Introduction to Python 3 Programming – Session 1

V 0.1

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 26/04/21

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 1

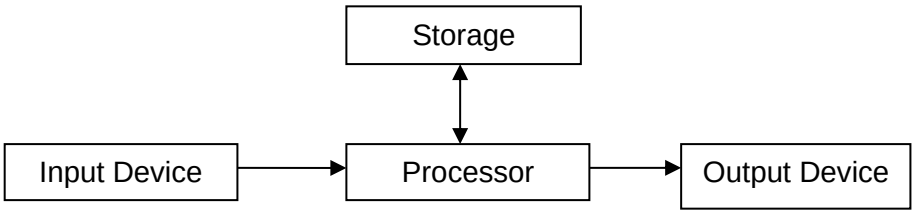
Session Objectives

At the end of this session you should 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
- Use arithmetic operators in Python
- Use If statements

1 Introduction

A computer is basically a simple machine that can perform a range of operations very quickly. The computer is essentially a stupid machine – you need to tell it exactly what to do. A computer program is a set of instructions for the computer.



A Simple Diagram of a Computer

The computer has a Central Processor (in the box), an Input Device (keyboard, mouse), an Output Device (screen, printer) and storage (Memory, disk drives etc).

In this course we will write programs using Python. Python makes it easy to write simple programs and can be used to write complicated programs. Python scripts (programs) are relatively easy to understand.

2 Write Your First Program/Script

First a little terminology, you will hear people calling python code by more than one name, some call the code programs and others call them script. For our purposes it does not matter, either term is acceptable. The word script is often used to describe code that needs to have an interpreter installed to be able to run, this is because the code is not "compiled" (turned into working code the computer can run) until the point in time that you run it. Python can work in this way, and so in this manual we will refer to our code as scripts.

Python can also be used to create pre-compiled code as well, it is a very flexible language, but we will worry about that further down the line.

a. Using IDLE

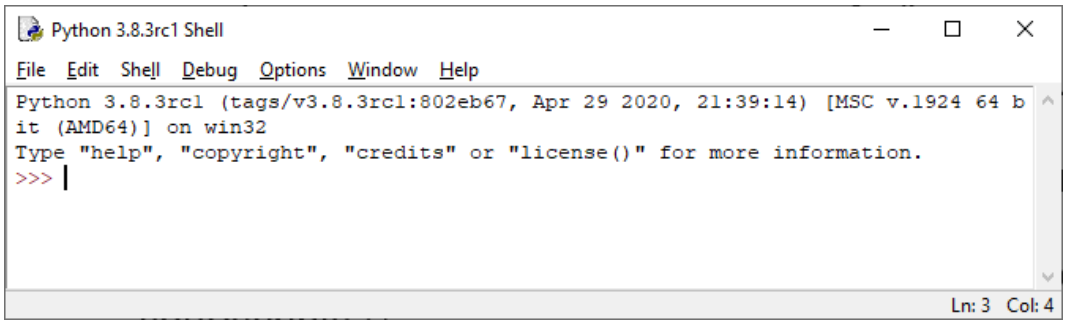
Python scripts are written in text files that can be easily written and edited using any text editor on just about any computer. So you could use notepad to create and edit scripts if you wanted to.

However during this course, for writing scripts we will use a special text processor called **IDLE** (**I**ntegrated **D**eve**L**opment **E**nvironment or **I**ntegrated **D**evelopment and **L**earning **E**nvironment). This is just like Notepad but makes it easier to lay out a script and offers tools that allow us to run our python scripts as well.

IDLE has two main window types, the **Shell** window and the **Editor** window. It is possible to have multiple editor windows simultaneously.

1 In the Start menu select, Python 3.8, IDLE
(The names will be similar to above, but don't worry if the numbers are a little different)

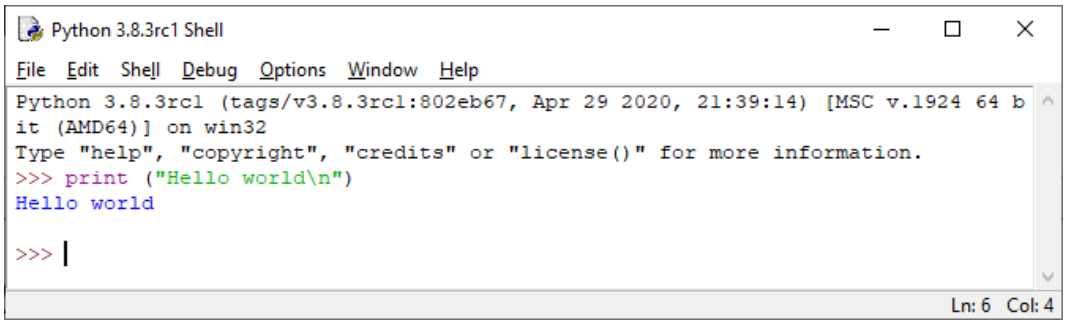
The **IDLE** program will start. **IDLE** opens an **interactive Shell window**. The interactive Shell window allows you to run Python commands live on your computer. It is very useful for trying out pieces of code that you intend to run within your scripts.



2 Try typing the following:

```
print (“hello world\n”)
```

3 Press Enter on your keyboard



You should see your first piece of code run within the interactive Shell! The shell is

OK for trying out snippets of code, but if you want to build a more substantial script, you will want to create your own file.

