Introduction to Python 3 Programming - Session 3

V1.0

e-mail: training@glasgow.ac.uk

web: gla.ac.uk/services/it/training

Course content created by Blair Thompson

Last edited by Blair Thompson on 08/09/21

Contents

Introduction

This course runs in three, three hour sessions. It is designed to be an introduction to simple programming in Python for non-programmers. It is not a complete Python programming course. It is intended as course which will enable you to write simple programs to manipulate and analyse data.

Objectives

On successful completion of this course participants will be able to:

- Understand what a computer program is.
- Use the IDLE Shell and Editor windows
- Write a simple print script
- Save your program as a Python program
- Run a Python script from the command prompt
- Include comments in Python scripts
- Assign values to variables
- Perform basic calculations
- Use If statements
- Use For and While Loops in Python.
- Use the Completion tool to speed up your coding
- Manipulate text using Python.
- Open and Save text based files within a Python script
- Define functions and use them in your scripts
- Import modules and use their contained definitions

Session 3

Session Objectives

At the end of this session you should be able to

- Open and Save text based files within a Python script
- Define functions and use them in your scripts
- Import modules and use their contained definitions

1 Programming Files Access

If we are going to be working with larger amounts of data during our research, there comes a point when we will need to read and write to external files,

Why do we need to:

- Data of non-trivial size is almost always stored in files,
- Making two hundred variables is tedious.
- Having lists with 10,000 elements is memory intensive.

Python provides read, write, and append operations on files. Once a file has been opened in python we can manipulate it in different ways.

a. The Basics of File Opening

When we are using python within research sometimes it is useful to be able to open files to read data from them. As standard Python is able to open and manipulate any text based file (any file you can open and read using a text editor such as notepad)

You can open other types of files within python by using a dedicated library.

We open files using the open command. This following line creates a variable named book and loads the contents of a file into it.

```
book = open("C:\\CourseFiles\\Introduction to Python\\Pride and Predudice.txt")
```

If we don't explain to Python what we intend to do with the file we are opening, Python will open the file in read mode. In read mode we can read the contents, but we are unable to edit that file. We can also when opening the file tell Python that we intend to edit the file.

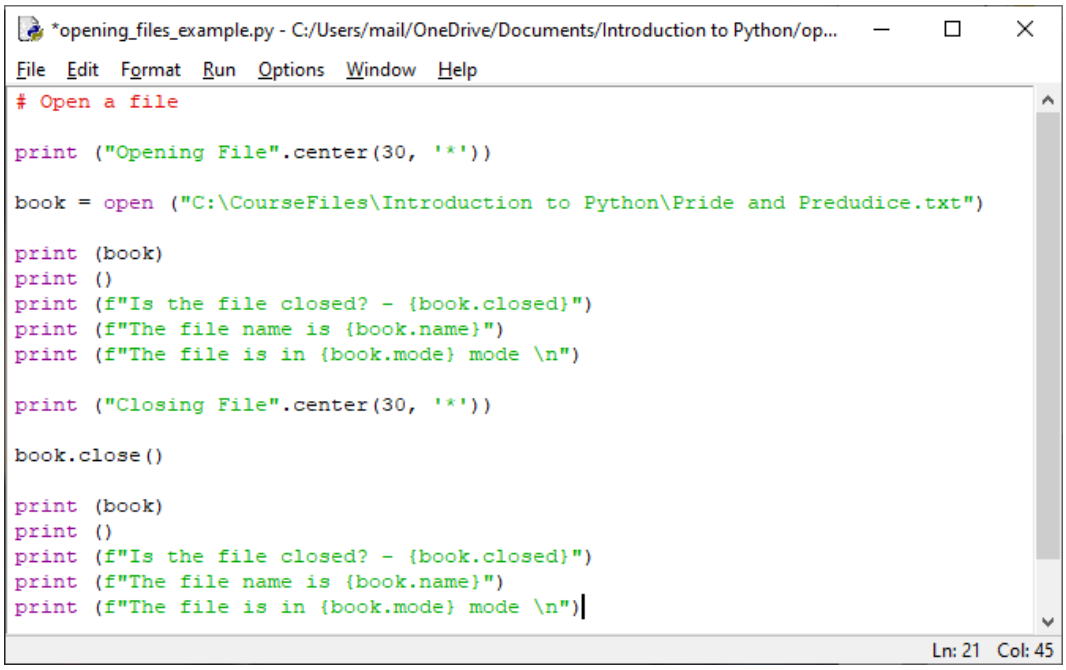
```
answers = open("C:\\CourseFiles\\Introduction to Python\\answers.txt", "w")
```

In this case the **"W"** argument is how we explain to Python what we indent to do with this file.

Task: Open a File

1 Open IDLE and Select **File** – **New File**

2 Input the following code



3 Save the script as **opening _files_example1.py**

4 Run the Script and observe the output

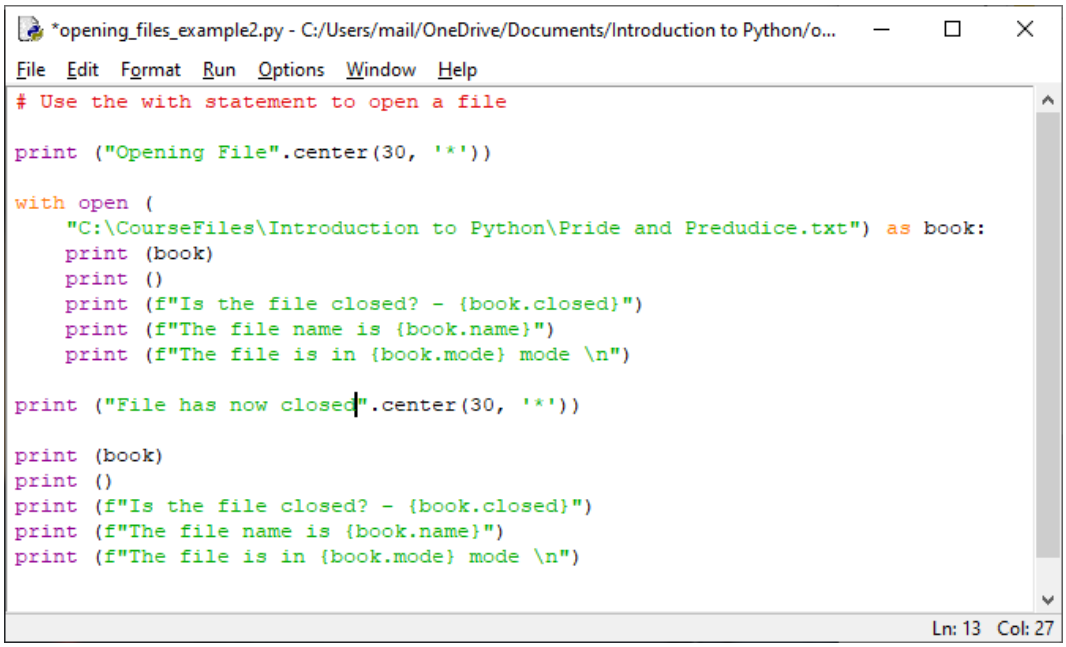
Note that the file is opened and then closed. When a file is open, it cannot be edited by the operating system or any other application. When it is closed with **.close()** it can no longer be manipulated by Python.

Additional Task: Open and Close a File

In the previous example we open a file and then close it. As mentioned before, while this file is open the file can become locked to other programs. If we forget to close the file this might cause issues. In Python there is another way that we can open a file. This following script opens the file for just enough time to do something with it.

