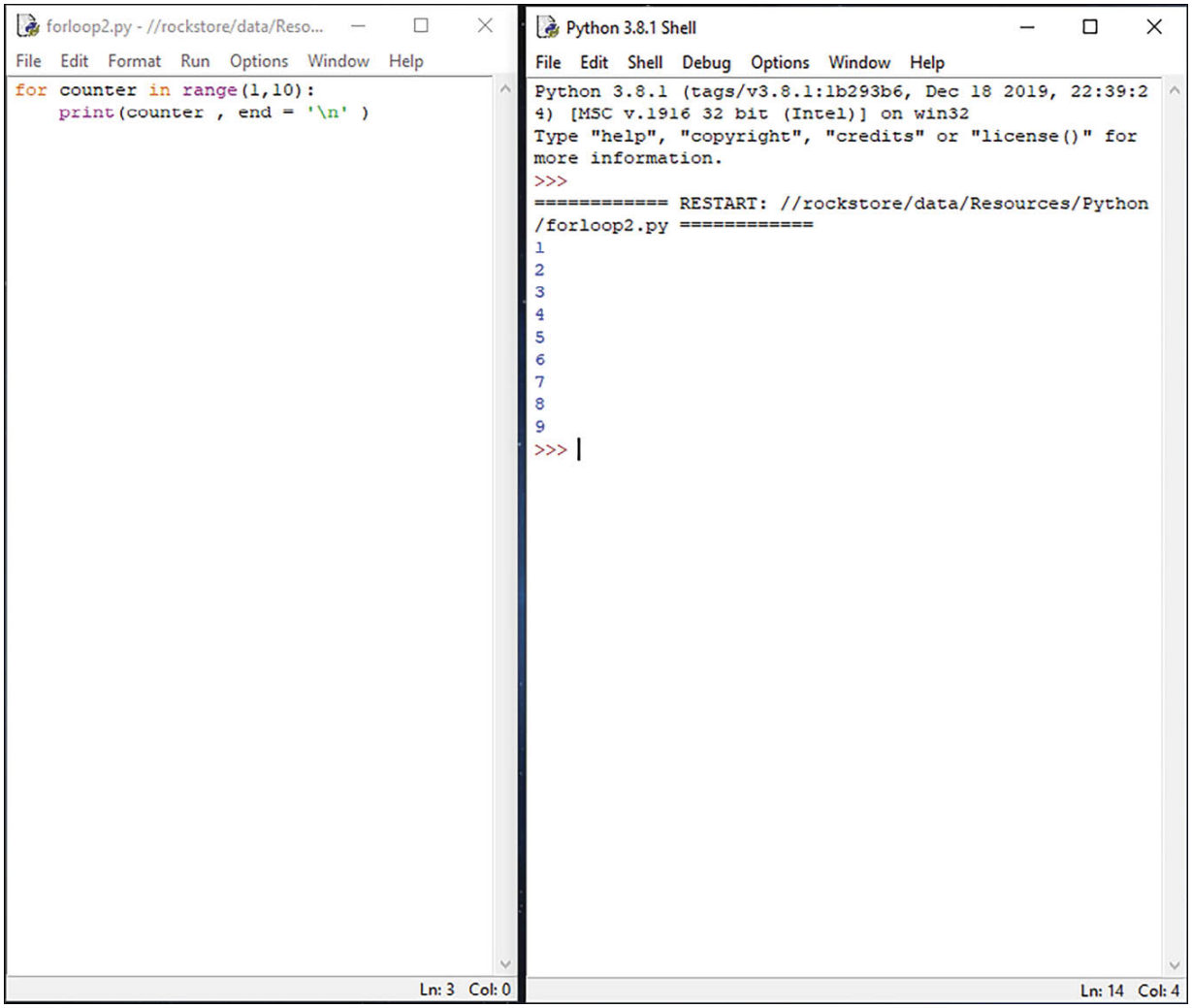
The interpreter will test the condition in the for loop again and if it is

true, the interpreter will execute everything inside the loop again. Each iteration of the loop, the counter moves to the next value or item.

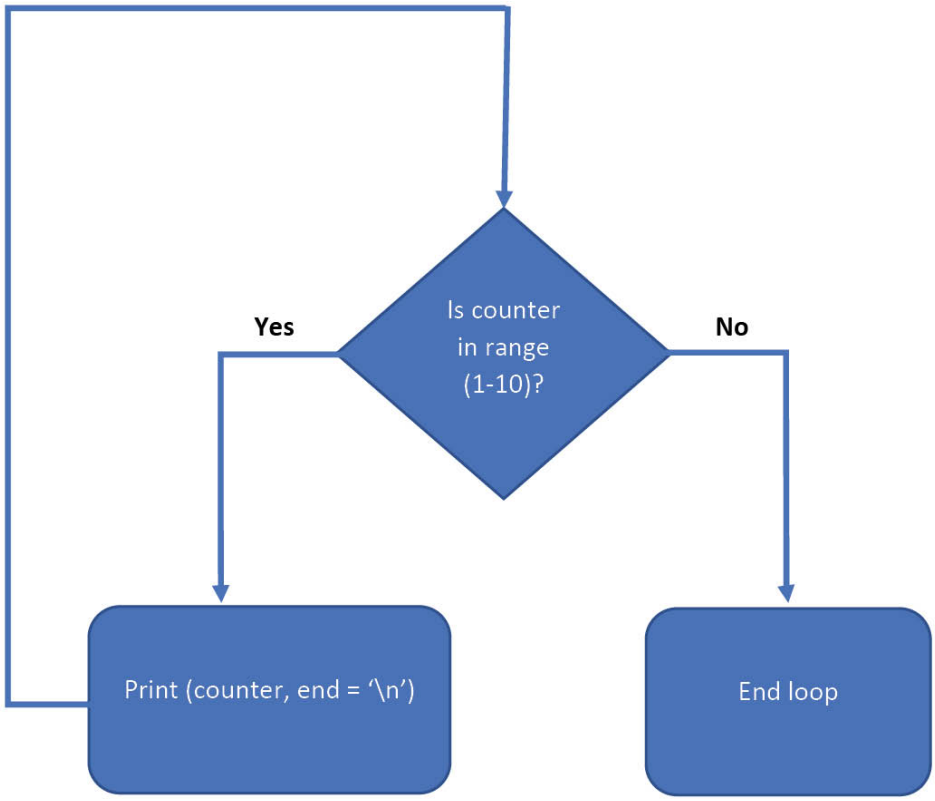


At the end of the sequence, the loop condition becomes false, so the loop terminates.

Lets look at another example. Open forloop2.py.

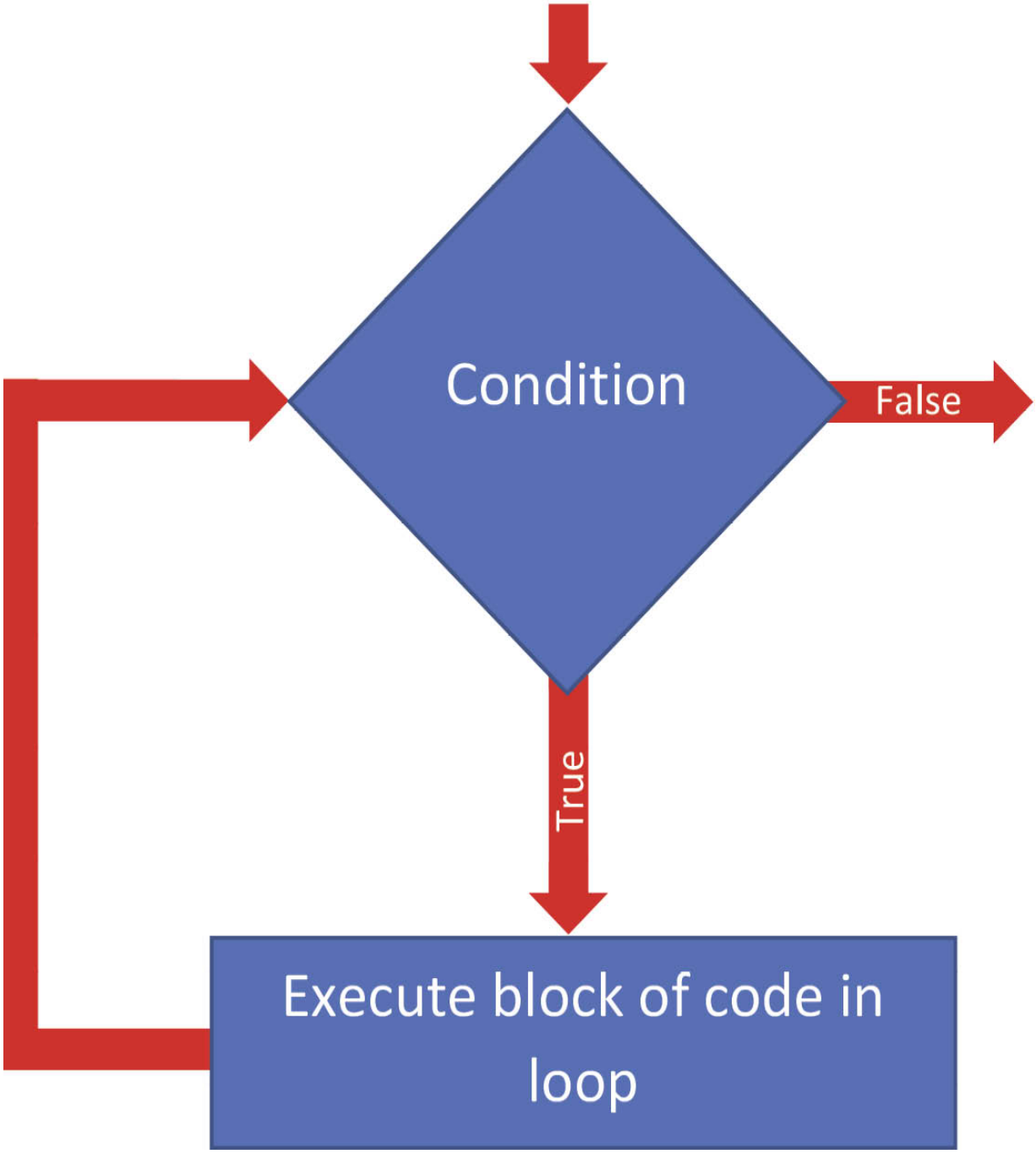


When you run through the program you can see what it's doing


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

While loop

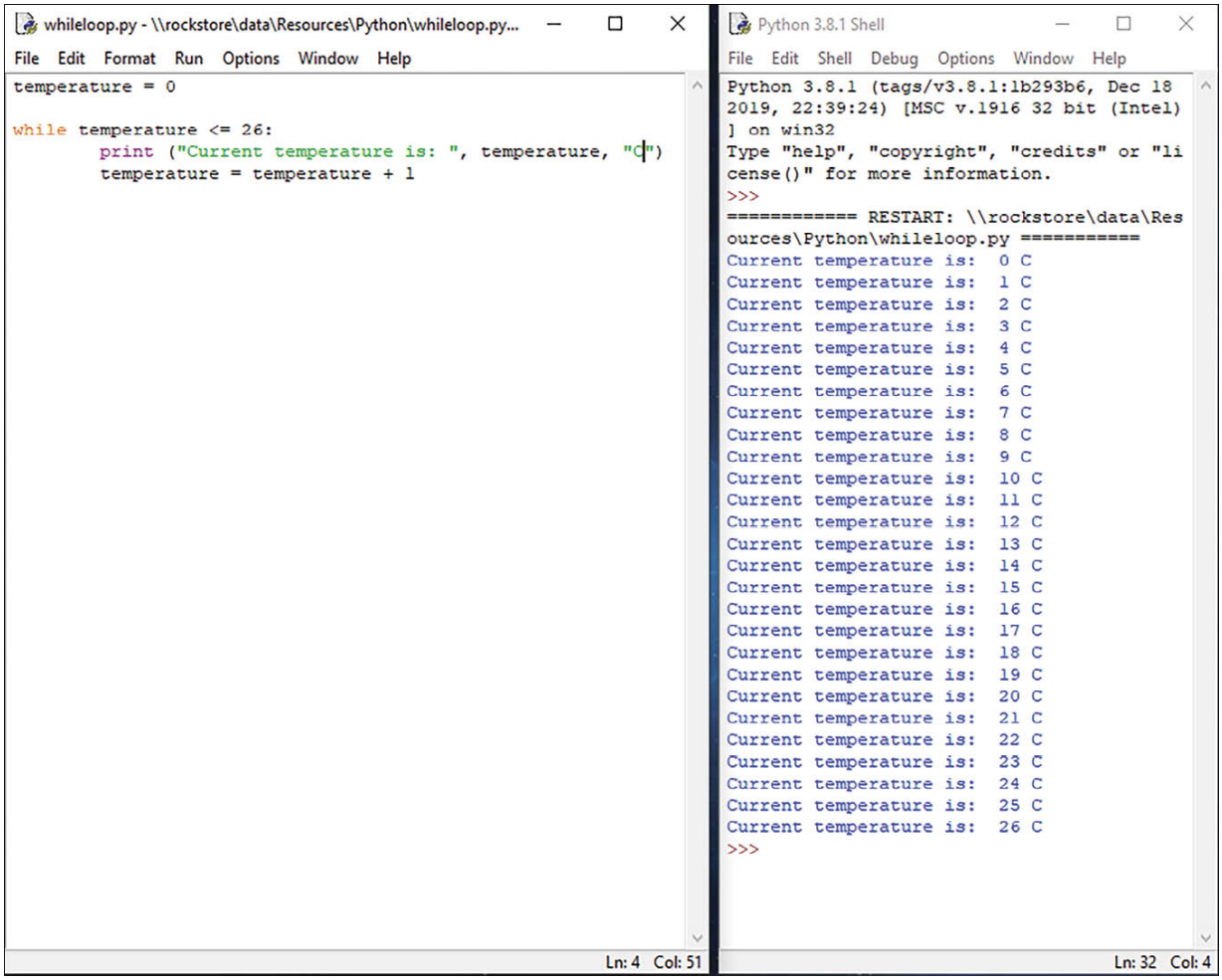
A while loop executes a set of statements while a certain condition is true. It allows the set of statements in the code block to be repeated an unknown number of times, and will continue to repeat while the condition is true.



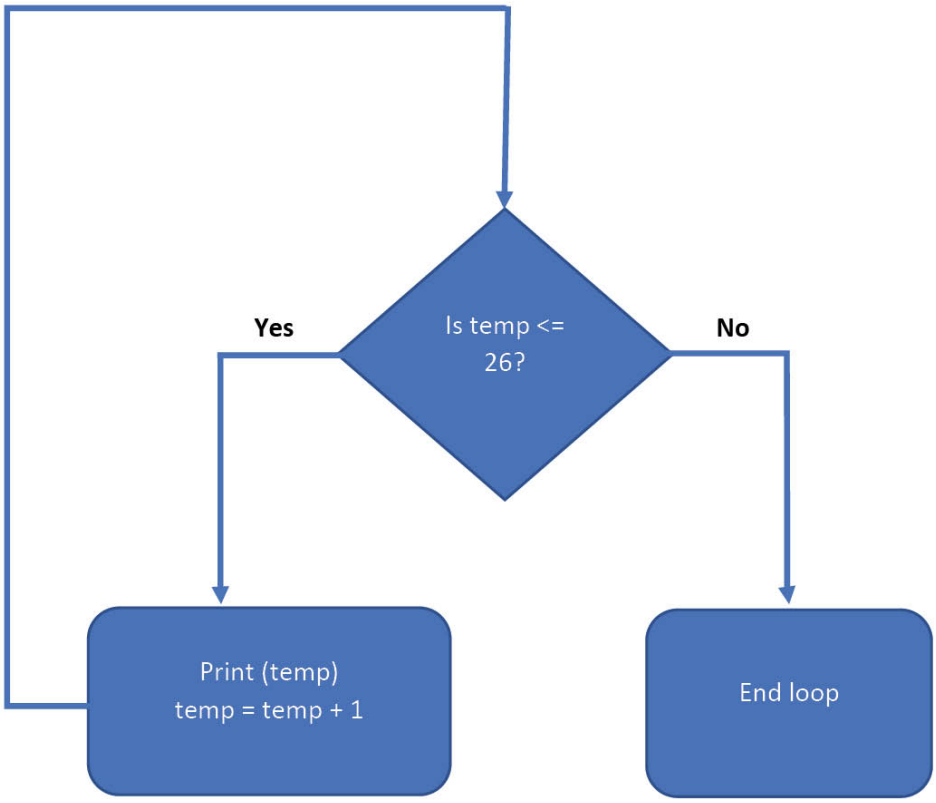
This particular loop will keep prompting the user for a string until the user enters the word ‘fire’.

```
userInput = ‘’

while userInput != ‘fire’:

userInput = input (‘Enter passcode: ‘)
```

Let’s have a look at a program. Open whileloop.py. The while loop contains a loop condition.



When you run through the program you can see what it’s doing



Break and Continue

The `break` statement breaks out of a loop. In this example, the loop breaks when the counter is equal to 5.

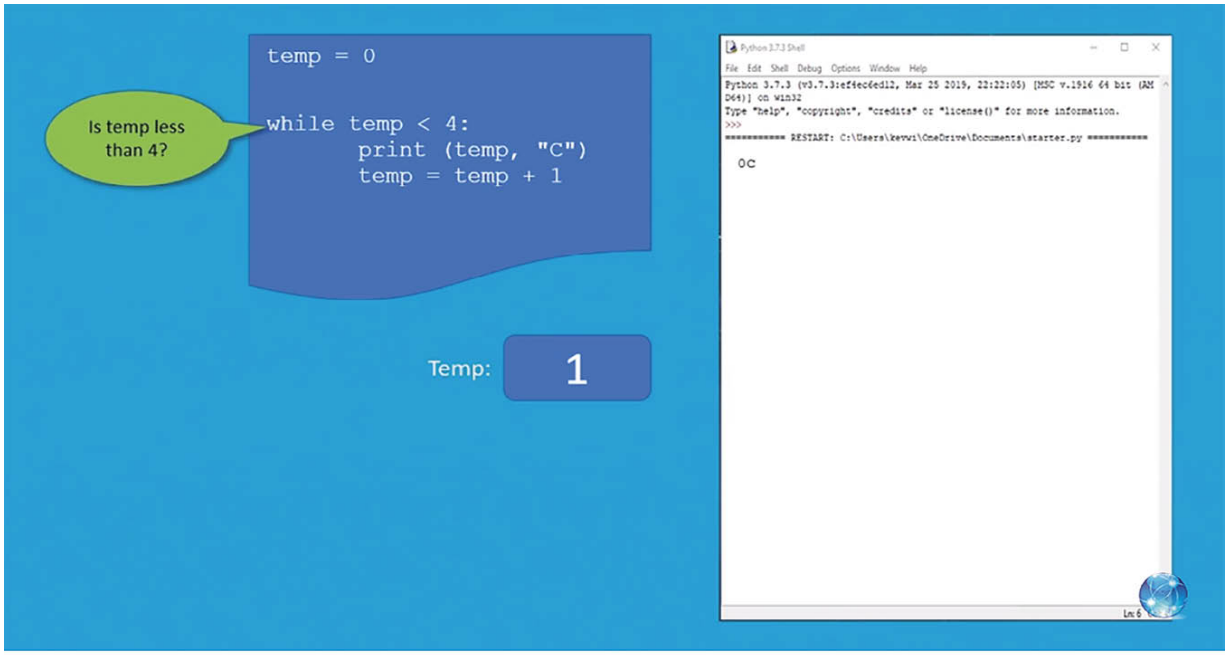
```
while (counter < 10):
if counter == 5:
break
counter = counter + 1
```

The `continue` statement jumps back to the top of the loop without executing the rest of the code after the continue keyword. In this example the loop restarts when 'number' is even.

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

Have a look at the video demos and familiarize yourself with loops and if statements.

www.elluminetpress.com/pythonflow



Lab Exercises

Take a look at the following exercises and use what you've learning solve the problems.

1. Write a program to print the numbers 1 - 10 to the screen.
2. Write a program to print a list of names to the screen.
3. Write a program to calculate and print the squares of the numbers from 1 to 10. Uses tabs to display them in a table.
4. Write a program that accepts an number from the user until a negative number is entered.
5. Write a program that accepts an integer and prints the specified range it belongs to.

   Range 1: 0 to 10

   Range 2: 11 to 20

   Range 3: 21 to 30

   Range 4: 31 to 40

Handling Files

Since the computer's memory (RAM) is volatile, it loses any stored data when the power is turned off. So any data that needs to be stored permanently must be saved in a file.

A file is a named location on a disk drive that is used to store data. Each file is identified by its filename.

Python contains inbuilt functions for reading data from files, as well as creating and writing to files.

For this section, take a look at the video demos

`elluminetpress.com/pyfiles`

You'll also need the source files in the directory Chapter 05.

File Types

There are two types of files. Text files and binary files. By default Python reads and writes data in a text file.

Text File

A text file stores sequences of characters stored in ASCII format. Plain text files, HTML files, program source code.

Use these file modes when opening a file in text mode:

- “r” opens a file for reading, error if the file does not exist
- “a” opens a file for appending, creates the file if it does not exist
- “w” opens a file for writing, creates the file if it does not exist
- “r+” opens a file for both reading and writing

Binary

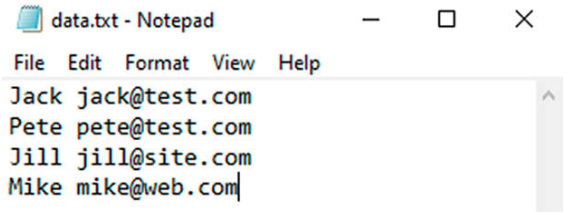
A binary file stores data in a sequence of bytes (1s and 0s) - the same format as the computer’s memory (RAM). For example, images such as JPEG or PNG, audio files such as WAV or MP3, video files such as MP4, and program executable files.

Use these file modes when opening a file in binary mode:

- “rb” opens a file for binary reading, error if the file does not exist
- “ab” opens a file for binary appending, creates the file if it does not exist
- “wb” opens a file for writing, creates the file if it does not exist
- “rb+” opens a file for both reading and writing

Text File Operations

By default, Python opens files as text files. Text files contain readable characters as shown in the example below.

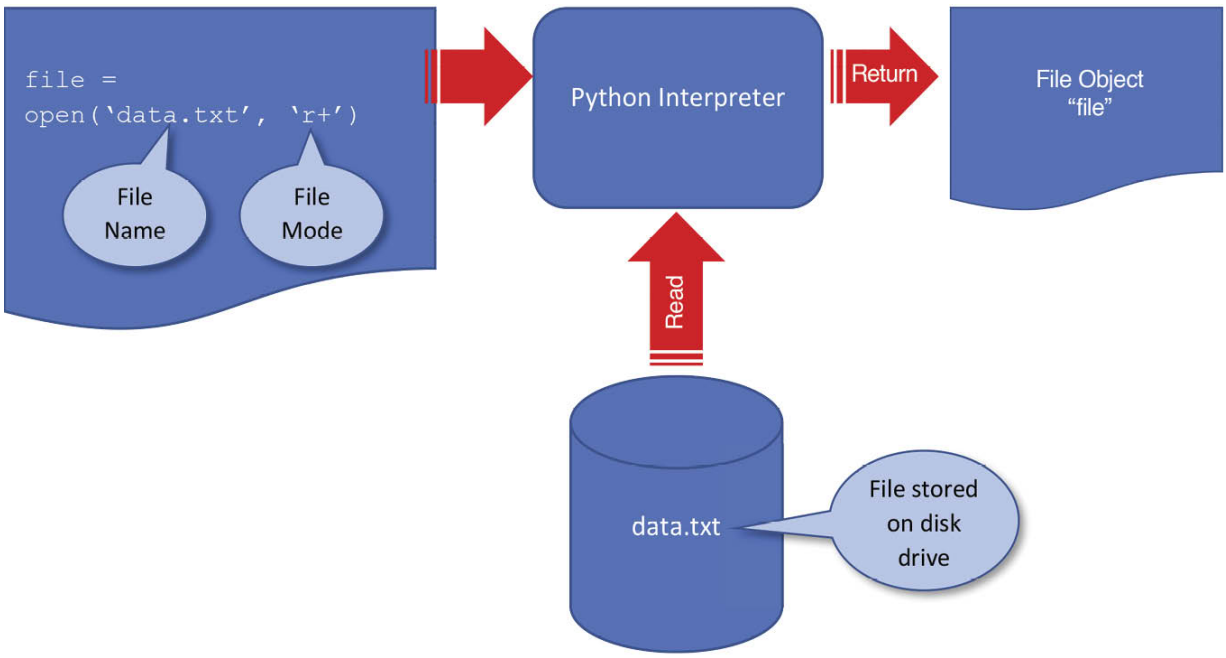


Open Files

To open a file use the `open( )` function. This function returns a file object that we can work on (called file in this example).

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

When you open a file, put the filename and the file mode in the parameters of the open( ) function.

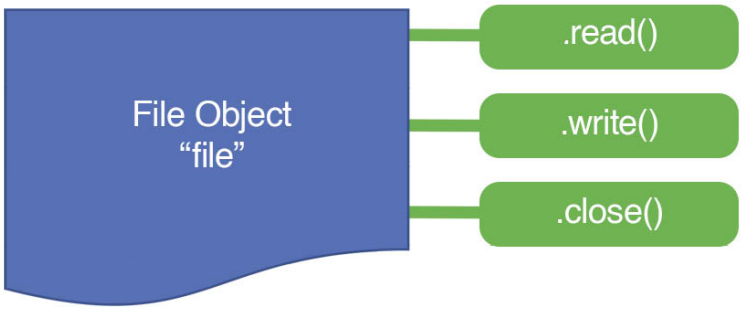


The file mode tells the Python interpreter what you intend to do with the file, ie read, write, or append.

- "r" opens a file for reading, error if the file does not exist

- “a” opens a file for appending, creates the file if it does not exist
- “w” opens a file for writing, creates the file if it does not exist
- “r+” opens a file for both reading and writing

Once the `open()` function returns a file object, we can work on the file using the object’s methods such as `.read()` `.write()` or `.close()`



Write to a File

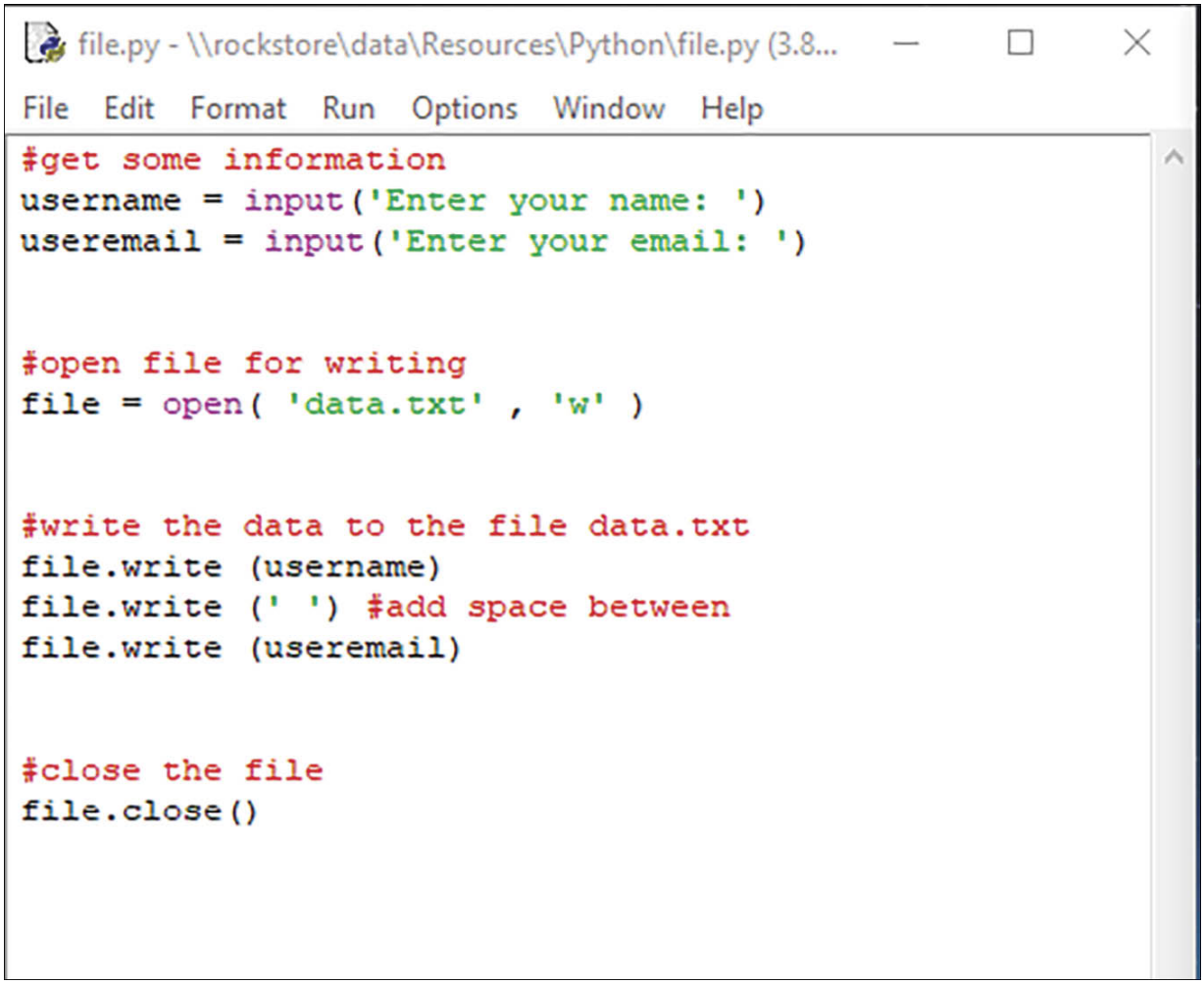
To write data to a file use the `.write()`. This method writes specified text to a file.

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

When opening a file for writing use either

- "a" opens a file for appending, creates the file if it does not exist. Adds new data to end of file.
- "w" opens a file for writing, creates the file if it does not exist. Overwrites any existing data in file.

Lets take a look at a program. Open file.py.

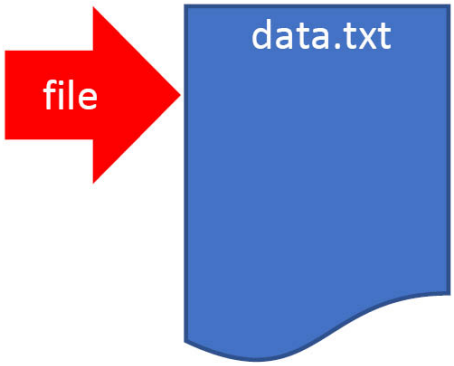


Here, we get some information from the user (a username and an email address).

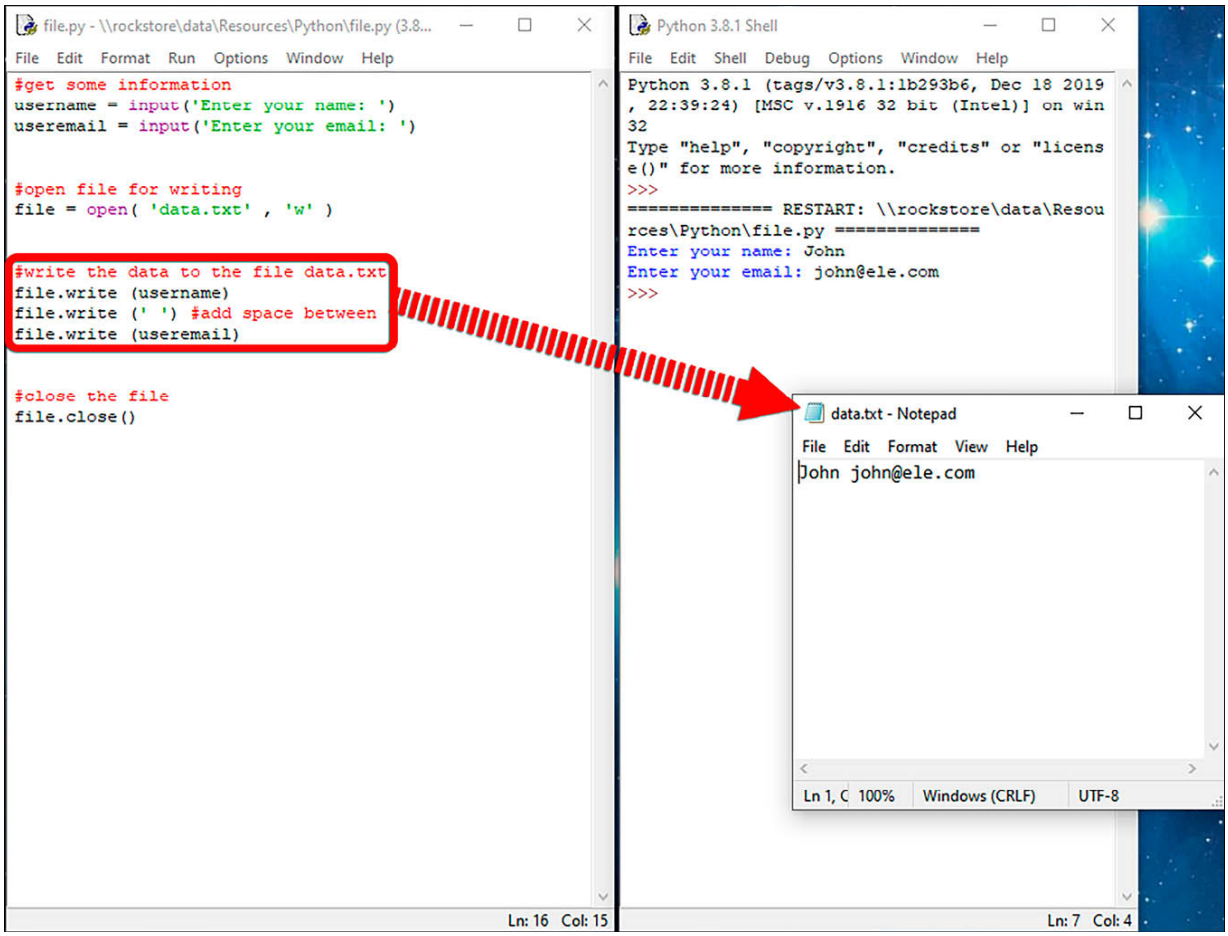
```
username = input('Enter your name: ')
useremail = input('Enter your email: ')
```

Next, we open a file called ‘data.txt’ for writing and assign it to an object called ‘file’.

```
file = open(‘data.txt’, ‘w’)
```



We then write the username and email address to the file using the file object’s `.write` method.



Now remember, the ‘w’ file mode opens a file for writing. This also

means any new data will overwrite any data already stored in the file.

After we've completed our file operations, we close the file.

```
file.close()
```

Read from a File

To read data from a file use the `.read()` method to read the whole file.

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

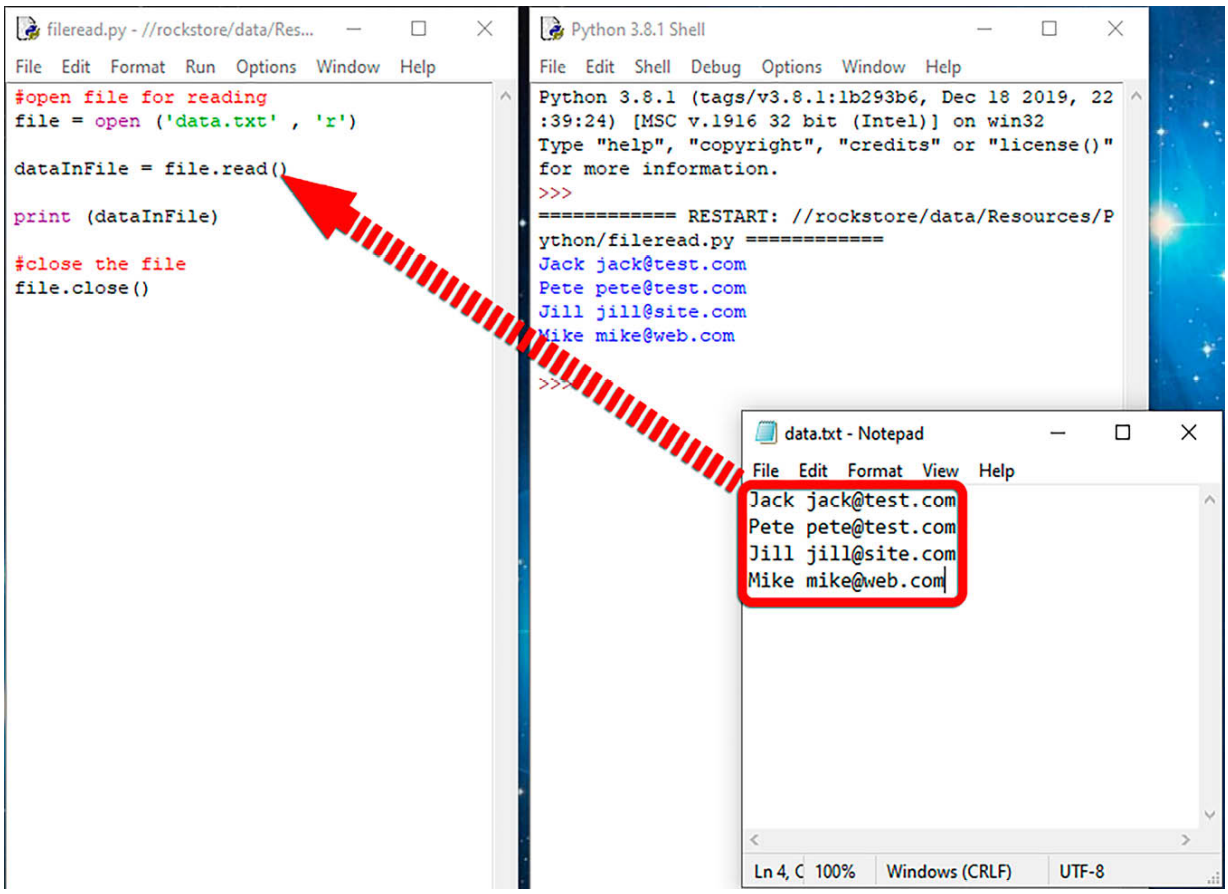
Use the `.readline()` method to read a line at a time.

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

When opening a file for reading use either

- "r" opens a file for reading, error if the file does not exist
- "r+" opens a file for both reading and writing

Lets take a look at a program. Open fileread.py.



Here, we open a file called ‘data.txt’ and assign it to an object called ‘file’.

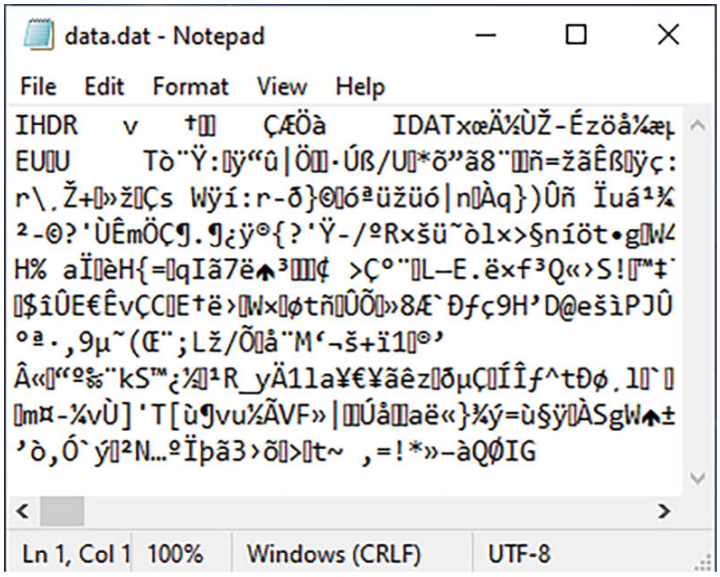
Next we read the data using the ‘file’ object’s `.read` method

Finally close the file.

Binary File Operations

Most digital data is stored in binary files as they are much smaller and faster than text files.

Binary files are not readable by humans as shown in the example below. Here, we've opened a binary file in a text editor.



Open Files

To open a file use the `open()` function.

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

When you open a file, put the filename and the file mode in the parameters of the `open()` function.

The file mode tells the Python interpreter what you intend to do with the file, ie read, write, or append.

- "rb" opens a file for reading, error if the file does not exist
- "ab" opens a file for appending, creates the file if it does not exist
- "wb" opens a file for writing, creates the file if it does not exist
- "rb+" opens a file for both reading and writing

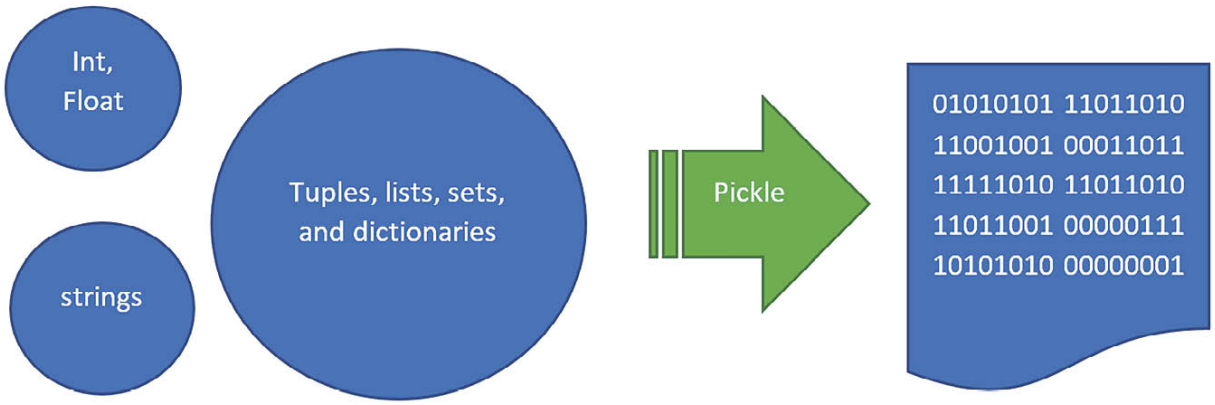
This depends on the purpose of your program.

It’s good practice to open your file, perform your operations, then close the file.

Write to a File

The `.write()` method writes data in text format and if you try to write data in binary mode using this method, you'll get an error message when you run your program.

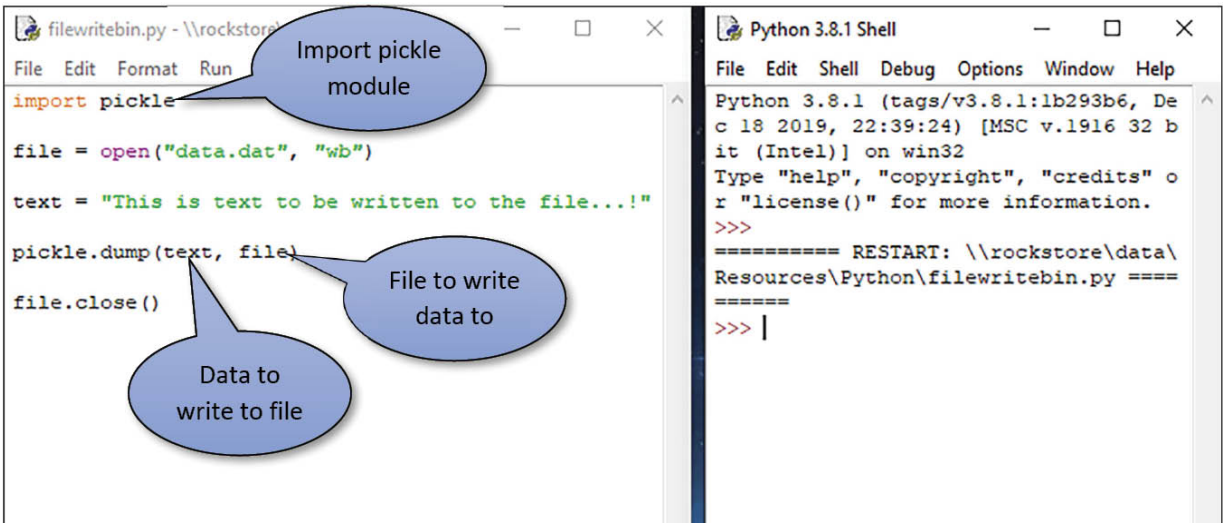
To write your data in binary format, first we need to convert it to a sequence of bytes. We can do this with the pickle module using a process called pickling. Pickling is the process where data objects such as integers, strings, lists, and dictionaries are converted into a byte stream.