1 Use the **Save As…** command to save the file from the last task as **opening_files_example2.py**

2 Adjust the code to look like this



3 **Save** the file

4 Run the script

Notice that the script produces exactly the same output as before, but this time you did not rely on the **.close()** method.

b. Reading the Lines of a File

It can be useful to break the contents of a file down into more manageable chunks. For this we use the .readlines method on our variable.

```
searchlines = book.readlines()
```

The **.readlines()** method works through the text file and whenever it finds a new line it adds its content to a list of strings. Now that we have that list we can then do things with it using string methods and functions.

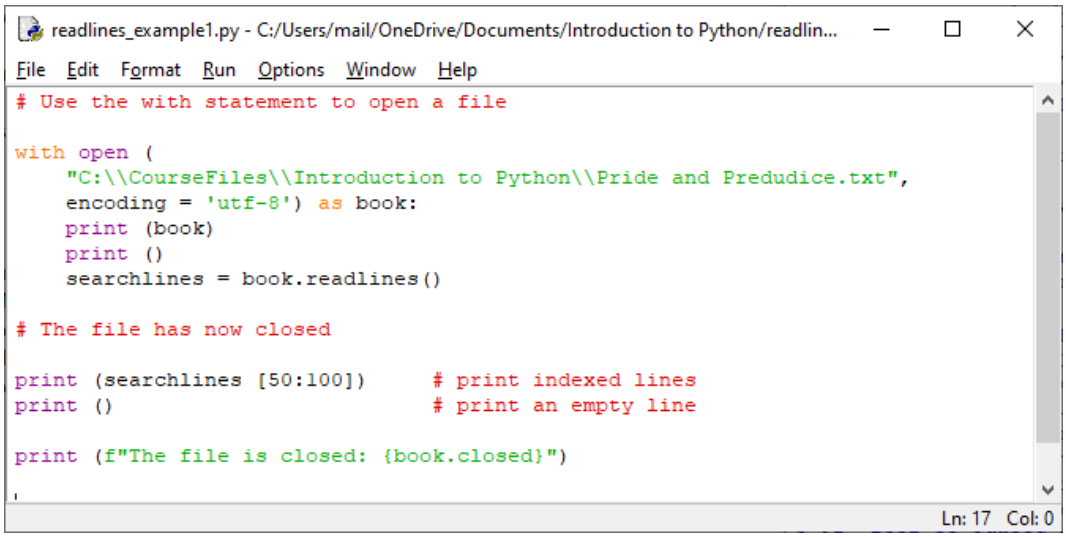
Task: Read some lines from our file

In this task we will open the file temporarily, read the lines from it and then print output them.

Note how we use the **with** command to open the file.

1 Create a new file **File** – **New File**

2 Input the following code:



3 Save the file as **readlines_example.py**

4 Run the script and observe the result.

Note: The .**readlines()** method in this example creates a list with each line of the file enclosed within. The **With** statement opens the file just long enough so that we can grab the contents and create our list.

We use indexing on the **searchlines** list to display the lines that we wish (The same way that we can with any other python **list**)

Another thing to note is the additional argument **encoding= 'utf-8'** that was used with the **Open** function. Try removing this and run the script again, depending on the computer you are using you may find that it does not run.

When we open text files Python 3 needs to understand how that file has been encoded. It will look at the operating system you are using (i.e. Windows, MacOS, Linux) and assume that the file will be encoded according to the default. For Windows that is **cp1252** you may have noticed that mentioned when you looked at the output of the previous opening_files_example2.py script

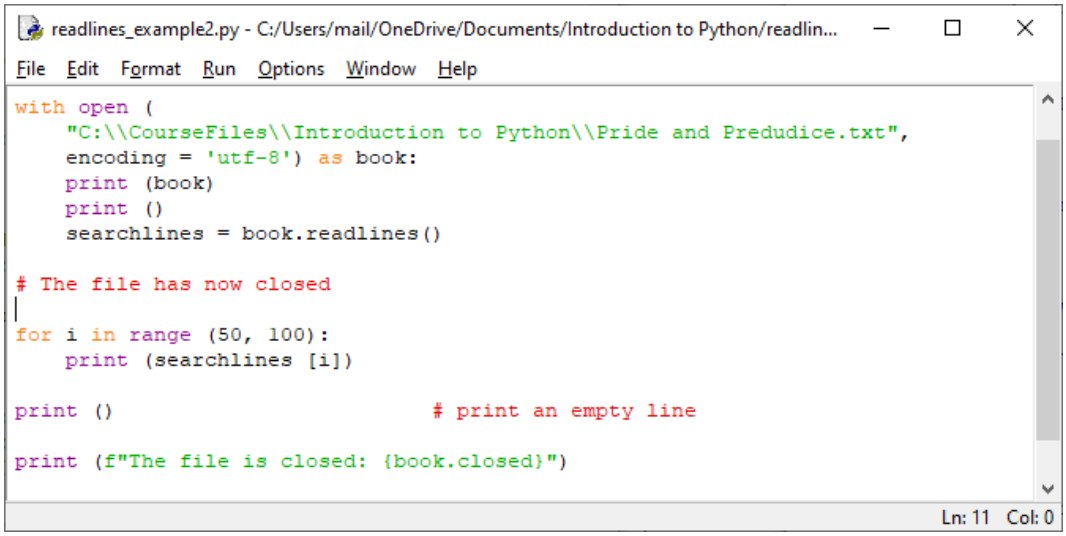
Sometimes the file will have come from somewhere else and/or may be encoded differently. Adding **encoding= 'utf-8'** to our open function arguments instructs Python 3 that this particular file uses that encoding.

Additional Task: Use a for loop to output the list

With the previous file try the following:

5 Use the **Save As…** command to save the file from the last task as **readlines_example2.py**

1 Adjust the script to look like the following:



2 **Save** the file

3 Run the script and observe the output

In this example, the output looks better as each line in the file is being output separately. You probably will notice that there are some extra characters being printed out from the file.

c. Writing to a File

Before we can write to a file using python we must open the file in the correct mode. In our previous examples we opened our files in read mode (this is the default mode used when you use the open command). To open a file in write mode we use the following.

my_file = open("C:\\coursefiles\Introduction to Python\\my_file.txt", "w")

Notice the **"w"** argument at the end of the open function.

We can set the following modes:

'r' – Read mode which is used when the file is only being read

'w' – Write mode which is used to edit and write new information to the file.

'a' – Appending mode, which is used to add new data to the end of the file; that is new information is automatically amended to the

'r+' – Special read and write mode, which is used to handle both actions when working with a file

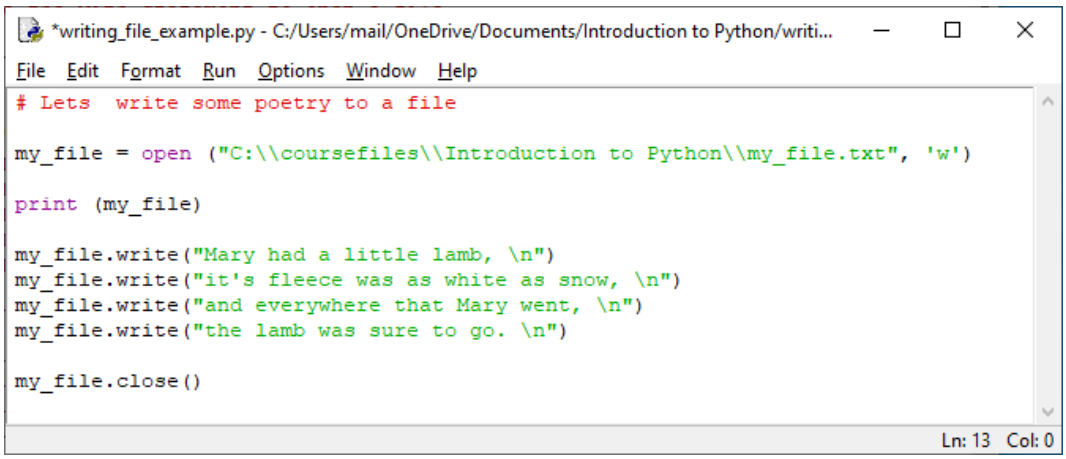
To write content we use the **.write()** method.

Note: **We must also remember to close our file when we are finished!**

Task: Write to a file

1 Choose **File** then **New**

2 Input the following code:



3 Save the file as **writing_file_example.py**

4 Run the script and observe the output.

5 From your windows explorer, navigate to the practice folder and look for **my_file.txt**

6 Open the file and review your results.

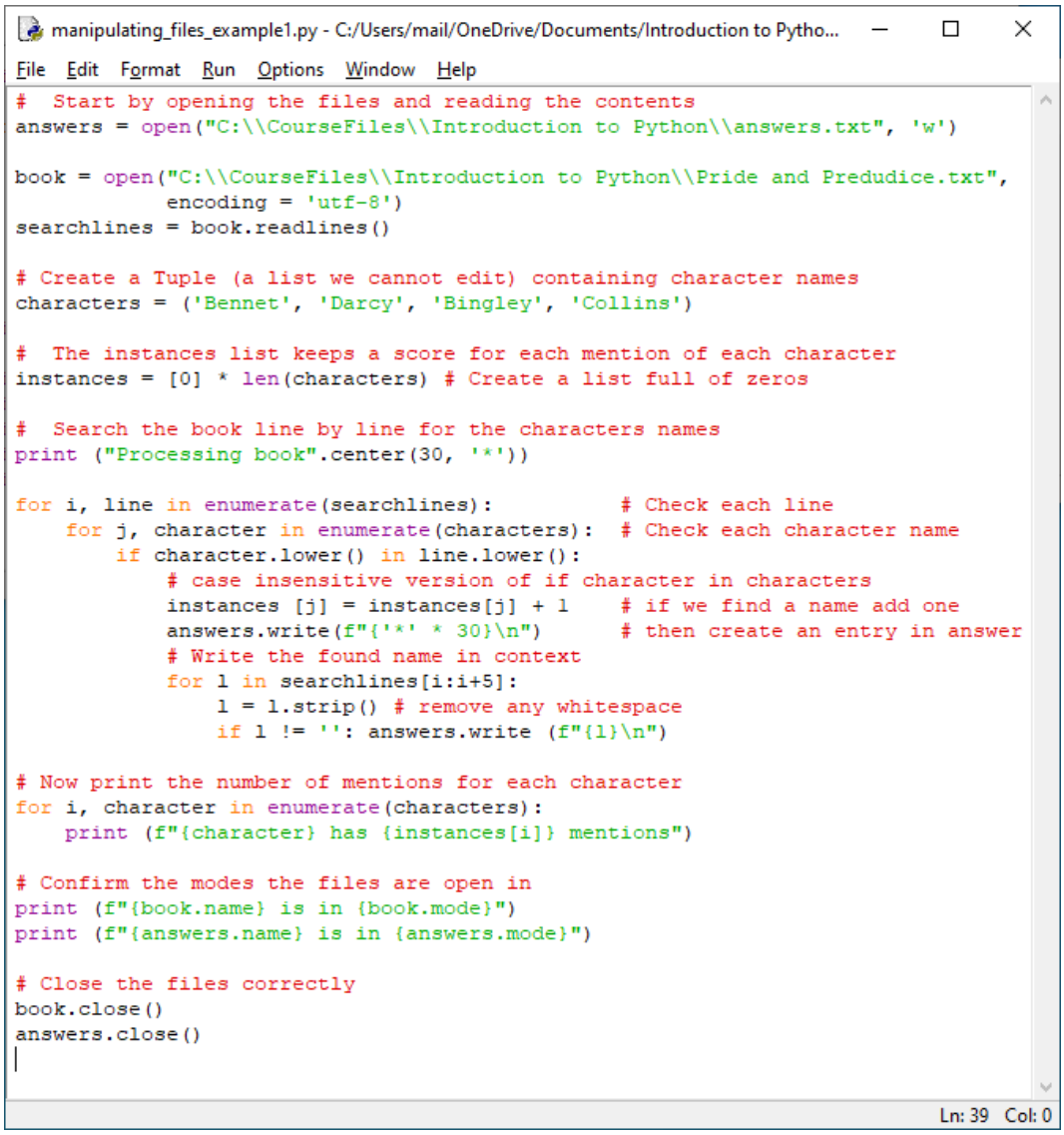
Note: The first time you run this script the file is created (if it does not already exist)

Each time you run this script the contents of the file are replaced. Use the **"a"** argument if you need to append data to an existing file.

Additional Task: Reading and Writing to Files

1 Create a new file

2 Write the following:



3 Save the file as **manipulating_files_example.py**

4 Run the script and observe the results.

5 Inspect the answers file created in your practice files folder.

Take some tine time to make sure that you understand the script you have just written. If there are any sections of code that you do not understand, try changing them or use the # to turn the code to comments and see what breaks!

d. CSV: Comma Separated Values.

CSV files are a (primitive but standard though un-standardised) data storage format. Each line of a CSV file is a number of fields separated by a separator like: comma, tab etc. By convention each row, has fields in the same order. We can access CSV files reasonably easily in Python by breaking each line for the file into separate fields using the split function.

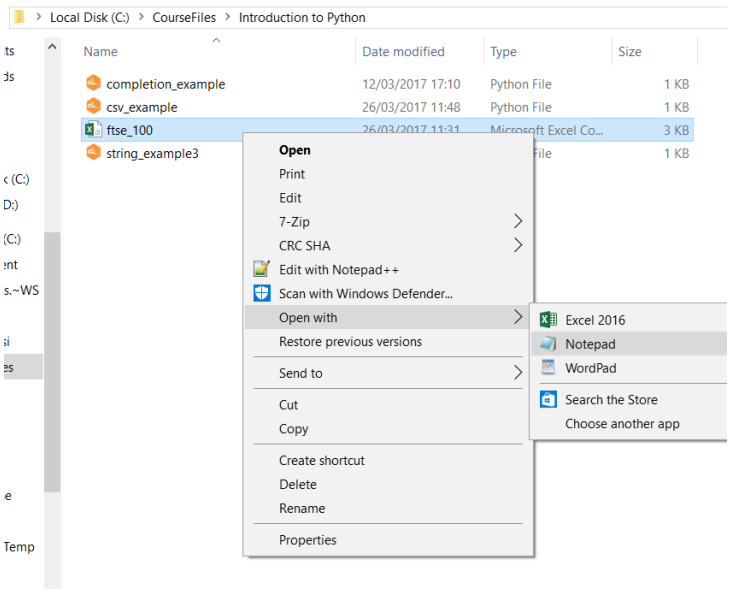
Accessing CSV Files:

- Open a file, read in the lines, use Split on the lines for a given delimiter. You've done it.
- Check first what the delimiter is, don't just assume there is one!
- Use that delimiter in **.split()**
- Do be aware of what data you expect in your columns.