To write to a file, use the `pickle.dump()` method

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

Let's take a look at a program. Open filewritebin.py. First we need to include the pickle module. You can do this using an import command.

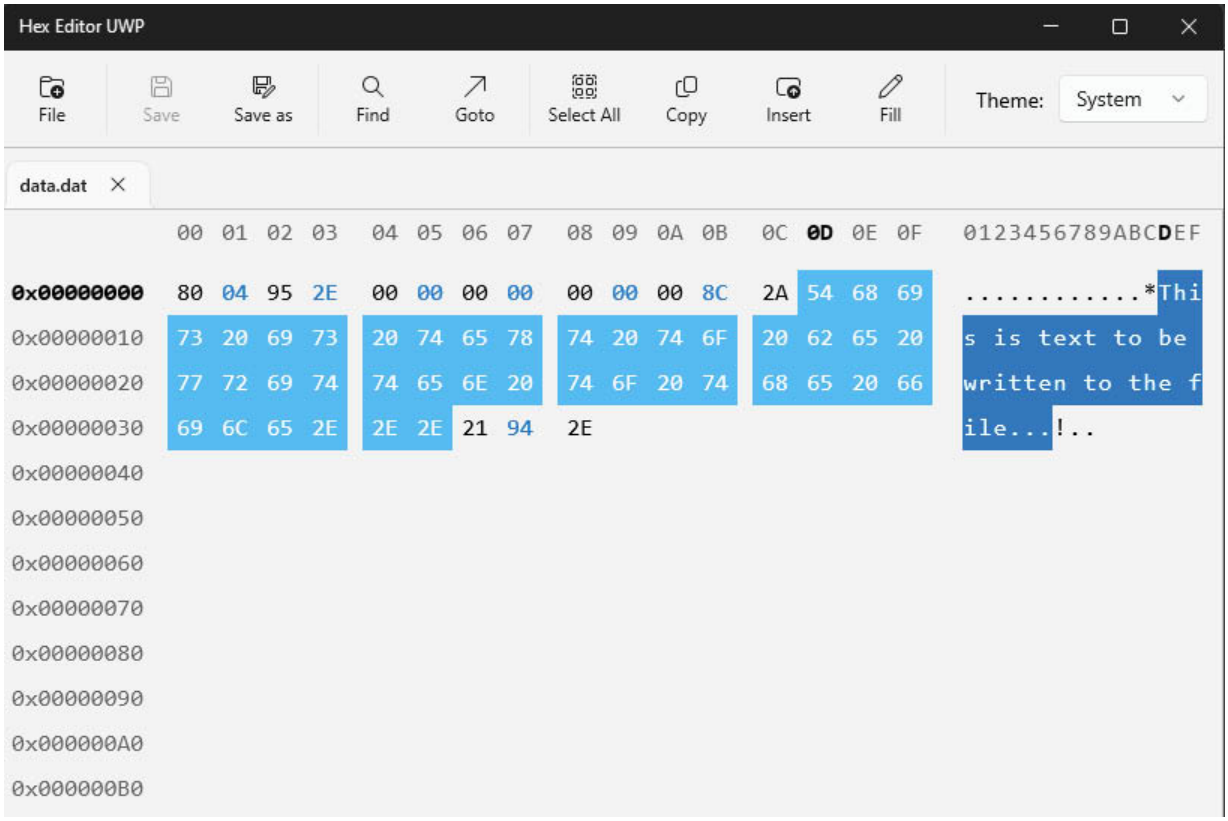


Next, we open a file in the usual way, except we set the file mode to binary write (wb).

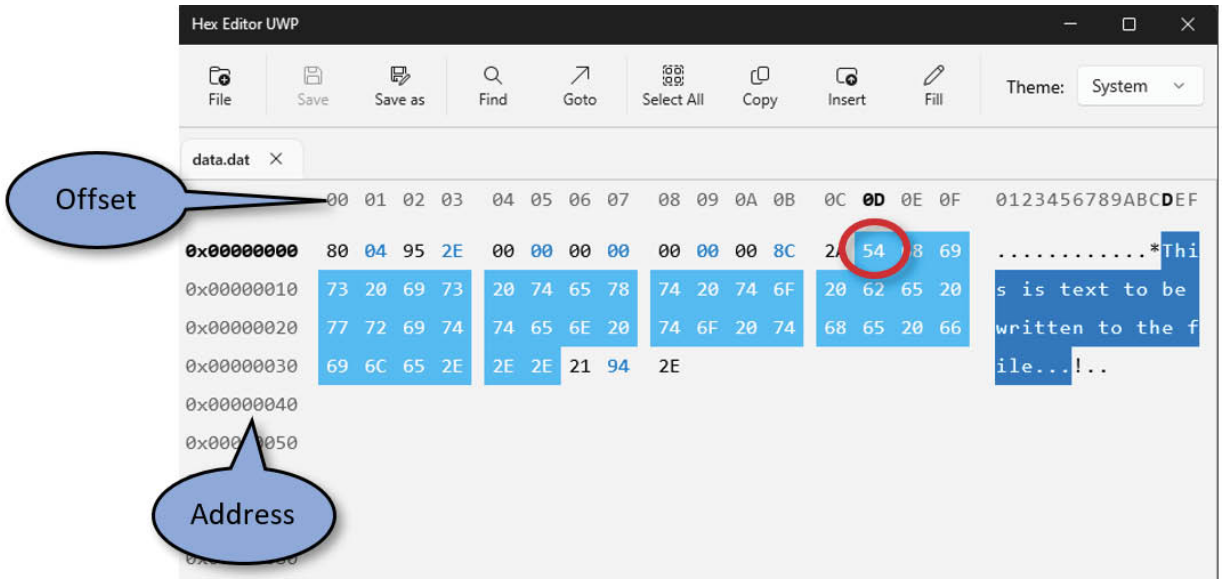
Now to write the data to the file, we pickle the 'text' object using the `pickle.dump()` method.

Finally close the file.

Here is the data written to the file. You can see in hex editor below, the string "Data that has been written to a binary file" has been converted into binary data.



Notice that the data in the middle is represented using hexadecimal numbers which are often used as shorthand for binary. Each hex number in the hex block represents 1 byte.



Read a File

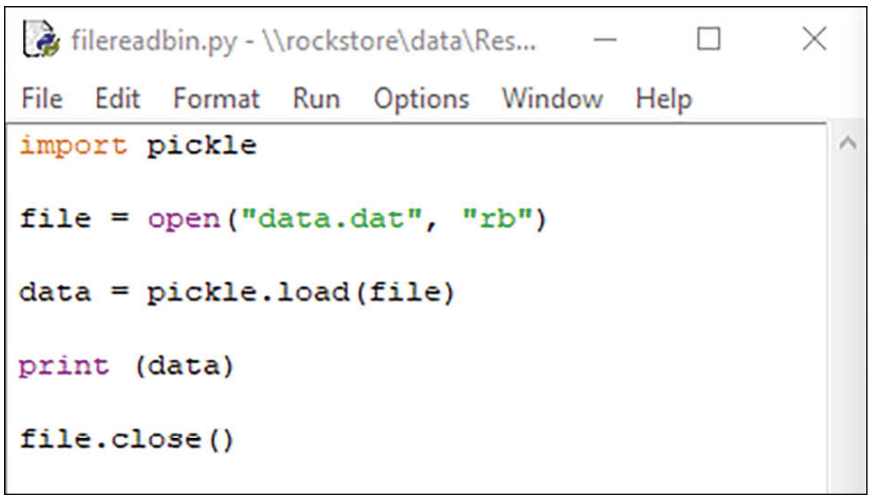
Remember when we wrote our data to our binary file, we used a

process called pickling. Well, to read the data from the file we use a similar process.

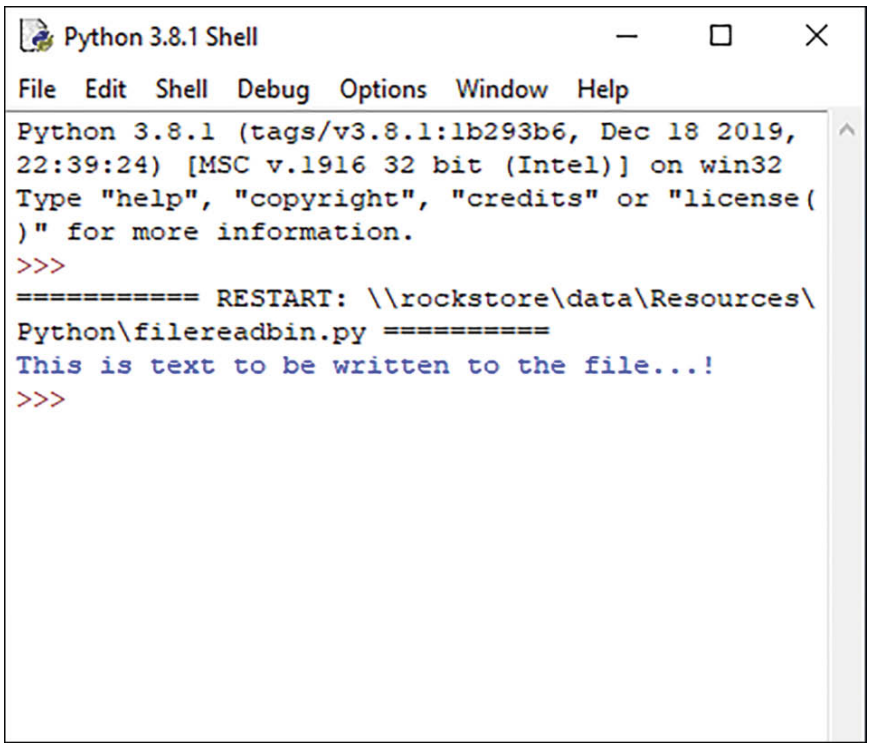
To read a file use the `pickle.load()` method

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

Let's take a look at a program. Open filereadbin.py. First we need to include the pickle module. You can do this using an import command.



When we run the program, the data is read from the file, unpickled, then assigned to the 'data' variable. We can then print the 'data' to the screen.


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\
Python\filereadbin.py ==========
This is text to be written to the file...!
>>>

Random File Access

When a file is opened, the Python Interpreter allocates a pointer within the file. This pointer determines the position in the file from where reading or writing will take place. The pointer can be moved to any location in the file.

To move the file pointer, use the `.seek()` method.

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

The first parameter (position-in-file) determines how many bytes to move. A positive value will move the pointer forward, a negative value will move the pointer backward. The position in the file is called an offset.

The second parameter (whence) determines where in the file to start from, and accepts one of three values:

- 0 : sets the start point to the beginning of the file (the default)
- 1 : sets the start point to the current position
- 2 : sets the start point to the end of the file

In a file, each position could contain one byte or one character. Remember, the numbering system starts with 0.

Using our text file as an example `file.seek(5)` would move the file pointer to the 6th byte.

	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)
```

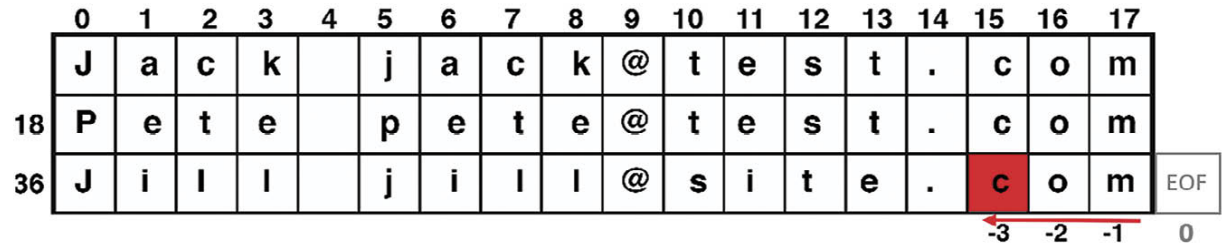
This will move the pointer to position 23 from the beginning of the file

	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

If you want to read from the end of the file, you'll need to count backwards from the end of the file, you do this with a negative offset. The `seek()` function with negative offset only works when file is opened in binary mode.

To move the pointer from the end of the file

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



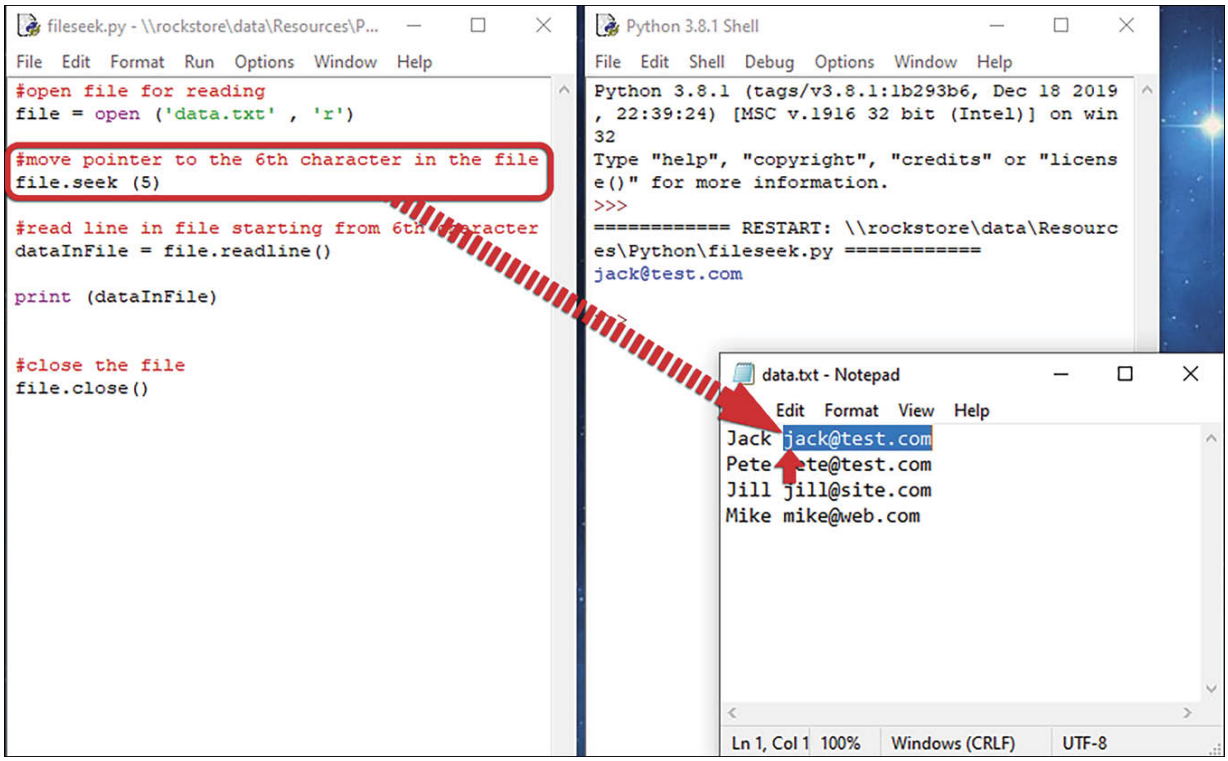
Use `.decode('utf-8')` to convert the binary back to text.

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

To find the current position of the file pointer in the file use the `.tell()` method.

```
file.tell()
```

Lets take a look at a program. Here, we're going to start reading the first line of the data.txt file starting from the 6th byte or character.


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
dataInFile = file.readline()
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 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
Mike mike@web.com
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8

File Handling Methods

Here is a summary of methods available for file objects. You can tag the method names below onto the object name using the following syntax:

`fileobject.method()`

Here we have various methods to close a file, as well as some other common methods.

Method	Description
`close()`	Closes the file
`detach()`	Returns the separated raw stream from the buffer
`fileno()`	Returns a number that represents the stream, from the operating system's perspective
`flush()`	Flushes the internal buffer
`isatty()`	Returns whether the file stream is interactive or not

Next, there are methods to read data from a file. You can read the whole file, or return lines from a file.

Method	Description
`read()`	Returns the file content
`readable()`	Returns whether the file stream can be read or not
`readline()`	Returns one line from the file
`readlines()`	Returns a list of lines from the file

Various methods to write data to a file. You can check whether a file is writable, or you can write data to a file.

Method	Description
`writable()`	Returns whether the file can be written to or not
`write()`	Writes the specified string to the file
`writelines()`	Writes a list of strings to the file

Various other methods to seek a file position, as well as a method to return the current position in a file and one to truncate the file to a specific size.

Method	Description
`seek()`	Change the file position
`seekable()`	Returns whether the file allows us to change the file position
`tell()`	Returns the current file position
`truncate()`	Resizes the file to a specified size

Lab Exercises

Take a look at the following exercises and use what you've learning solve the problems.

1. Write a program that gets a string from the user then writes it to a file along with the user's name.
2. Modify the program from exercise 1 so that it appends the data to the file rather than overwriting.
3. Write a program to write a list of names to a file.
4. Write a program to read a file line by line and store it in a list.
5. What is the difference between a text file and a binary file?

Using Functions

Functions help break a program into smaller pieces. This avoids repetition of code, making larger programs more efficient and easier to maintain.

For this section, take a look at the video demos

`elluminetpress.com/pyfunctions`

You'll also need the source files in the directory Chapter 06.

What are Functions

A function is a block of code that is only executed when it is called. Python comes with a library of functions to perform a wide range or tasks. These are called built in functions. For example:

```
print ()

input ()

open ()
```

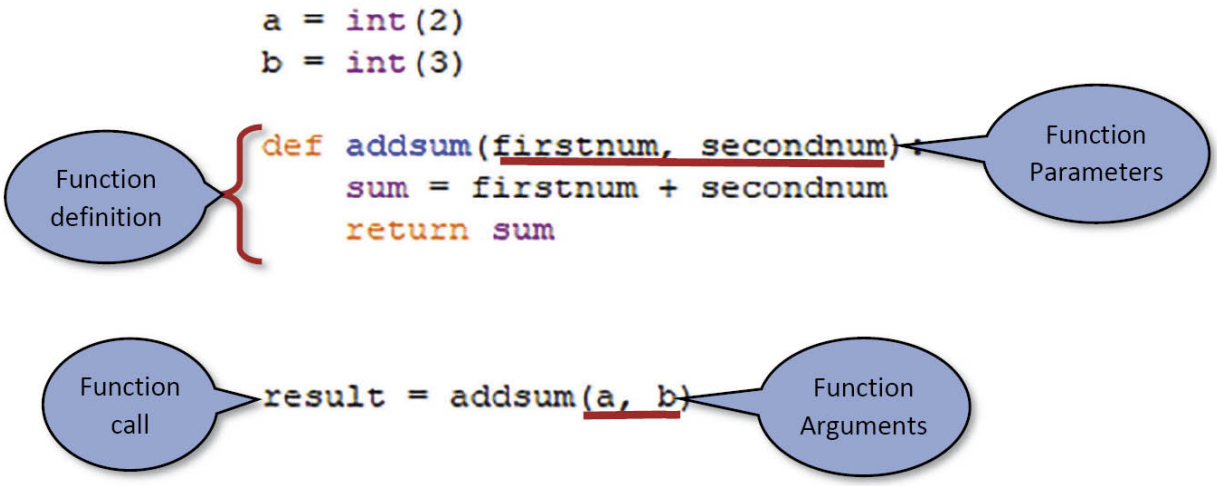
You can also create your own functions. These functions are called user-defined functions. For example:

```
def functionName (parameters):

<function code>
```

A function takes parameters or arguments enclosed in parenthesis and returns a result. Most people seem to use both these terms interchangeably but they aren't the same thing.

- A parameter is a variable in the function definition.
- An argument is a value passed during a function call.

For example, in the following code. The function definition of 'addnum' takes two parameters 'firstnum' and 'secondnum'.



In the line at the bottom we call the 'addnum' function, and pass two arguments 'a' and 'b'.

Built in Functions

Python has various built in functions. Here are some of the more common ones.

Function	Description	Example
abs()	Returns the absolute value of a number	`abs(val)`
bin()	Returns the binary version of a number	`bin(val)`
bool()	Returns the boolean value of the specified object	`bool(val)`
bytearray()	Returns an array of bytes	`bytearray(val)`
bytes()	Returns a bytes object	`bytes(val)`
chr()	Returns a character from the specified Unicode.	`chr(65)`
dict()	Returns a dictionary (Array)	`dict(id = "A34", name = "USA")`
divmod()	Returns the quotient and the remainder when argument1 is divided by argument2	`divmod(17, 3)`
filter()	Use a filter function to exclude items in an iterable object	`filter(function, item-to-be-filtered)`
float()	Returns a floating point number	`float(val)`
format()	Formats a specified value Substitute 'format' for 'b' - Binary format 'd' - Decimal format 'e' - Scientific format, with a lower case e 'E' - Scientific format, with an upper case E 'f' - Fix point number format 'F' - Fix point number format, upper case 'o' - Octal format 'x' - Hex format, lower case 'X' - Hex format, upper case 'n' - Number format '%' - Percentage format	`format(value, format)`
help()	Executes the built-in help system	
hex()	Converts a number into a hexadecimal value	`hex(val)`
input()	Allowing user input	`val=input('enter…')`
int()	Returns an integer number	`int(val)`
len()	Returns the length of an object	`len(list)`
list()	Returns a list	`list(vals)`
max()	Returns the largest item	`max(2,43)`
min()	Returns the smallest item	`min(2,43)`
oct()	Converts a number into an octal	`oct(val)`
open()	Opens a file and returns a file object	`open("file", "mode")`
pow()	Returns the value of x to the power of y	`pow(2, 3)`
print()	Prints to the standard output device	`print(val to print)`
range()	Returns a sequence of numbers, starting from 0 and increments by 1 (by default)	`range(start, stop, incr)`
round()	Rounds a number	`round(number, digits)`
set()	Returns a new set object	
slice()	Returns a slice object	`slice(start, end, step)`
sorted()	Returns a sorted list	`sorted(iterable, key, reverse)`
str()	Returns a string object	`str(object, encoding)`
sum()	Sums the items of an iterator	`sum(vals)`
super()	Returns an object that represents the parent class	`super().__init__()`

User Defined Functions

You can declare a new function using the def keyword followed by the function name.

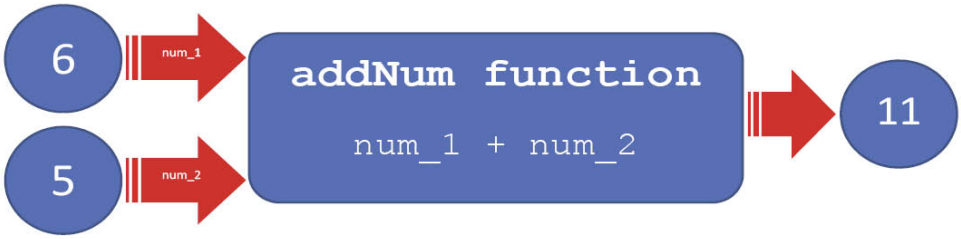
```
def functionName(parameters):
code to be executed in function
```

If the function takes parameters, you can include these in parenthesis next to the function name.

So for example, if we wrote a function to add two numbers together, we could write something like this:

```
def addNum(num_1, num_2):
return num_1 + num_2
```

This function takes two numbers as parameters ‘num1’ and ‘num2’, adds them together, and returns the result.



You can call the function like this with the arguments 6 and 5 in parenthesis:

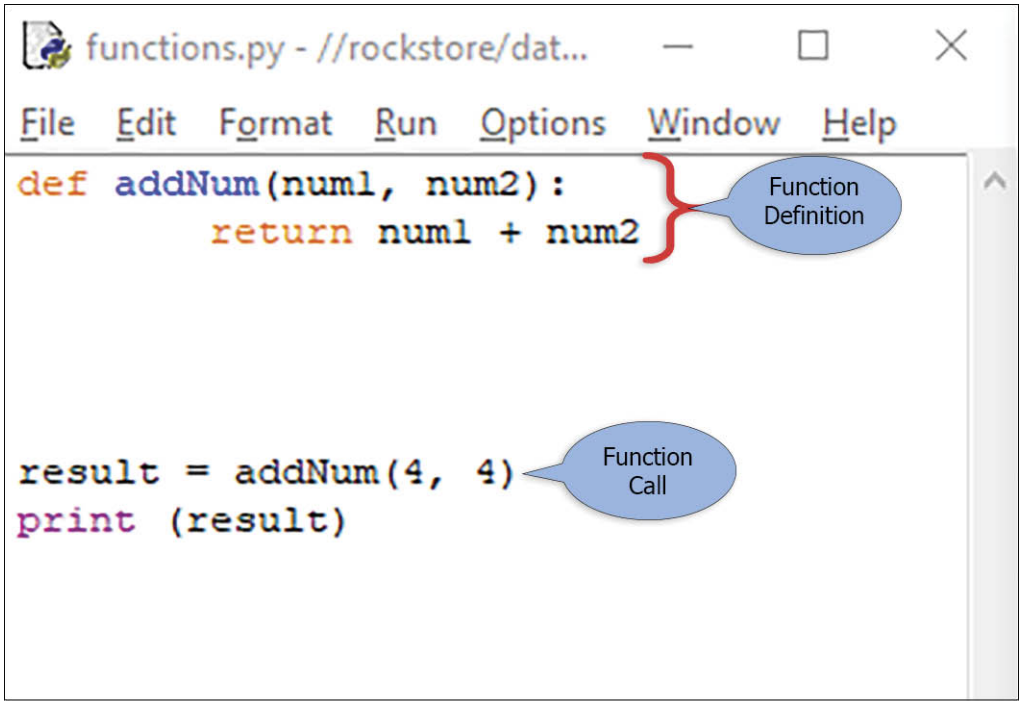
```
result = addNum(6, 5)
```

For smaller programs you can declare your functions in the same file - usually at the top, but as programs become larger and more complex, you should declare your functions in a separate file, then include the file in your main script. This allows you to modularize and reuse code - it is good programming practice for larger projects.

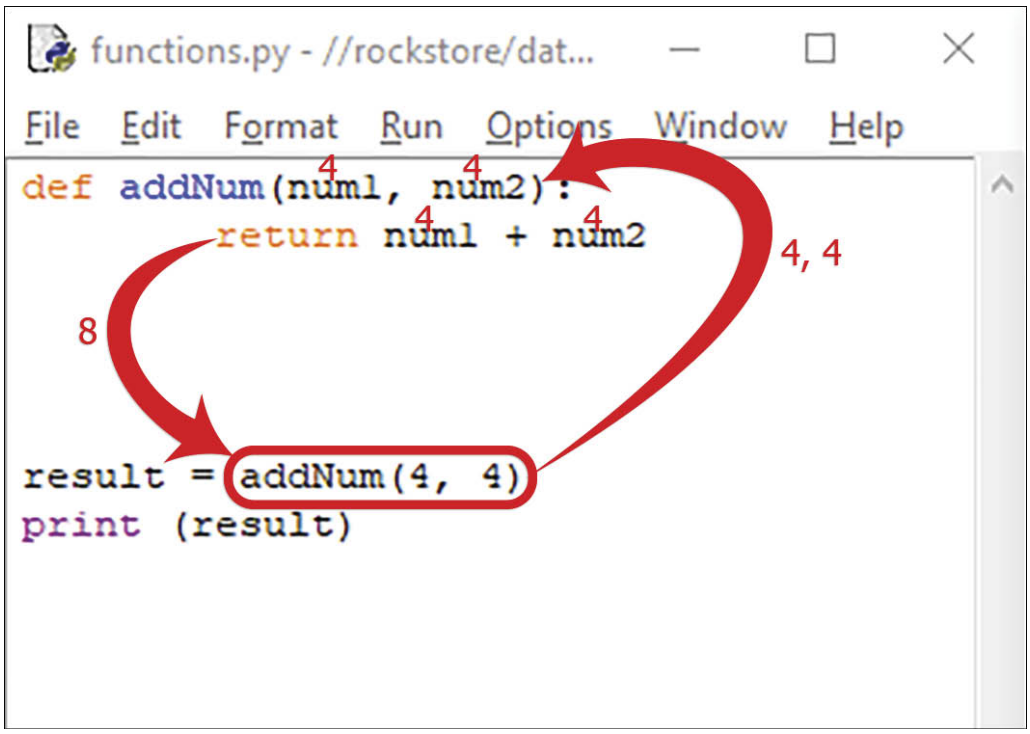
We can declare our addNum function in our myfunctions.py file and include it in our functionsmain.py file. This is called a module (more about modules in chapter 7). To include functions in another script use the import keyword.

```
import myfunctions
```

Lets have a look at a program. Open functions.py. Here, at the top of the script, we've defined a simple function to add two numbers together.



At the bottom of the script, we call our function addNum and pass two values as arguments (4, 4). We use the 'return' keyword to return the result.

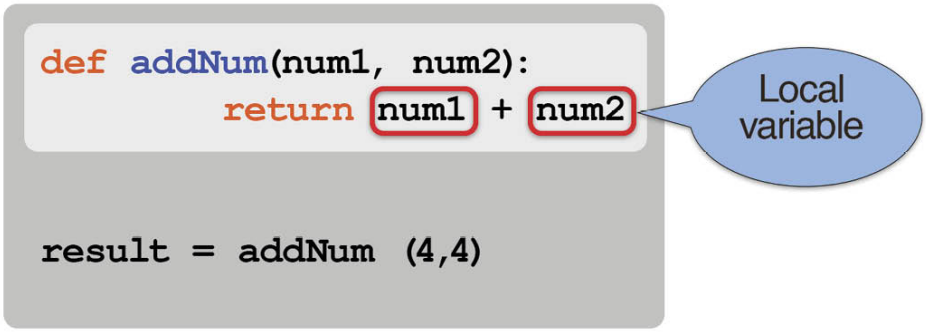


The result returned from the function is then assigned to the variable ‘result’. Finally we print the contents of the variable ‘result’ to the screen, so we can see what is happening.

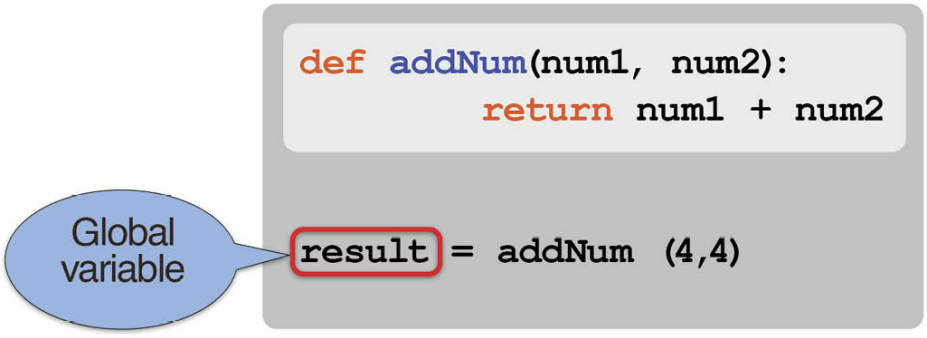
Scope

The part of a program where a variable is accessible is called its scope. In this section, we're going to take a look at local scope, and global scope.

If a variable is only available from inside the region it is created, for example a variable created inside a function, it belongs to the local scope of that function, and can only be used inside that function. This is called local scope.



A variable created in the main body of the Python code is a global variable and belongs to the global scope. Global variables are available from within any scope, global and local.



The period in which a variable exists in memory is called its lifetime.

Variables defined inside a function exist only while the function executes. Once the function returns, the variables inside the function are destroyed.

Recursion

A recursive function is a function that can call itself. This enables the function to repeat itself several times.

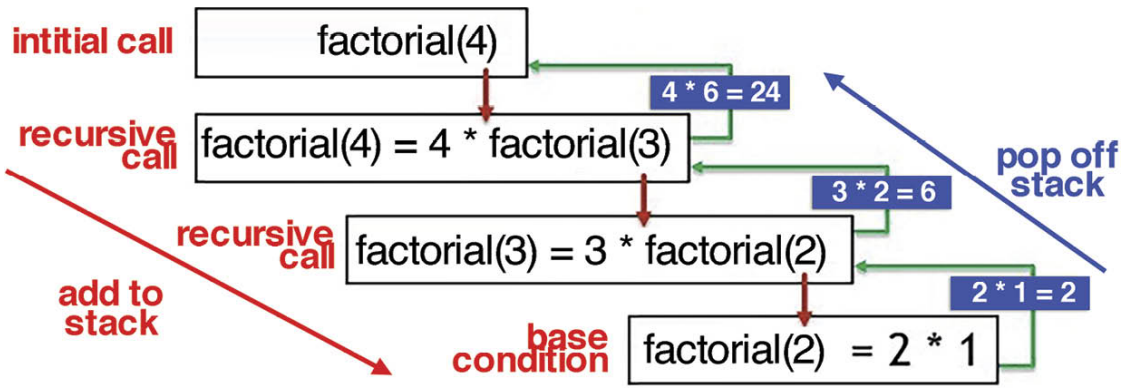
Open the file recursion1.py. Here we have a recursive function that calculates the factorial of a number. Remember, to calculate the factorial you multiply all the numbers from 1 to the given number.

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

When we call the factorial function and pass a positive integer (n)

```
factorial(4)
```

It will recursively call itself by decreasing the number passed to the function (n) by one each time, then add the call to a call stack.



If we enter 4, the factorial function will call itself and pass (n-1 which is 3) as an argument.

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

On the next call the function will pass (n-1 which is 2)

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

On the next call the function will pass (n-1 which is 1)

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

The recursion ends when the number (n) reduces to 1.

```
return 1
```

This is called the base condition (remember we had **`if n <= 1`** in the function). This returns 1 and ends the recursive calls.

Once we hit the base condition, each call is popped off the stack and evaluated.

First pop off factorial(1) which is currently 1, then multiply by 2 = 2

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

Next, pop off factorial(2) which is now 2, then multiply by 3 = 6

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

Finally, pop off factorial(3) which is now 6, then multiply by 4 = 24

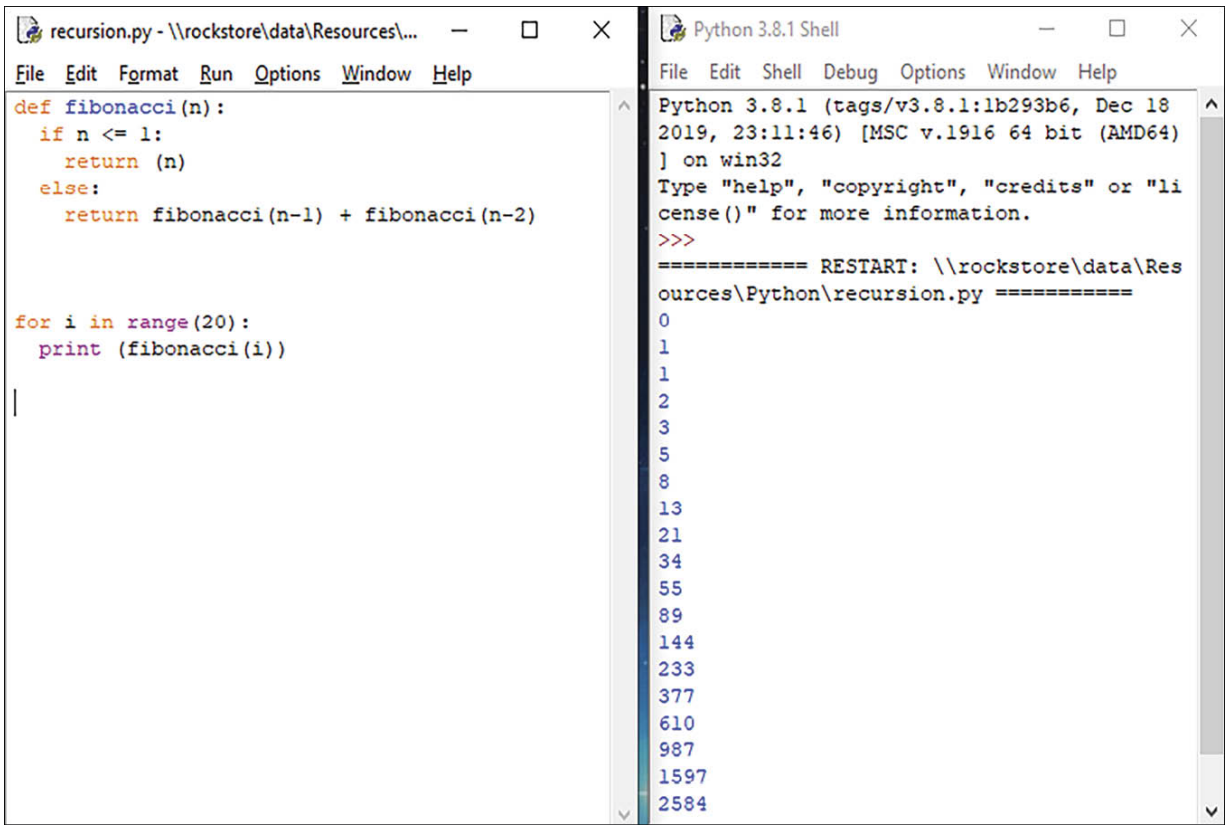
```
4 * factorial(3)
```

Have a look at the recursion demo. Here, we've recorded the recursive function as we step through the code, so we can see the stack and the variables as it executes line by line. Navigate to the following website:

```
elluminetpress.com/pyfunctions
```

Select 'recursion functions'. Study the procedure to see how it works.

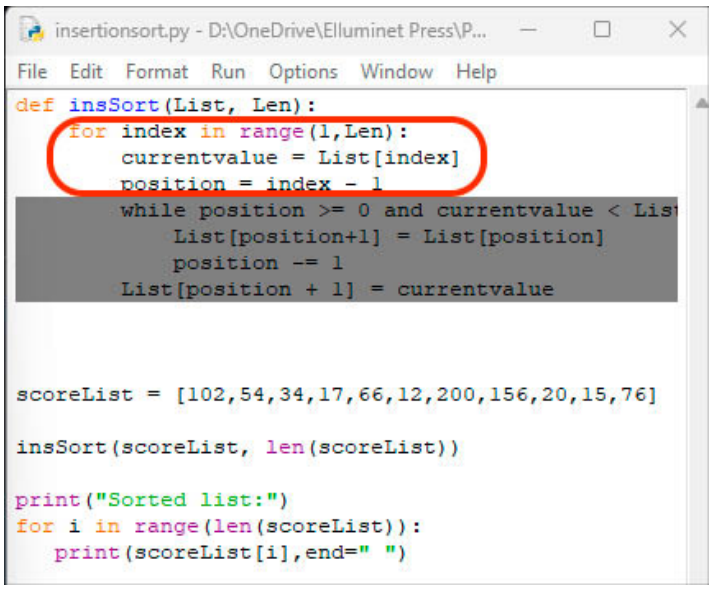
Here's another example, a recursive function to print out the Fibonacci numbers. Have a look at recursion.py. Step through the code, see how it works.



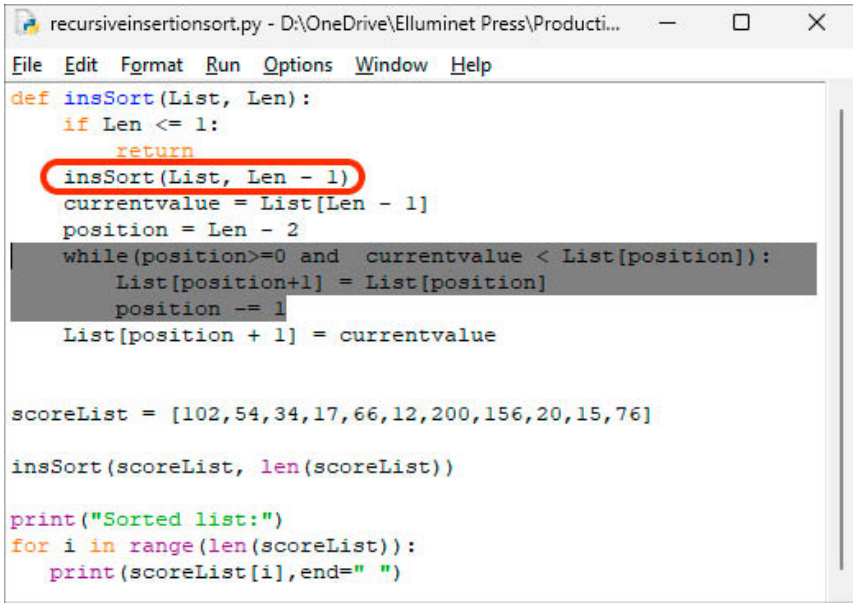
Recursive programs can also be written using iteration, so why bother with recursion? Recursive programs allow programmers to write efficient functions using a minimal amount of code and is useful technique that can reduce the length of code and make it easier to read and write.

Recursion works well for algorithms such as traversing a binary tree, or a sort algorithm and generating fractals. However, if performance is vital, it is better to use iteration, as recursion can be a lot slower.

Here's an example of an insertion sort written using iteration (while loop). Take a look at insertionsort.py



Here's the same algorithm written using recursion. Take a look at recursiveinsertionsort.py



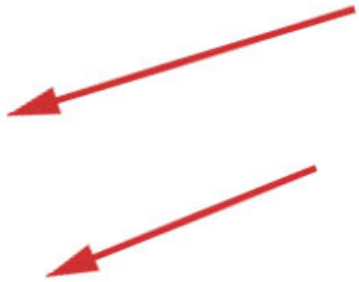
Spot the difference.

Lab Exercises

1. Write a program that accepts a number from the user and uses a function to square the number then return the result. Print the result to the screen.
2. Write a function that returns the largest of two numbers. Test the function and print the results to the screen.
3. What is the difference between a local and a global variable?
4. Write a program that prints first 10 positive numbers using a recursive function
5. Whats the difference between a parameter and an argument?
6. What is a built in function?
7. What is a user defined functions?
8. What makes a function recursive?
9. What are the advantages and disadvantages or recursion?
10. Have a look at the following program

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

Trace the function calls of the program.



Using Modules

When developing more complex python applications, as the program grows in size, it's a good idea to split it up into several files for easier maintenance, and reusability of the code. To do this, we use modules.

Modules are simply files with the Dot PY extension, containing code that can be imported into another program.

In doing this we can build up a code library that contains a set of functions that you want to include when developing larger applications.

In this section, we'll take a look at how to create modules and include them in our python programs. Have a look at the video demos.

`elluminetpress.com/pyfunctions`

You'll also need the source files in the directory Chapter 07.

Importing Modules

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

- **math** — Mathematical functions
- **turtle** — Turtle graphics
- **tkinter** — GUI interface toolkit
- **pygame** — Tooklit for creating games and other multimedia applications.

To import the modules into your code use the import keyword. In this example, I'm going to use the import keyword to import the turtle graphics module into a python program. To do this we enter the following line at the top of the program:

```
import moduleName
```

To call a function from an imported module use

```
moduleName.function()
```

For example, if we wanted to use turtle graphics we'd import turtle