We have not yet covered using modules in Python, but there is a module named csv that automates some of the following task (We will look at modules later)

Exercise: Read a CSV File

1 Navigate to your practice folder and right click on the file named **ftse_100.csv**



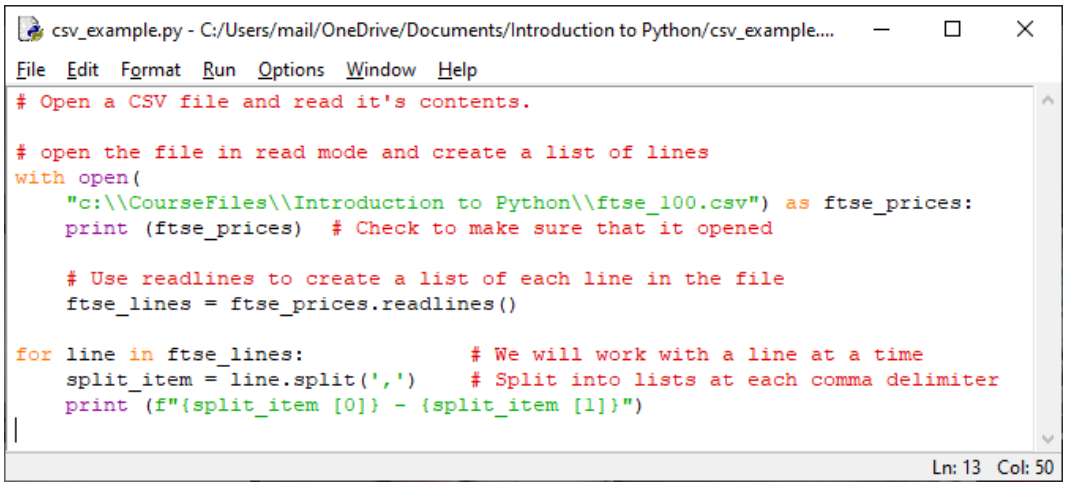
2 Choose **Open with**, and then **Notepad**

3 Observe the file, and take a note of the delimiter used (comma)

4 Close the file

5 Create a new file

6 Input the following code:



7 Save the file as **csv_example1.py**

8 Run the script and observe the output

Note: We read this file just like any other text files using .**readlines()** to create a list of each line in the file. The **.split()** command then broke this list down into smaller lists each time it found a comma.

Additional Task

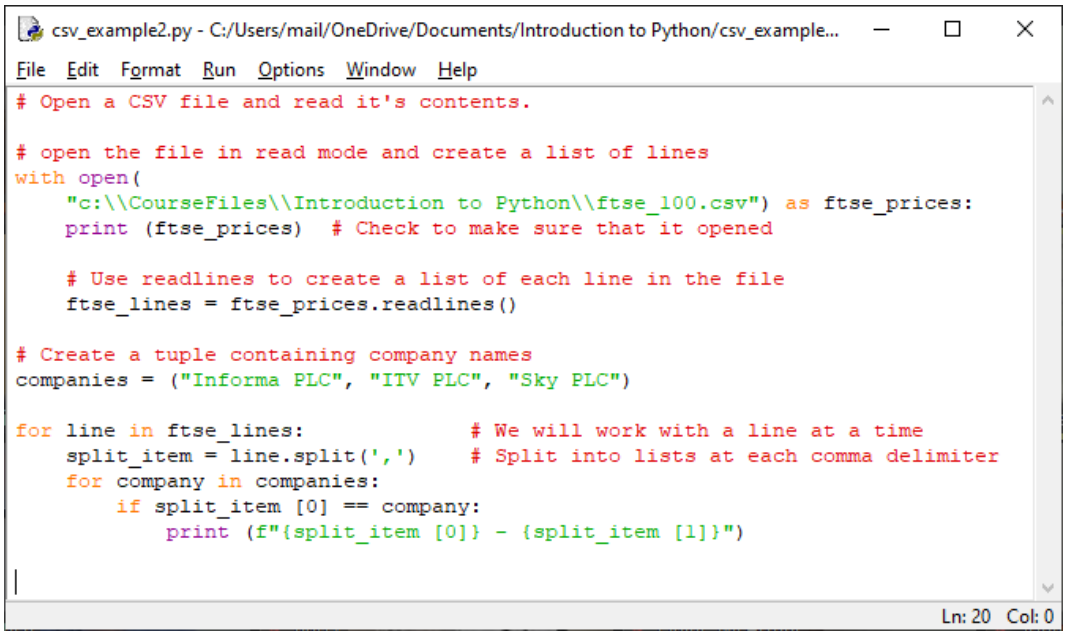
Using the **csv_example1** file from the last task:

1 Use the **Save As…** command to save the file from the last task as **csv_example2.py**

2 Create a tuple

companies = (“Informa PLC”, “ITV PLC”, “Sky PLC”)

3 Modify the for loop so that Python only outputs the prices for items in the companies tuple



4 Save and run the file

2 User Defined Functions

When producing any code we frequently find ourselves wishing to do the same thing over and over again. In Python we can use functions and methods to do this. The functions that we have used so far in this course have been ones that are already inbuilt to the Python programming language.

We can also define our own functions and use them in our Python code. By doing so we can use our code more efficiently, avoiding repetition in the code.

To define a function we use the following syntax:

```
def functionname( parameters ):
   "function_docstring"
   your code
   return [expression]
```

- The keyword **def** introduces the function definition, the **functionname** is what we will call our function. Then in parenthesis we have the arguments (if there are any) of the function
- The function **docstring** (in speech marks) is where we add a description of the function for documentation standards. Although optional it is important that we do this.
- We then write the code that makes up the function
- Finally, we can complete the function with a return statement. This allows us to return the results of the function when we use it. This is optional depending on what you are trying to achieve.

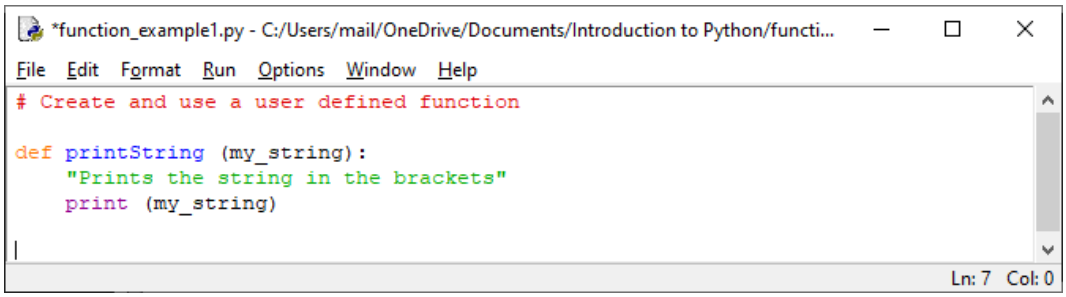
Notice like all of our code how important indents are when we are working with functions.

Let's create a function and use it.

Task: Create a user defined function

1 Create a new file using IDLE

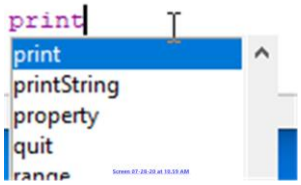
2 Input the following code:



3 Save the script as **function_example1.py**

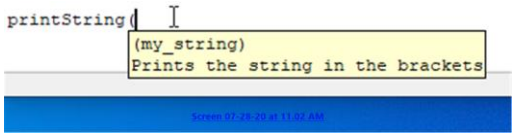
4 Run the script and observe the output. (spoiler alert! Nothing much happens)

5 At the bottom of your script type **pri** and then **TAB**



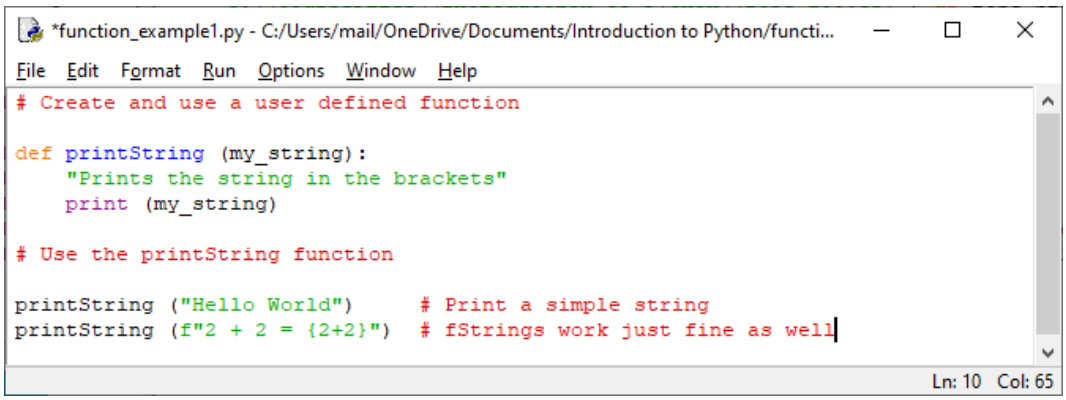
6 Use the down arrow to select your function and then press **TAB**

7 Type **(**



Note: when you were using the **printString** function, IDLE included your **docstring** "Prints the string in brackets" in the yellow box

8 Add more code until your script looks like this

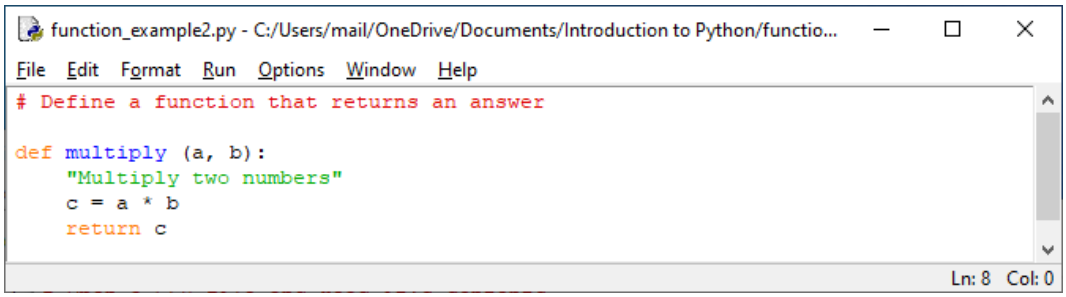


9 Close the file

Additional Task: Return an answer in your function

1 Create a new File

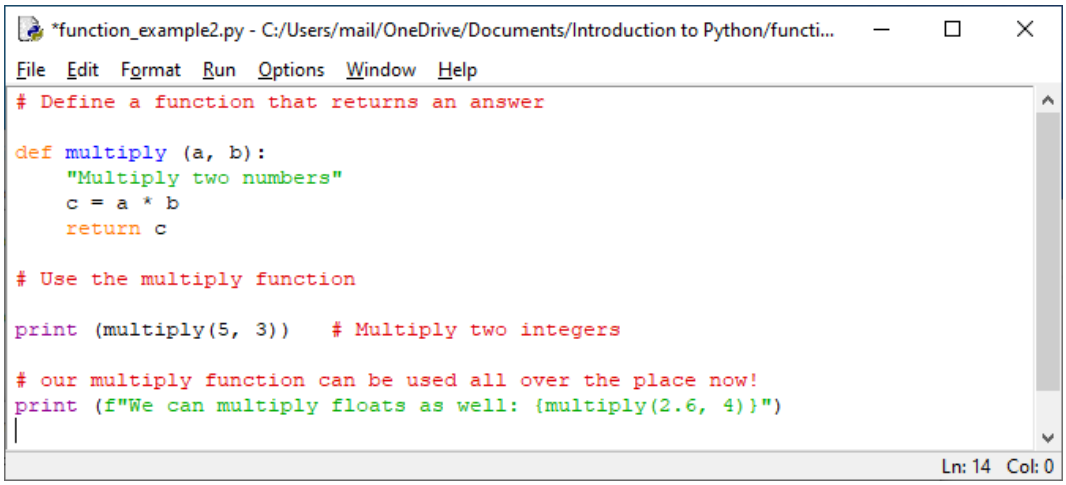
2 Input the following code:



3 Save the file as function_example2.py

4 Run the script and observe the output.

5 Add more code until your script looks like this:



6 Run the script and observe the output.

7 Close the file

Our user defined functions were rather simple and are quite basic examples of what we can do with user defined functions. We have only scratched the surface of what is possible using user defined functions.

If you want to learn more, visit

https://docs.python.org/2/tutorial/controlflow.html#defining-functions

3 Modules

In our last section we saw that you can add user defined functions to Python using **def**. Another way of extending the capabilities of Python is to import modules into your scripts. Modules are files that contain Function definitions, variables, constants and more.

Modules can be imported from various sources. Python when is installed on your computer with a library of **Standard Modules**. These offer a lots of additional functions to your scripts. As well as **Standard Modules**, there are also other modules that can be installed on your computer (usually via the web). These are installed via a program or are downloaded with their own self-installers.

You can also write your own modules and import functions from them in your scripts. This allows you to break your code up into separate files to organise complex programs.

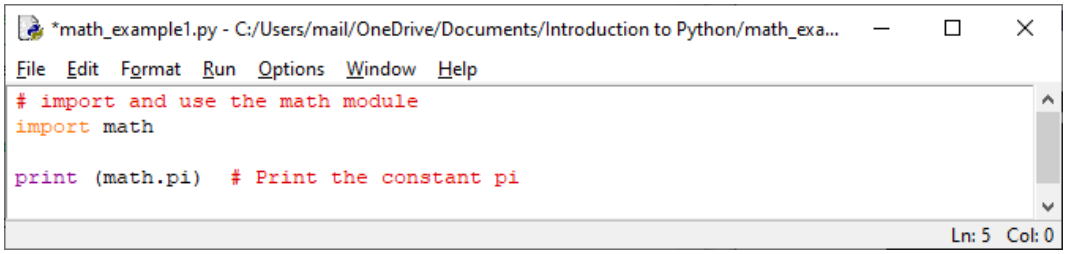
a. Import and use the Math Module

One of the many **Standard Modules** is called math. It extends the mathematical capabilities of the Python programming language giving us access to functions that will calculate Sin, Tan etc., Think about the buttons that you find on a mathematical calculator, the math library includes functions that do more or less the same things.