```
import turtle
```

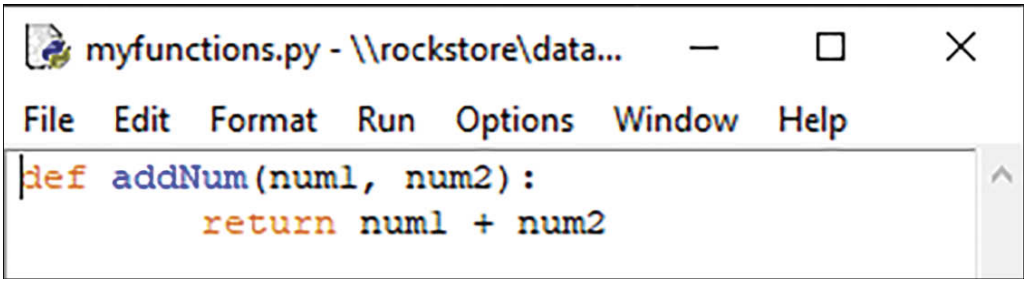
To use a function from that module

`turtle.functionname()` eg: `turtle.forward(100)`

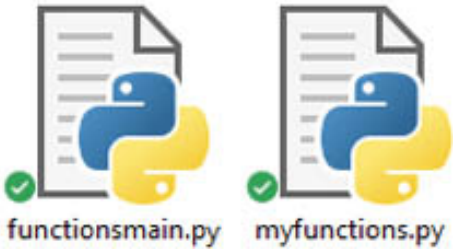
Creating your Own Modules

You can declare and store your functions in a separate file and import them into your main program.

All function definitions can be stored in a file eg myfunctions.py

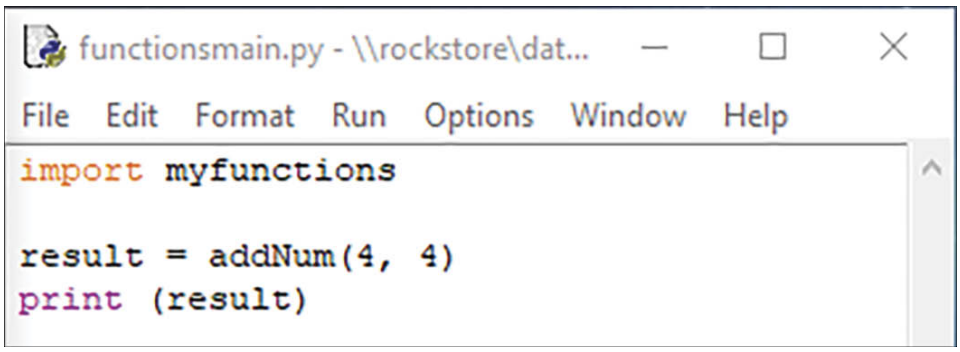


Here we end up with two files: functionsmain.py and myfunctions.py

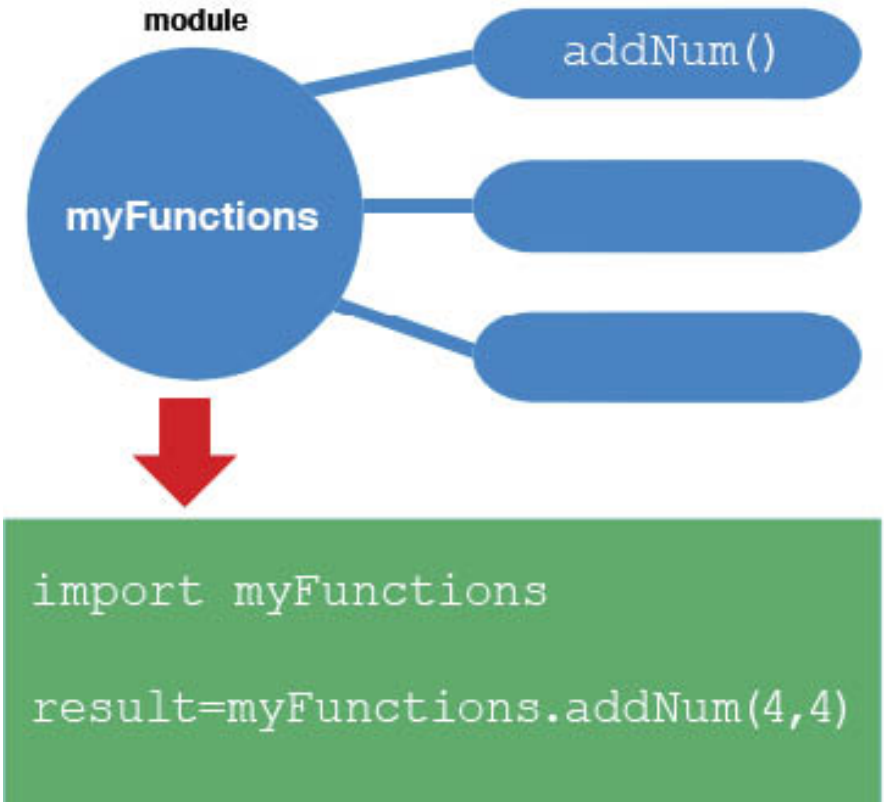


The main program could be called functionmain.py. All the functions for this program are stored in the file myfunctions.py

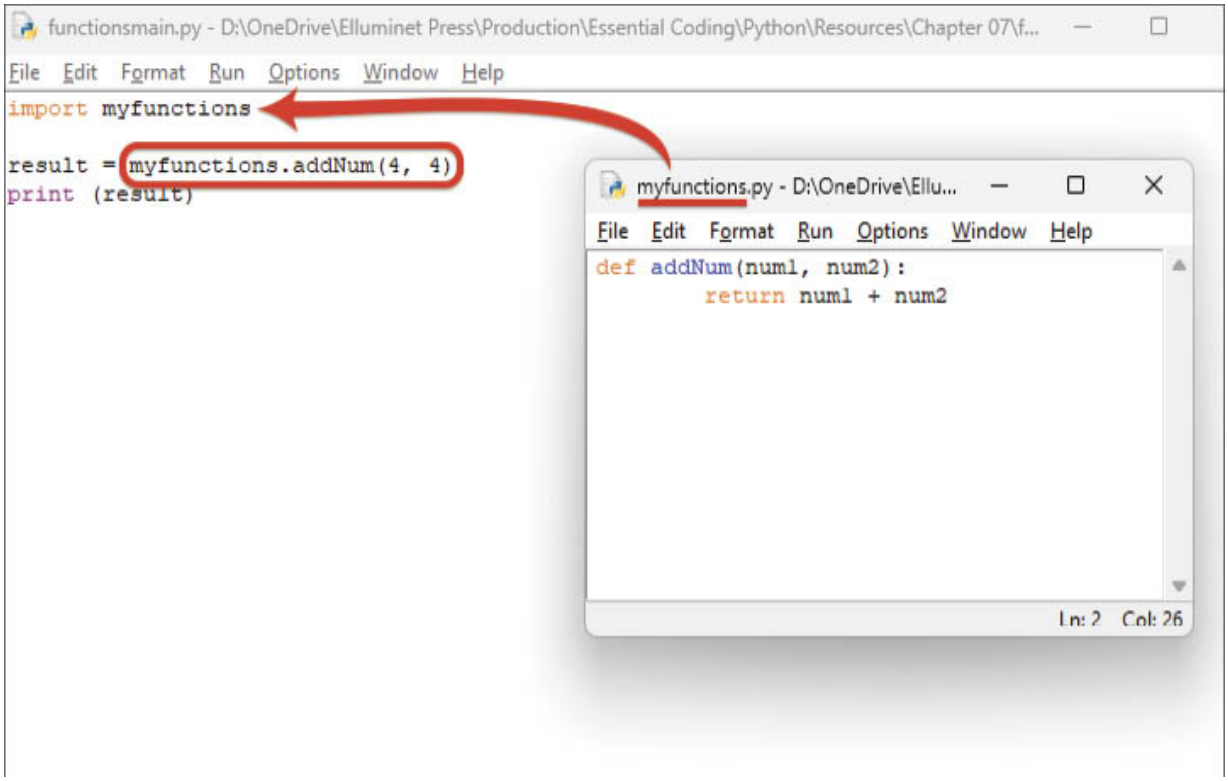
At the top of the main program, you'll need to import your functions stored in the other file (myfunctions.py). Strip off the file extension (.py).



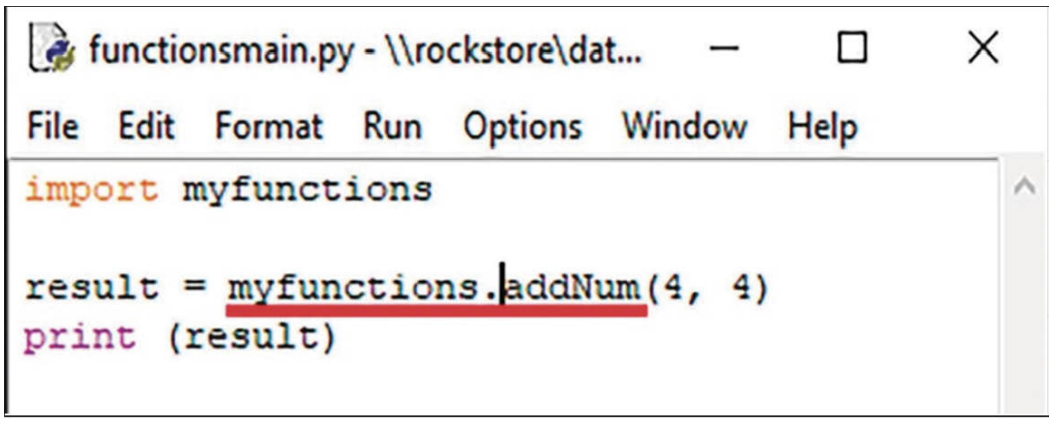
This is called a module. Any functions declared will be included in the main program.



You can include these functions in any program you need to. This makes maintenance easier.



Now to call any functions from that module, you need to specify the module name followed by the function name.



Lab Exercises

1. Write a function that accepts a number from the user and uses a function to square the number then return the result.
2. Save this file as a module
3. Import the module you just created into a new program.
4. Call the function in the module

Exception Handling

An exception is an error that occurs during execution of a program, sometimes called a runtime error. This could be a ‘file not found’ error if you are trying to load a file that doesn’t exist, or a ‘type error’ if you type text into a field when the program is expecting a number.

Exceptions are useful for handling errors encountered with file handling, network access, and data input.

These errors can be handled gracefully using Python’s exception handling procedures.

You’ll also need the source files in the directory Chapter 08.

Types of Exception

Here's a list of built in exceptions according to the Python documentation.

Exception	Cause
AssertionError	Raised when assert statement fails.
AttributeError	Raised when attribute assignment or reference fails.
EOFError	Raised when the input() functions hits end-of-file condition.
FloatingPointError	Raised when a floating point operation fails.
GeneratorExit	Raise when a generator's close() method is called.
ImportError	Raised when the imported module is not found.
IndexError	Raised when index of a sequence is out of range.
KeyError	Raised when a key is not found in a dictionary.
KeyboardInterrupt	Raised when the user hits interrupt key (Ctrl+c or delete).
MemoryError	Raised when an operation runs out of memory.
NameError	Raised when a variable is not found in local or global scope.
NotImplementedError	Raised by abstract methods.
OSError	Raised when system operation causes system related error.
OverflowError	Raised when result of an arithmetic operation is too large to be represented.
ReferenceError	Raised when a weak reference proxy is used to access a garbage collected referent.
RuntimeError	Raised when an error does not fall under any other category.
StopIteration	Raised by next() function to indicate that there is no further item to be returned by iterator.
SyntaxError	Raised by parser when syntax error is encountered.
IndentationError	Raised when there is incorrect indentation.
TabError	Raised when indentation consists of inconsistent tabs and spaces.
SystemError	Raised when interpreter detects internal error.
SystemExit	Raised by sys.exit() function.
TypeError	Raised when a function or operation is applied to an object of incorrect type.
UnboundLocalError	Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
UnicodeError	Raised when a Unicode-related encoding or decoding error occurs.
UnicodeEncodeError	Raised when a Unicode-related error occurs during encoding.
UnicodeDecodeError	Raised when a Unicode-related error occurs during decoding.
UnicodeTranslateError	Raised when a Unicode-related error occurs during translating.
ValueError	Raised when a function gets argument of correct type but improper value.
ZeroDivisionError	Raised when second operand of division or modulo operation is zero.

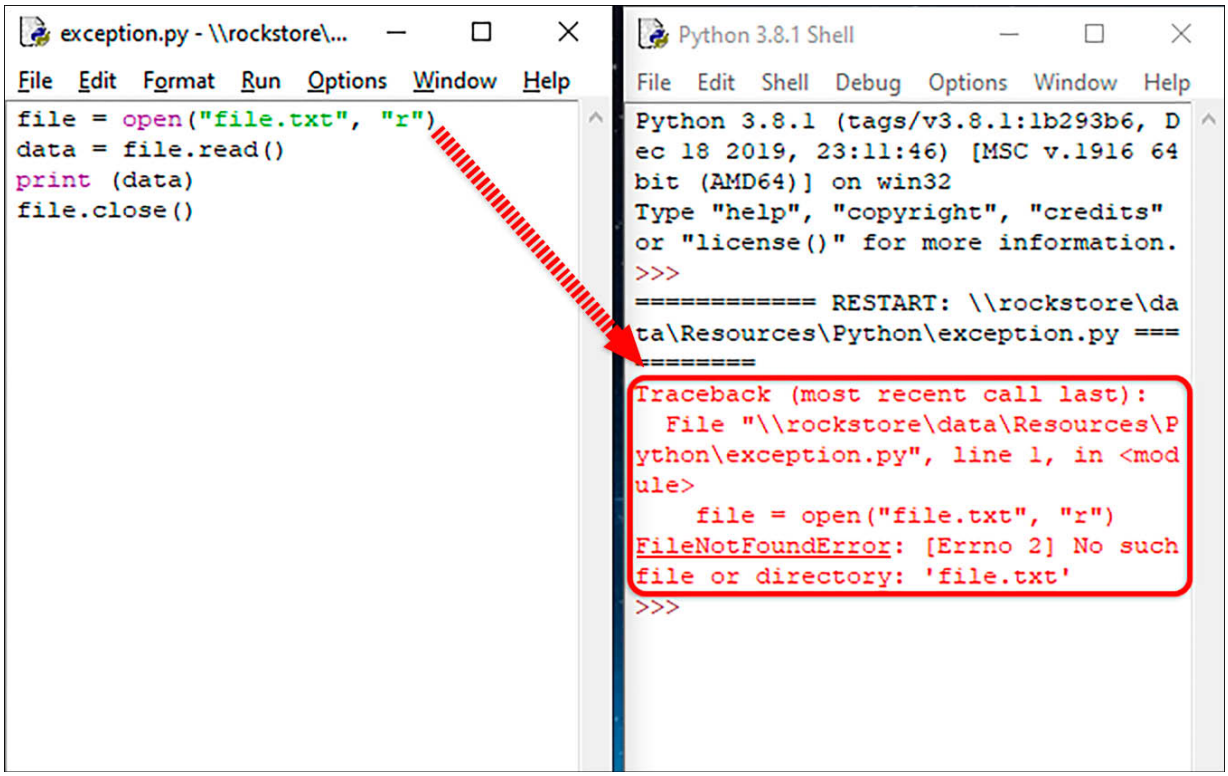
Whenever an exception occurs, the interpreter halts the execution of the program, and raises an exception error as shown in the table

above.

You can catch these exceptions using the `try` and `except` keywords and provide code to handle the error.

Catching Exceptions

If we run the following code, the Python interpreter will raise a `FileNotFoundError` exception because there is no file called 'file.txt'. This will cause the program to crash.



You can catch exceptions using the `try` and `except` keywords. Just put your code in the 'try' block and your error handling code for each exception in the 'exception' block as shown below.

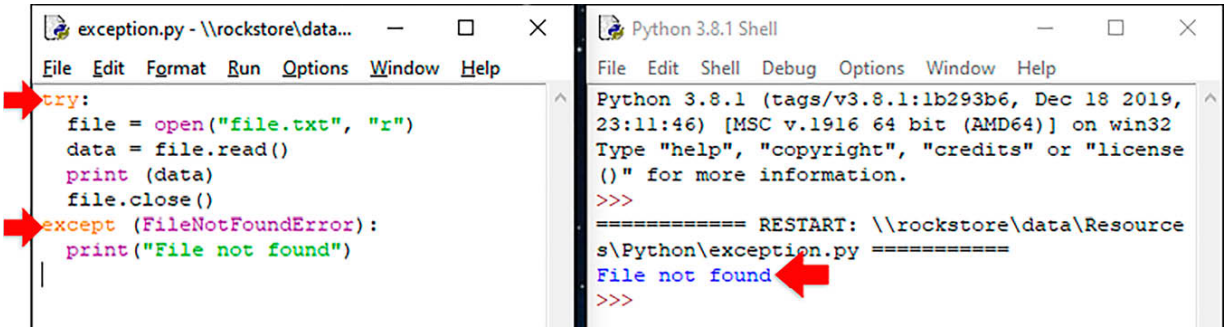
```
try:
# Code to execute as normal
except [exception (see table above)]:
# Code to deal with exception
```

The `try` block contains the code to execute. The `except` block contains the code to handle the error.

Let's take a look at the program again. We can take our code and place it in the 'try' block. Then add an 'except' block to deal with the

error. If we look at the error message in the shell, we see this is a `FileNotFoundError`. We can add this after the 'except' keyword.

Now, when we run the program, we get a simple message rather than an ugly error.



Use the `finally` block to perform any clean up. The `finally` statement runs regardless of whether the `try` statement produces an exception or not.

```
try:
file = open("file.txt", "r")
data = file.read()
except (FileNotFoundError):
print("File not found")
finally:
f.close()
```

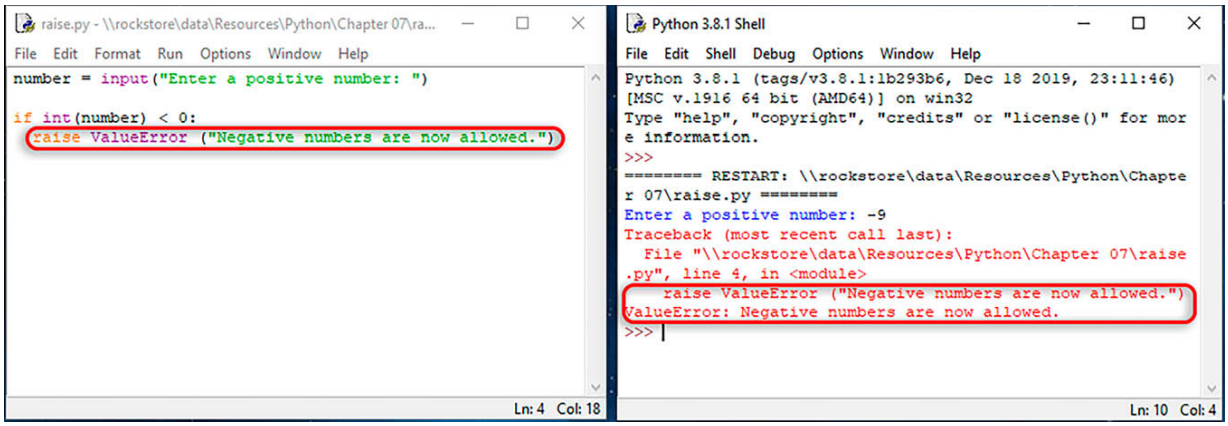
Raising your Own Exceptions

Use the `raise` keyword to force a specified exception to occur followed by the type of error using the table on page 103

```
if number < 0:
```

raise `ValueError` `("Negative numbers only.")`

Here in the file raise.py, we've raised a ValueError exception.



Object Oriented Programming

Python is an object oriented programming language. This means that the program design is based around objects, rather than functions and logic.

Each object is created using a blueprint known as a class. Each class has attributes to contain the data an object needs to work with.

Each class also contains functions, called methods that perform operations on objects.

An object is an instance of a class. So, you could have a class called ‘car’ and use it to define an object called ‘merc’.

For this section, take a look at the video demos.

`elluminetpress.com/pyoop`

You’ll need the source files in the directory Chapter 09.

Class

A class is a user-defined blueprint or template that defines the attributes (variables) and methods (functions), and contains them all in a single unit called a class.

```
class ClassName :
```

For example, we could create a class called Person.

```
class Person :
```

Object

An object is an instance of a class. So, from the class called ‘person’ declared above, we can use it to create objects called ‘staff’, ‘student’, ‘lecturer’ and so on.

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

For example

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

Attribute

An attribute, is similar to a variable and is used temporarily store data. For example, name, dob and email.

```
class Person :

def __init__(self, name, dob, email):

self.name = name

self.dob = dob

self.email = email
```

The attributes are defined inside a special method called a constructor.

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

This method is called automatically when an object is created. Any parameters that are passed to the object are enclosed in parenthesis.

‘Self’ is a reference to the current object, and is used to access attributes that belongs to that object.

Method

A method is a function that is used by an object to perform an action. Methods are usually referenced by stating the object name dot method name. For example, `getAge()`

```
class Person :

...

def getAge(self):

currentDate = date.today()

age = currentDate.year - self.dob.year

return age
```

Principles of OOP

The four principles of OOP are: encapsulation, inheritance, polymorphism and abstraction.

Encapsulation

With encapsulation, you restrict access to methods and attributes within a certain class. This prevents accidental modification of data and unwanted changes to other objects.

Inheritance

A class can inherit all the methods and attributes from another class. If a class called ‘person’ had name, age, and dob. We could define two other child classes called ‘student’ and ‘staff’. Both can inherit the methods and attributes from the ‘person’ class.

Polymorphism

Polymorphism allows us to define methods in the child class with the same name as defined in the parent class. This is known as method overriding. Polymorphism also allows us to define methods that can take many forms.

Abstraction

Abstraction is the process of reducing objects to their essence so that only the necessary elements are represented. In other words you remove all irrelevant information about an object in order to reduce its complexity.

Classes & Objects

As we saw in the introduction, you can define a class using the `class` keyword. Let's take a closer look.

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

Here, we have created a class called Person.

```
class Person :
```

All classes have a function called `__init__()` which is automatically executed when the class is initiated. Use the `__init__()` function to initialize attributes.

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

Remember the `self` keyword represents the current instance of the class (ie the object created from the class). By using the `self` keyword you can access the attributes of the object itself.

When you declare a method, you pass the current instance of the class (ie the object itself), along with any other parameters required, to the method.

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

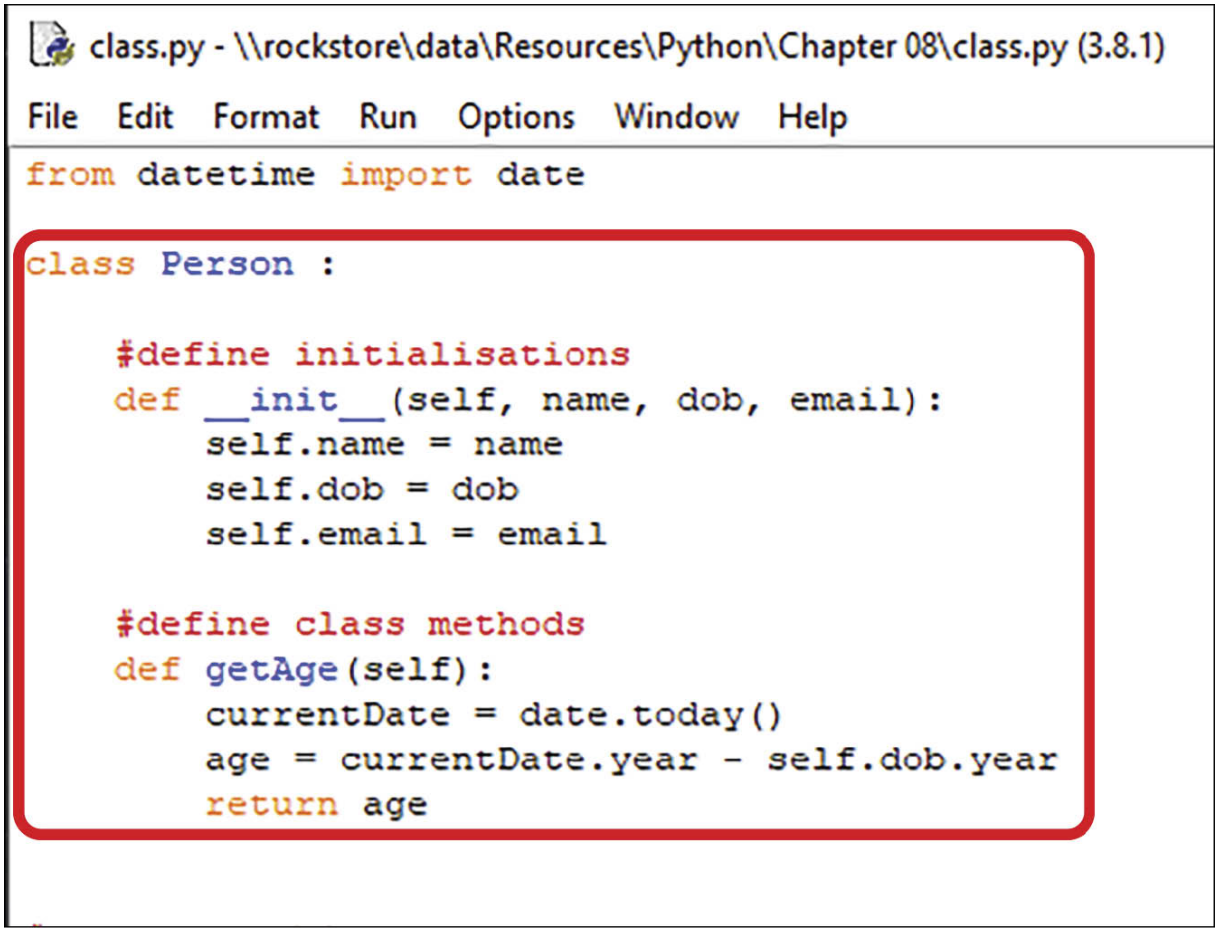
When you need to use any attribute, you use `self` followed by the attribute name.

```
self.attribute-name
```

So for example:

```
self.email
```

Lets take a look at a program. Open the file class.py. Here we've defined our 'Person' class.



To create an object from the class, call the class Person(...) and pass any data using parenthesis ( ). Assign the new object to a variable eg: person.

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

To use the object use the dot notation.

```
classname.method()
```

or

`classname.attribute`

So in our example, to use our attributes:

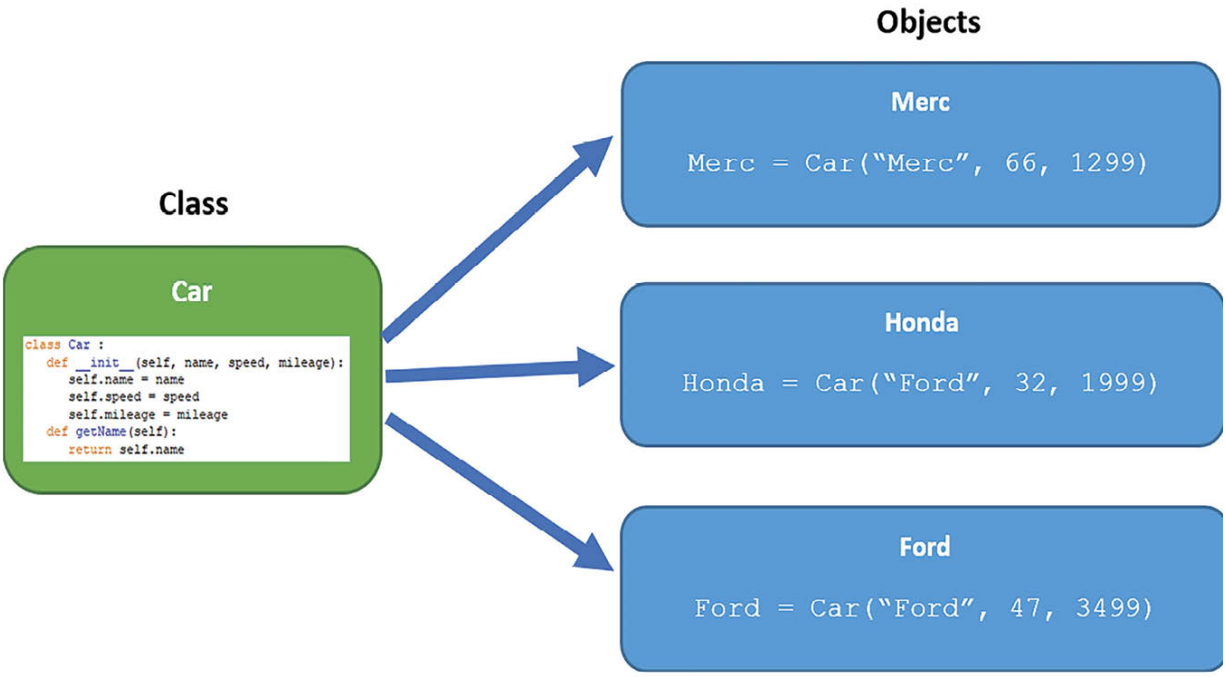
```
print(person.name)
print(person.email)
```

...and to use our methods:

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

Lets have a look at another example. You could use the Car class to create some objects such as Merc, Honda, and Ford.

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



When an object is created, the Python interpreter automatically calls the `__init__()` method. This method is called a constructor and is used to initialize objects created with the class.

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

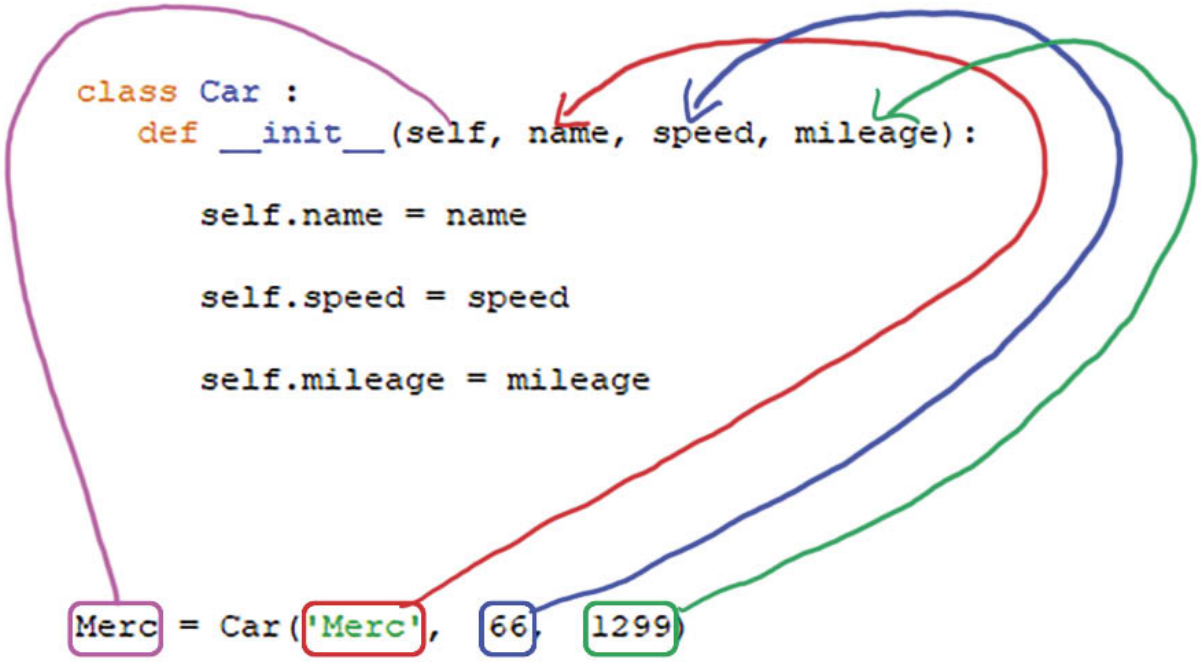
Within the `__init__()` method, we initialise the attributes.

```
self.name = name
```

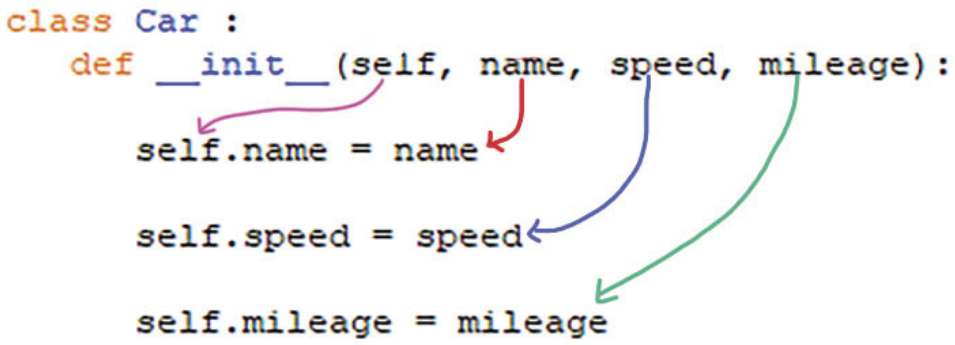
```
self.speed = speed
self.mileage = mileage
```

Let's take a look at what is happening. In the `__init__()` method, self refers to the instance of the object itself and is automatically passed.

Here at the bottom of the diagram below, when we create an object (Merc) from the Car class, we pass some data name, speed, mileage.



Next we assign the attributes passed to the `__init__()` method to the attributes of the object.



We can use the object of a class to perform actions. For example:

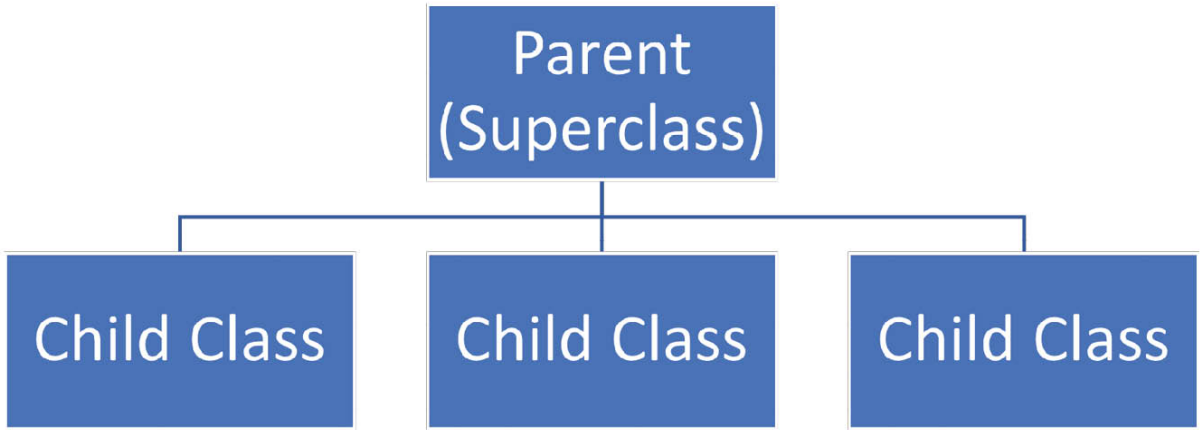
```
Merc.getName()
```

or

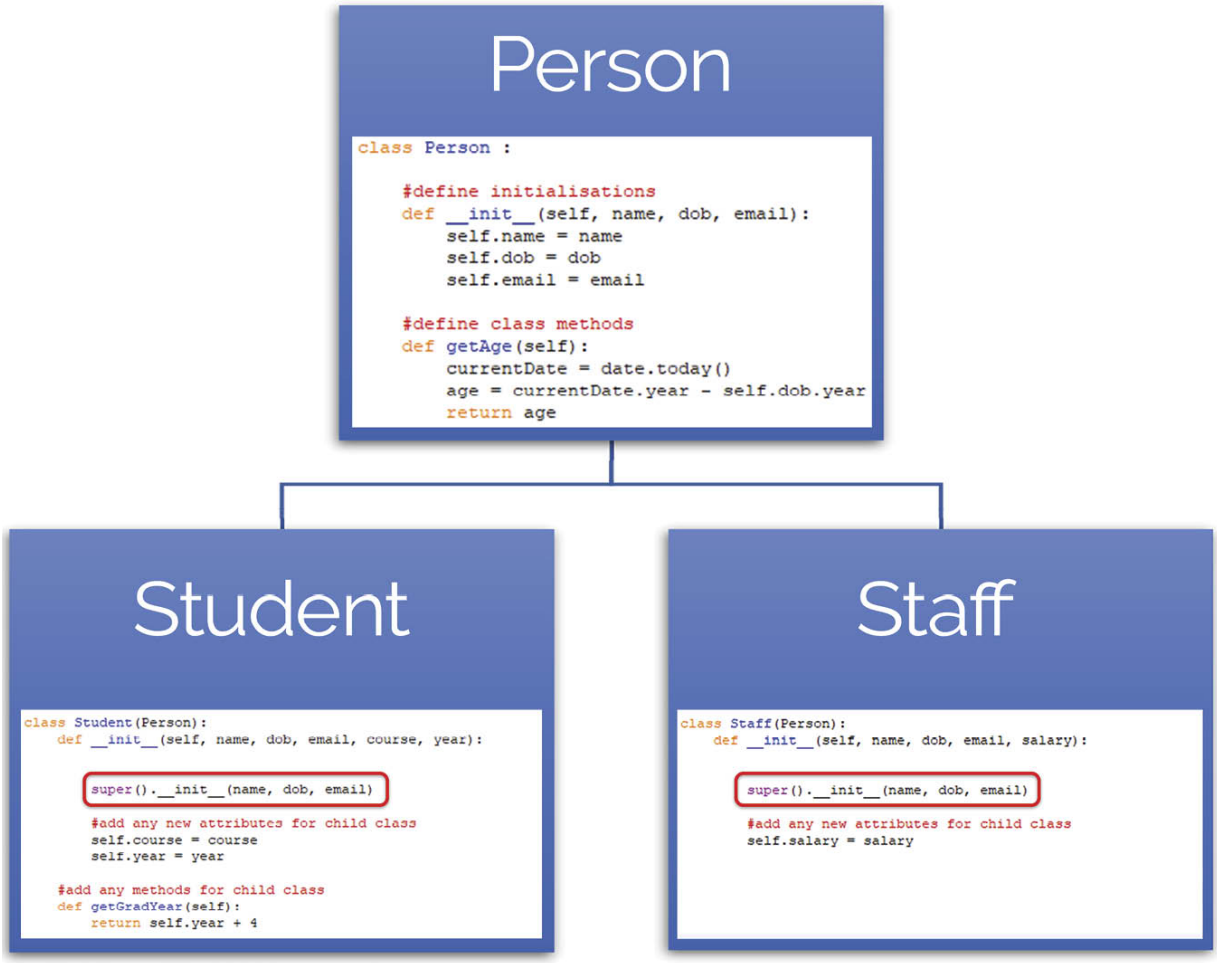
```
Merc.speed = 66
```

Inheritance

A class can inherit all the methods and attributes from another class.



If a class called ‘person’ had name, age, and dob. We could define two other child classes called ‘student’ and ‘staff’. Both can inherit the methods and attributes from the ‘person’ class.



When defining your child classes, you might need to access methods defined in the parent. To do this, use the `super()` function. The `super()` function allows us to refer to the parent class explicitly.

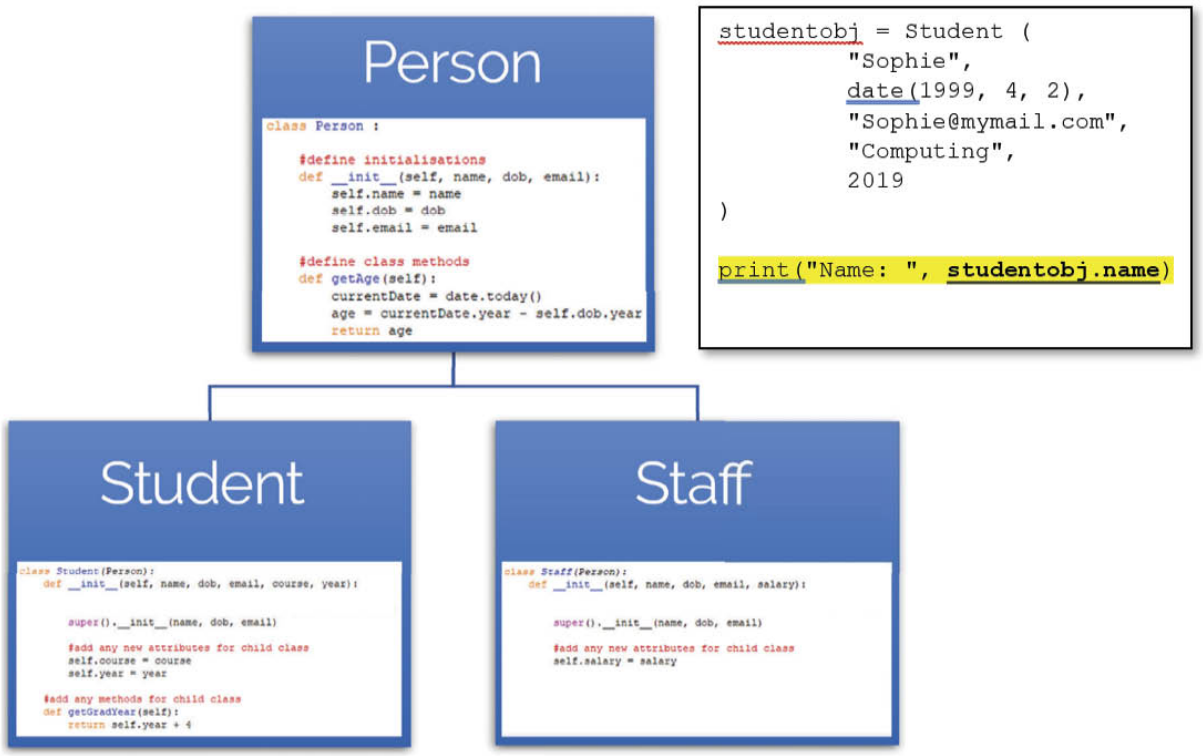
To create a child class, declare the class as normal, except include the parent class in parenthesis after the class name. So:

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

For example, if you need to access the attributes in the parent, you'll need to call the parent's `__init__()` constructor using `super()` in the child class.

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

In the example below, in the print statement, we are calling the "name" attribute which is defined in the parent class Person, from the object (studentobj) created using the Student class.

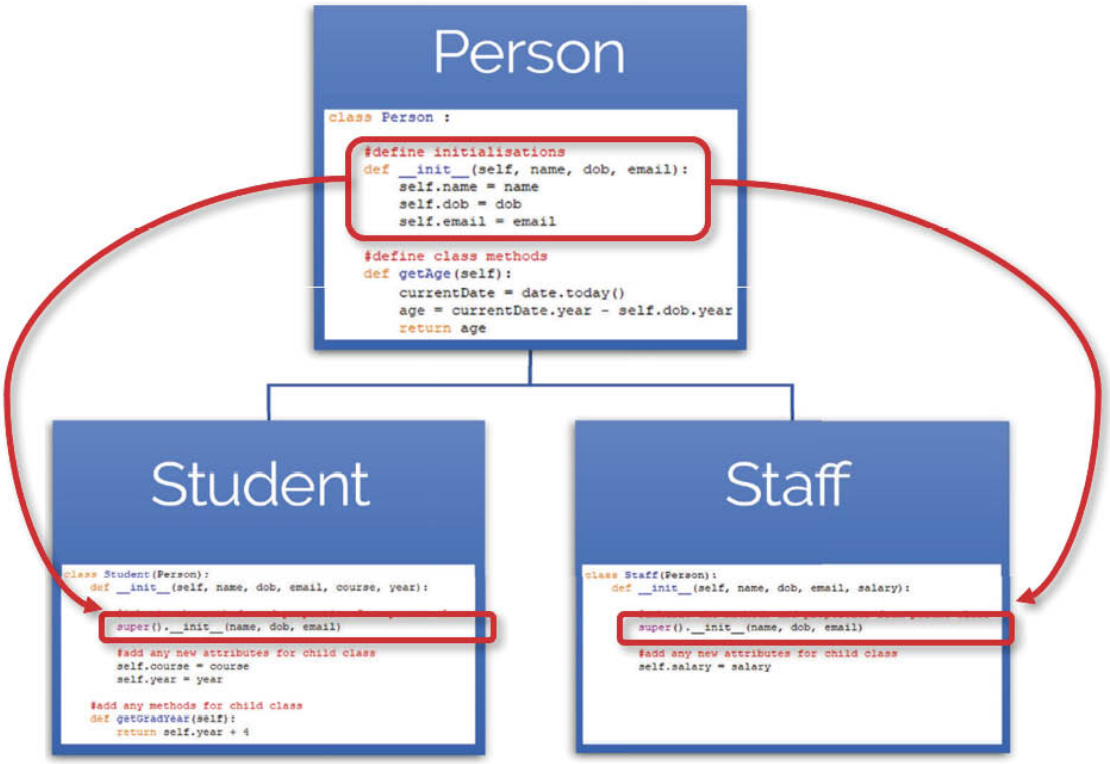


To access this we need to use `super()` to call the `__init__()` constructor method defined in the parent class (Person).

```
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
```

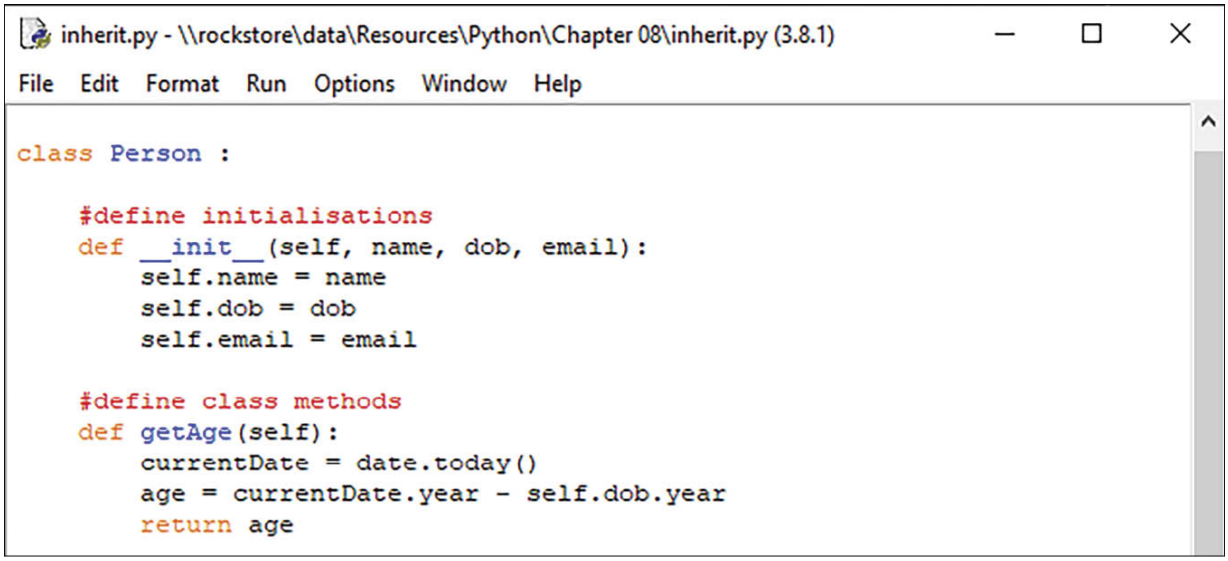
```
class Staff(Person):
    def __init__(self, name, dob, email, salary):

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

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

Here in the two child classes (student & staff) we've called the `super()` function in the `__init__()` method of the child class.



The `super()` function calls the __init__() method of parent class, which gives the child class access to all the attributes of the parent class. If we don't need to access the methods defined in the parent class from an object created from the child class, then we don't need the `super()` function.

Open the file inherit.py. He, we've created a class called person. We've also created to child classes called student and staff.



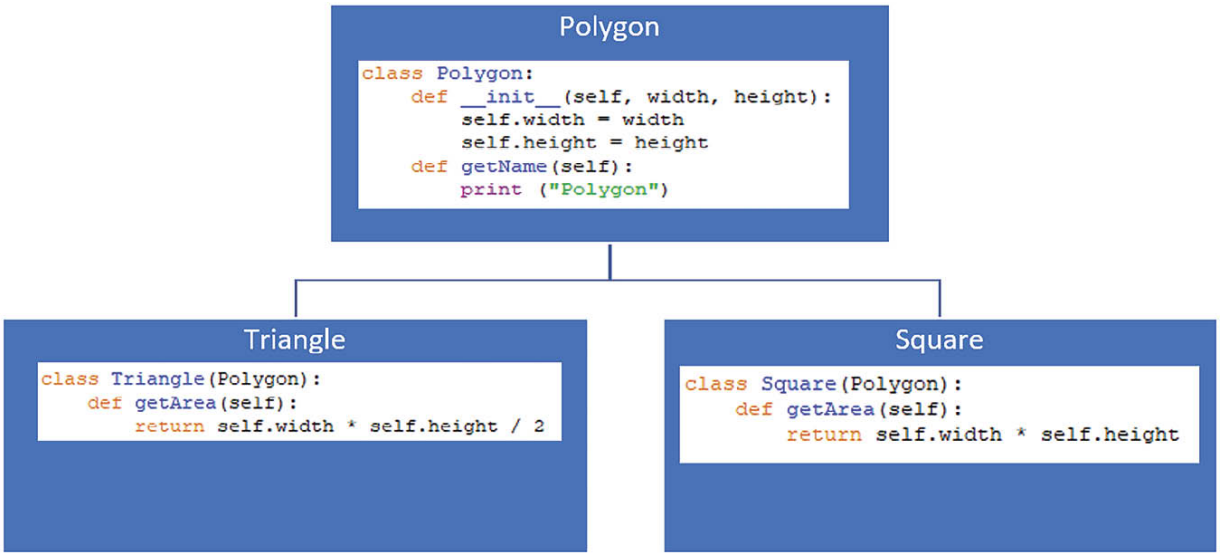
We can create a lecturer object from the class Staff

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

To reference the attributes in the object we use the dot notation.

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

Lets take a look at another example, here we've defined a parent class called Polygon and created a child class called Triangle.



When we create an object from the child class Triangle…

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

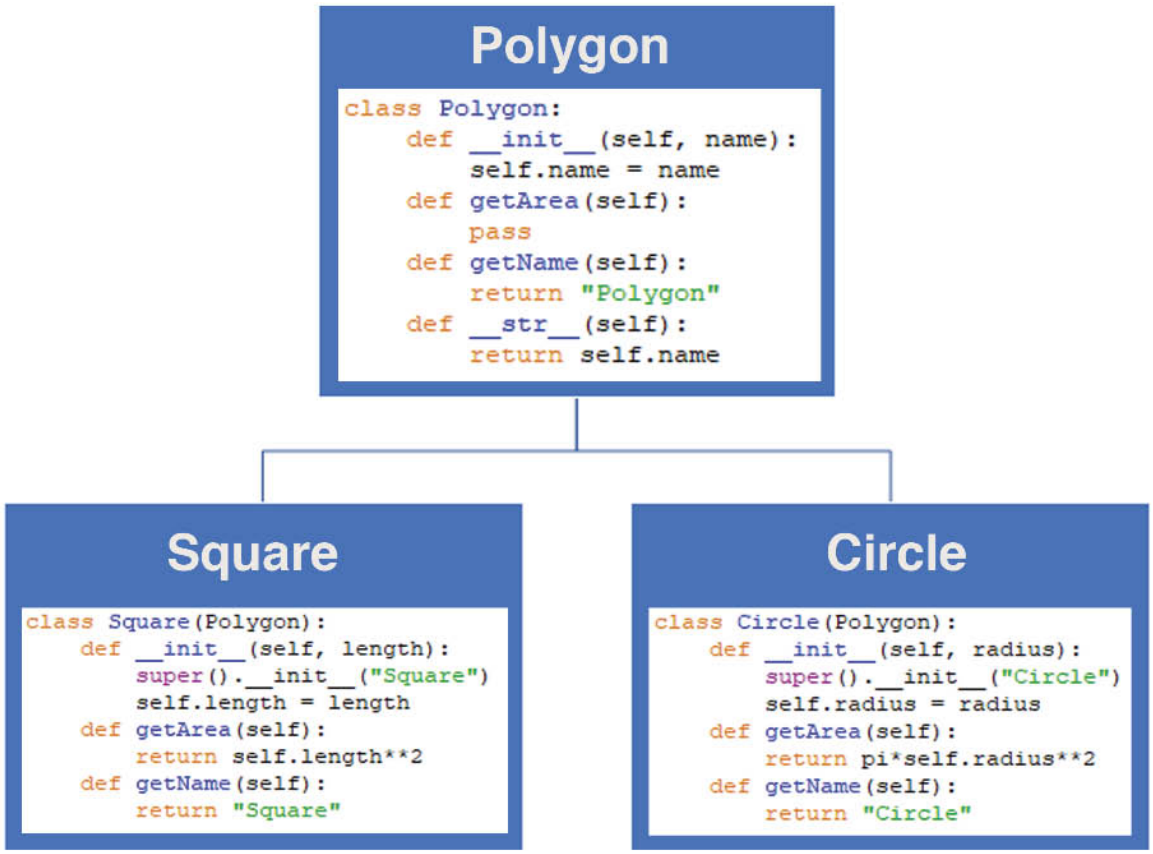
…we are not calling any methods defined in the parent class (Polygon). Here, we're calling the `getArea()` method from the triangle object (triobj) defined in the Triangle class.

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

Polymorphism

Polymorphism combined with inheritance allows us to define methods in the child class with the same name as defined in the parent class. This is known as method overriding.

In this example we're going to create three classes. To be polymorphic, each of these classes needs to have an interface in common. So we define methods for each class that have the same name. In this case, we have defined a method that calculates the area in each class (square and circle).



Open the file inhertit.py.

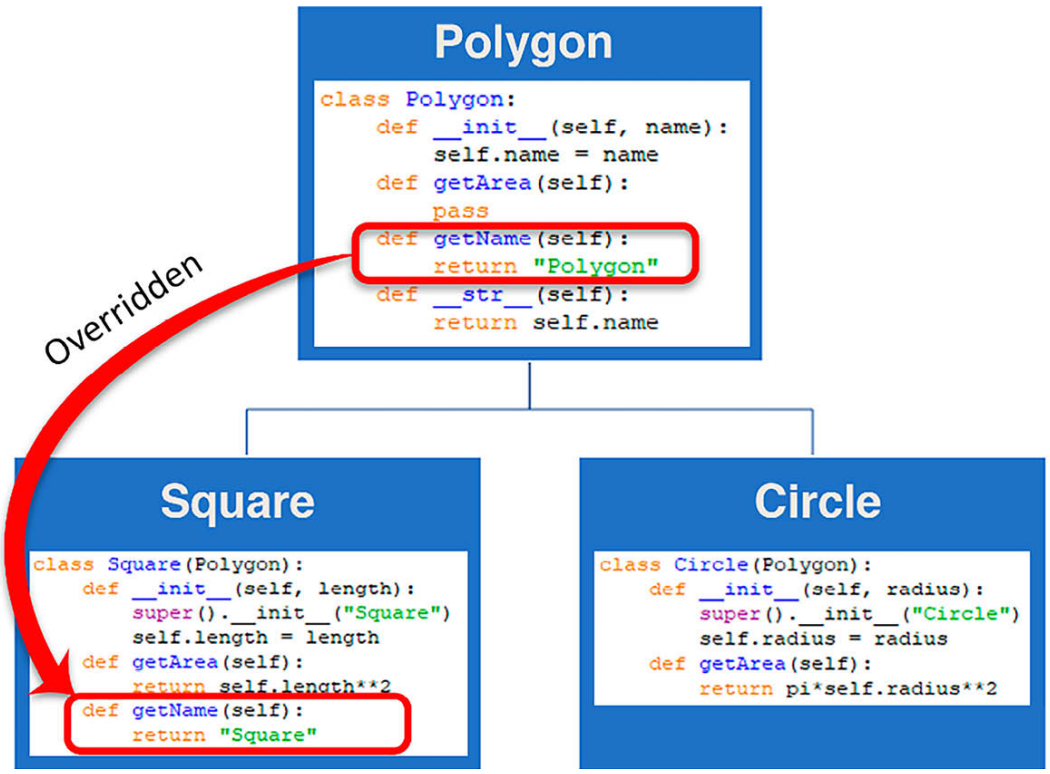
Polymorphism allows us to access these overridden methods and attributes that have the same name as defined in the parent class. For example, if we create an object called squareObj from the Square class:

```
squareObj = Square(7)
```

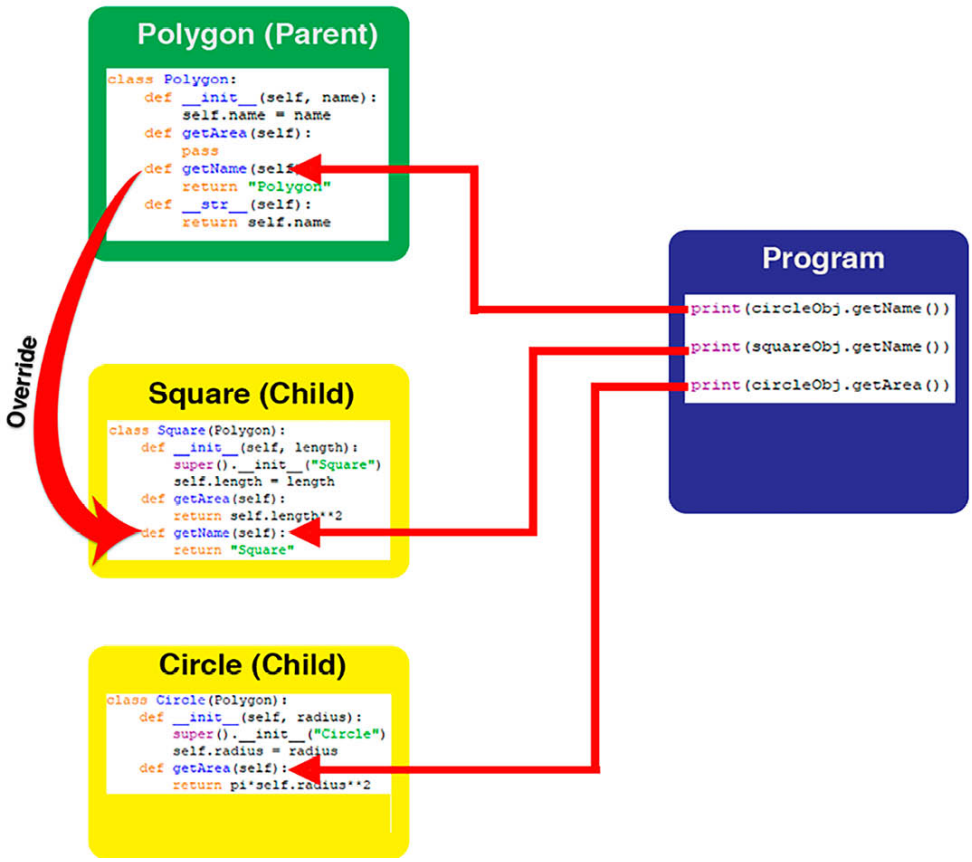
Then call the `getName()` method.

```
print(squareObj.getName())
```

The interpreter automatically recognizes that the `getName()` method for squareObj is overridden.



So the `getName()` method defined in the child class (Square) is used. The `getName()` method defined in the Square class is overriding the `getName()` method defined in the parent (Polygon). Similarly with `getArea()`.



You'll also notice that we have called the `getName()` method from the circle class object circleObj.

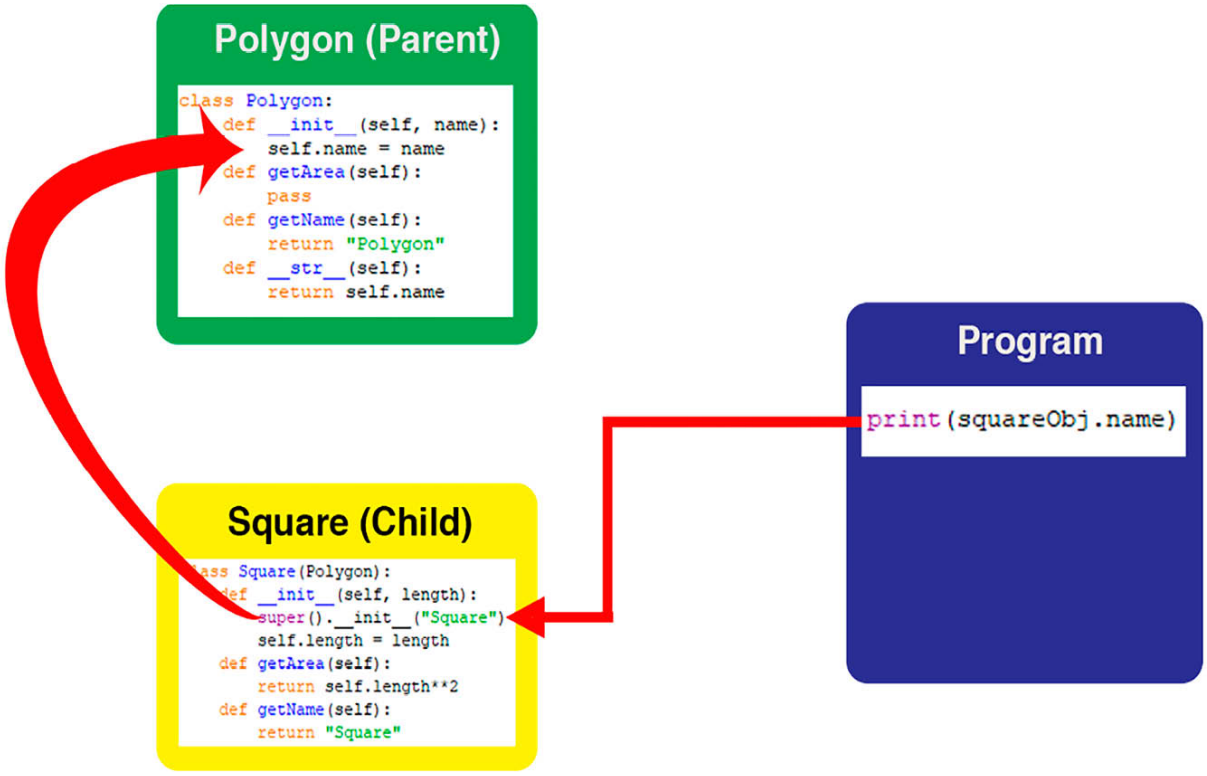
```
print (circleObj.getName()
```

In this case, there is no `getName()` method defined in the circle class, so the interpreter will call the `getName()` method from the parent.

If we reference the "name" attribute from squareObj

```
print(squareObj.name)
```

You'll notice that "name" is defined in the parent class (Polygon) not in the child class (Square).



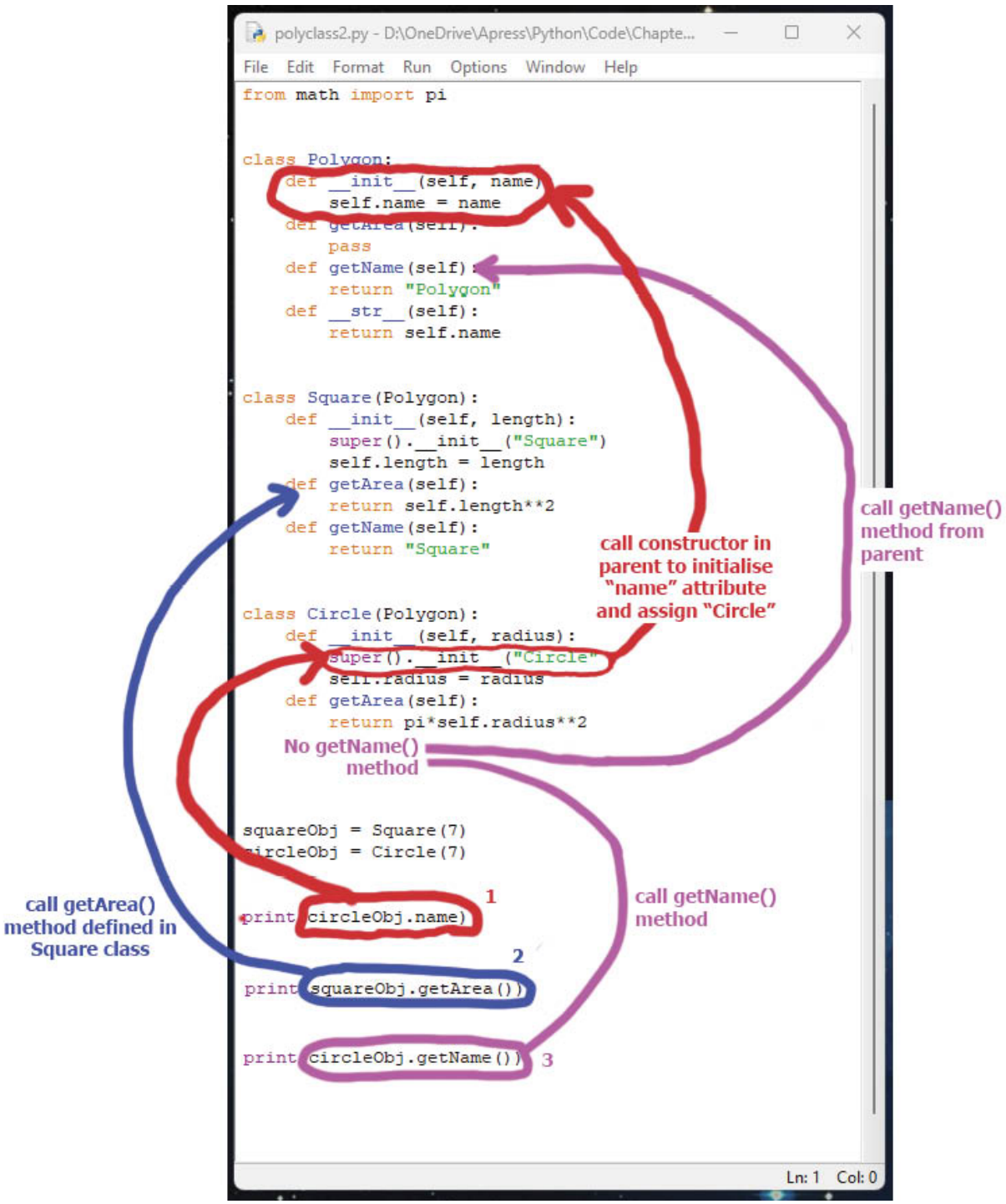
To access this attribute from the child, we need the `super()` function in the child class to call the `__init__()` constructor method defined in the parent class (Polygon).

```
super().__init__("Square")
```

this calls the following method from the Polygon class

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

Have a look at polyclass2.py


polyclass2.py - D:\OneDrive\Apress\Python\Code\Chapte...
File Edit Format Run Options Window Help
from math import pi
class Polygon:
def __init__(self, name):
self.name = name
def getArea(self):
pass
def getName(self):
return "Polygon"
def __str__(self):
return self.name
class Square(Polygon):
def __init__(self, length):
super().__init__("Square")
self.length = length
def getArea(self):
return self.length**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(7)
circleObj = Circle(7)
print(circleObj.name)
print(squareObj.getArea())
print(circleObj.getName())
Ln: 1 Col: 0
call constructor in parent to initialise "name" attribute and assign "Circle"
call getName() method from parent
No getName() method
call getArea() method defined in Square class
call getName() method
1
2
3

Lab Exercises

1. Declare a new class called Vehicle without any attributes and methods
   Add some attributes to the Vehicle class such as

   1. Name
   2. Speed
   3. Mileage
4. Add a method to the Vehicle class to return the vehicle name
5. Create a child class called Taxi
6. Add a method to the Taxi class to collect the fare. Fare is calculated at 20c a mile.
7. What is a class? Show an example.
8. What is an object? Show an example.
9. What is a constructor? Show an example.
10. What is a method? Show an example.
11. What is `__init__()` used for? Show an example.
12. What is `super()` used for? Show an example.

Turtle Graphics

Turtle graphics operate much like a drawing board, in which you can execute various command to move a turtle around. We can use functions like forward() and right(). The turtle will travel along the path that you define using these functions, leaving a pen mark behind it.

For this section, take a look at the video demos.

`elluminetpress.com/pygraphics`

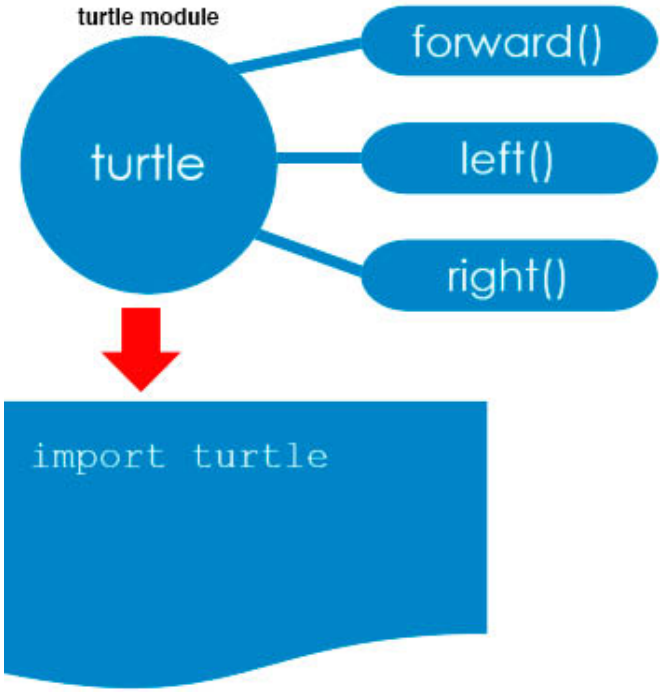
You'll also need the source files in the directory Chapter 10.

Importing Turtle Graphics Module

To begin using turtle graphics, we first need to import the modules

```
import turtle
```

This statement imports all the turtle graphics functions into the program.



Turtle Commands

We can move the turtle around using various commands. We can access these using the module's functions.

```
moduleName.functionName()
```

For example:

```
turtle.forward(distance)
turtle.right(degrees)
```

'turtle' is the name of the turtle module we imported earlier, and `forward()` is a function defined within the turtle graphics module. The argument distance is how far you want the turtle to move, and degrees is the number of degrees you want the turtle to turn (eg right 90º).

Let's put this into a program. Below we have our import statement to import all the turtle graphics modules. Below that we have a statement that moves the turtle forward, and one to finish the program.

Notice that we use the dot notation to access the functions in the turtle module.

```
moduleName.functionName()
```

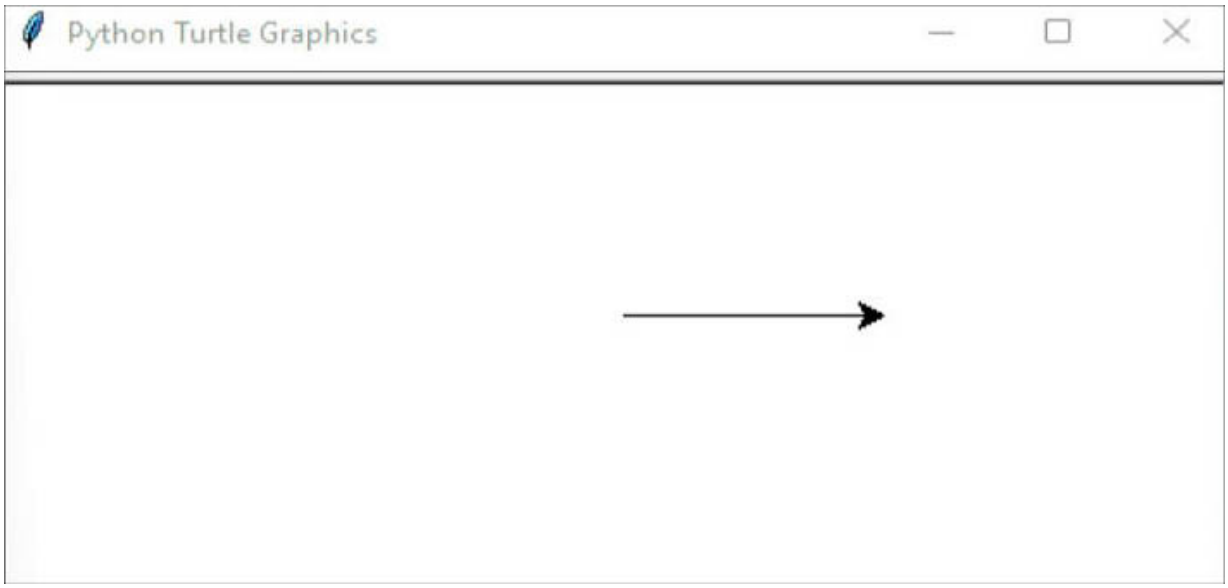
We want to access the forward function, so we specify the module

name it's in (turtle), followed by a dot, then the name of the function we want (forward). So we get

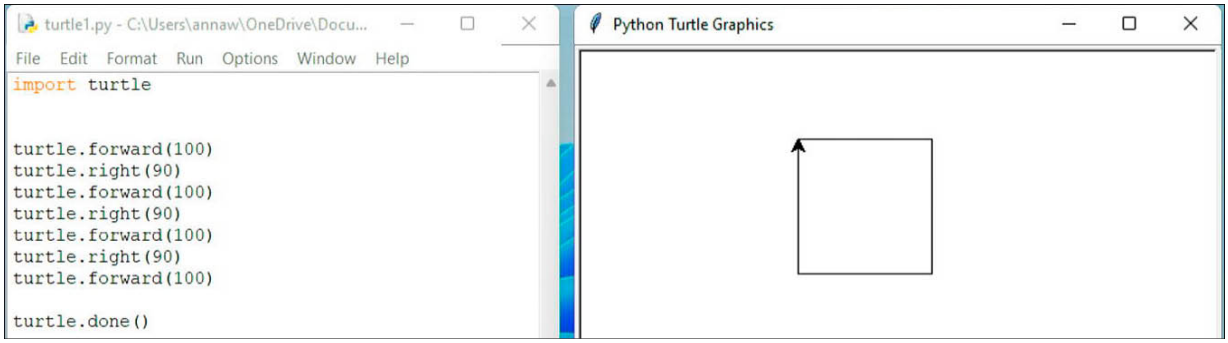
```
turtle.forward(100)
```

This will move the turtle 100 pixels forward.

Here we can see the output to the program, the turtle has moved 100 pixels to the right.



We can complete the program to draw a square.



You can also use to go backwards

```
turtle.backward(distance)
```

You can turn left, or turn right.

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

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

You can also use other commands to pickup and drop the turtle's pen. When you use `pendown()` the turtle will draw. If you execute `penup()` the turtle wont draw a line when he moves.

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

You can also change the pen color

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

You can jump to a specific co-ordinate on the screen

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

Execute .done() at the end of your program

```
turtle.done()
```

Customize the Turtle Window

We can customize the turtle window. First we create a turtle window.

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

Change the background color

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

Create a turtle object

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

Looping Commands

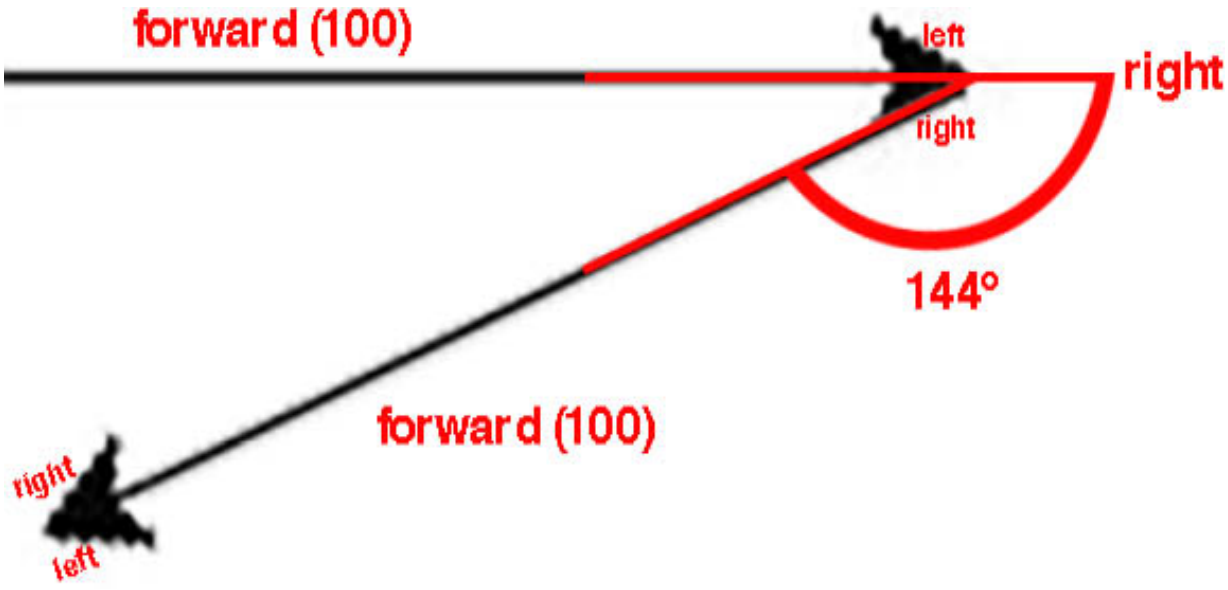
You can use loops to repeat commands to draw shapes. For example, if we wanted to draw a hexagon, we can use a for loop to repeat the commands for each side.

```
sides = 6
angle = 360 / sides

for index in range(sides):

polygon.forward(200)

polygon.right(angle)
```

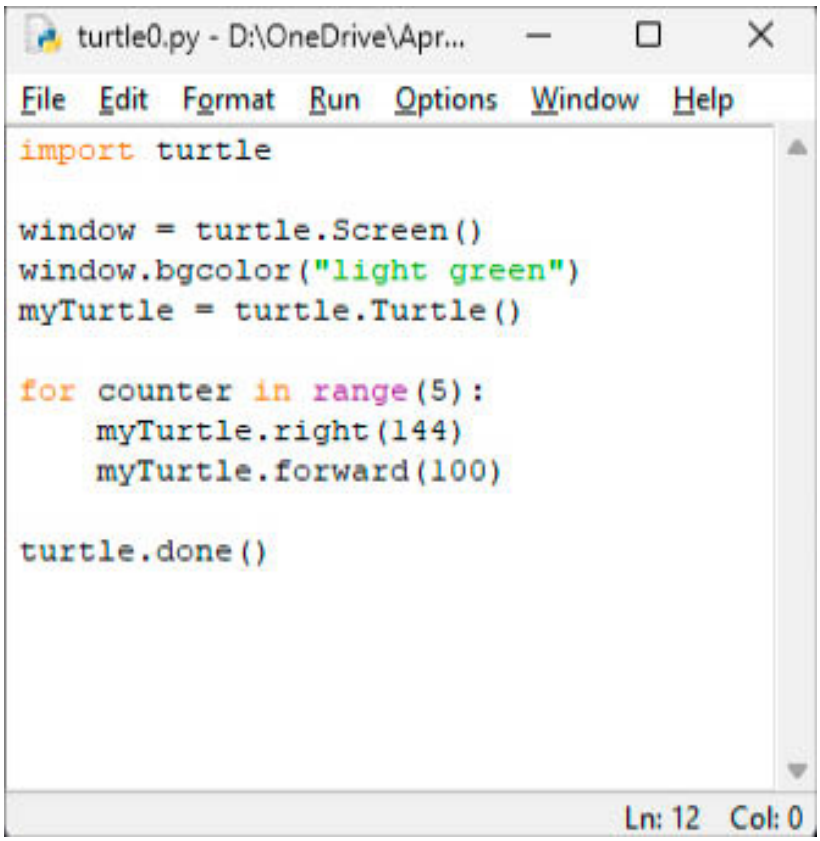
Or if you wanted to draw other shapes. How about a star?

```
for counter in range(5):

myTurtle.right(144)

myTurtle.forward(100)
```

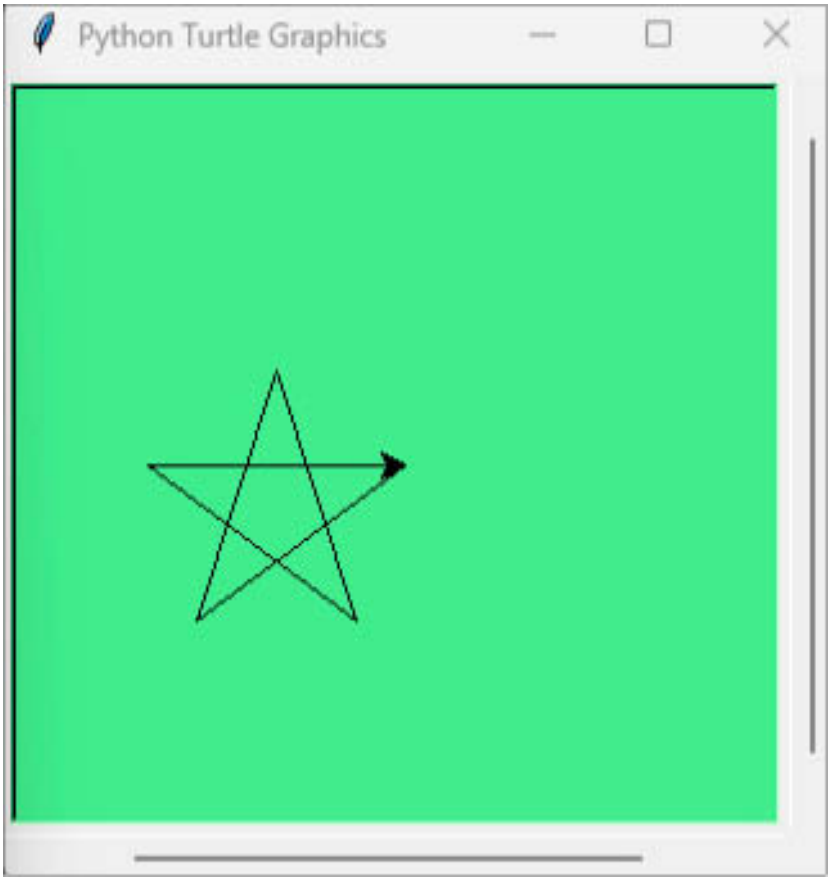
Here, we moved the turtle 100 forward, then rotated the turtle 144º to the right.



We repeat this to run 5 times with a for loop.



When you execute the program...



Notice that background color has changed

Try out some of the other Turtle Commands.

Command	Description
turtle.forward(distance)	Move the turtle forward by the specified distance
turtle.backward(distance)	Move the turtle backward by distance,
turtle.home()	Move turtle to the origin – coordinates (0,0)
turtle.penup()	Pull the pen up
turtle.pendown()	Pull the pen down
turtle.pencolor(colorstring)	Set pencolor to colorstring, such as "red", "yellow" etc.
turtle.circle(radius)	Draw a circle with given radius.
turtle.shape("turtle")	Sets the turtle shape to turtle.
turtle.undo()	Undo (repeatedly) the last turtle action(s)
turtle.clear()	Erases all drawings that currently appear in the graphics window.

See what patterns you can draw. How about a spiral?

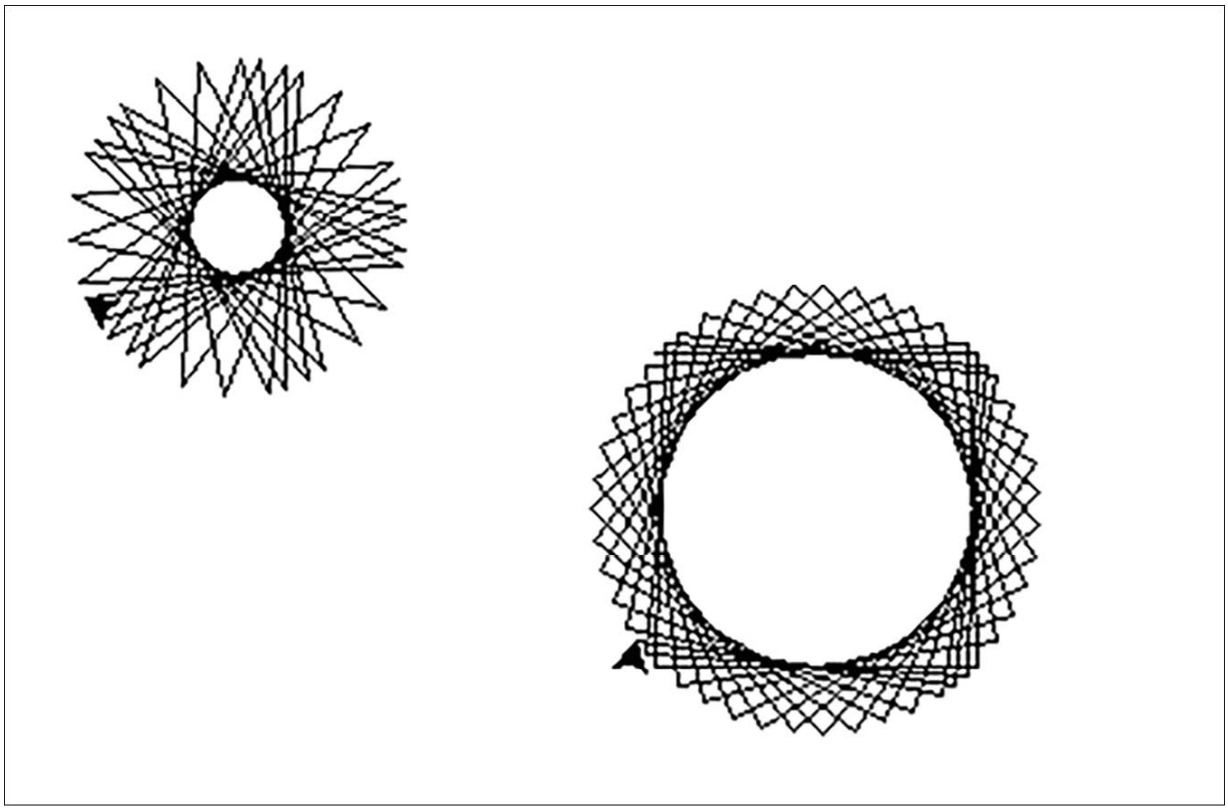
```
for i in range(50):
turtle.forward(100)
```

```
turtle.right(91 + 1)
```

Or maybe this one?

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

Give it a try

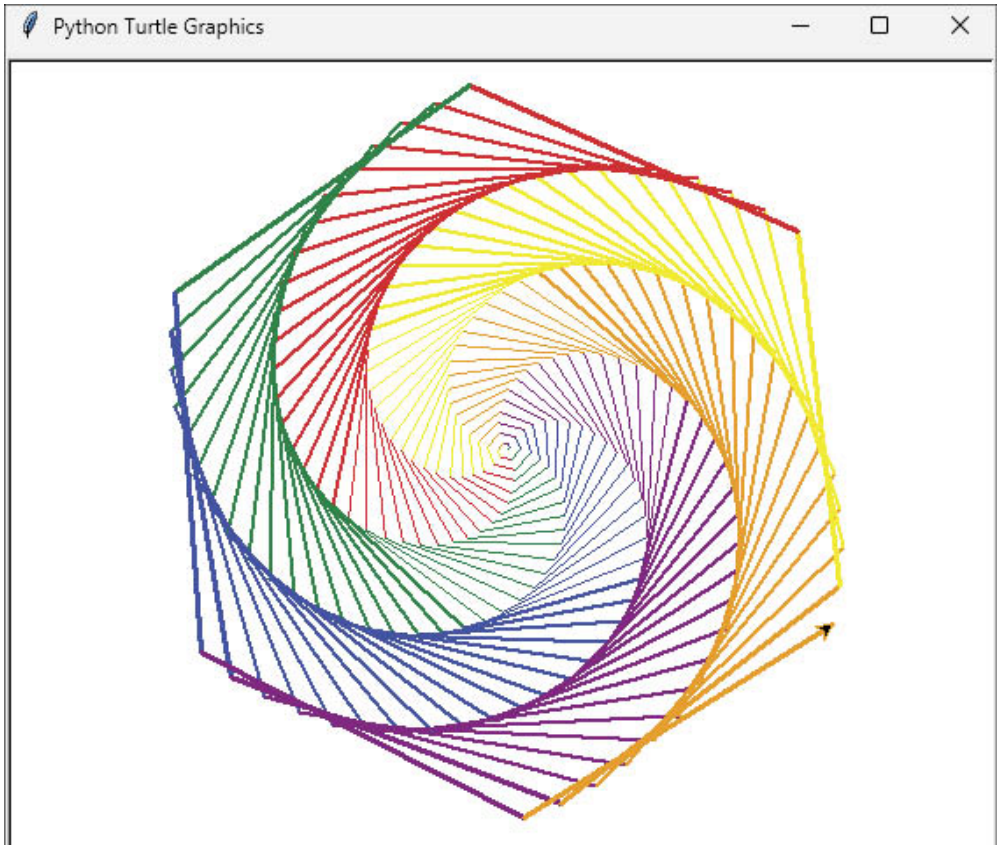


A little something more advanced. Try a colored spiral.

```
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)
```

Will look something like this



Lab Exercises

1. Create a new program and import the turtle graphics module.
2. Experiment with drawing different shapes using some of the turtle graphics methods.
3. Use the turtle commands to draw some shapes.