Task: Import the Math Library

1 Create a new file

2 Input the following code:



3 Save the file as **math_example.py**

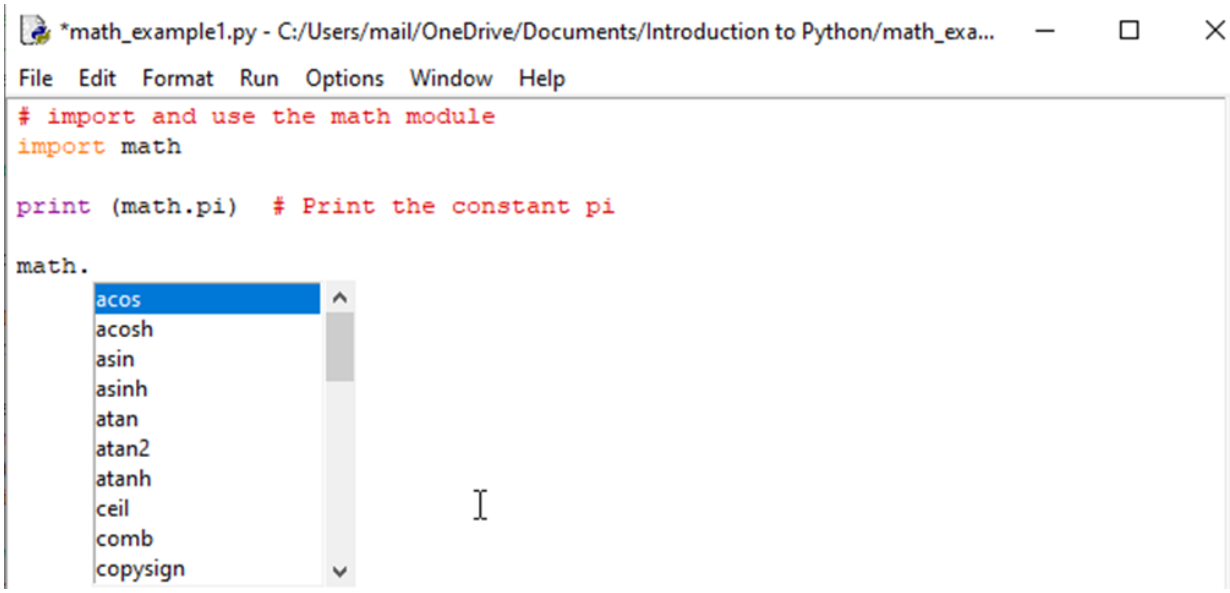
4 Run the code and observe the result.

5 At the end of the script type the following:

math.

6 Click **TAB** or **CTRL + SPACE**

7



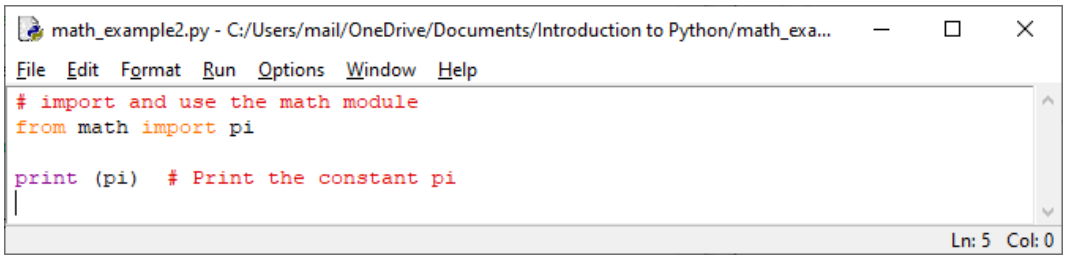
8 From the autocomplete drop down box use the **up** and **down** arrows to examine the functions and constants that are now available from the math module.

Additional Task: Import a single function from the Math Module

1 With the math_example.py file

2 Use the **Save As…** command to save the file from the last task as **math_example2.py**

3 re-write the code to read:

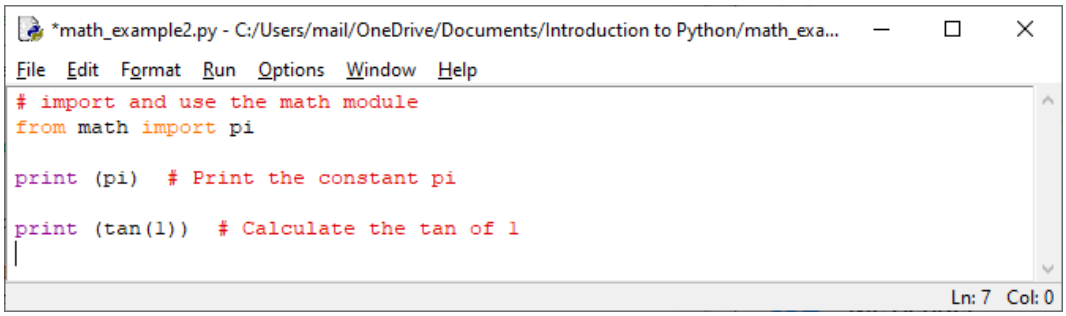


4 **Save** and **Run** the script

5 Observe the results

The code should run exactly the same as it did in the previous task.

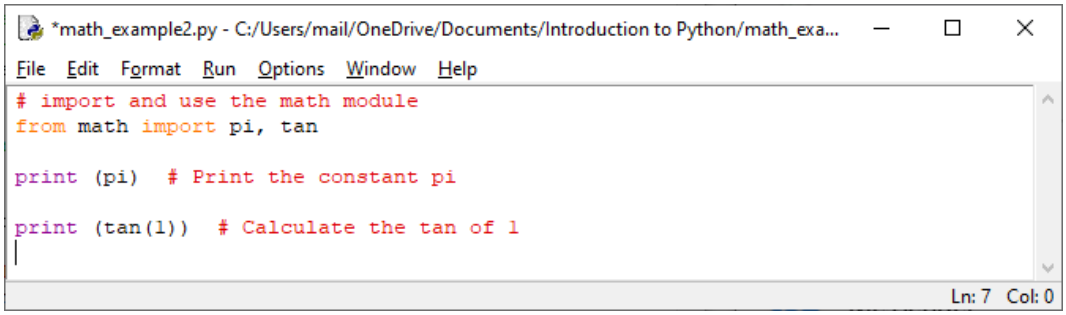
6 Add the following to the code:



7 Save and run the code again. You should receive an error message

When we use the **from/import** commands to add a function, we do not add the full module to our code.

8 Modify the code to add tan to the import statement:



9 Save and run the code

this time your script should run without error.

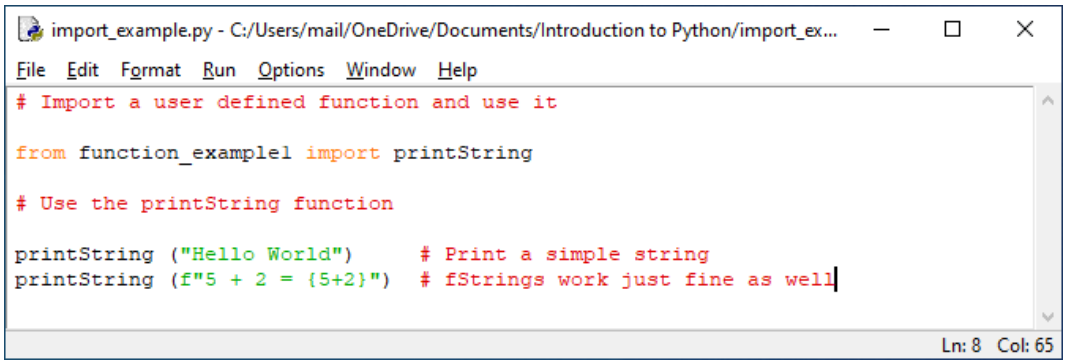
b. Create and use your own module

Creating a module within Python is actually very easy. All you need to do is to create a file with a python extension that contains functions. We have already done this when we created the function_example1.py script.