Building an Interface

Modern computer applications are built with graphic user interfaces in mind. The user interacts with the application using windows, icons, menus and a mouse pointer rather than console-based I/O.

To create a graphical user interface using Python you'll need use Tkinter (Tk interface). This module is bundled with standard distributions of Python for all platforms.

For this section, take a look at the video demos

`elluminetpress.com/pygraphics`

You'll also need the source files in the directory Chapter 11.

Creating a Window

The first thing you need to do is import the tkinter module into your program. To do this, use

```
from tkinter import *
```

To create a window use the `Tk()` method

```
window = Tk()
```

Add a title

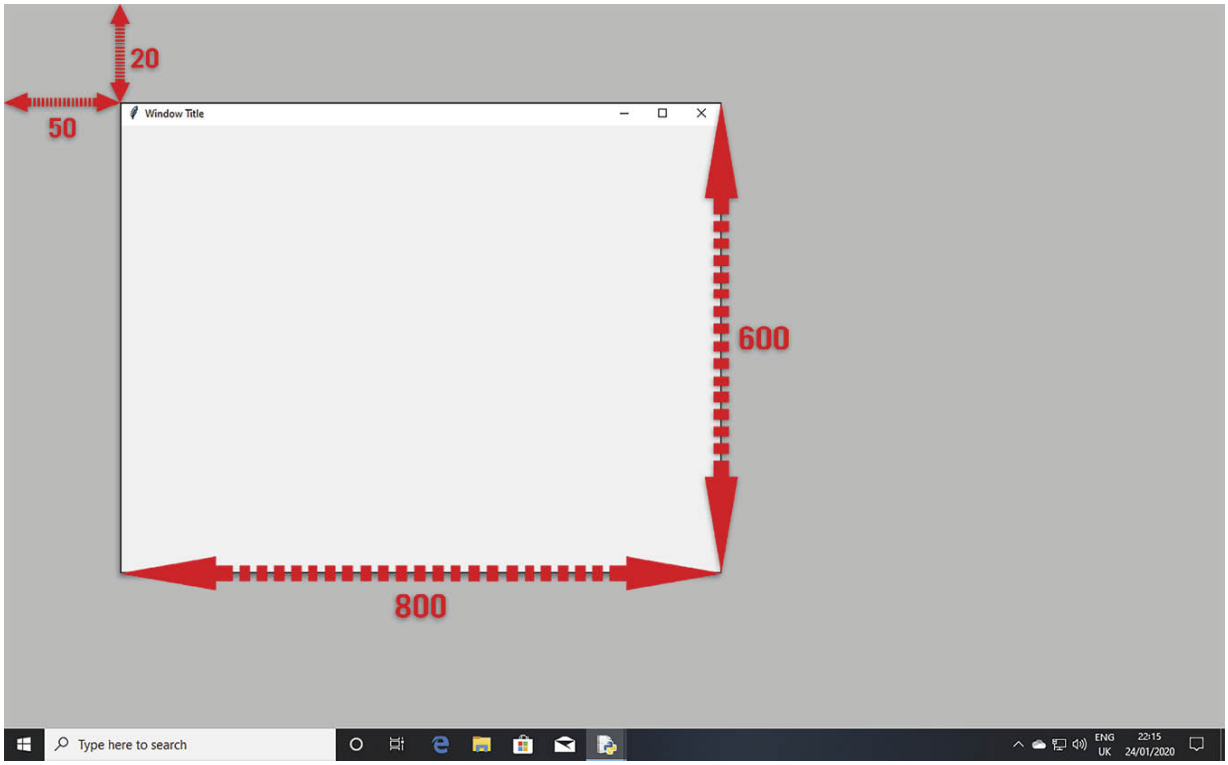
```
window.title('Window Title')
```

Set the initial size and position of the window. Use the .geometry() method.

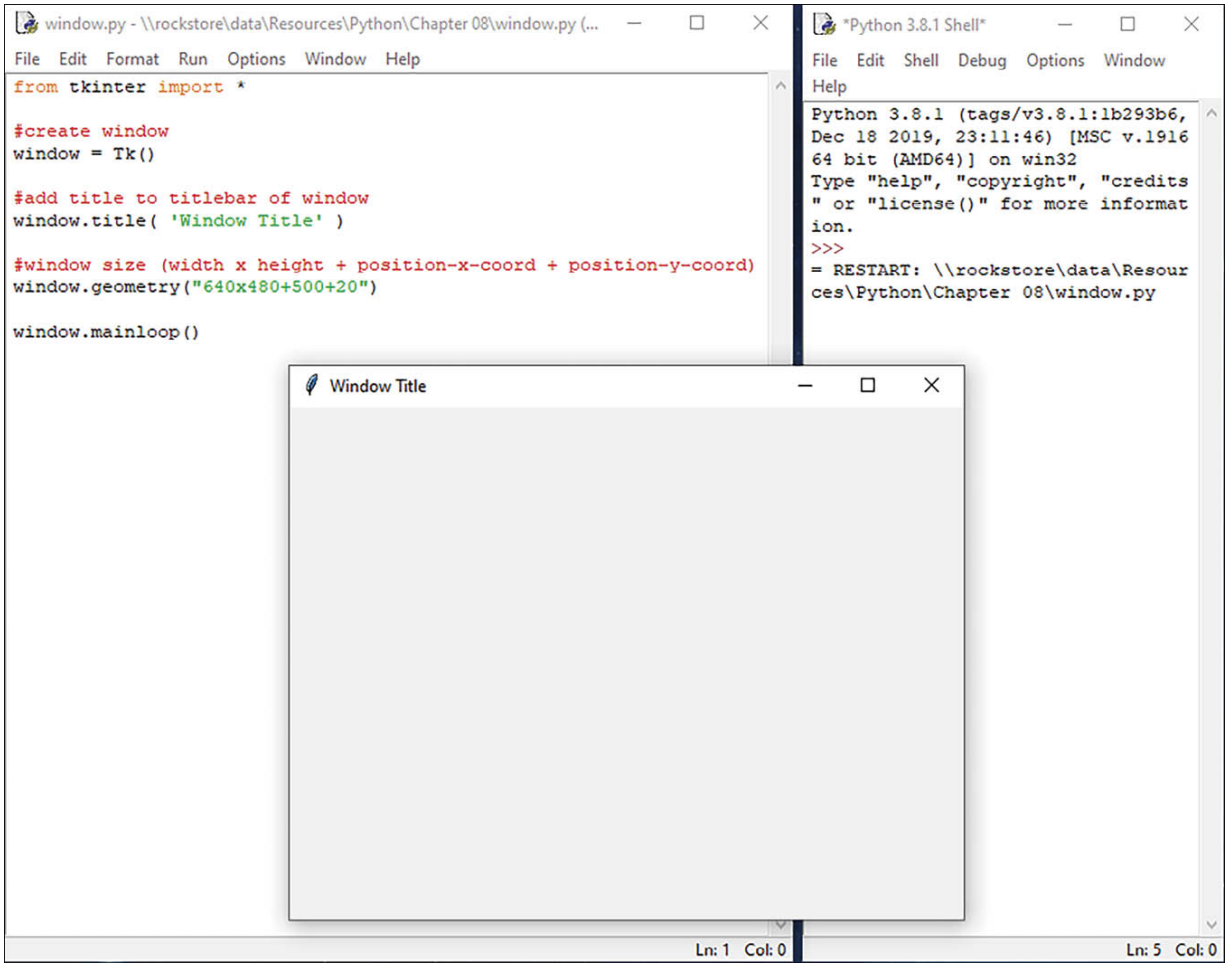
```
window.geometry("800x600+50+20")
```

The first two numbers in this case '800x600', sets the window size. Set this to the desired window size in pixels. This could be 1280x720, 1920x1080, and so on.

The second two numbers in this case '50+20', sets the initial position of the window on the screen using *x* and *y* co-ordinates


20
50
Window Title
600
800
Type here to search
ENG
UK
22:15
24/01/2020

Lets take a look at a program. Open window.py. Here, we've created a window. You can do this using the `Tk()` function and assign it to a window object.



We've sized the window so that is 640 pixels wide by 480 pixels high. We've also positioned the window 500 pixels across from the top left by 20 pixels down. You can do this using the `.geometry()` method. This is the initial size and position of the window on screen.

We've also added a window title. You can do this using the `.title()` method. This helps to identify your app.

Finally to make the window appear, we need to enter the Tkinter event loop. You can do this with the `.mainloop()` method.

`window.mainloop()`

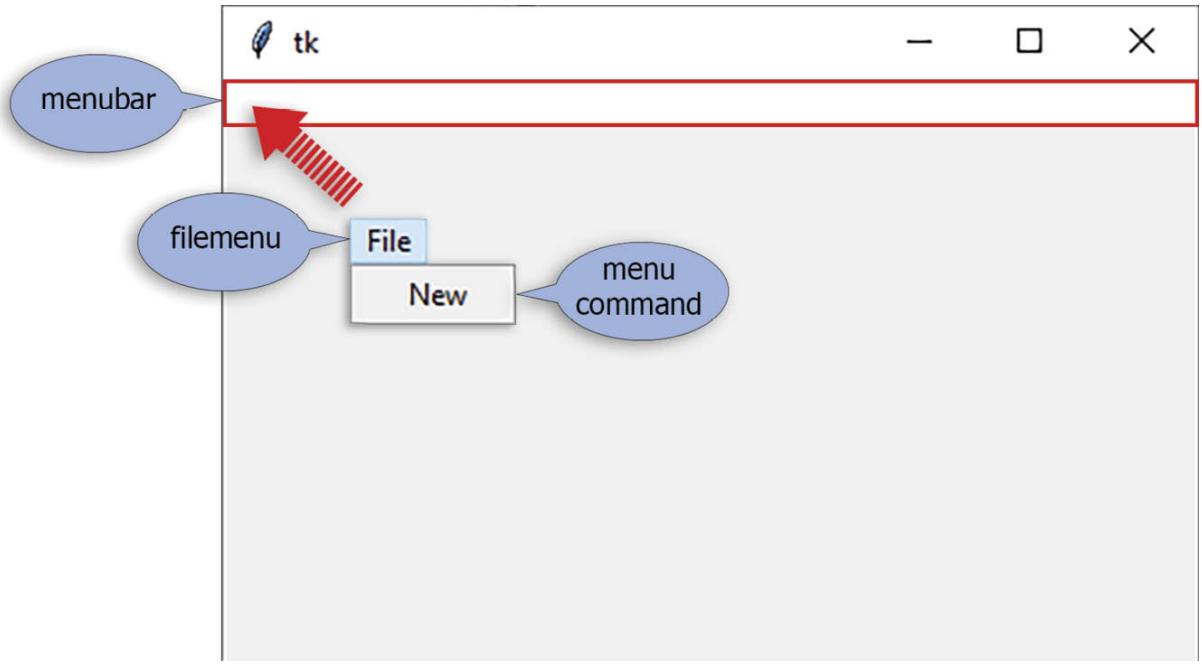
This is an infinite loop used to run the application and is called an event loop. The `.mainloop()` method waits for an event such as a key press, or mouse click events from the window system and dispatches them to the application widgets (frames, buttons, menus, etc).

Adding Widgets

Widgets are standard items for building a graphic user interface using Tkinter.

Menus

Let's add a menu. You can create a menu bar using the `Menu()` function. Assign it to an object (eg menubar) and attach it to your main window.



Now you need to create the individual menus (such as 'file', 'edit', etc) on the menubar.

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

Add the menus to the menubar

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

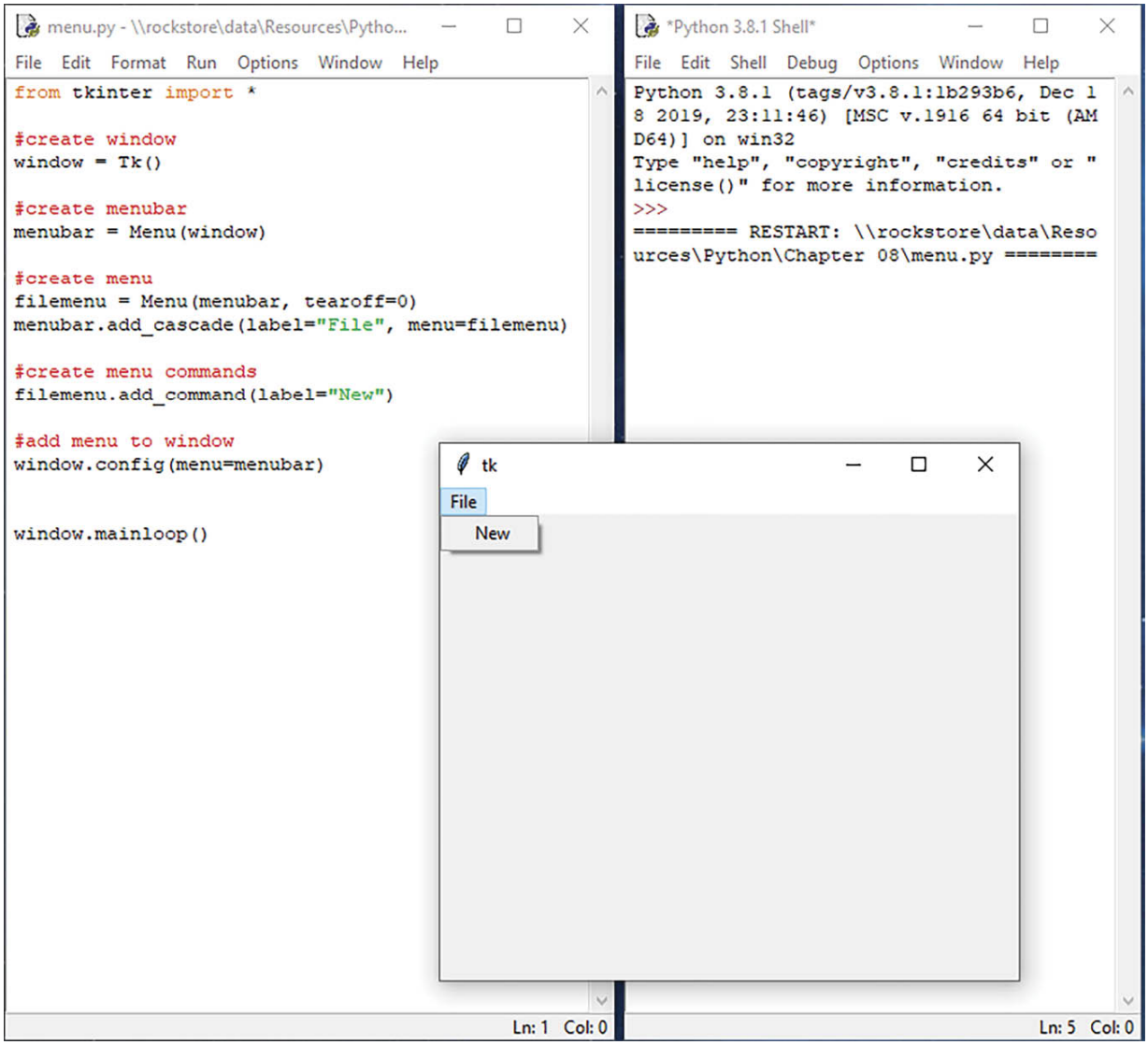
For each menu you create (eg filemenu), you need to create each menu command (such as 'new', 'save', 'exit' etc.)

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

Finally add the menu bar you've created to the main window.

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

Let's take a look at a program. Open the file menu.py. Here, we've written the program as described above.



The Canvas

The canvas is used to draw, create graphics, and layouts. This is where you place your graphics, text, buttons and other widgets to create your interface.

To create a canvas use:

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

Use `parent-window` to attach the canvas to your window. Use `height` and `width` to size your canvas.

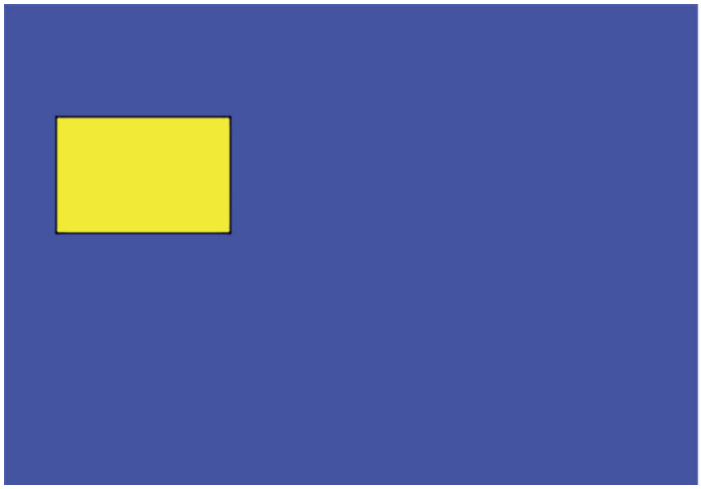
Use `bg` to set the background colour. Open `colorchart.py` and run the program. This will create the colour chart shown below.

Select the name of the color from the chart to use in the `bg` parameter.

Lets draw a shape on the canvas

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

The first two numbers are the x & y co-ordinates on the canvas. The second two numbers are the length and width.

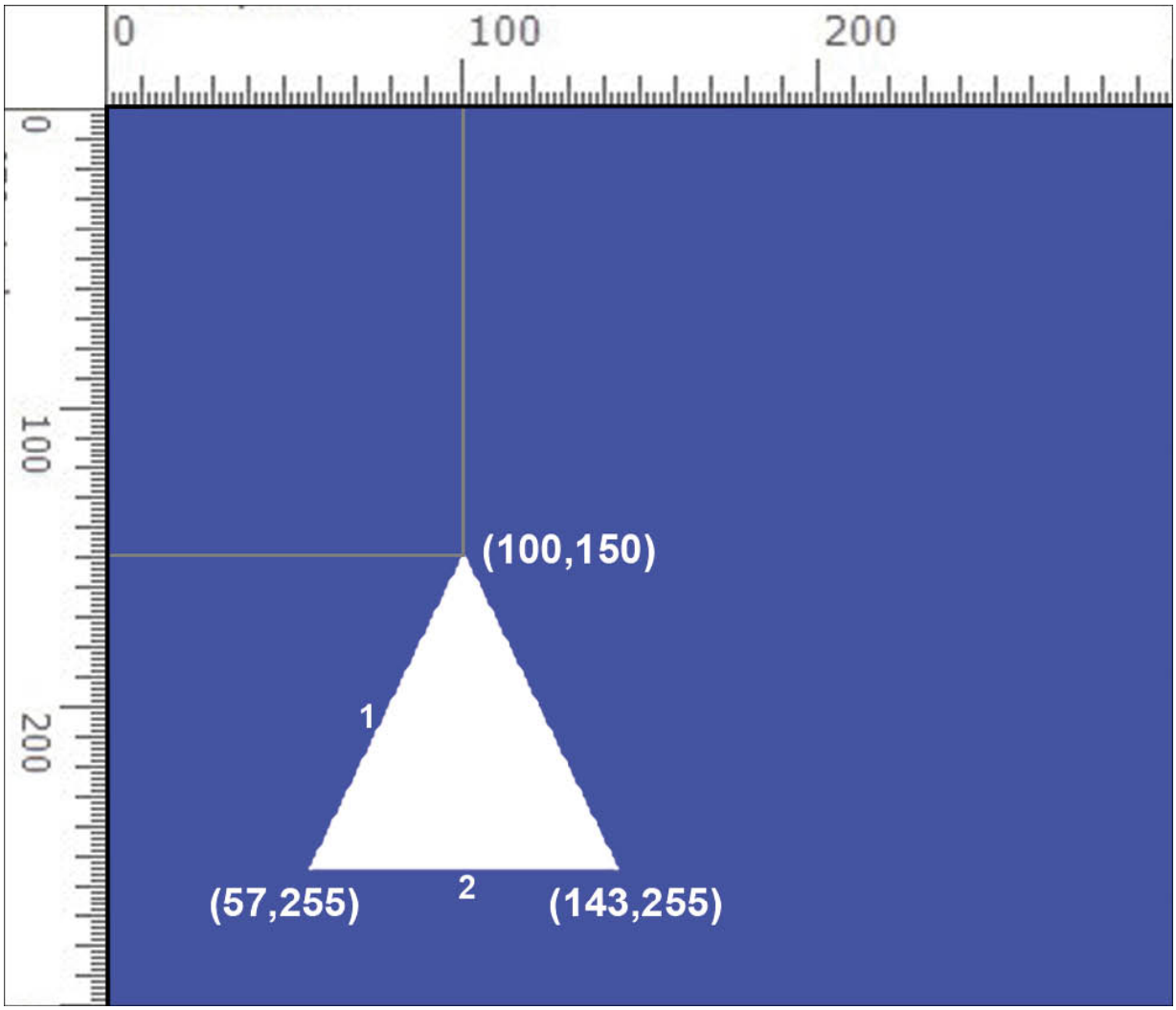


You can also draw a polygon

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

1 2

In this example, we're creating a triangle. A triangle has three sides, so we need to draw three lines. The first two numbers indicate the start point of the first line; the second two numbers indicate the end point of the first line, and so on. Let's take a look:

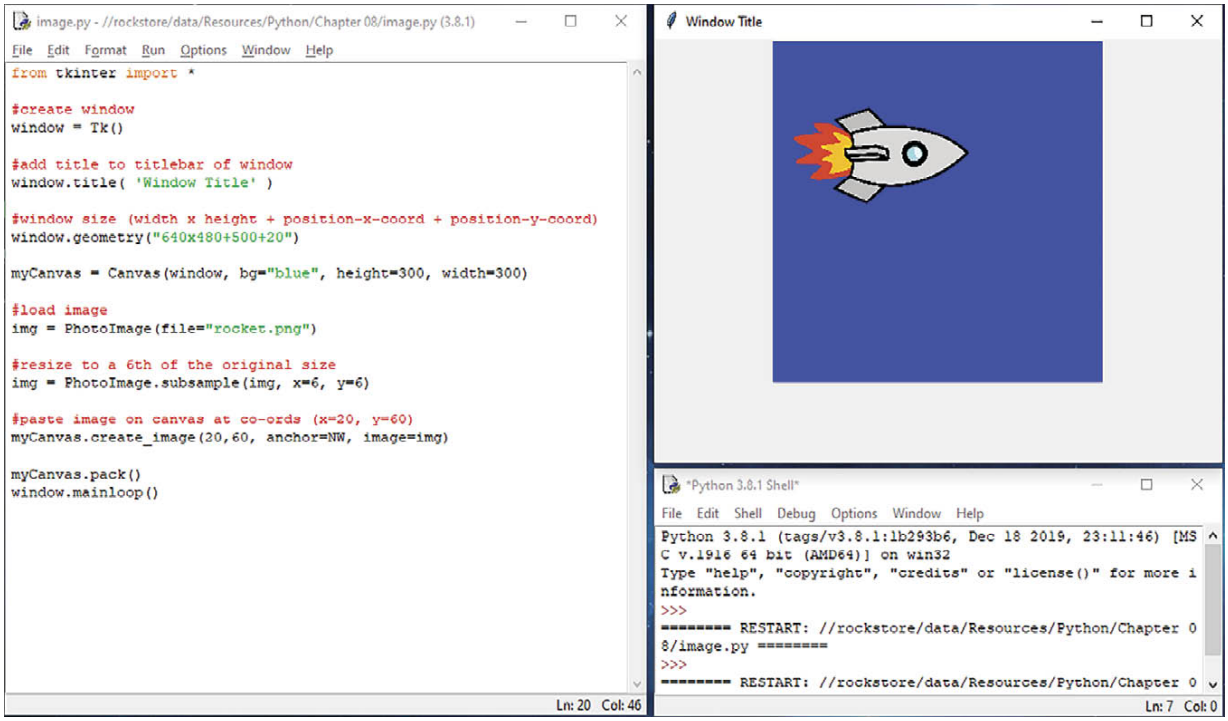


Try drawing a pentagon. A pentagon has five sides, so you need to draw five lines.

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

Images

You can add images to your canvas. Have a look at images.py. To load the image use the `PhotoImage()` function.



To paste the image on your canvas use the `.create_image()` method.

Buttons

You can add command buttons to your canvas. To do this use the `Button()` function.

Have a look at buttons.py.

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

Use `window` to specify the name of the window the button is to go on.

Use `command` to specify the function you want to call to handle what the button does. You can call existing functions or define your own functions to do this.

Use the `.pack` method to add the button to your window.

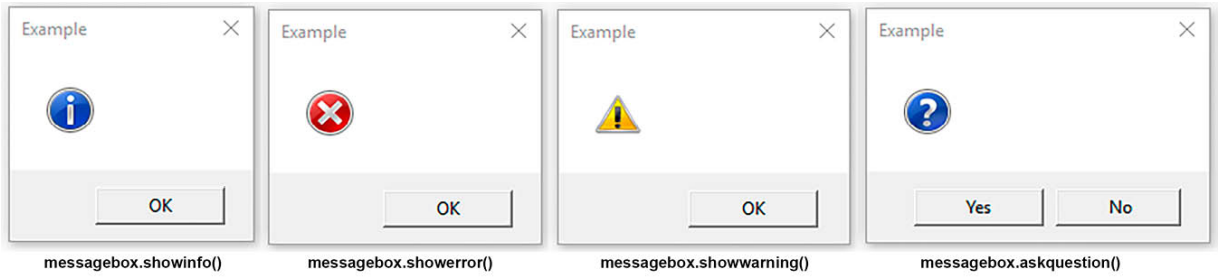
```
myButton.pack()
```

Message Boxes

You can add message boxes to your programs. To do this you will need to import the messagebox functions from the tkinter module. You can to this using the import command.

```
from tkinter import messagebox
```

You can create different types of message boxes. An info box, a warning box, an error box, and a box that asks for a yes/no response.



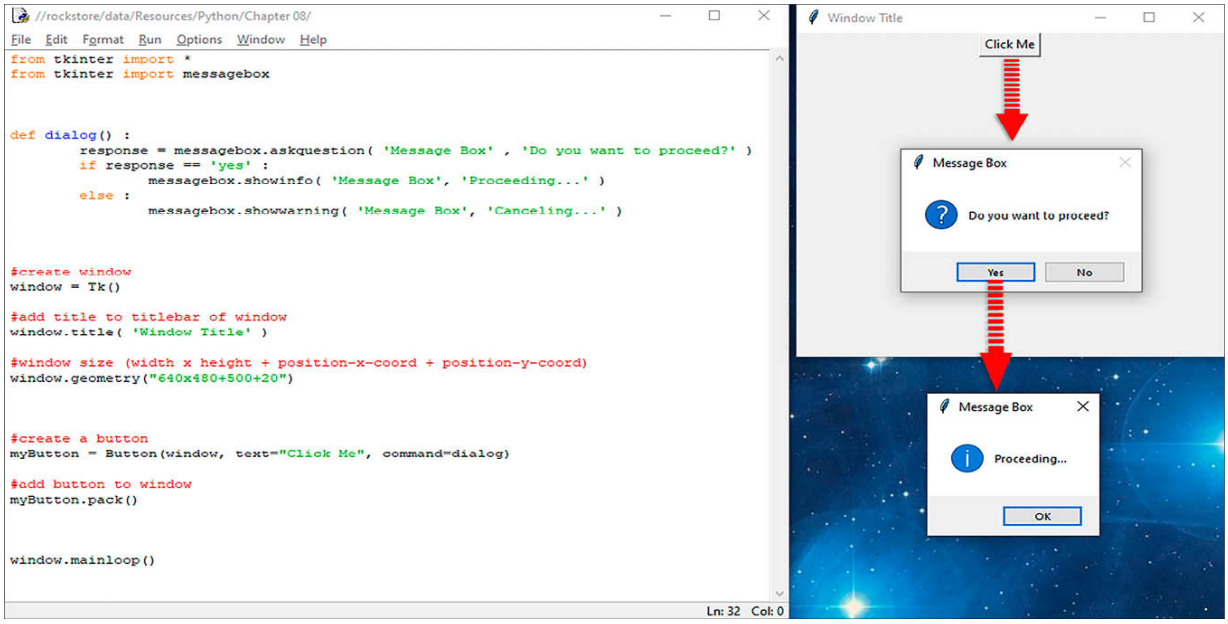
messagebox.showinfo() messagebox.showerror() messagebox.showwarning() messagebox.askquestion()

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

If you're asking the user for a yes/no response, you'll need to process this.

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

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

else :

executed if user clicks 'no'
```

Lets have a look at messagebox.py



Text Field

Use the `Entry()` function to create your text field.

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

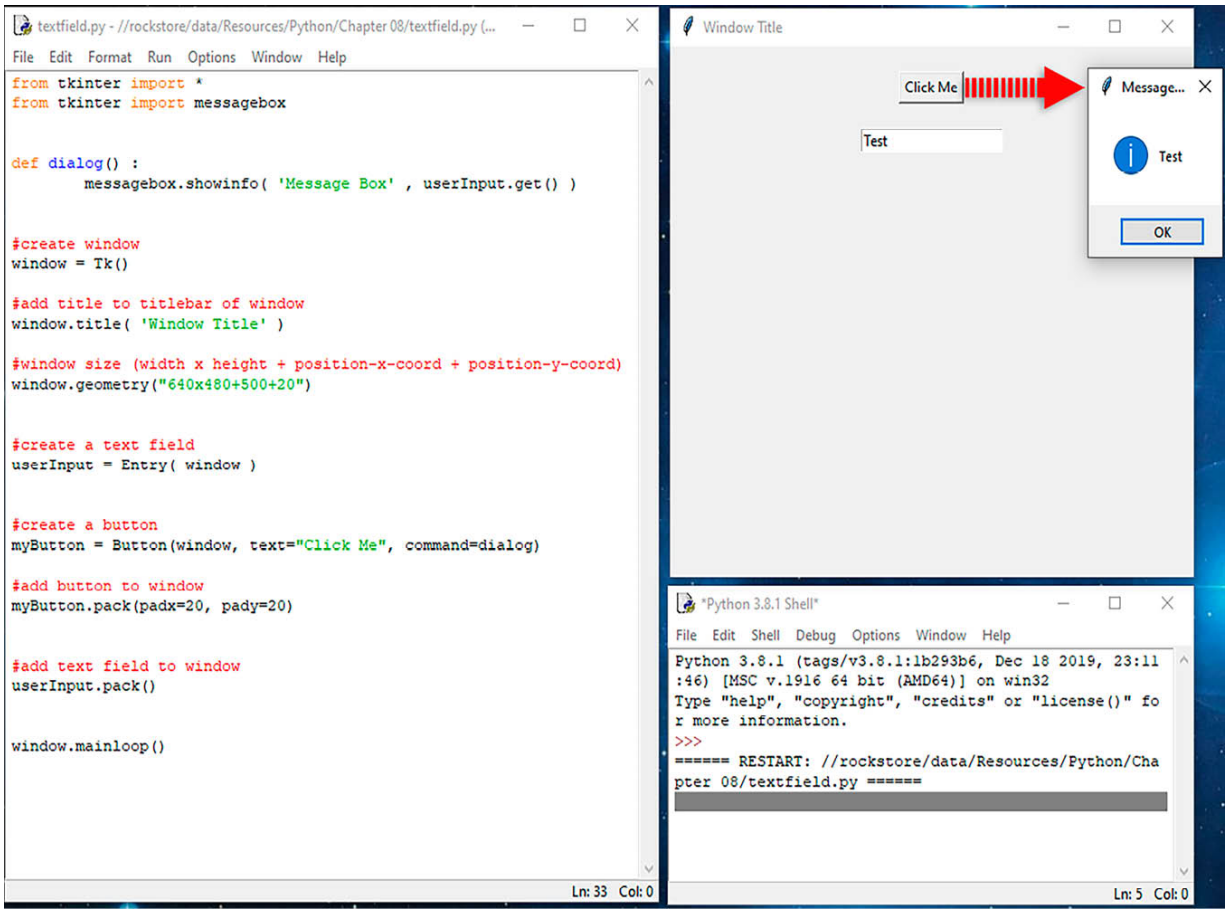
Use the `.pack()` method to add the field to your window.

```
userInput.pack()
```

To get the data from the text field use the `.get()` method

```
userInput.get()
```

Lets add these to the program. Have a look at textfield.py. Here, we've added a text field to the canvas under the command button.



We've also added code in the `dialog()` function to get the data from the text field and display it in a message box.

The `dialog()` function is called when the 'click me' button is pressed.

Run the program and see what it does.

Listbox

Use the `Listbox()` function to create your list box.

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

Use the `.insert()` method to add items to the listbox.

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

Use the `.pack()` method to add the listbox to your window. Use the padx and pady parameters to add some padding to space out

your listbox in the window.

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

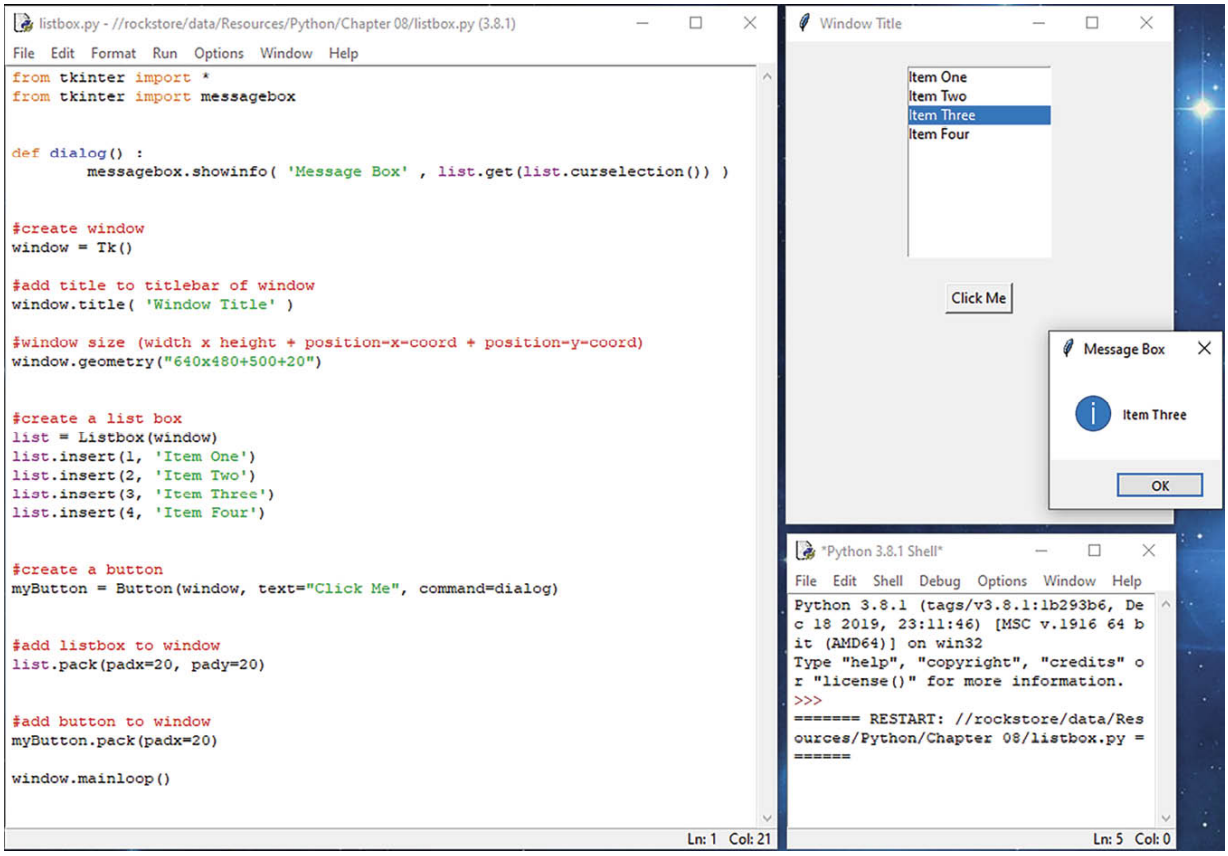
Use the `.curselection()` method to get the index of the item selected by the user. Remember, the first item's index is 0.

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

Use the `.get()` method to return the item

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

Lets take a look at a program. Open listbox.py.



Checkbox

Use the `Checkbutton()` function to create each of your checkboxes.

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

You'll need to create a variable for each checkbox to assign its 'onvalue' if the user clicks the checkbox.

```
box1Checked = IntVar()
```

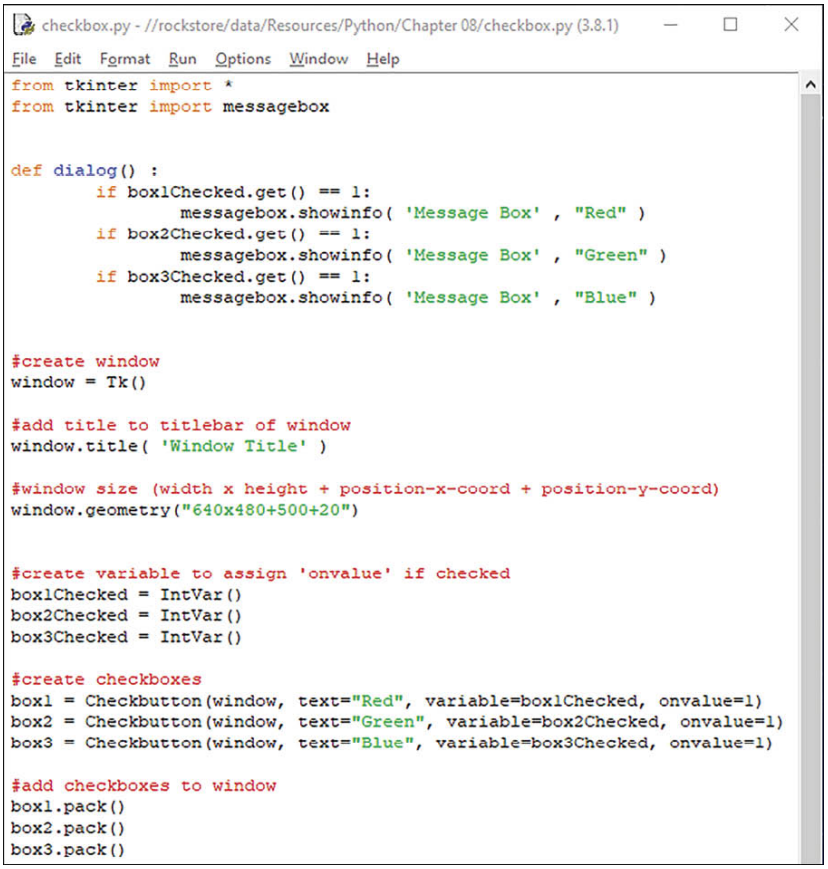
The variables you created above will either be 1 or 0. Onvalue is set to 1, so the variable will be set to 1 when the user clicks the checkbox. Use the `.get()` method to get the value.

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

Use the `.pack()` method to add each of your checkboxes to your window.

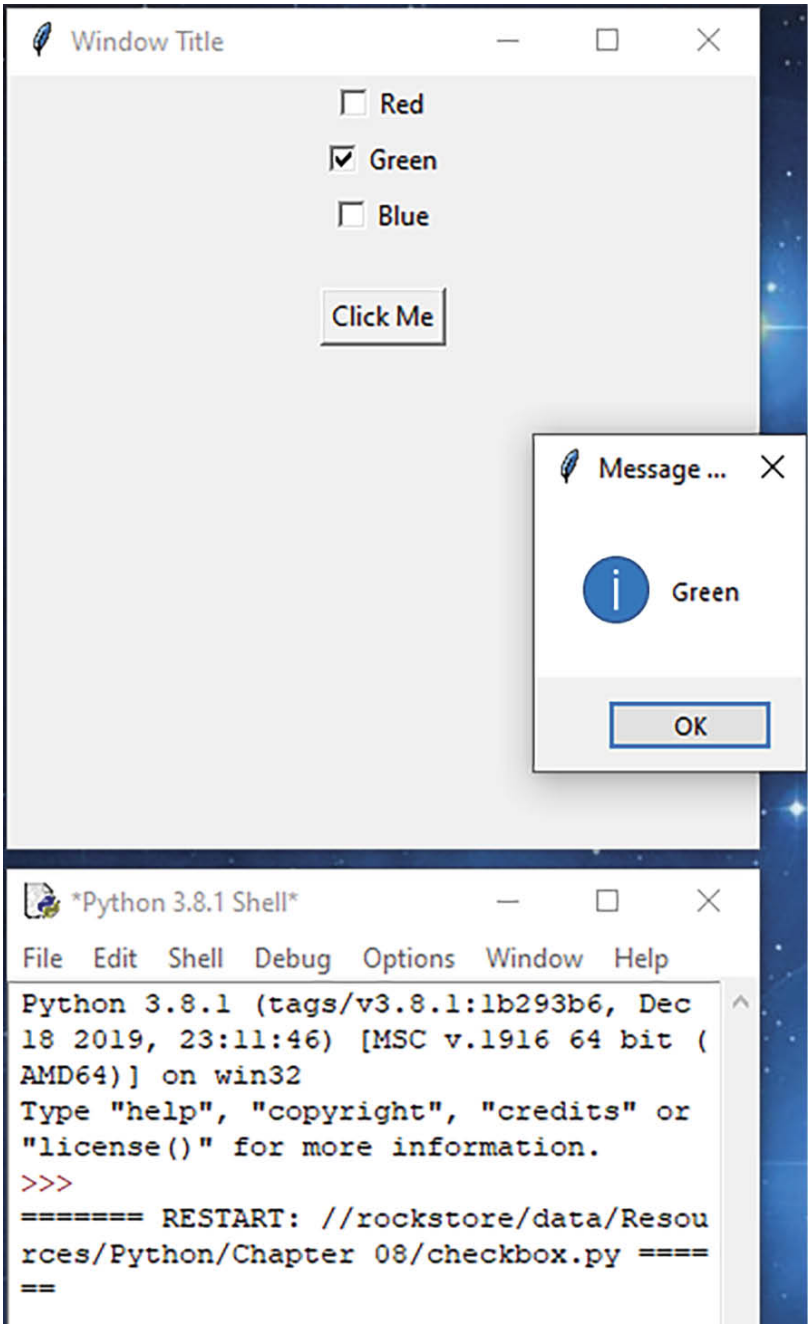
```
box1.pack()
```

Lets take a look at a program. Open checkbox.py.



When you run the program you can select any of the checkboxes. When you click the button, the function reads which checkbox is

selected and returns the value.



Labels

You can create labels to label text fields and other items on your interface. To do this, use Label()

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

Use pack() to add the label to your window.

```
textLabel.pack()
```

Label Frame

The LabelFrame is used to group related widgets, such as checkboxes, radiobuttons or text fields.

First you need to create your label frame group. You can do this with `LabelFrame()` as follows

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

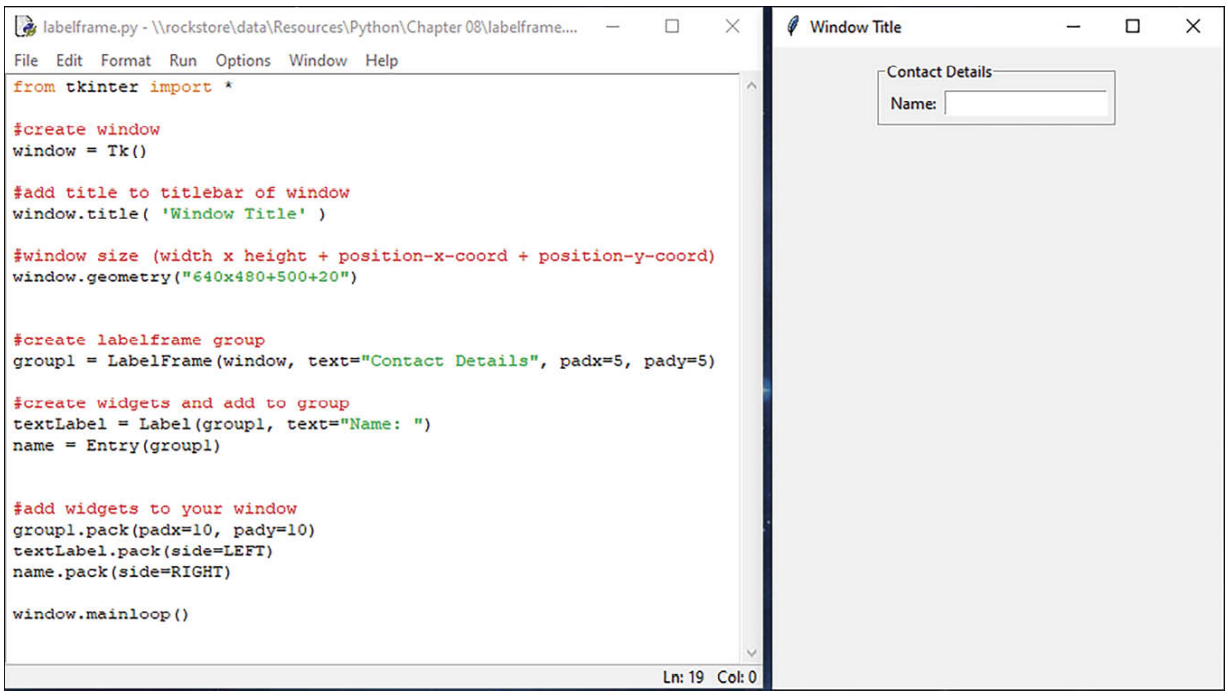
Use the first parameter `window` to attach the group to your main window. Next you need to add your widgets to your group. You can do this in the usual way, except you need to specify in the widget functions, which widget to attach to. So to add our text label, specify the widget to attach to using the first parameter (our labelframe defined above is called group1, so use group1 underlined below).

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

Add your widgets to your window in the usual way.

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

Lets take a look at a program. Open labelframe.py.



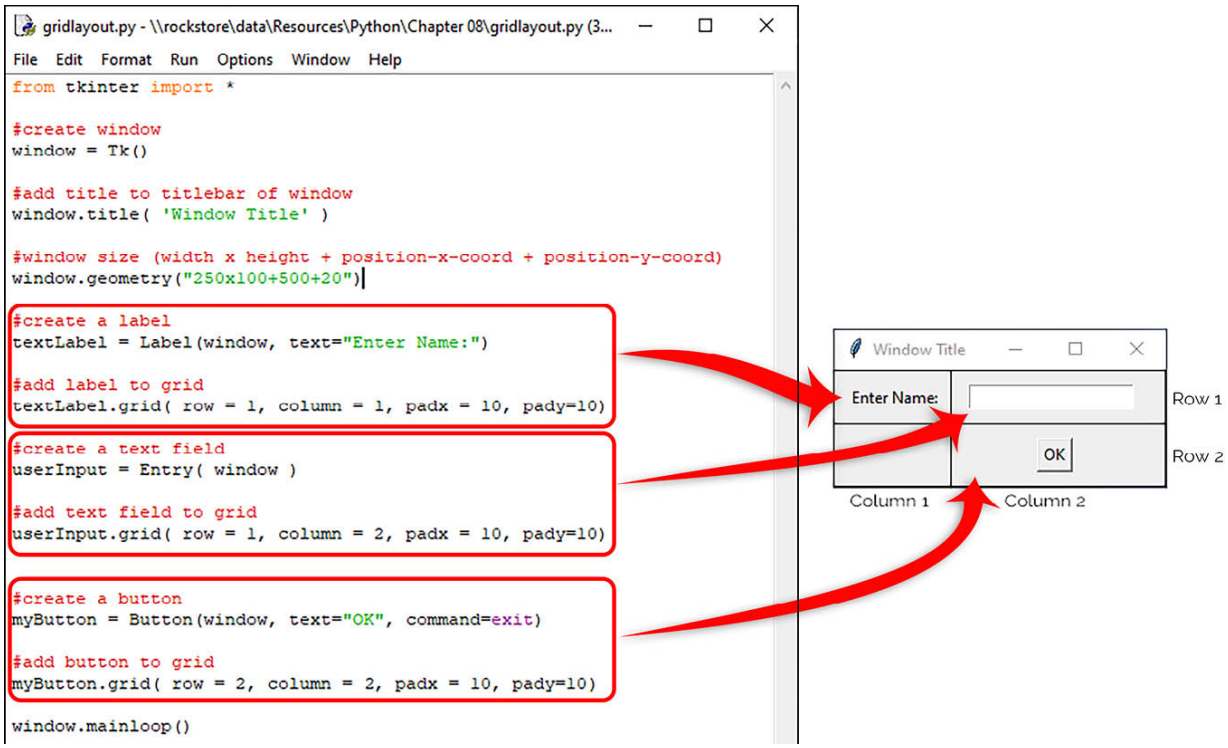
Here, we've created a text label and a textfield inside the labelframe group (group1).

Interface Design

Now that we know how to create a window, menus, and add different types of widgets, we'll take a look at how to lay them out in the window to create a usable interface.

You can do this using the grid layout manager. Lets take a look at an example. Open the file gridlayout.py.

Use the `.grid()` method to placing the widgets in the window according to the grid layout. Use `row` and `column` parameters to specify which cell in the grid to place the widget. Use the `padx` and `pady` parameters to add some spacing around your widgets in the grid.

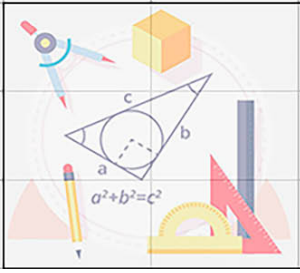


Here, we've placed a text label in row 1, column 1. There is an text field in row 1, column 2. We've placed a command button in row 2, column 2.

When you run the program, you'll see the result as shown below.

Enter Name:	
	OK

Lets design a simple interface for a unit converter app. To design this interface, we'll divide the window up into 3 rows and 5 columns.

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

Now, we'll place a logo on the left hand side and span it across 2 columns and down 3 rows.

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

imgLbl = Label(window, image=img)

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

We'll also place a label in row 1 column 3.

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

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

A drop down box in row 1, column 4,

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

A text field in row 2, column 4, with a button in row 2, column 5.

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

A label at the bottom in row 3, column 4, to show the result.

```
textLabel.grid( row = 3, column = 4,
```

```
padx = 10, pady=5)
```

Add a command button to row 2 column 6

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

Lets take a look at the program. Open the file converter.py. Here, we're adding the widgets to the grid using the `.grid()` method.

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

File Edit Format Run Options Window Help

```
#load image and add to imgLbl label
img = PhotoImage(file="logo.png")
imgLbl = Label(window, image=img)

#add image to grid
imgLbl.grid( row = 1, column = 1, padx = 10, pady=10, columnspan=2, rowspan=3)

#create a label
textLabel = Label(window, text="Convert:")

#add label to grid
textLabel.grid( row = 1, column = 3, padx = 10, pady=10)

#create drop down menu
conversions = Combobox(window, values=[
                                       "Cm to Inch",
                                       "Inch to Cm",
                                       "Km to Miles",
                                       "Miles to Km"])

#set combobox default selection
conversions.current(0)

#add combo box to grid
conversions.grid( row = 1, column = 4, padx = 10, pady=10)

#create a text field
userInput = Entry( window )

#add text field to grid
userInput.grid( row = 2, column = 4, padx = 10, pady=10)

#add label for result
textLabel = Label(window, text="")

#add result label to grid
textLabel.grid( row = 3, column = 4, padx = 10, pady=5)

#create a button
myButton = Button(window, text="OK", command=convert)

#add button to grid
myButton.grid( row = 2, column = 6, padx = 10, pady=10)

window.mainloop()
```

Ln: 14 Col: 29

We've used the padx and pady parameters to space out the widgets in the grid layout.

That's the interface sorted. As it stands, the program won't do anything if you click the button, or enter a number into the text field.

We need to write a function to take care of this and call it when the button is clicked.

Declare the function in the usual way. We'll call this one `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)
```

You'll need to read the selection from the combo box. You can do this with the `.current()` method. The first item in the combo box has an index of 0, the second is 1, and so on. Use an if statement to separate the calculations for each selection in the combo box.

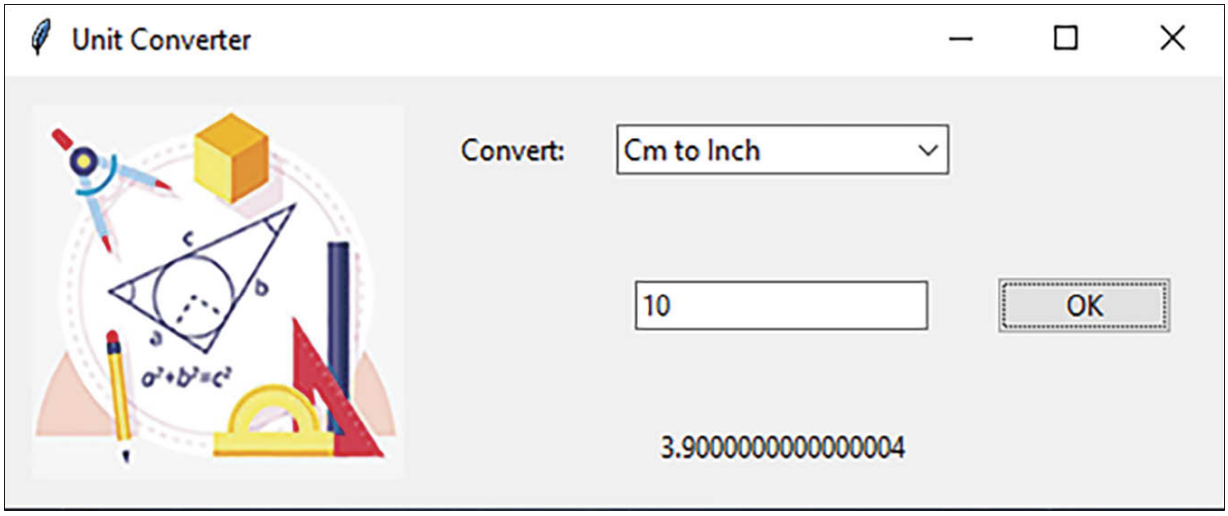
```
if conversions.current() == 0:
```

Next you'll need to get the data from the text field. You can do this with a `.get()` method on the text field. Remember to cast the data type to a float, as the data from a text field is a string.

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

Perform the calculation and return the result to the blank text label in row 3, column 4 of the grid.

Now, when you run the program, you'll get a nicely laid out interface


Unit Converter
Convert:
Cm to Inch
10
OK
3.9000000000000004

Developing a Game

To start creating your own games using Python you'll need use the PyGame module. This module isn't bundled with standard distributions of Python, so you'll need to install it before you start.

For this section, take a look at the video demos

`elluminetpress.com/pygraphics`

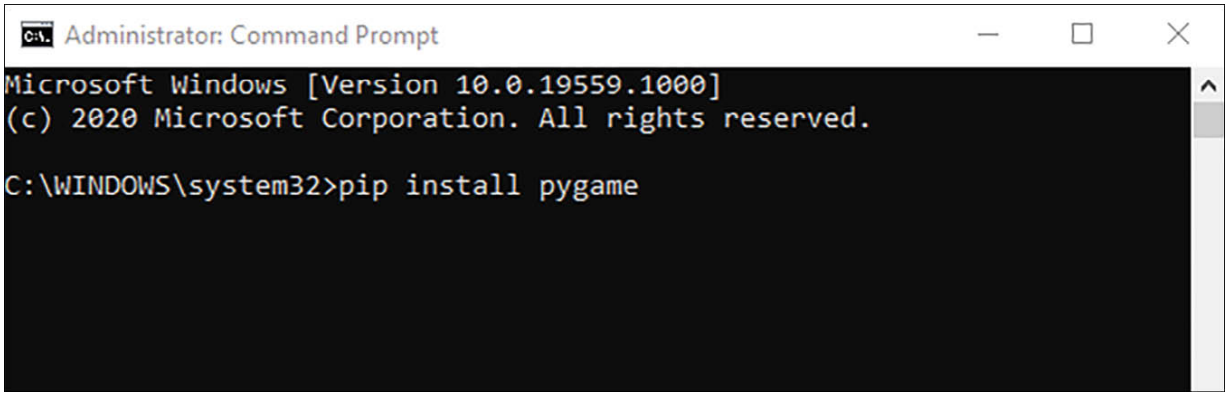
You'll also need the source files in the directory Chapter 12.

Installing PyGame

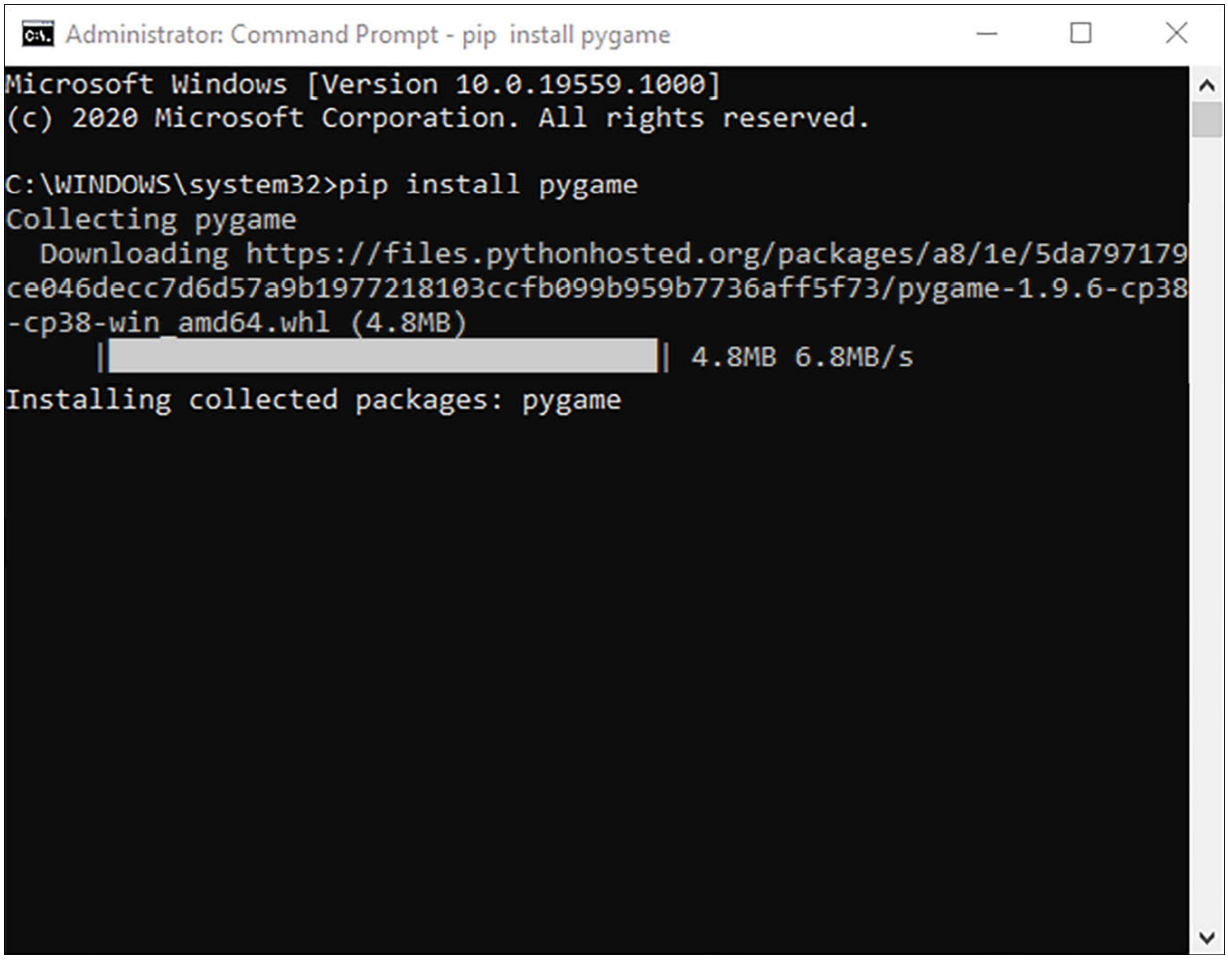
To install the module, open a command prompt. Make sure you run this as an administrator. On the command prompt type

```
pip install pygame
```

Once you press enter, the install will begin.



Allow the pip utility to download and install the module.



Once the process is complete, you can start using pygame.

Opening a Window

The first thing you'll need to do is import the pygame module.

```
import pygame
```

Initialist pygame using the `.init()` method

```
pygame.init()
```

Open a window. This sets the window size 640 pixels wide by 480 pixels high. This could also be 800x600, 1920x1080 and so on.

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

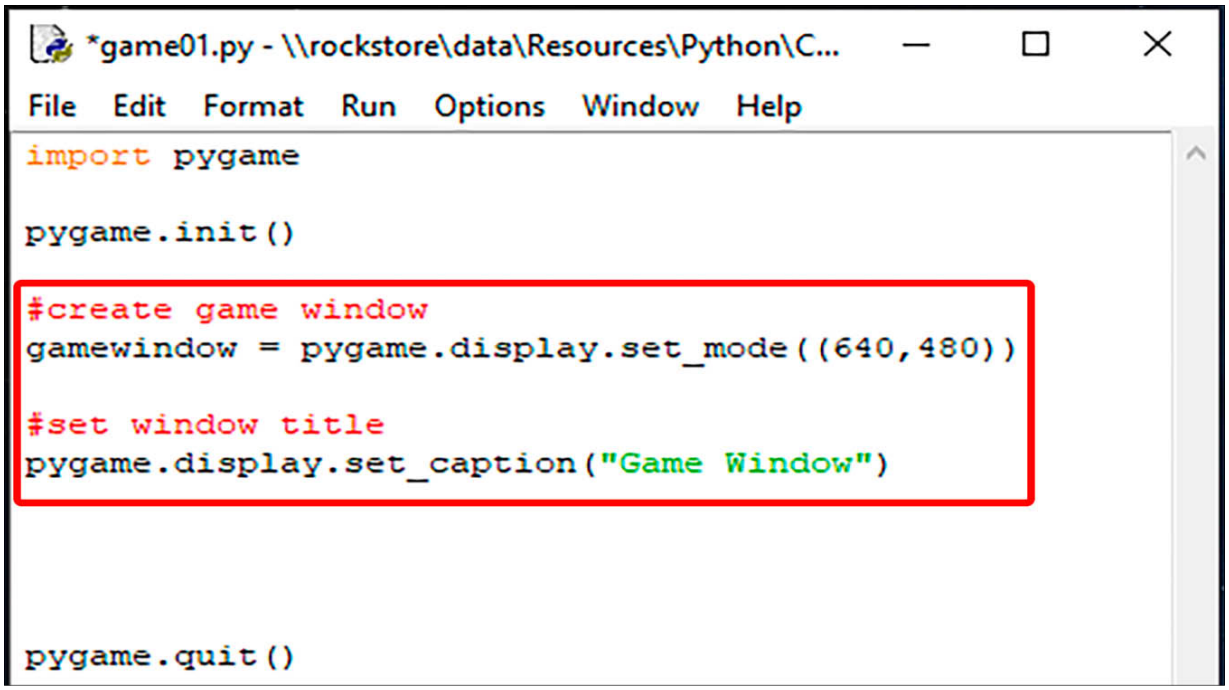
Set the window's title. This is the title that appears in the title bar of the window.

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

You should always end your pygame scripts with the `.quit()` method.

```
pygame.quit()
```

Lets take a look at the program so far.



If you run this program, the game window will initialize, open, then

immediately close. This is fine since there is no other code to execute.

Adding an Image

Lets add an image to the game window. This is going to be our character in the game. In this case we're going to use a space rocket. We can add the image load statement to our program.

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

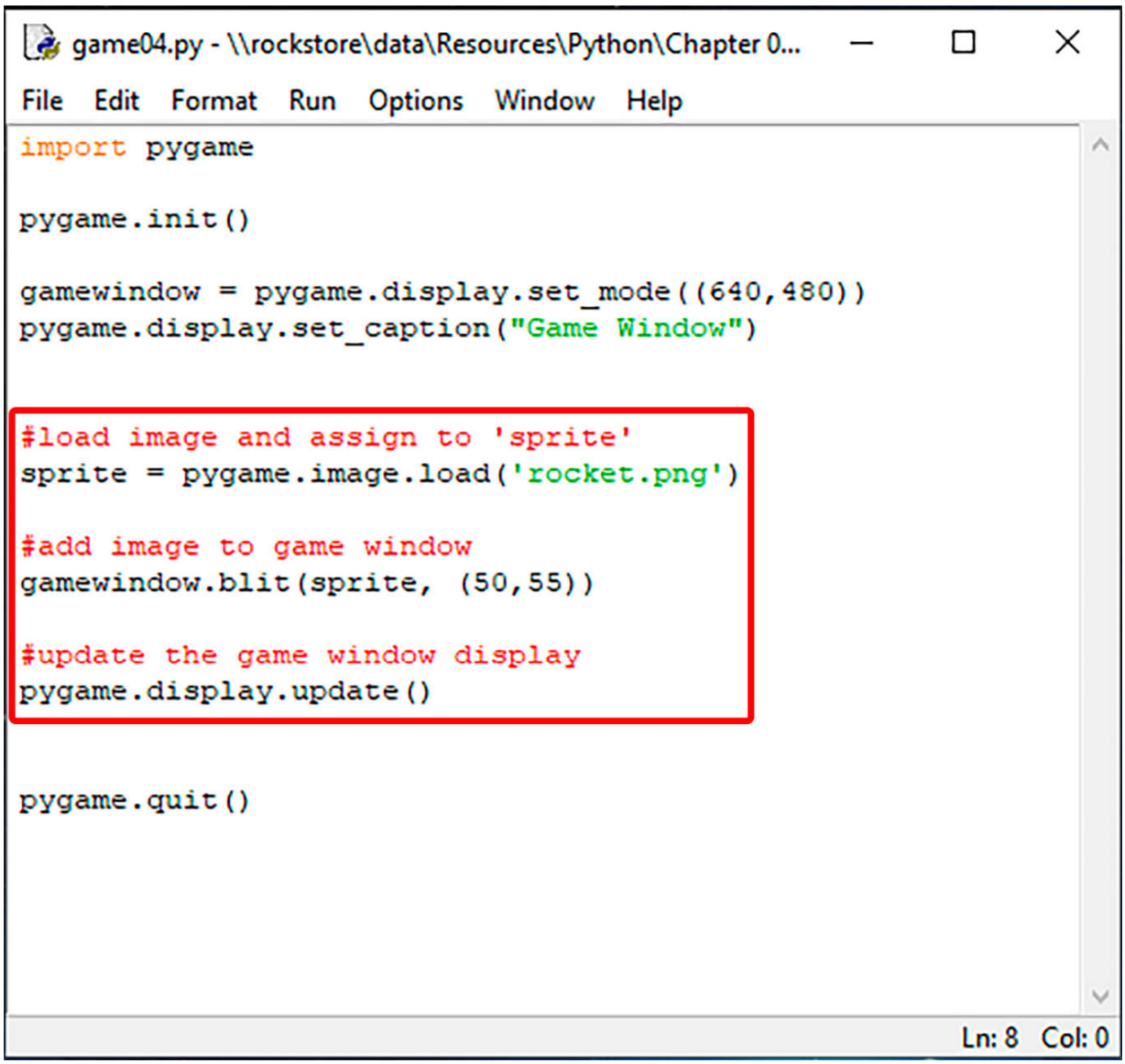
Paste the image (sprite) onto the game window using the `.blit()` method, and assign the initial position on the screen (x, y)

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

Update the display to show the image

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

Lets take a look at the program.


game04.py - \\rockstore\data\Resources\Python\Chapter 0...
File Edit Format Run Options Window Help
import pygame
pygame.init()
gamewindow = pygame.display.set_mode((640,480))
pygame.display.set_caption("Game Window")
#load image and assign to 'sprite'
sprite = pygame.image.load('rocket.png')
#add image to game window
gamewindow.blit(sprite, (50,55))
#update the game window display
pygame.display.update()
pygame.quit()
Ln: 8 Col: 0

The Game Loop

Now, lets get our rocket ship to actually do something. To do this we need to create a game loop to draw our sprites, update the screen and keep the program running.

We can take the following two statements and add them to our game loop. For this bit we'll use a while loop.

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

Update the display to show the image

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

Put these inside the while loop.

```
while running == 1:

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

pygame.display.update()
```

We'll also need to initialize some variables: x & y - the initial position on the screen, and running - to indicate whether the program is running or not.

```
running = 1

x = 250

y = 280
```

Lets take a look at the program.

```
#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()
```

The Event Loop

In this simple game, we want the rocket ship to move left and right when the user presses the left and right arrow keys on the keyboard. To do this we need something to handle these events.

To do this, we can put a for loop inside our while loop (the game loop). We're monitoring for each event that occurs. You can use the `.get()` method to read each event.

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

Now inside the for loop, we need some selection depending on which key is pressed. We can use an `if` statement for this.

First we need to check whether a key has been pressed.

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

Inside this if statement we need to identify which key has been pressed. You can use another if statement.

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

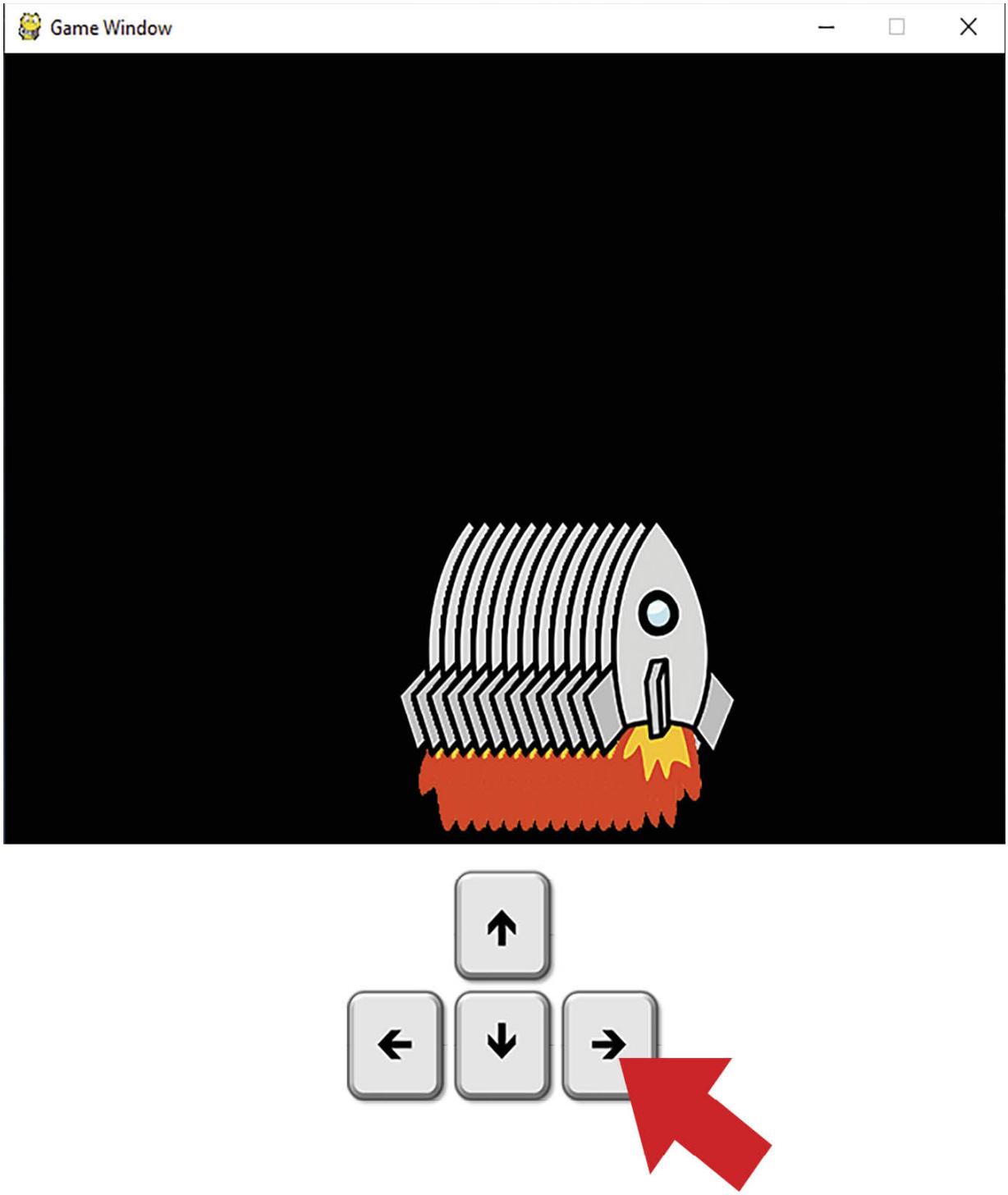
We do the same for all other keys we're going to use. Just add `elif` statements to the if statement.

Lets take a look

```
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

Now, when we run the program, you'll see your rocket move left when you press the left key, and right when you press the right key.



You'll also notice something else. The image repeats on the screen. To fix this, you need to clear the screen (refresh) each time you move the object. You can use:

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

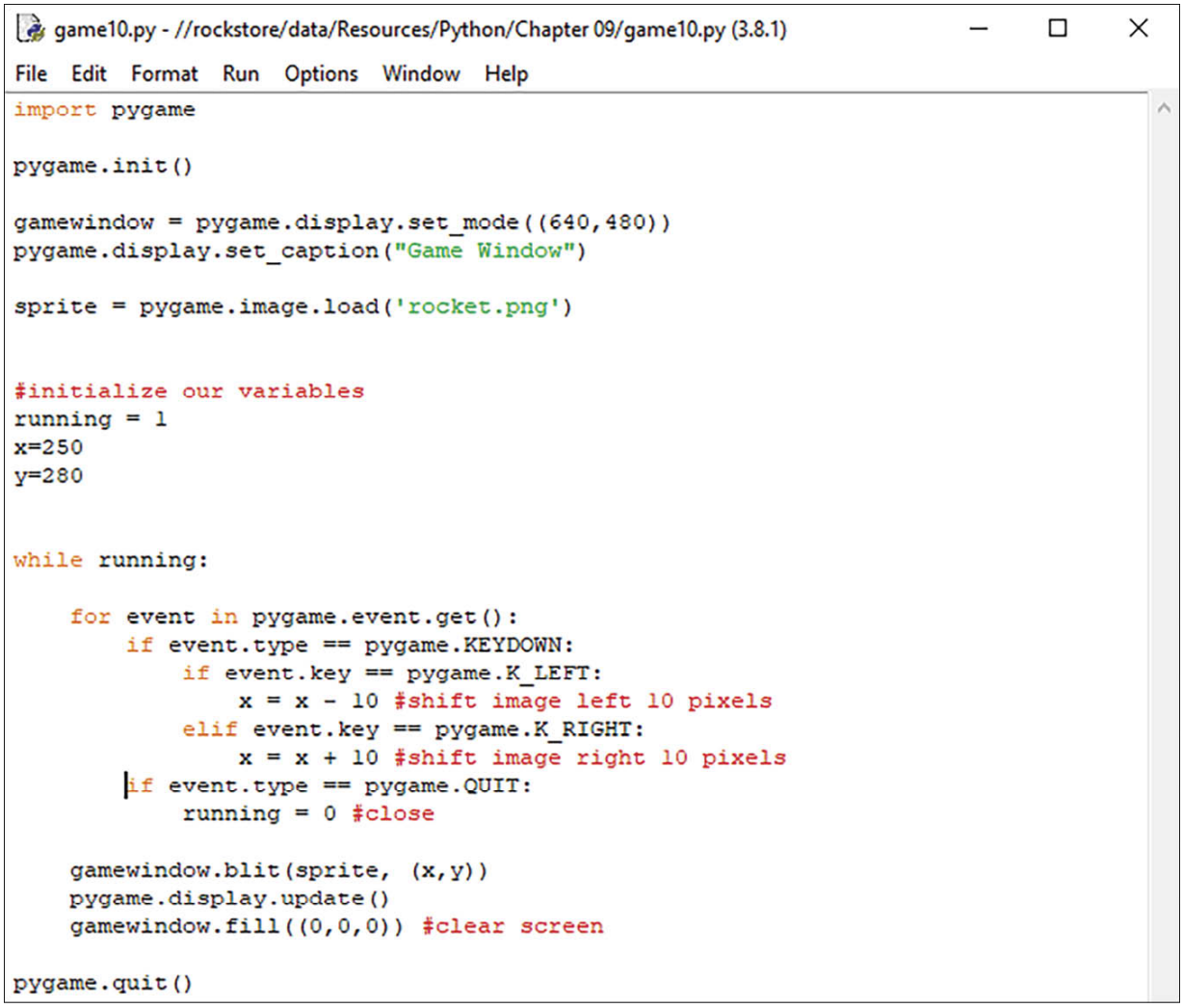
This returns the screen to black at the end of each iteration of the game loop (while loop).

It is also good practice to include a quit event in your event loop, so the program terminates gracefully.

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

This will set our running variable to 0 meaning the game loop will terminate and the program will close. This event will happen when you click the close icon on the top right of the window.

Lets take a look at the program so far.



You'll also notice that the rocket doesn't move when you hold they

key down. This is because the key-repeat feature is turned off. To turn this on, add the following line before your game loop.

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

The first parameter is the delay before the key event is repeated. The second parameter is the interval between repeats.

Shapes

You can add shapes such as circles, ellipses, rectangles and other polygons.

To draw a rectangle, use the `.rect()` method. Specify the surface or window you want to draw on, the colour, then specify the x & y position, followed by the width and length of the rectangle.

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

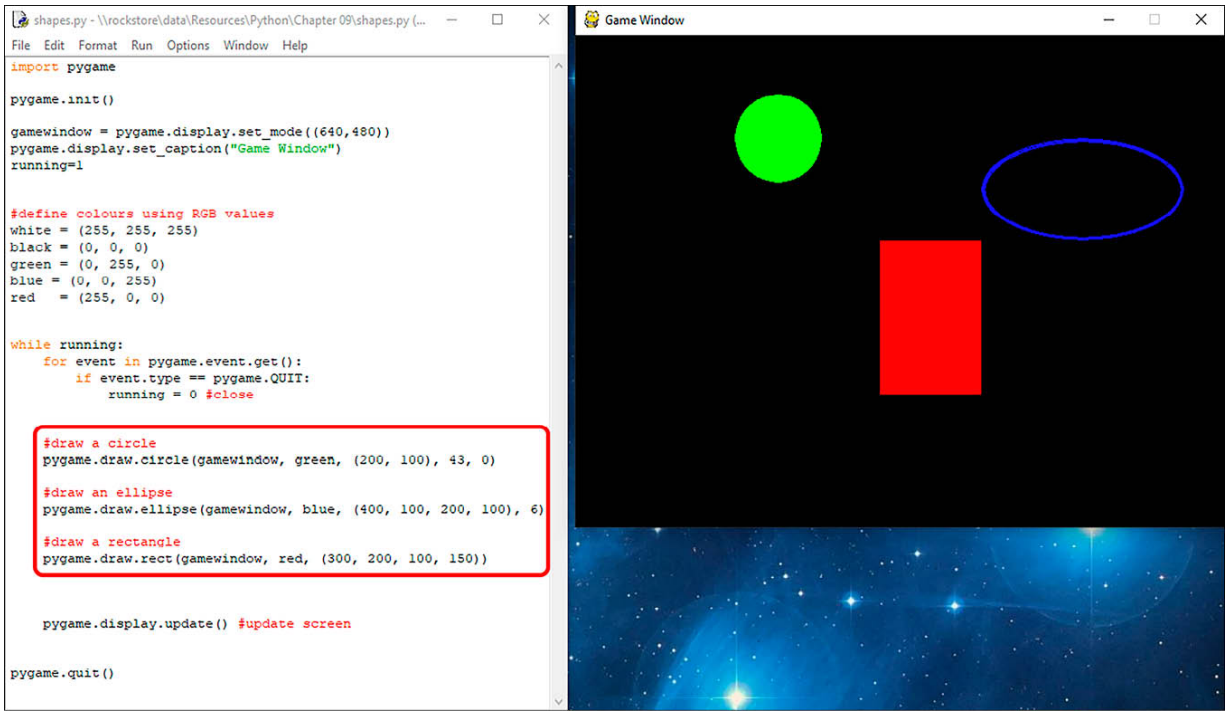
To draw an ellipse use the `.ellipse()` method. When drawing an ellipse, you're actually drawing it inside an invisible rectangle. This is why you specify width and length when drawing your ellipse.

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

To draw a circle use the `.circle()` method. Specify the surface or window you want to draw on, the colour, then specify the x & y position, followed by the radius of the circle.

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

Have a look at 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

Basic Animation

To demonstrate basic animation, we're going to move our ufo object on the screen.

First we need to load in our image. You can do this using the `.load()` method as we've done before.

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

Now, because an image is loaded onto a surface object by default, we can't move it or manipulate it. To get around this we assign the image to a rectangle. You can do this using the `.rect()` method.

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

We also need to define some speed and direction variables. We can do this with a list. This is a list containing the `[x, y]` co-ordinates on the screen *(speed[0] is x, speed[1] is y)*.

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

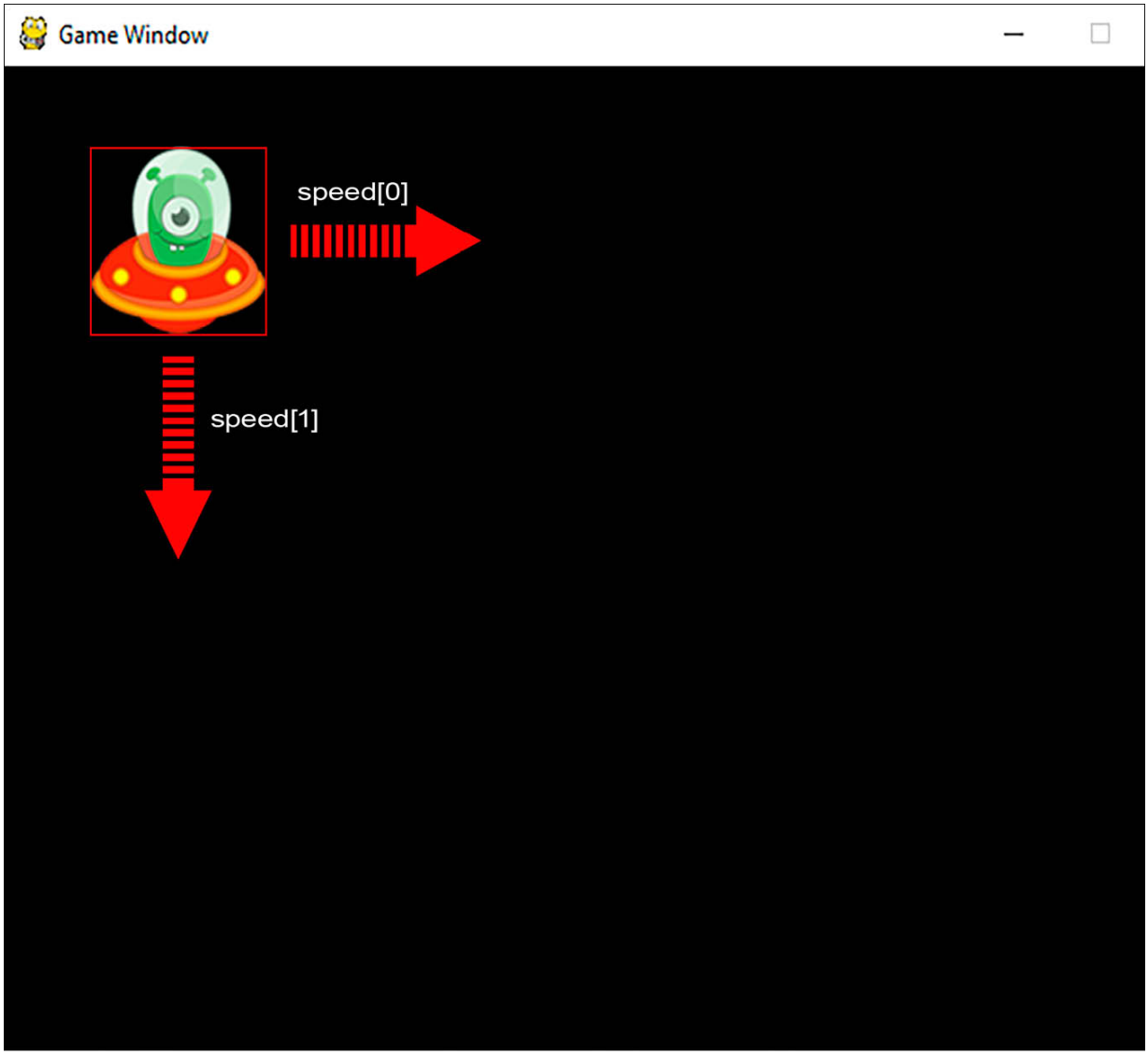
To move the object use the `.move_ip()` method

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

Lets take a look at the program. Have a look at 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)
```

Now, when you run this program, the ufo will fly from left to right across the screen.



Because we set `speed = [10, 0]`, this means we move our ufo 10 pixels along the x axis each time we execute `ufo_rect.move_ip(speed)` in the game loop.

If we set `speed = [0, 10]` this means we move our ufo 10 pixels along the y axis each time we execute `ufo_rect.move_ip(speed)` in the game loop.

Try changing the values in the program anim02.py and see what happens.

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

Try larger values.

Lets take our program a step further. Let's make the ufo bounce around the screen.

To do this we need to check whether the left edge, right edge, top edge and bottom edge of the ufo_rect goes beyond the edges of the screen. We can use if statements for this.

If the left edge of the ufo goes off left edge x reverse direction

To reverse the direction all you need to do is change the speed[0] to a negative number.