Task: Import the printString Function

1 Create a new file

2 Input the following code:

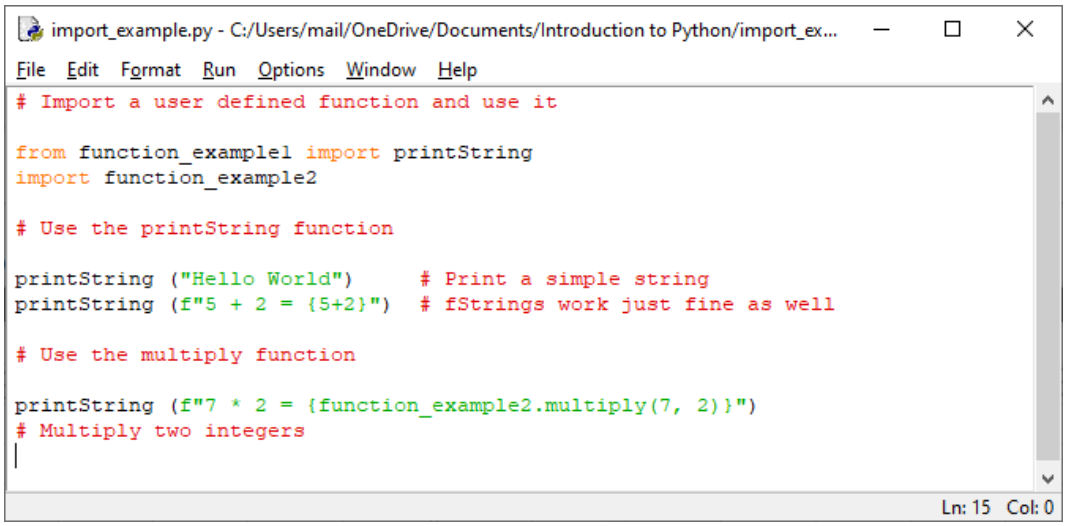


3 Save the script as import_example

4 Run the script

Note: Have a careful look at the output for this script. What do you notice? What does this tell you about what the import statement actually did? If you need to reopen the function_example1.py script and have a look at it

5 Modify the script to look like this:



6 Save and run the script.

Note Again we are seeing code from the imported being run at the point where you import the file. This is perhaps not the most useful thing to happen in this example. How would you go about preventing this?

Additional Task: Tidy Up the Imported Scripts

To make the functions in the function_example files more useful go back through and tidy them up so that they only contain functions. You do not have to delete any code you could simply use # to turn lines of code into comments (sometimes called “commenting out” code)

Alternatively you could create a new python3 file that contains both functions and use that instead.

c. Import and use the Request module

Previously we used the math module, one of the inbuilt modules that is supplied with Python. Often when we are working with Python we will want to use code that others have developed that is not part of the standard library of modules. Requests is such a module. Python contains two modules by default that can access web pages (urllib and urllib2), but the more recent requests library offers an improved set of tools for manipulating web pages.

However before we can use requests me must install it.

There are various ways to install libraries to your python environment and each library that you find and want to use will have its own documentation. Some may have their own installable package, others will rely on Python's inbuilt installer **PIP** (an acronym for **P**ip **I**nstalls **P**ython or **P**IP **I**nstalls **P**ackages)

Task: Install Requests

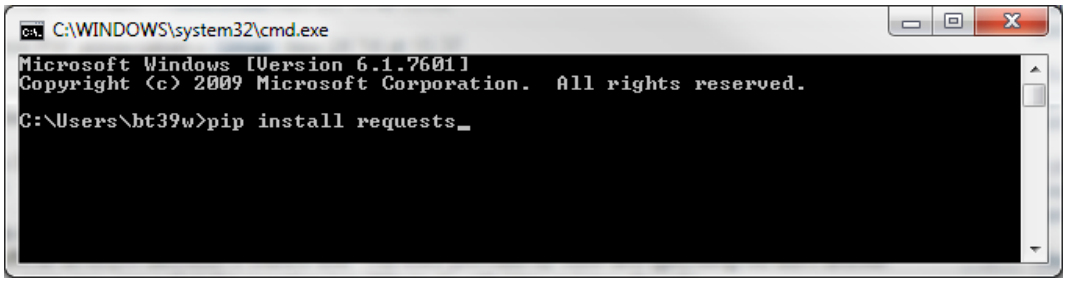
Depending on how your machine is setup this and the next task may not work properly. If not please study the code and ensure that you are confident on how it is supposed to work. You can try this on a different environment later.

1 Click on the start menu and type **cmd.exe**

2 Hit **ENTER**

The command window will appear

3 Type **pip install requests**

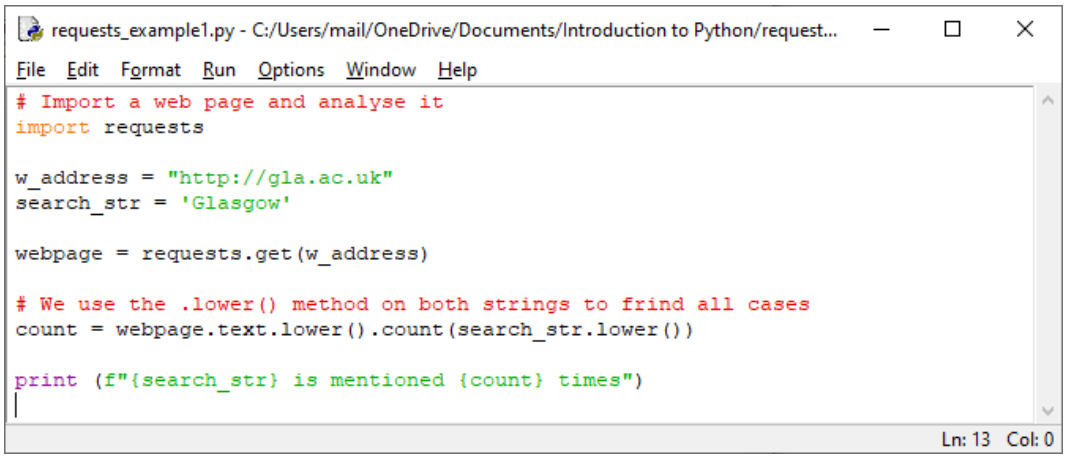


4 After a brief installation, the requests library will be installed. If you receive an error message at this stage try typing **python -m pip install requests**

Task: Use the requests module

1 Open IDLE and create a New file

2 Type in the following code:

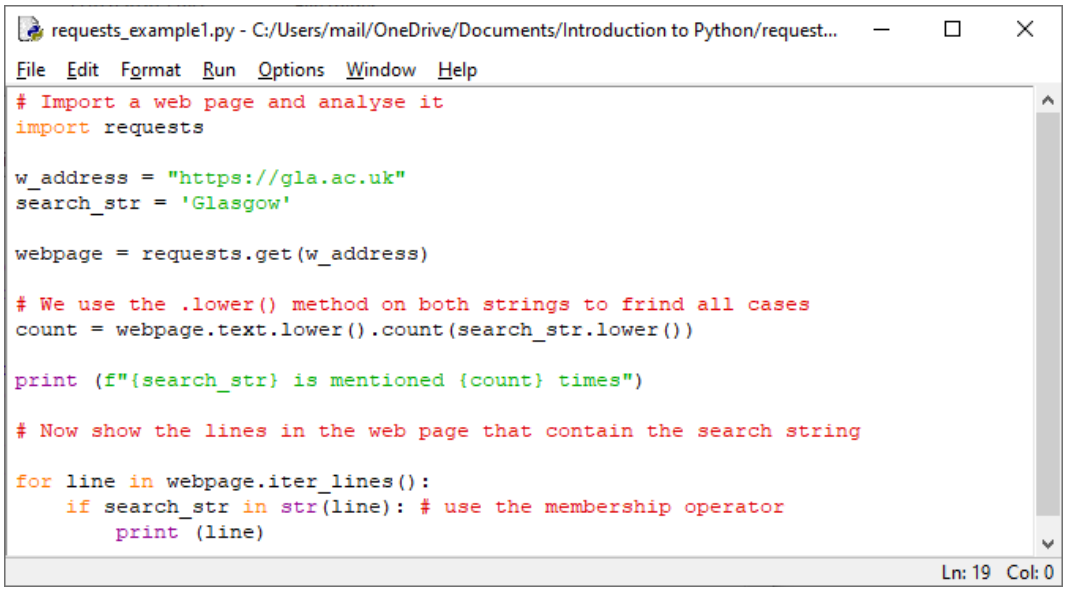


3 Save the script **as requests_example1.py**

4 Run the script and observe the result

The code should run and you will find out how many times the word Glasgow is mentioned on the web page.