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

If the right edge of the ufo goes off right edge reverse x direction

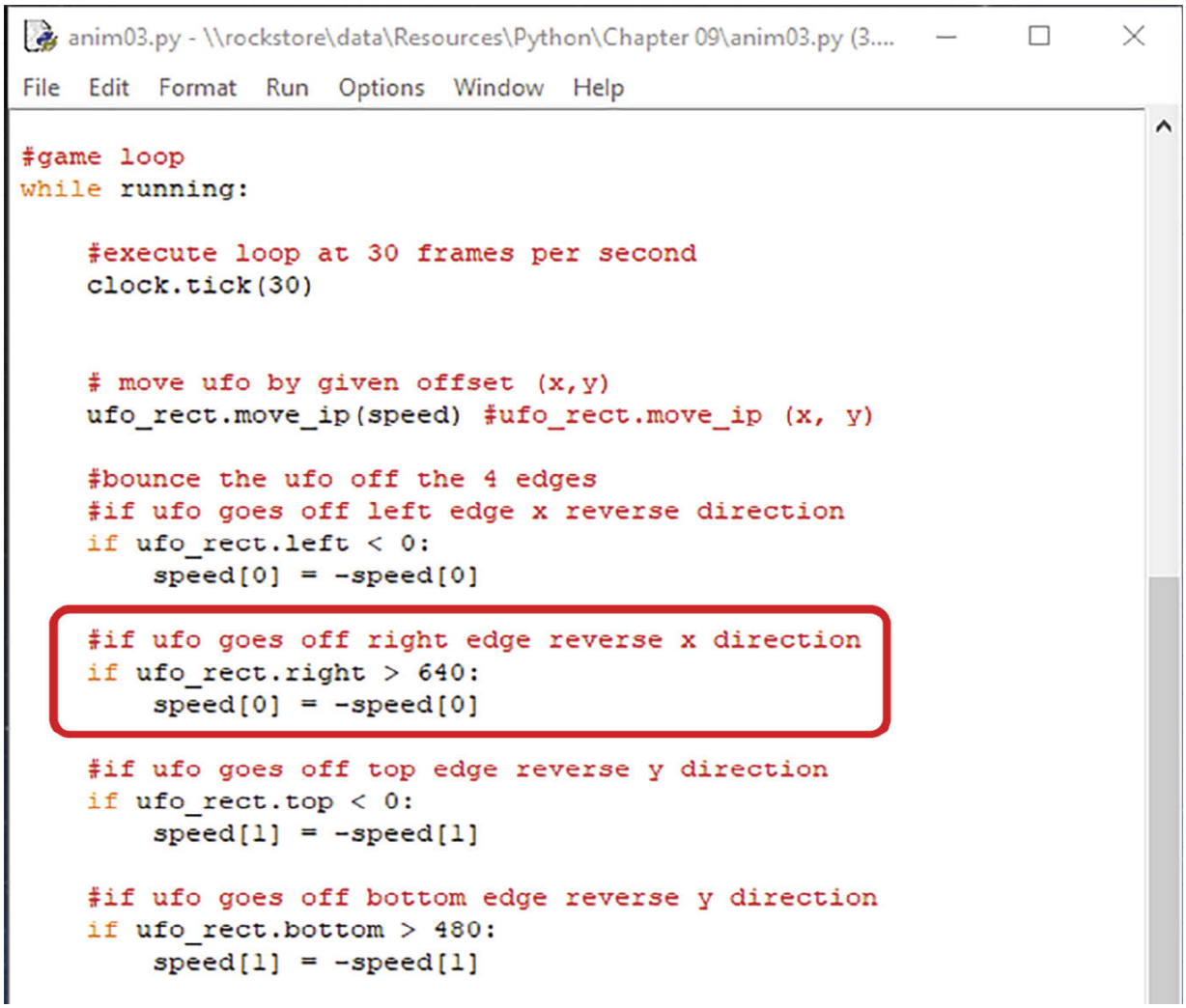
Again, change the speed[0] to a negative number.

```
if ufo_rect.right > 640:
```

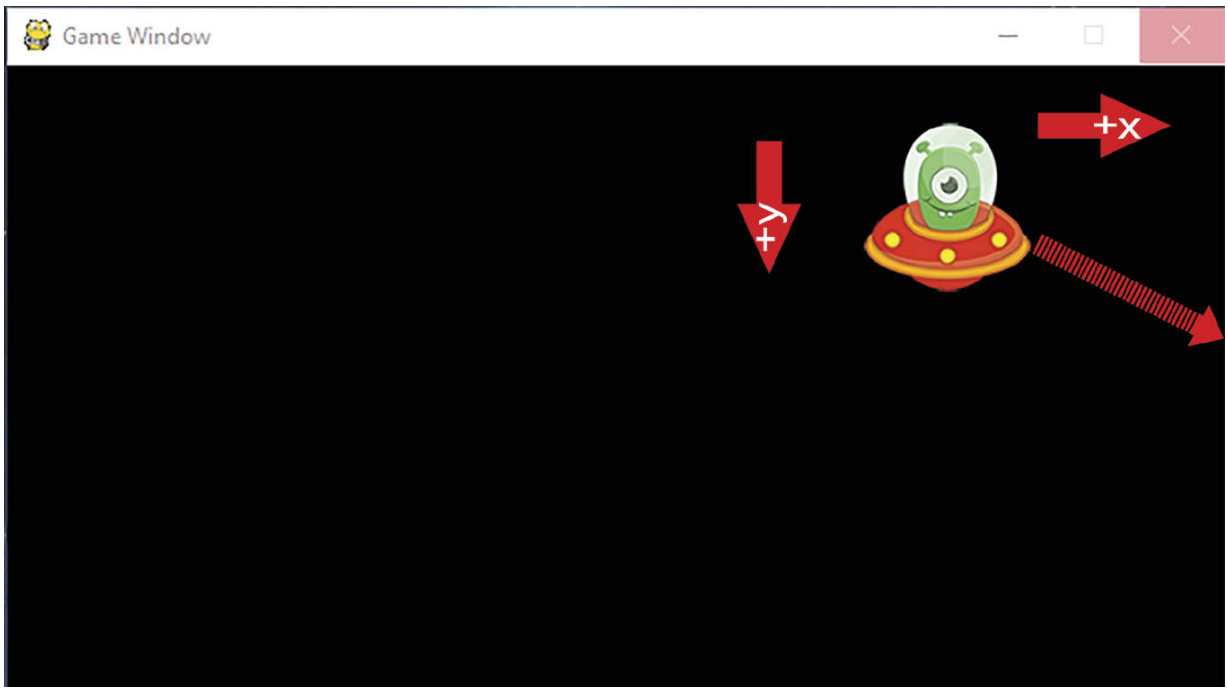
```
speed[0] = -speed[0]
```

Do the same with the top and bottom. Give it a try.

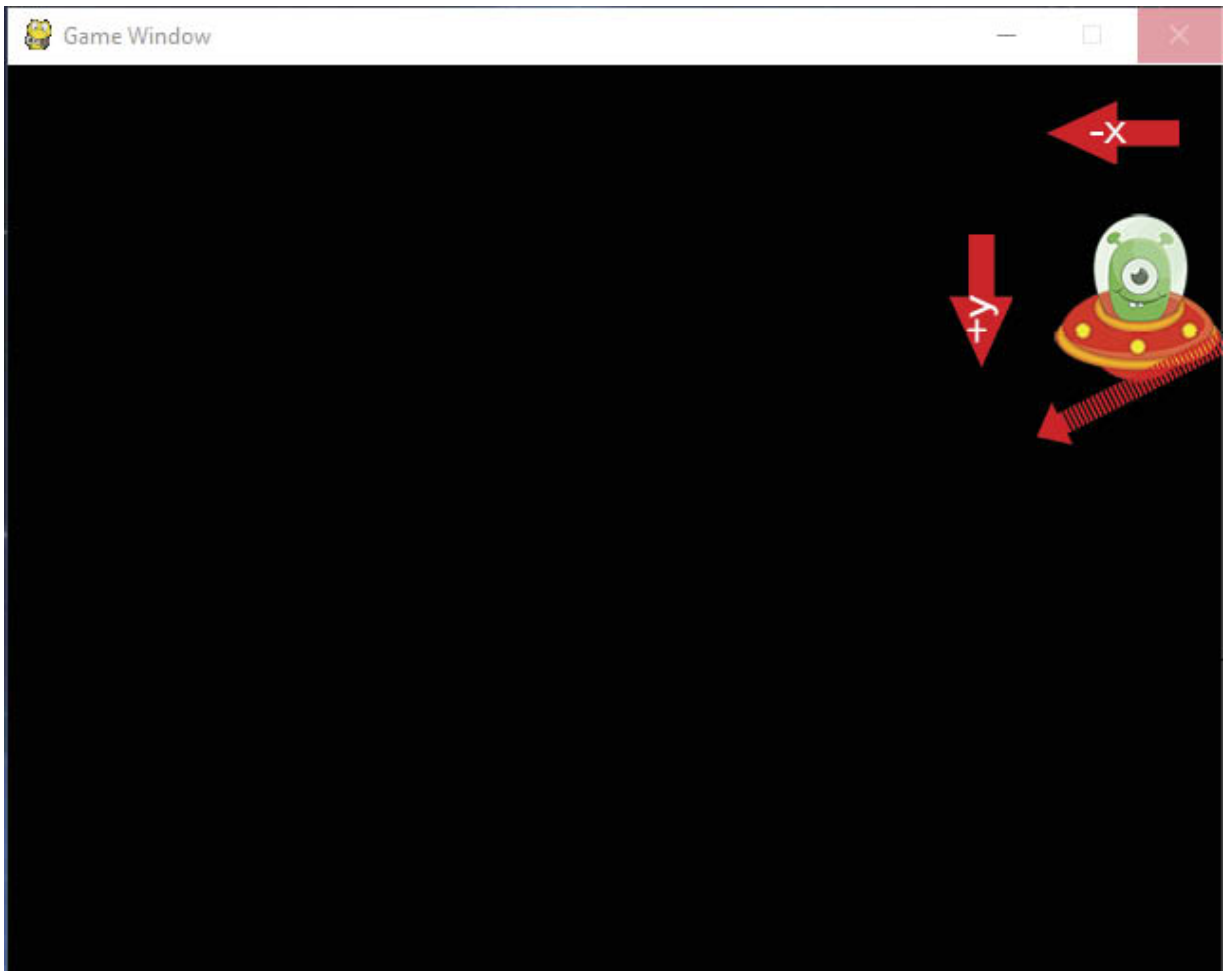
Lets take a look at the program. Open anim03.py. Here, you'll see the ufo bounce around the screen.



When the ufo moves toward the right wall, x (speed[0]) is increasing and y (speed[1]) is increasing.



When the ufo hits the right wall, we change the direction of x (speed[0]), but not y (speed[1]).



Now x (speed[0]) is decreasing, but y (speed[1]) is still increasing.

Similarly for the other three sides.

What happens if you change the `speed[ ]` variables?

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

How would we add another ufo?

How would we add our rocket ship from the previous section?

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')]
```

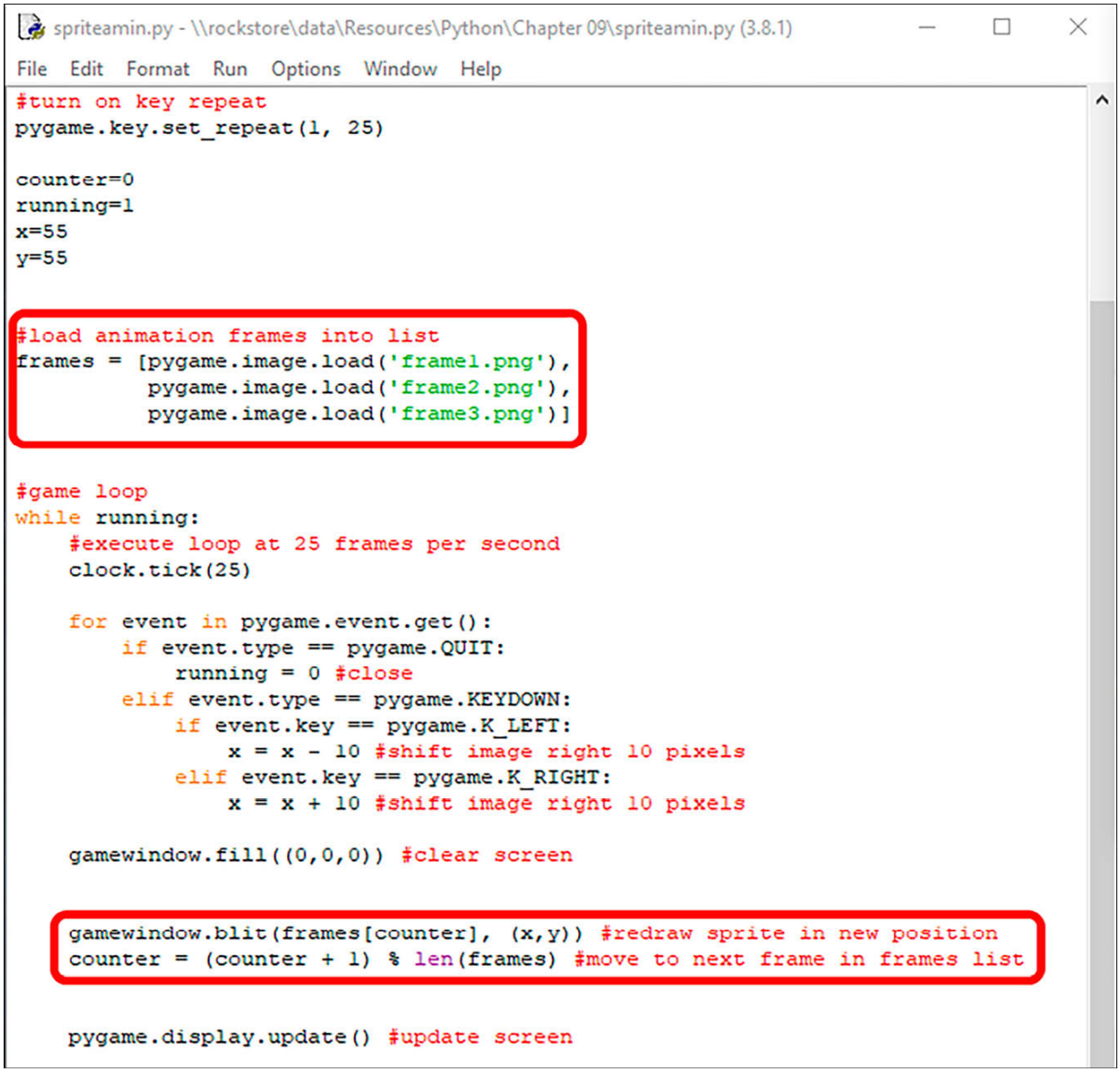
Now, inside your main game loop, you can draw the frame using the `.blit()` method to draw the frame from the frames list.

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

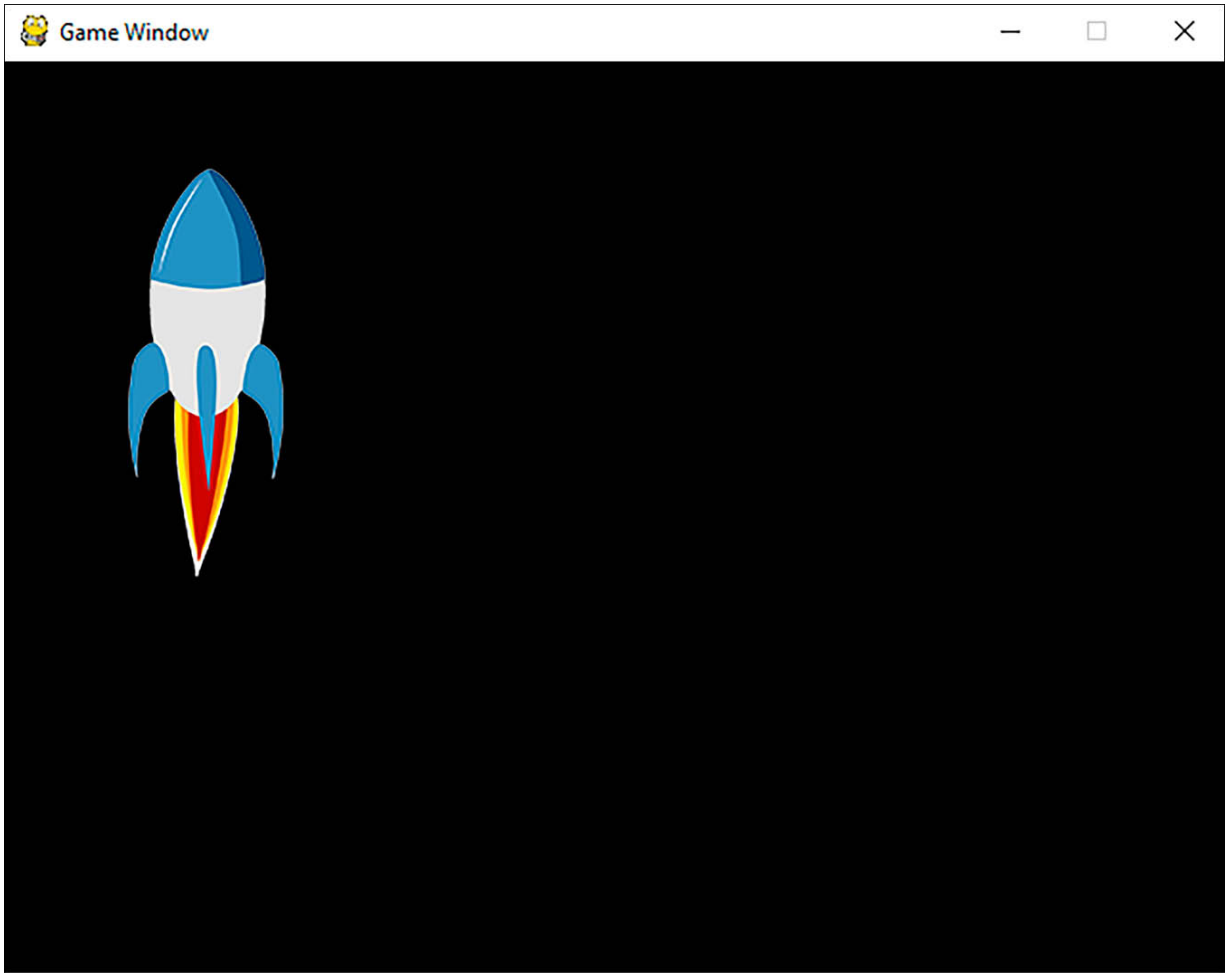
Select next frame in list, loop back to first frame at the end of the list. We can do this with the `len()` function to return number of frames in the list and modulus division.

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

Lets take a look at the program. Open spriteanim.py



With this particular animation, we have three frames to animate the flame effect on the rocket.


Game Window

Putting it all together

Now that we have learned some of the basics of how to use PyGame, lets have a look at how we can put all these skills together and create a simple game. In this particular version we won't be using any object oriented programming techniques, we'll just use procedures.

Have a look at final.py

To finish off the game, we need to add some bullets for the rocket to fire at the UFO. We can use a bullet png image.

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

We also need some variables for the position of the bullet on screen

```
bullet_X = 0
bullet_Y = y #the y position of the rocket
```

How many pixels to move the bullet at a time

```
bullet_Xchange = 0
bullet_Ychange = 5
```

Also need is a condition as to whether a bullet has been fired or not

```
bullet_state = "nofire"
```

Next, we need to update the event handler, so we can fire the bullet with the space bar. We move the position of the bullet image to the same position as the rocket plus about 30 pixels so it looks like its coming out of the front of the rocket. Draw the bullet on the screen and set the bullet state to "fire". This is how the program knows when a bullet has been fired or not.

```
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”
```

Next, we need to move the bullet up until it goes off the top of the screen which is 0 on the y axis

```
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
```

Now, what happens when a bullet hits the ufo? This is called a collision. First, we need to add some variables to contain the actual position of the ufo. Here, we've taken the center of the rectangle containing the ufo image.

```
ufo_X_pos = ufo_rect.centerx
ufo_Y_pos = ufo_rect.centery
```

We need to define a function. We take the x pos and y pos of the ufo, along with the x and y position of the bullet and run it through a formula to calculate the distance between them.

$$distance = \sqrt{(\text{ufo\_X\_pos} - \text{bullet\_X})^2 + (\text{ufo\_Y\_pos} - \text{bullet\_Y})^2}$$

If the distance is less than 35 pixels, we consider that a hit and return 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
```

After we've defined the collision detection function, we can call this

function to check the distance between the bullet and the ufo.

```
collision = isCollision(ufo_X_pos, ufo_Y_pos, bullet_X, bullet_Y)
```

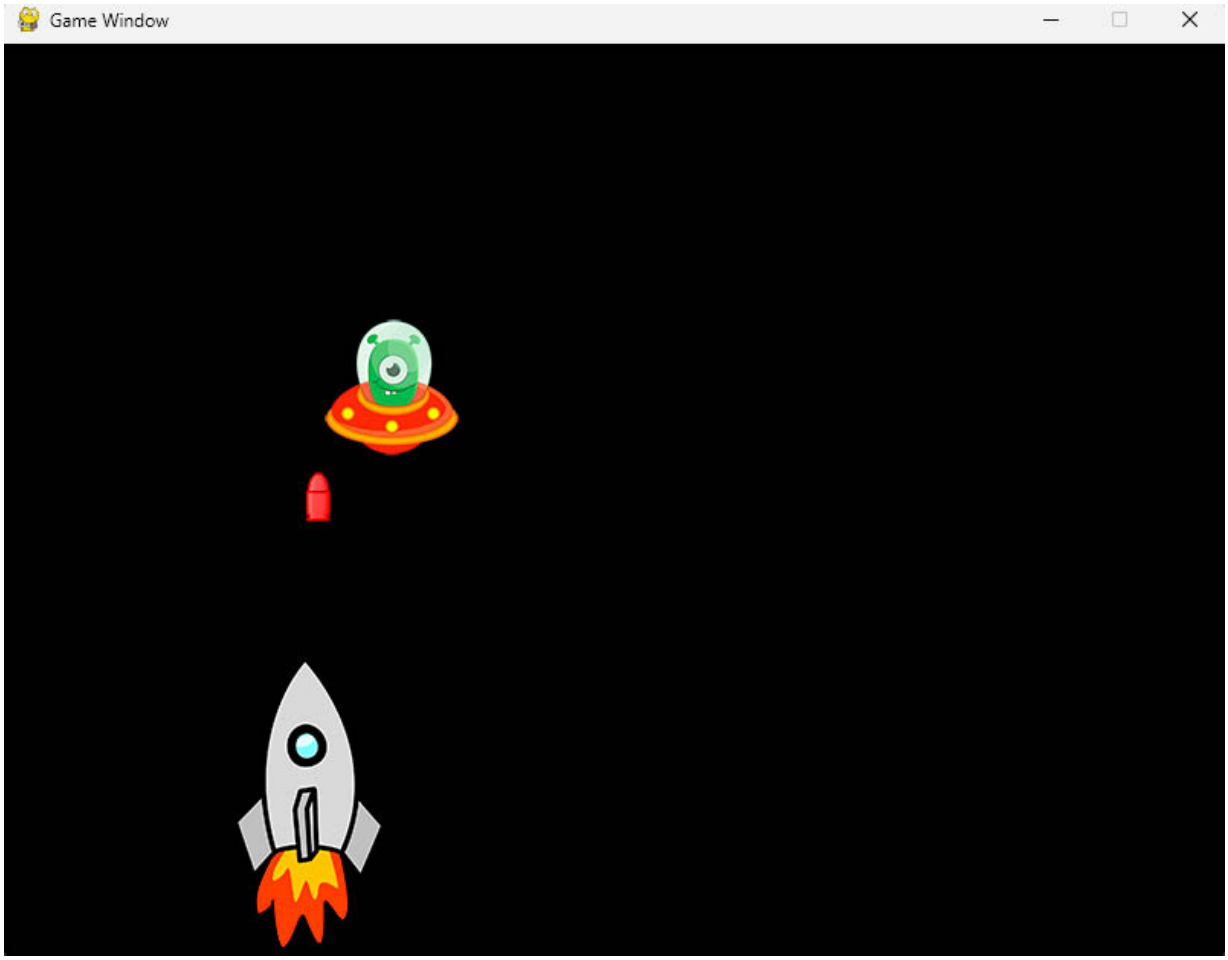
Once the UFO is hit, we can reset the bullet position, and the bullet state then show an explosion image to show the player the ufo has been hit

```
if collision:
bullet_Y = y
bullet_state = “nofire”
gamewindow.blit(exp, (ufo_X_pos, ufo_Y_pos))
```

Once the UFO has been hit, we can redraw the UFO on a random position on the screen.

```
ufo_rect.centerx = random.randint(0, 700)
ufo_rect.centery = random.randint(0, 500)
```

When you execute the program, you can move the rocket ship left and right, as well as fire a bullet.



The UFO will bounce around the screen. When you hit the UFO with the bullet, you'll see an explosion and the game will restart, placing the UFO in a random position on screen.

Lab Exercises

1. There are a few bugs in the code, try it out and see if you can find them and improve the program.
2. How would you improve the program?
3. Could you add a scoring system? Check out text.py
   For text you'll need (specify typeface and size):

   ```
   gameFont = pygame.font.Font('fontfilename', 32)
   ```

   To specify text to display and color name in quotes:

   ```
   gameText = gameFont.render('text…', True, 'color')
   ```

   To render the text in the window at x,y co-ordinates:

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

4. Could you add another UFO?
5. What about adding a background image?
6. Try adding Sound Effects. Check out audio.py
   You'll need to load sounds:

   ```
   bulletSound = pygame.mixer.Sound('bullet.wav')
   hitSound = pygame.mixer.Sound('hit.wav')
   ```

   To play the sound:

   ```
   bulletSound.play()
   hitSound.play()
   ```

Mini Project

Most of the time, larger programs like this will be written using object oriented programming. Lets take a look at an object oriented version. Go to the following website

`elluminetpress.com/python`

Download and unzip `My Invaders CH12 OOP Project`. Have a look at `My Invaders Notes.pdf` along with the code files.

Python Web Development

Python is widely used for developing large scale web applications that are not possible to build using .NET and PHP.

Python supports features that are executed with different frameworks such as Django, Flask, Pyramid and CherryPy commonly used in sites such Spotify and Mozilla.

Web Servers

Most Python web applications are executed on a web server through an interface called WSGI (web server gateway interface). Other Python scrips are executed through CGI (common gateway interface).

Here, we've installed the Python interpreter and enabled the WSGI adapter module for the Apache web server.

For this section you'll need access to a web server with Python support.

Upload your scripts to your public_html directory on the server, then on your computer, open a web browser and enter the URL to your script.

```
http://server-name/script-name.py
```

Eg:

```
http://titan/script.py
```

To start writing your Python scripts, you'll need to tell the web server where to find the Python interpreter. This is usually:

```
#!/python/python
```

or on a Linux server:

```
#!/usr/bin/python
```

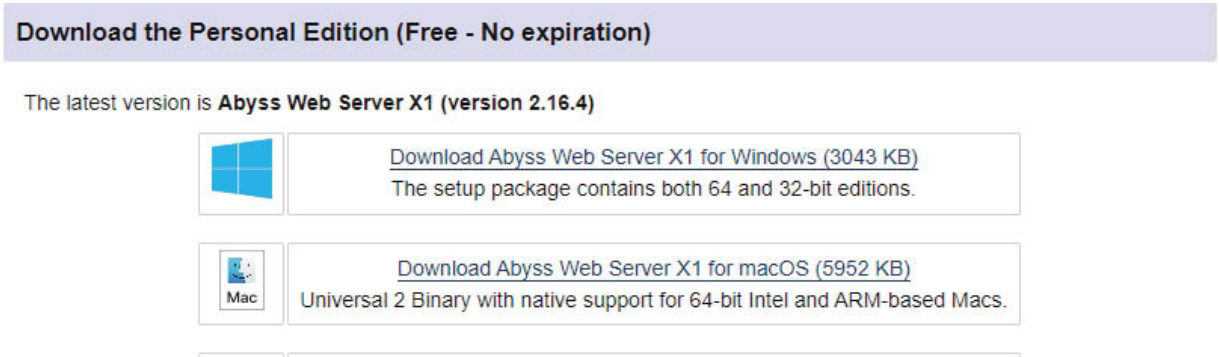
This is the first line of your script

Installing a Web Server

There is a free webserver available from Aprelium called Abyss Web Server X1 that you can install on your computer to develop and test websites

`aprelium.com/abyssws/download.php`

Select your version, Windows, Mac or Linux.



Open the directory where you have saved the software package. Double-click on the software icon (abwsx1.exe) and follow the on screen prompts.

Deselect components you do not want to install. Auto Start enables Abyss Web Server auto starting when a Windows session starts – deselect this. Start Menu Shortcuts enables adding Abyss Web Server shortcuts in the Start Menu. Documentation installs help files. Click Next.

Choose a directory where you want to install Abyss Web Server files. From now on, <Abyss Web Server directory> will refer to this directory.

`c:\Abyss Web Server`

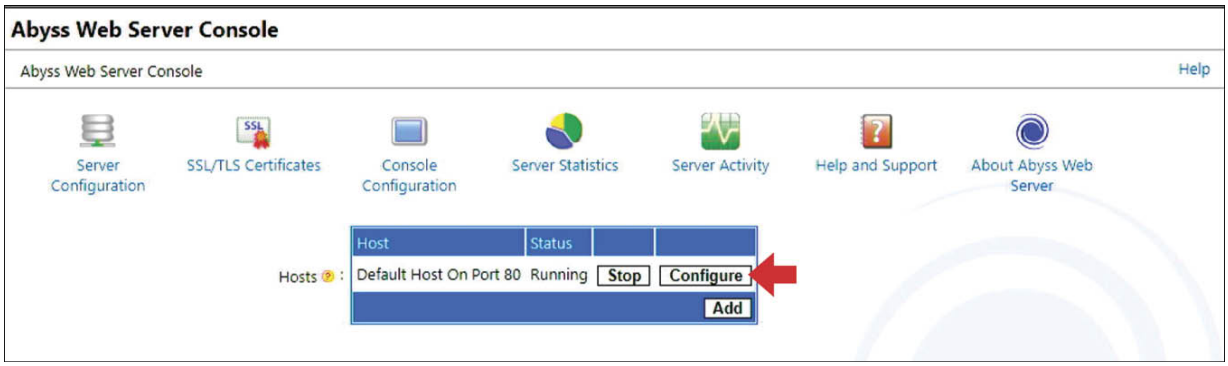
Click Install.

Set up Python Support

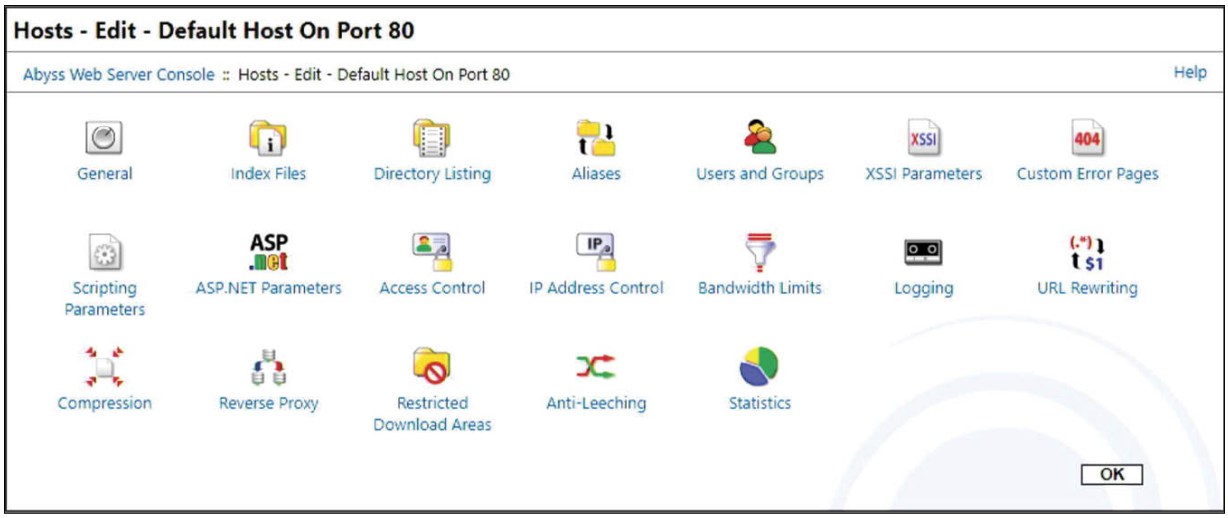
Start the web server (select Abyss Webserver X1 from the start menu), then open Abyss Web Server's console. Type the following into your web browser

```
127.0.0.1:9999
```

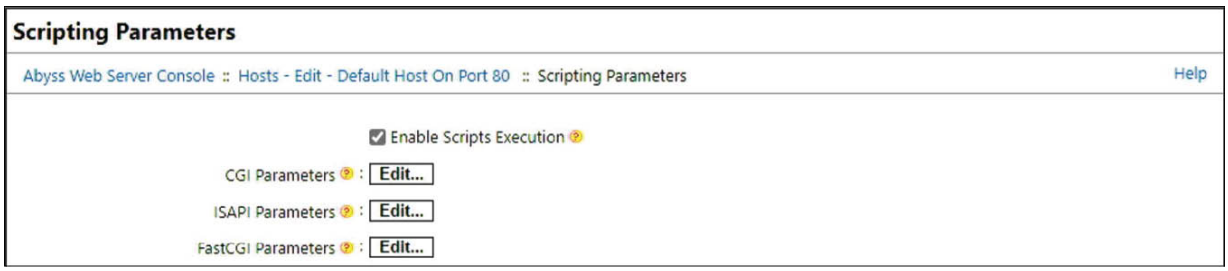
In the Hosts table, click Configure in the row corresponding to the host to which you want to add Python support.



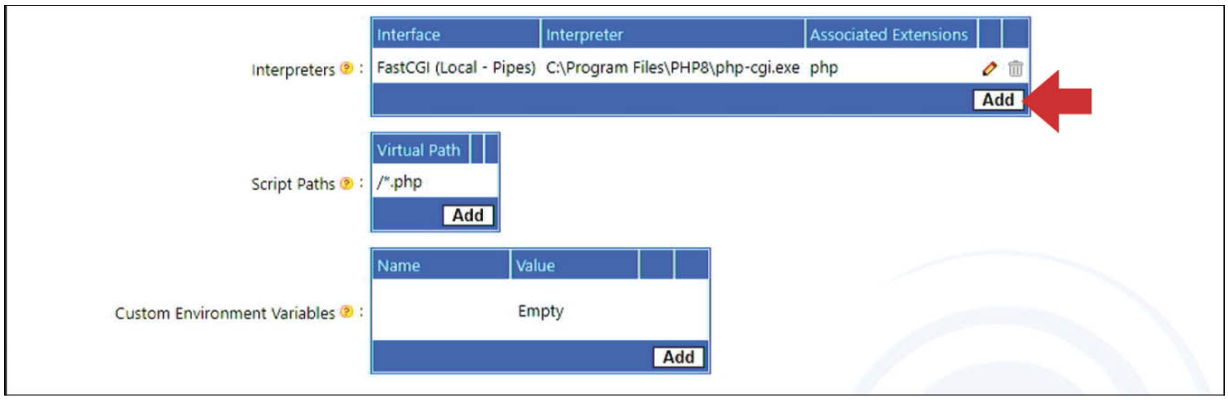
Select Scripting Parameters:



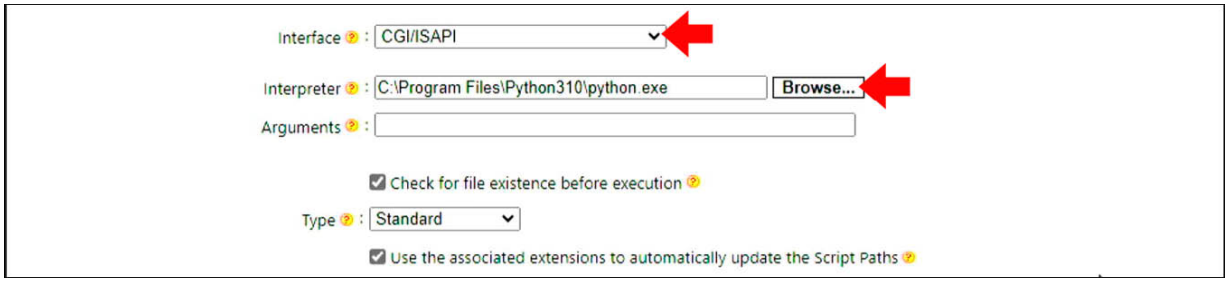
Check Enable Scripts Execution.



Click Add in the Interpreters table.



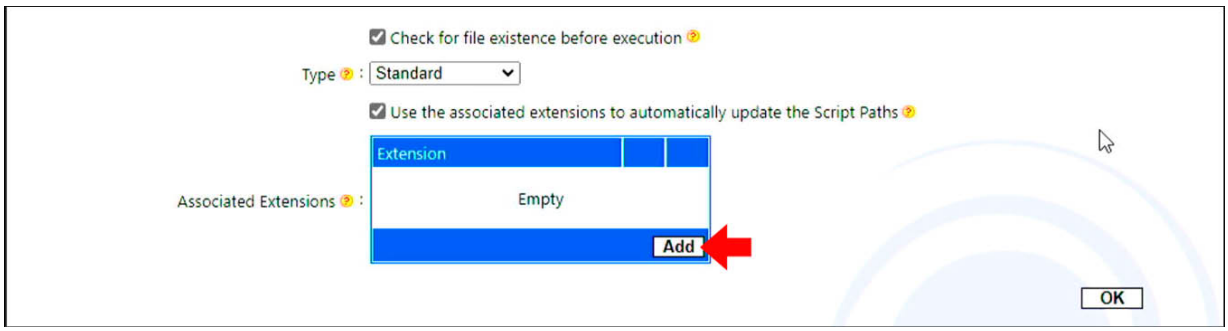
Set Interface to CGI/ISAPI.



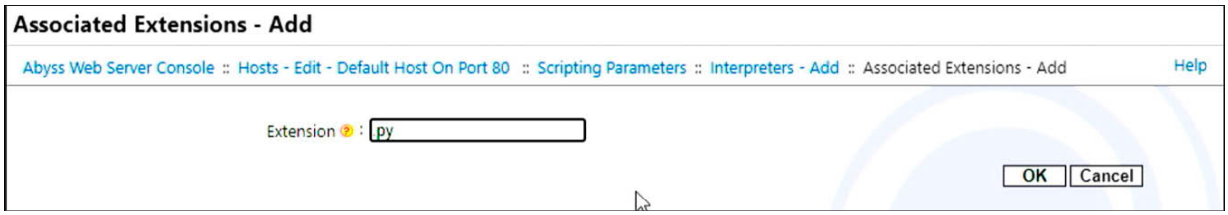
In the Interpreter field, click Browse... Go to the directory where you have installed Python, and click on python.exe.

Check Use the associated extensions to automatically update the Script Paths.

Click Add in the Associated Extensions table.

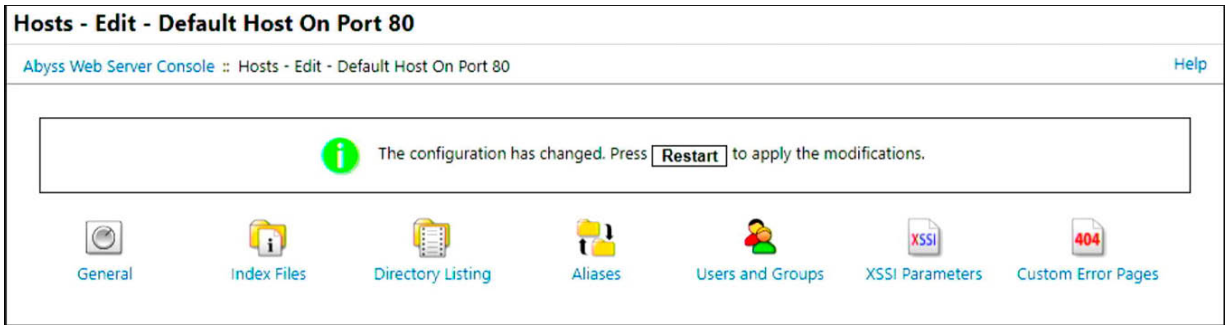


Enter py in the Extension field and click OK. Then click OK again.

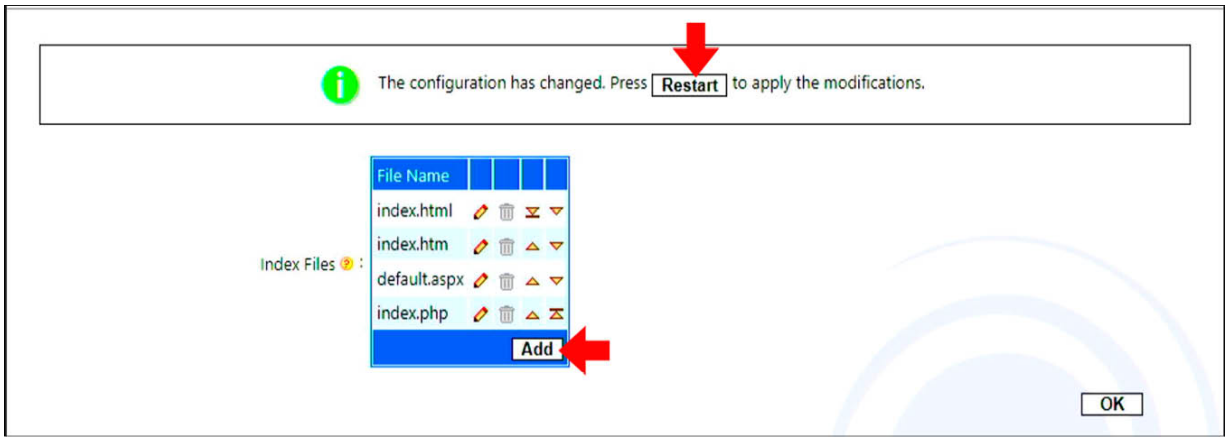


Click OK in the Scripting Parameters dialog to get back to the hosts screen.

Select Index Files.



Click Add in the Index Files table.

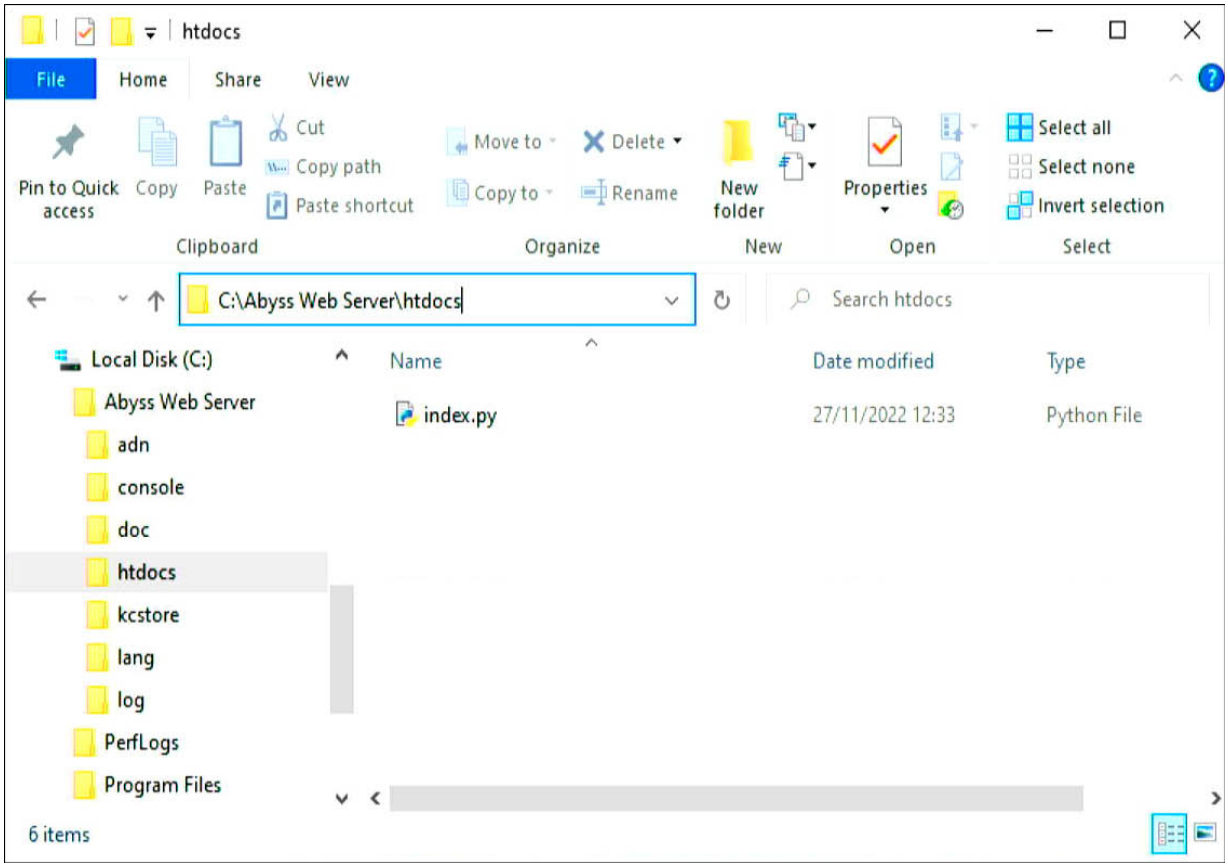


Enter index.py in the File Name field and click OK.