5 Modify the code:



6 Save and run the script, observe the result.

Hopefully your code will run without errors. To learn more about the requests library visit:

http://docs.python-requests.org/en/master/

Using other people's libraries, you can vastly extend the capabilities of your Python scripts. For each library that you wish to use you need to read the documentation to work out how exactly to use the code. You can also use search engines to find help in solving your programming problems.

Additional Task: Add User Input to Our Website Analyser

If you have time, modify requests_example.py so that you can analyse any user inputted website searching for any user inputted string. Have a think about how users are used to entering web addresses into browser address bars and see if you can make your code tolerant of commonly inputted addresses. (i.e. **www.gla.ac.uk** vs **gla.ac.uk**)

4 Next Steps in Python

We hope that you have found the Introduction to Python course both useful and enjoyable. Like all introductions to a subject, the course can only introduce you to some of the concepts of programming in Python. Your learning starts here.

a. Find a Project

The most important next step that you can take is finding a reason to practice your skills and learn new ones. One of the best steps you can take now is to find and start coding your own Python script/program/project.

To complete your first Python coding project you will need to learn about more functions, modules, libraries and programming techniques. Thankfully, Python is one of the best-documented languages with plenty of books, websites and open sourced projects available for study.

Try and choose a project that:

- Solves a problem that you have experienced
- Interests you
- Is not too complex (you can build on complexity as your skills grow)

When you start writing your code:

- Document, document and document some more. Comments are critical to understanding and learning (Google Rubber duck debugging to see why!)
- Save and run the code frequently. Simple mistakes are often revealed when you test incremental changes to your code.
- Use tools such as code completion as much as you can. Typos and badly remembered commands (as you probably found out during the course) cause problems.
- Read the documentation carefully.
- Always look out for code examples that others have written.
- Your search engine is your best friend

b. Online Resources

- docs.python.org/2/
- learnpythonthehardway.org/
- https://www.cs.hmc.edu/csforall/
- http://opentechschool.github.io/python-beginners/en/index.html
- http://anandology.com/python-practice-book/index.html
- http://gawron.sdsu.edu/ Python for Social Scientists
- http://pbpython.com/ Practical Business Python

Appendix 1 Python Built in Functions

Function	Description
abs()	Return the absolute value of a number.
all()	Return `True` if all elements of the iterable are true (or if the iterable is empty).
any()	Return `True` if any element of the iterable is true. If the iterable is empty, return `False.`
ascii()	Return a string containing a printable representation of an object, but escape the non-ASCII characters.
bin()	Convert an integer number to a binary string.
bool()	Convert a value to a Boolean.
bytearray()	Return a new array of bytes.
bytes()	Return a new "bytes" object.
callable()	Return `True` if the object argument appears callable, `False` if not.
chr()	Return the string representing a character.
classmethod()	Return a class method for the function.
compile()	Compile the source into a code or AST object.
complex()	Create a complex number or convert a string or number to a complex number.
delattr()	Deletes the named attribute of an object.

Function	Description
dict()	Create a new dictionary.
dir()	Return the list of names in the current local scope.
divmod()	Return a pair of numbers consisting of quotient and remainder when using integer division.
enumerate()	Return an enumerate object.
eval()	The argument is parsed and evaluated as a Python expression.
exec()	Dynamic execution of Python code.
filter()	Construct an iterator from elements of iterable for which function returns true.
float()	Convert a string or a number to floating point.
format()	Convert a value to a "formatted" representation.
frozenset()	Return a new `frozenset` object.
getattr()	Return the value of the named attribute of an object.
globals()	Return a dictionary representing the current global symbol table.
hasattr()	Return `True` if the name is one of the object's attributes.
hash()	Return the hash value of the object.
help()	Invoke the built-in help system.
hex()	Convert an integer number to a hexadecimal string.

Function	Description
id()	Return the "identity" of an object.
input()	Reads a line from input, converts it to a string (stripping a trailing newline), and returns that.
int()	Convert a number or string to an integer.
isinstance()	Return `True` if the object argument is an instance.
issubclass()	Return `True` if class is a subclass.
iter()	Return an iterator object.
len()	Return the length (the number of items) of an object.
list()	Return a list.
locals()	Update and return a dictionary representing the current local symbol table.
map()	Return an iterator that applies function to every item of iterable, yielding the results.
max()	Return the largest item in an iterable.
memoryview()	Return a "memory view" object created from the given argument.
min()	Return the smallest item in an iterable.
next()	Retrieve the next item from the iterator.
object()	Return a new featureless object.

Function	Description
oct()	Convert an integer number to an octal string.
open()	Open file and return a corresponding file object.
ord()	Return an integer representing the Unicode.
pow()	Return power raised to a number.
print()	Print objects to the stream.
property()	Return a property attribute.
range()	Return an iterable sequence.
repr()	Return a string containing a printable representation of an object.
reversed()	Return a reverse iterator.
round()	Return the rounded floating point value.
set()	Return a new set object.
setattr()	Assigns the value to the attribute.
slice()	Return a slice object.
sorted()	Return a new sorted list.
staticmethod()	Return a static method for function.
str()	Return a str version of object.

Function	Description
sum()	Sums the items of an iterable from left to right and returns the total.
super()	Return a proxy object that delegates method calls to a parent or sibling class.
tuple()	Return a tuple
type()	Return the type of an object.
vars()	Return the `__dict__` attribute for a module, class, instance, or any other object.
zip()	Make an iterator that aggregates elements from each of the iterables.
__import__()	This function is invoked by the `import` statement.

Appendix 2 DOS Commands

Command	Function	Example
cd	Change Directory	cd.. go back a directory cd c:\perl\myscripts\
Mkdir	Make a new directory in the current folder	“Mkdir newfolder”
Copyfile	copies file1.pl to file2.pl	“Copy file1.pl file2.pl”
Del	deletes file1.pl – be careful!	Del file1.pl
Doskey	starts remembering commands.	
Exit	: closes command prompt	

Useful Shortcut keys

Using keyboard shortcuts can help you become more efficient when creating documents in Microsoft applications. Most keyboard shortcuts require you to use two or more keys at the same time. To use a keyboard shortcut first press and hold down the modifier key or keys (i.e. SHIFT, CTRL, ALT) and then press the corresponding standard key on your keyboard.

Function	Shortcut
Save and run your script (in IDLE)	F5
Open	CTRL+O
Save	CTRL+S
Close	ALT + F4
Cut	CTRL+X
Copy	CTRL+C
Paste	CTRL+V
Select all	CTRL+A
Indent line	CTRL+I or Tab
Cancel	Esc
Undo	CTRL+Z
Re-do	CTRL+SHIFT+Z
Find	CTRL+F
Replace	CTRL+H
Show Completions	CTRL+SPACE