Click Restart at the top of the screen to restart the server.

Where to Save Python Scripts

Upload your scripts to your public_html, or htdocs directory on the server. If you're using Aprelium it will be htdocs.



On the Aprelium webserver it will be

`c:\Abyss Web Server\htdocs`

Executing a Script

To start writing your Python scripts, you'll need to tell the web server where to find the Python interpreter. This is the first line of your script.

On Windows/Mac use

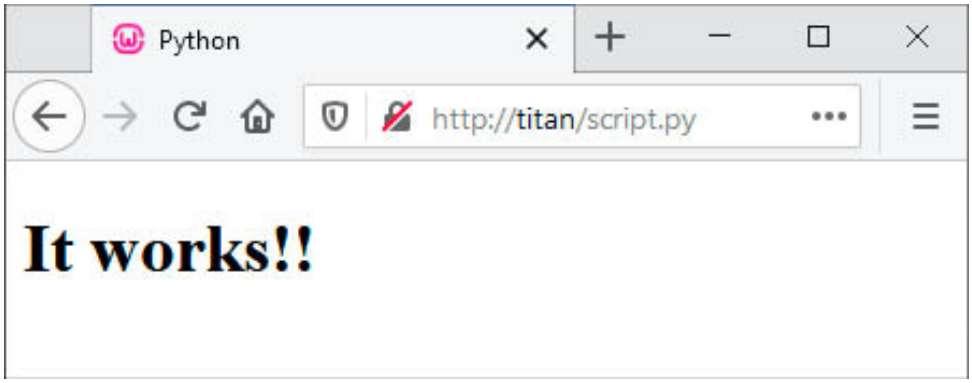
```
#!/python/python
```

Or on a Linux server:

```
#!/usr/bin/python
```

Lets take a look at an example. Have a look at script.py. Here we've written a script to output a simple HTML page.

This page simply outputs the heading 'It works!!'. Upload the script into your public_html directory on your web server, then navigate to the script URL using your web browser on your computer.



In our lab URL this would be:

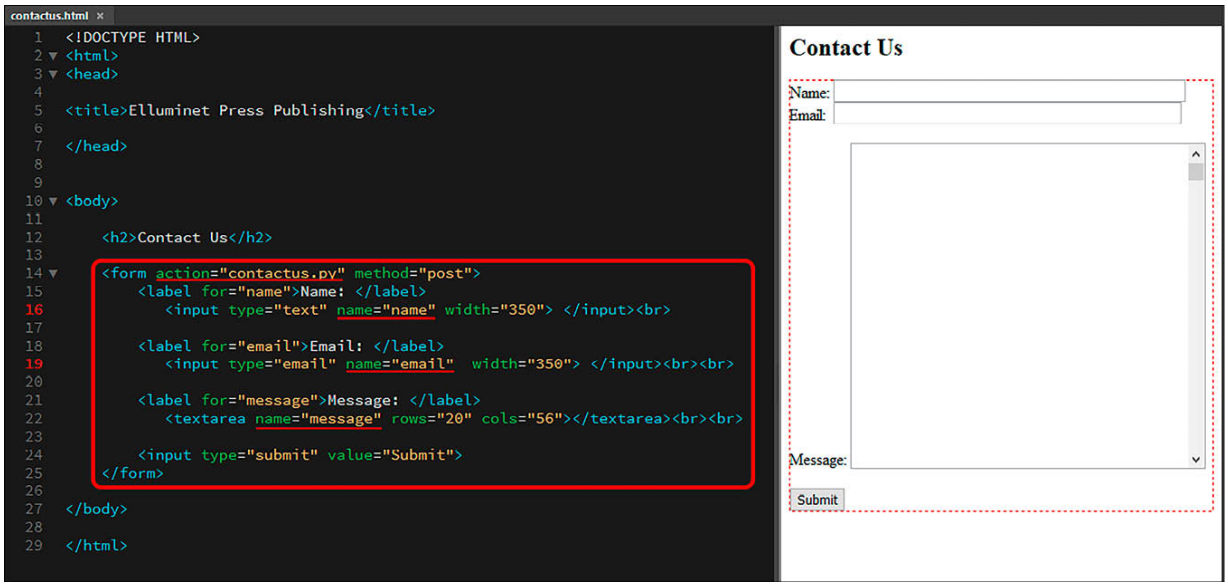
`http://titan/scriptfilename.py`

If you're using aprelium personal server on your own computer you can use

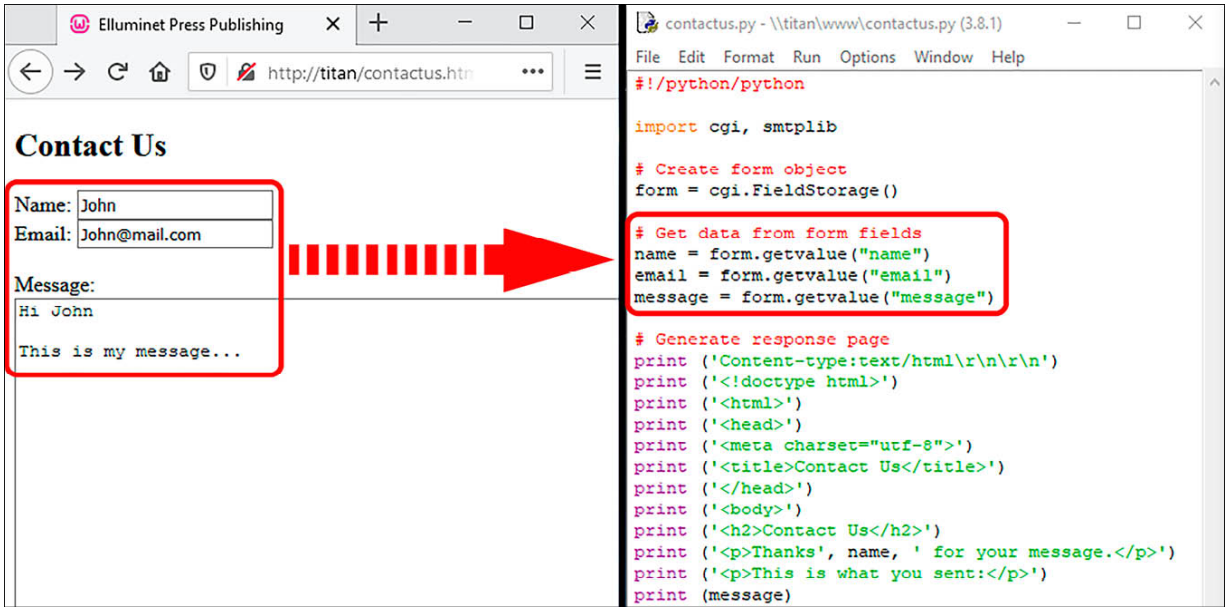
`http://localhost/scriptfilename.py`

Substitute 'scriptfilename.py' with the filename of your script.

Lets take a look at a practical example. Here, we're creating a simple contact form. The user is presented with an HTML form that asks for their name, email address and a message.



When the user clicks the 'submit' button, the HTML form calls a python script called contactus.py



The python script processes the data passed from the HTML form and stores it in a form object.

We can then get the values passed from the HTML form and store them in this object.

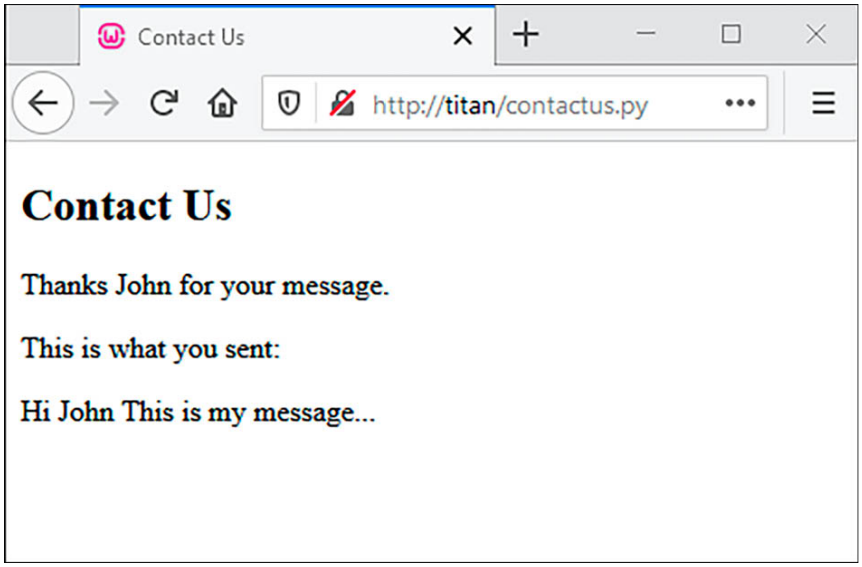
The python script generates another HTML page using print statements for the response to the user.

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>')
```

You can see the output in the browser below.

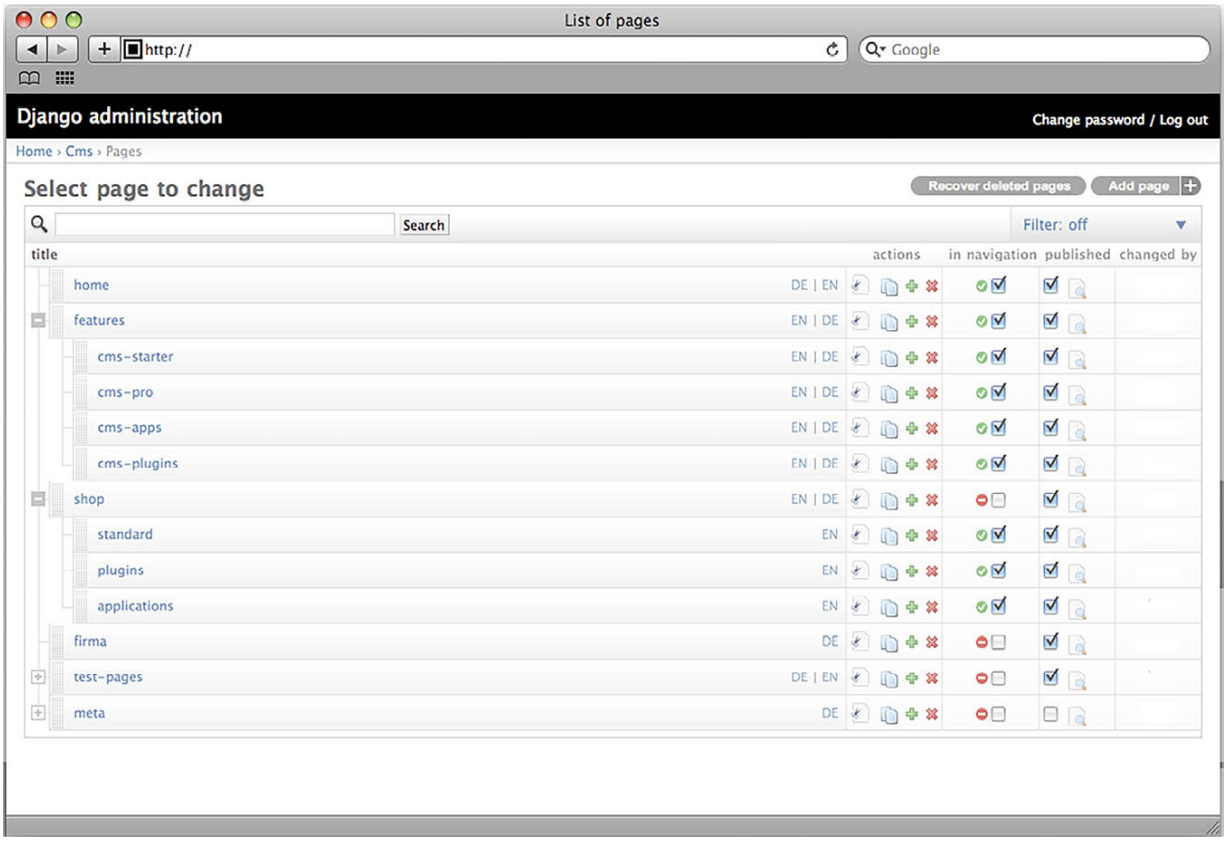


Python Web Frameworks

If you are using Python in web development, you'll more than likely be using a Python web framework rather than the old CGI we looked at in the previous section.

A Python web framework is a collection of tools, libraries and technologies that allow you to build a web application.

One example of a Python web framework is Django (pronounced "Jango").



Another example is Flask. Let's take a look at how to create a simple web app using this framework.

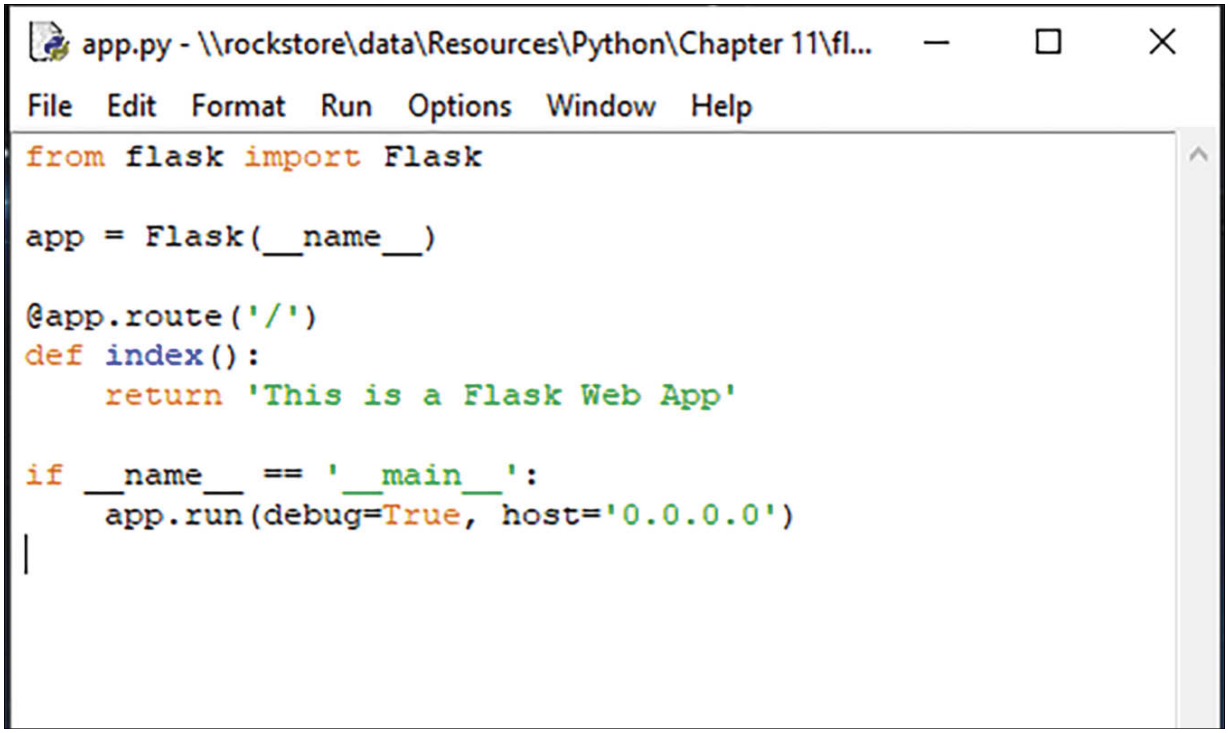
The first thing you'll need to do is install the Flask module if you haven't already done so. Use the following command in the admin command prompt

```
pip install Flask
```

Use this if you're on a linux based machine

```
sudo pip install Flask
```

Lets create an app. First thing we need to do is create our main program. To do this we create a new file called app.py. We've included all these files in the Flask directory in the resource files.



Here, we've imported our Flask module.

Modern web apps use a technique called routing. This means instead of having a URL to a page

`localhost/resources.php`

...we use a route

`localhost/resources/`

So our first route is the root of our website and is usually the index page. We use `@app.route('/')` to determine this.

The '/' means the root of the website:

```
http://localhost:5000/
```

`def index()` is the name you give to the route defined above. This one is called index, because it's the index (or home page) of the website.

`host='0.0.0.0'` means the app is accessible from any machine on the network.

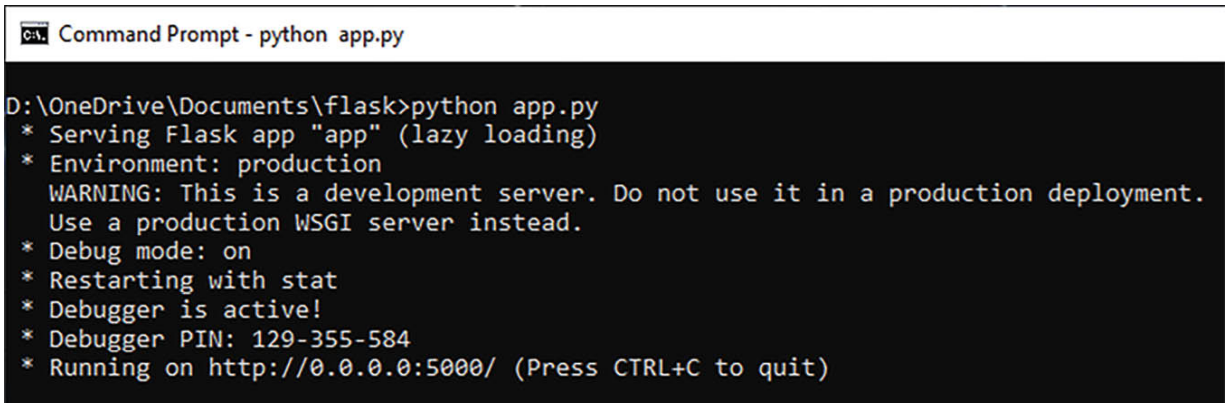
To run and test the app, we need to open it using the development environment. This is a simple web server that allows you open the app in a web browser for testing.

To do this, open your app directory in the command prompt. In this particular example, the app files are in OneDrive/Documents/Flask

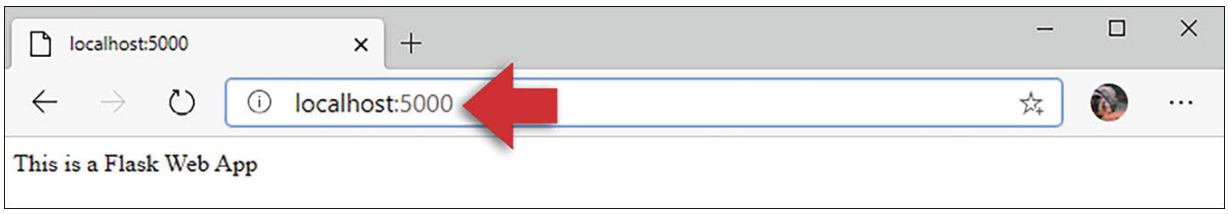
To start the app type

```
python app.py
```

Once you press enter, the server will start.



You can open the app in a web browser. On your own workstation you can use `localhost:5000`



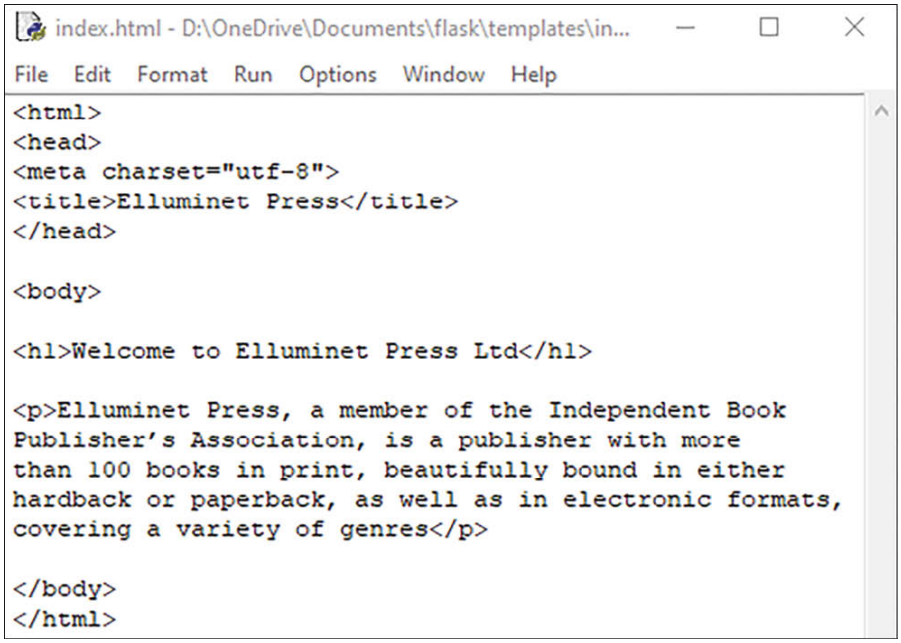
To add another page, add another route.

```
@app.route('/shop')
def shop():
    return 'This is the shop page'
```

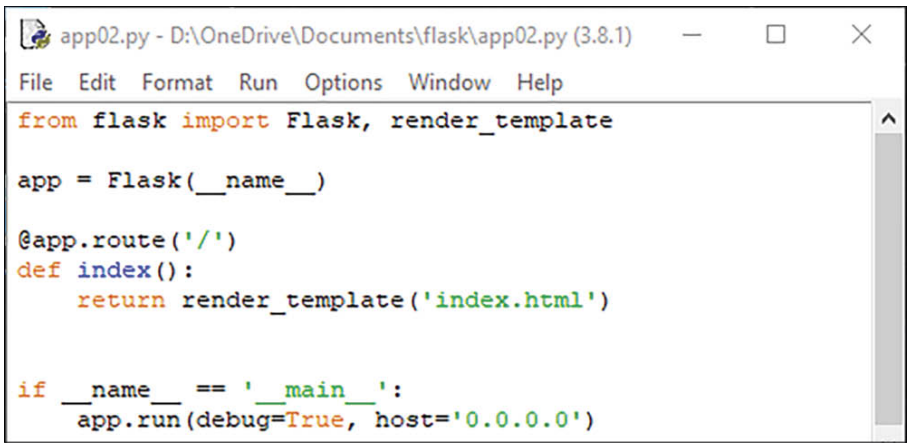
Now in the web browser you can use

`localhost:5000/shop`

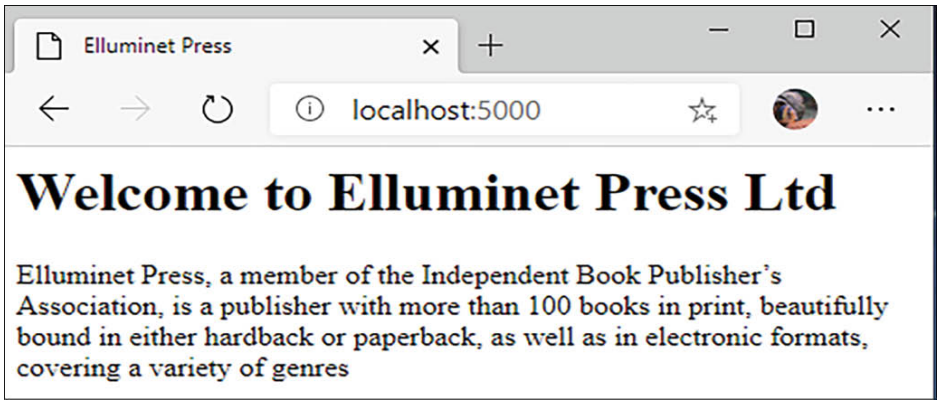
Now that we have our base app, we can develop web pages for the app to call. These web pages are called templates and we store these in a template directory. Here's a simple HTML page we've created an saved in the templates directory.



Lets call our HTML page from our app. We can use the `render_template()` function.



When we view the page in a browser, we'll see a rendered version of the HTML page.

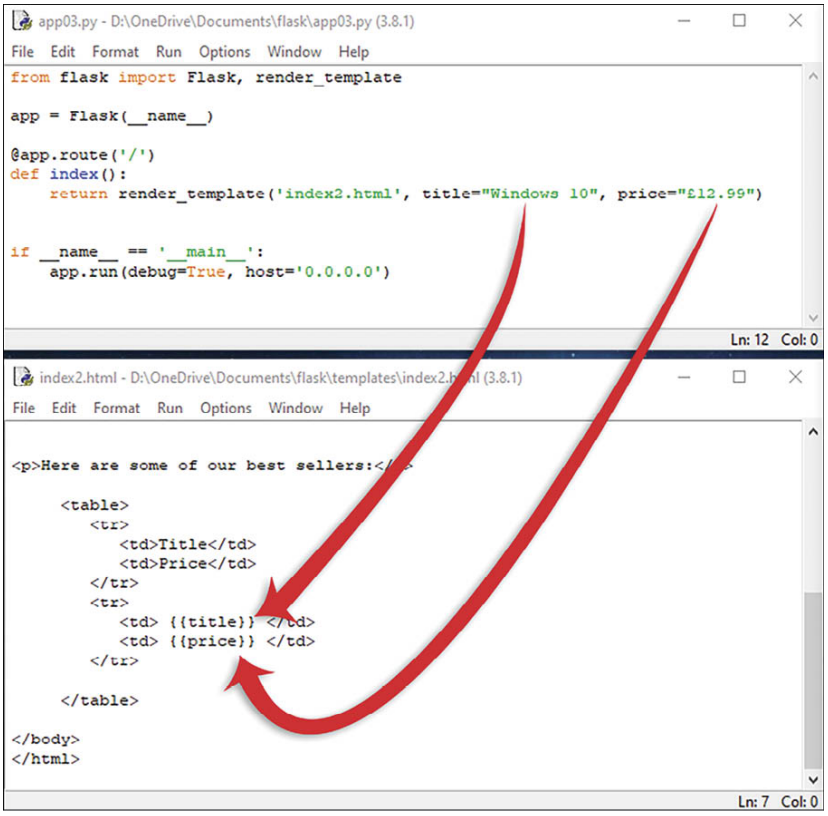


You can pass variables to your HTML templates. To do this embed the variable using `{{variable-name}}` in your HTML.

Then add the variable as a parameter to the `render_template()` function.

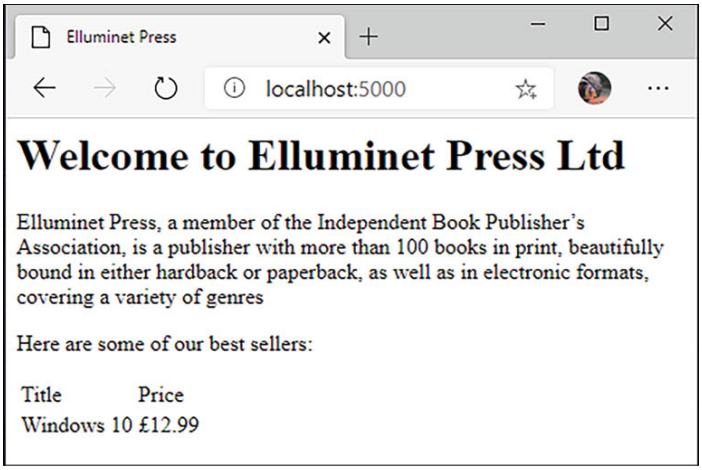
```
render_template('index.html', variable-name = "...")
```

Lets take a look at an example. Open the file app3.py and index2.html



Here, we've passed the title and price as variables to the HTML template.

You can add images to your templates using HTML and CSS code. You can also embed python code.



Resources

To help you understand the procedures and concepts explored in this book, we have developed some video resources and app demos for you to use, as you work through the book.

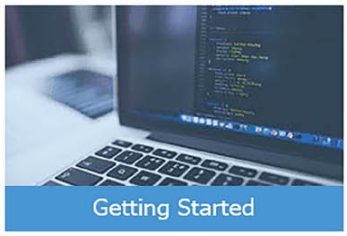
To find the resources, open your web browser and navigate to the following website

`elluminetpress.com/python`

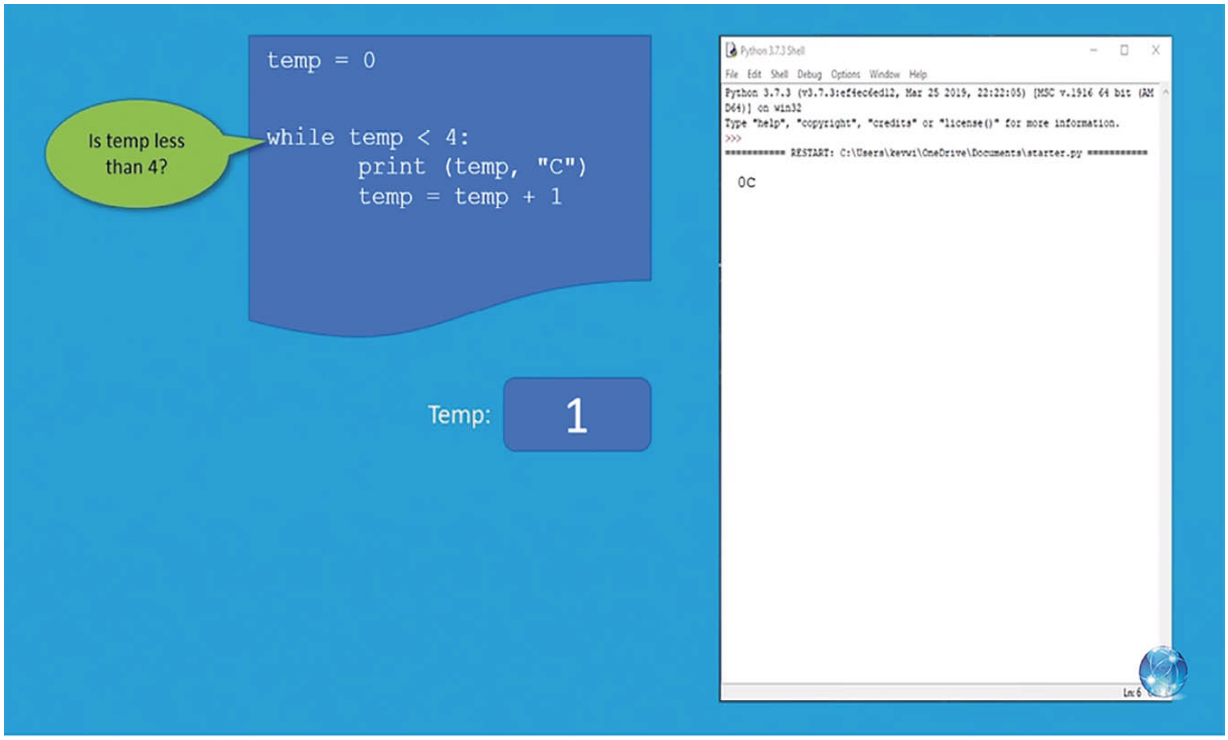
At the beginning of each chapter, you'll find a website that contains the resources for that chapter.

Using the Videos

You'll find video resources for each section of the book. Just click on the links to open the sections.



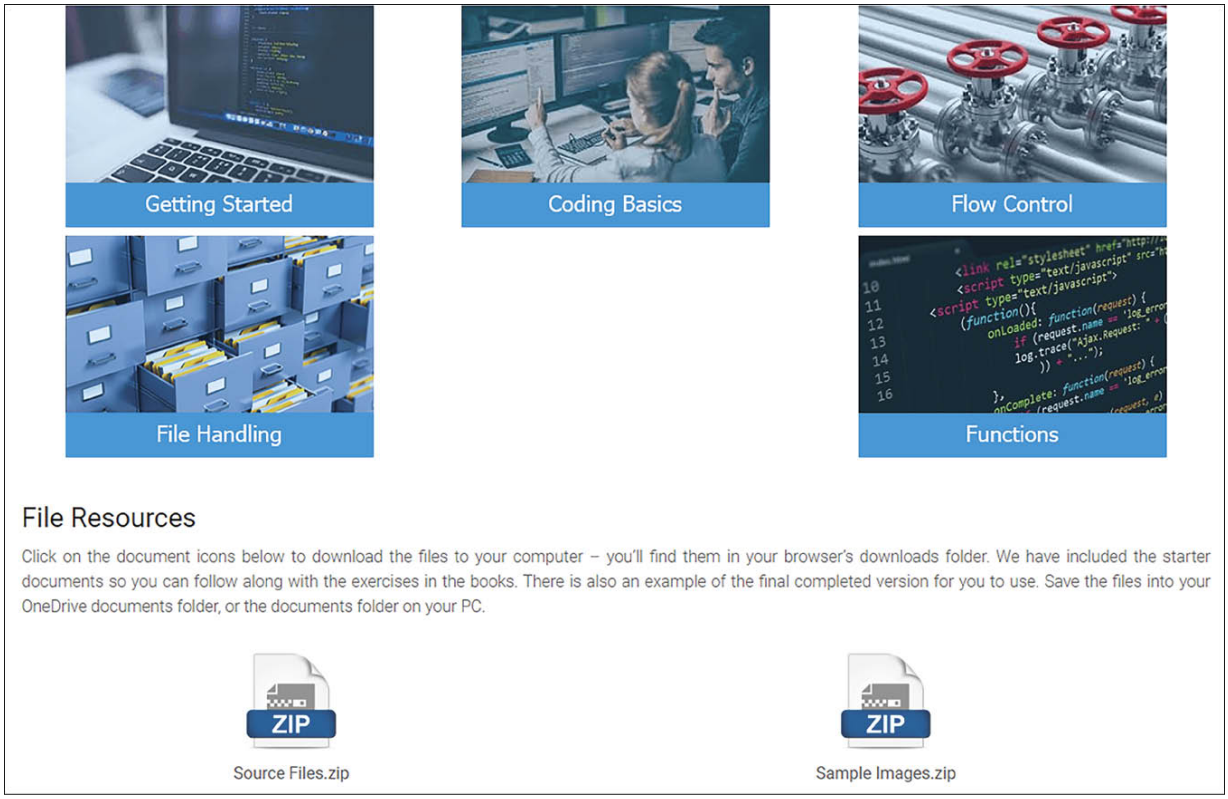
When you open the link to the video resources, you'll see a thumbnail list at the bottom.



Click on the thumbnail for the particular video you want to watch. Most videos are between 30 and 60 seconds outlining the procedure, others are a bit longer.

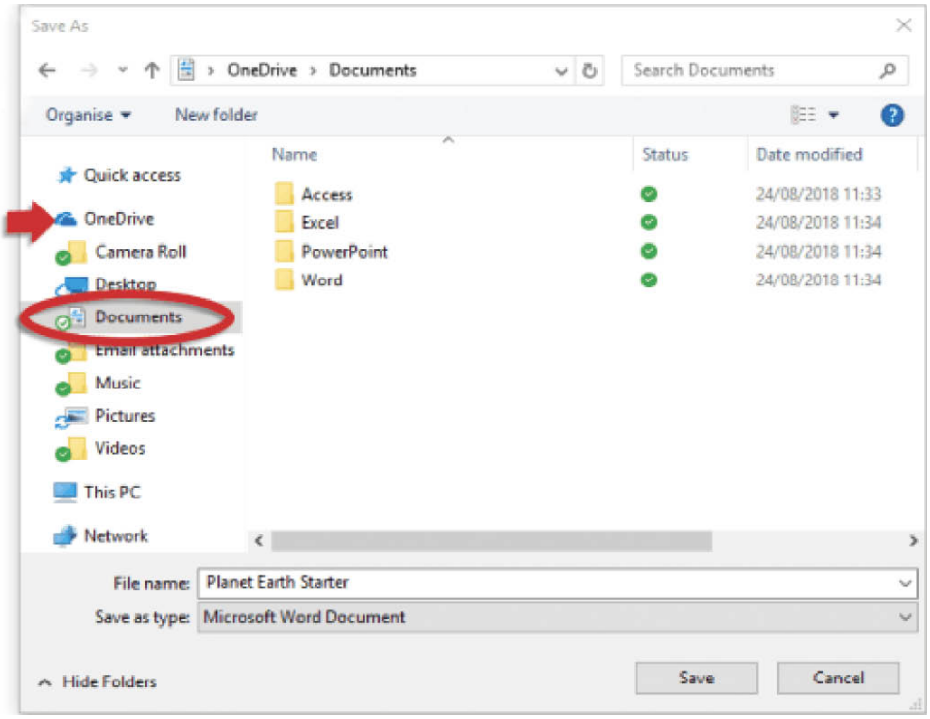
Downloading Example Code

You'll also find the source code files at the bottom of the main resource page for the book under 'file resources'.

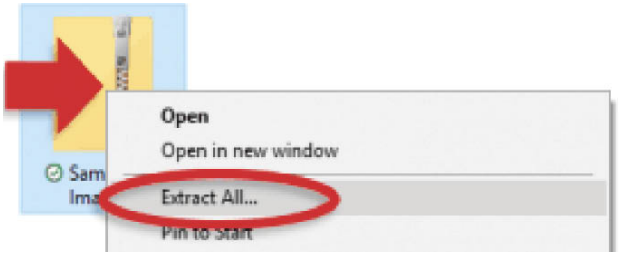


To download the zip file, right click on the zip icon on the page above, 'pythonfiles.zip'.

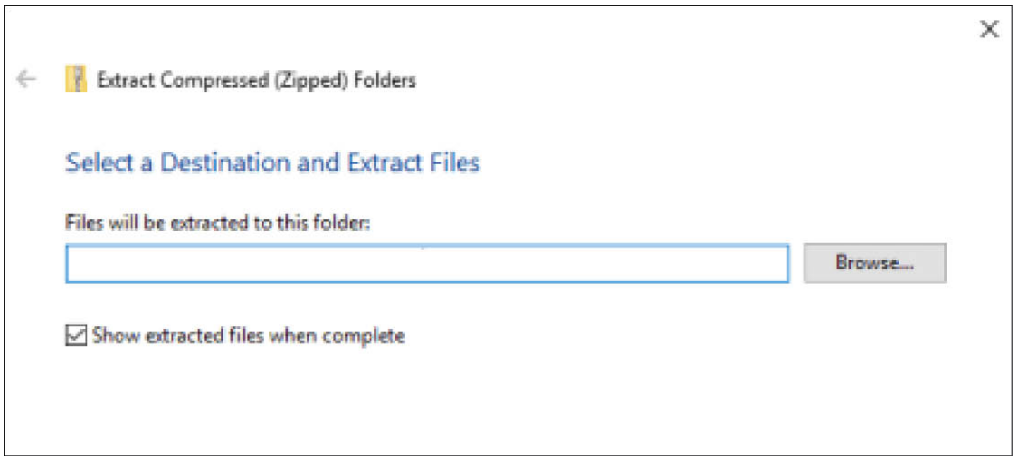
Select 'save target as' (or 'save link as', on some browsers) and save it into 'documents' on your OneDrive folder.



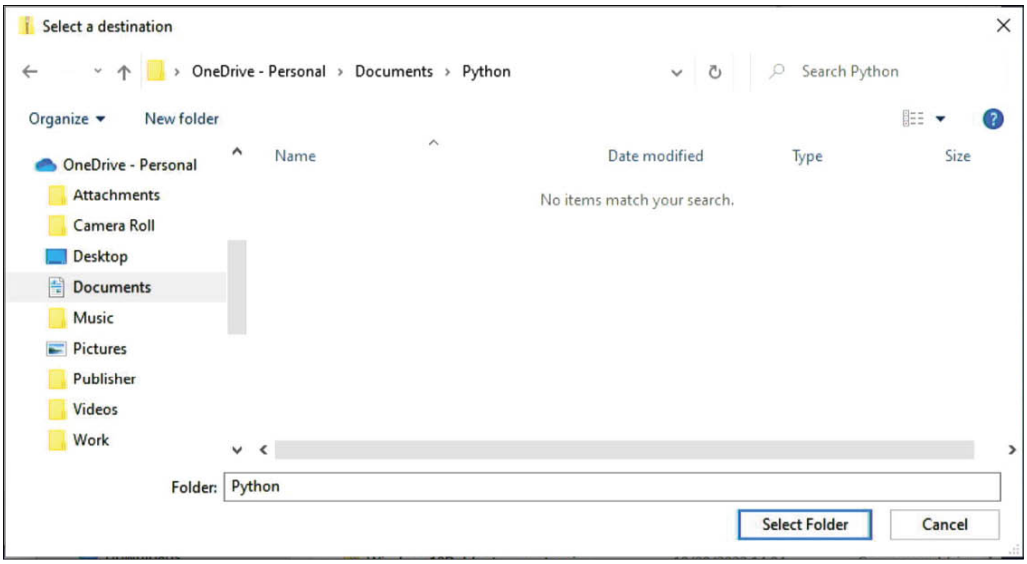
Once you have downloaded the zip file, go to your 'documents' folder in your OneDrive, right click on the zip file, and select 'extract all' from the menu.



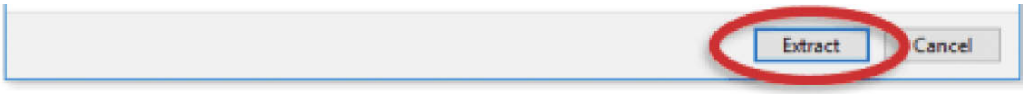
From the dialog box click 'browse'.



Navigate to the folder you want to extract the files into. Eg: OneDrive -> Documents folder. If you want to create a new folder ‘click ‘new folder’ - give it a name. Click ‘select folder’



Click ‘extract’.

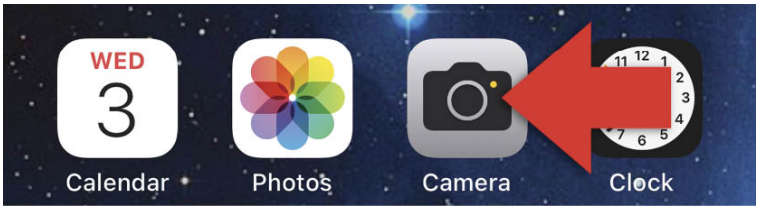


Scanning the Codes

At the beginning of each chapter, you'll a QR code you can scan with your phone to access additional resources, files and videos.

iPhone

To scan the code with your iPhone/iPad, open the camera app.

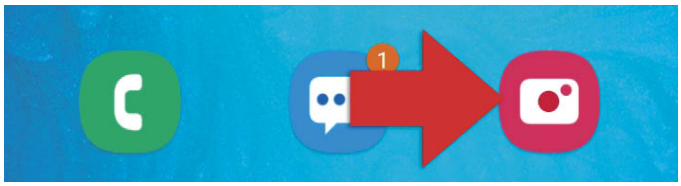


Frame the code in the middle of the screen. Tap on the website popup at the top.


Website QR Code
Open "elluminetpress.com"
in Safari
SLO-MO
VIDEO
PHOTO
PORTRAIT

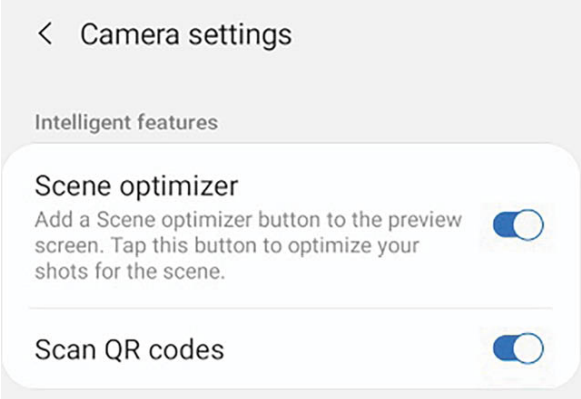
Android

To scan the code with your phone or tablet, open the camera app.



Frame the code in the middle of the screen. Tap on the website popup at the top.

If it doesn't scan, turn on 'Scan QR codes'. To do this, tap the settings icon on the top left. Turn on 'scan QR codes'.



If the setting isn’t there, you’ll need to download a QR Code scanner. Open the Google Play Store, then search for “QR Code Scanner”.