4 From the **File** menu select **New File**.

A new window will open, this is the **Editor** window. The editor window allows you to write your module and then to run the code.

5 Enter the following lines of code in text editor that appears:

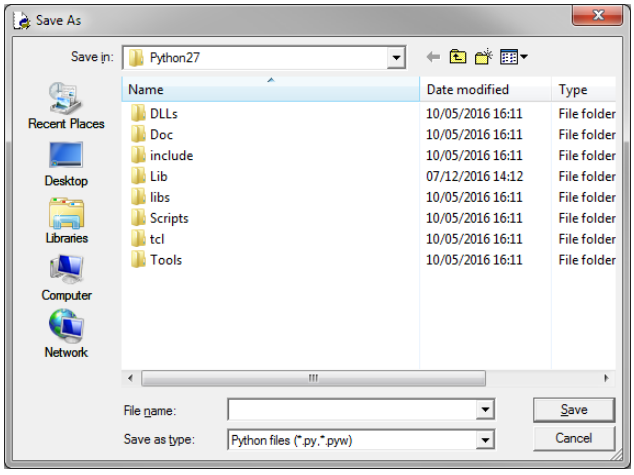
```
# First Script
print ("Hello World\n")
print “This is my first Python script”
```

b. Saving Your Program

Before we can run our first Python script we must first save it.

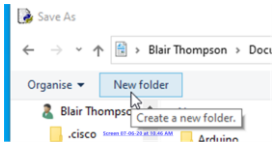
1 Click **File** then **Save**

2 The **Save As** dialogue box will appear

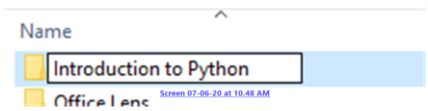


3 Navigate to where you wish to save your files during this course, If you are learning in our classroom **C:\coursefiles** is a good location, at home you might prefer to place the files in your **Documents** folder

4 Use the **New Folder** tool to create a new folder



5 The folder will be created and you will be prompted to name it. Type **Introduction to Python** then **Enter**



6 Double click the folder that you have just created

7 In the **File name:** box type helloWorld

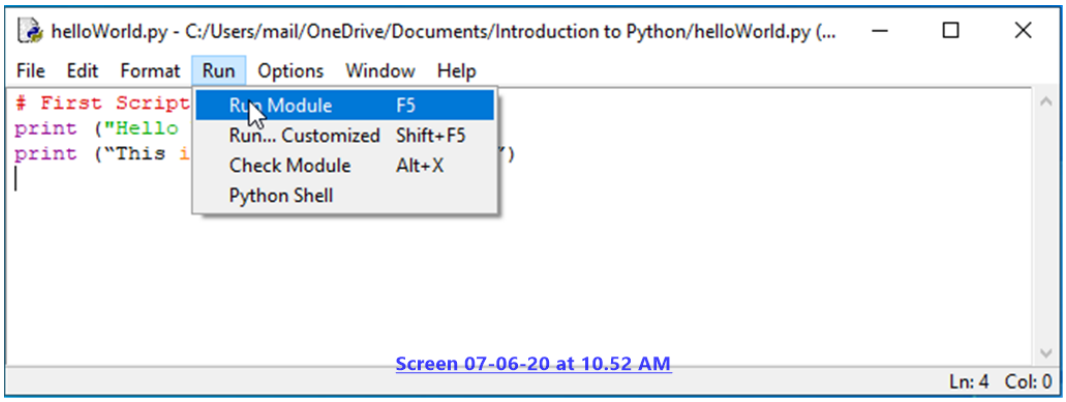
8 Check that **Save as type** is set to Python files

9 Click **Save**

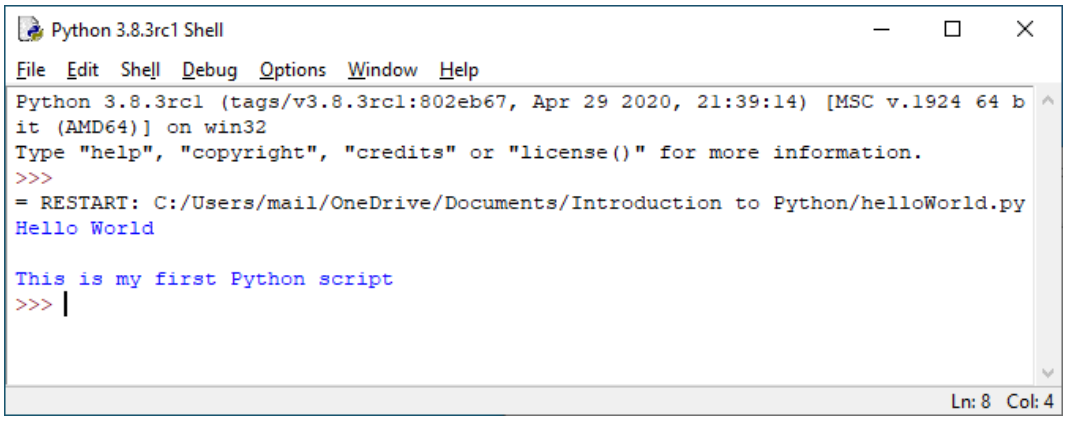
c. Run Your Program

You can run a python program in a number of ways. If you are editing it using IDLE, the simplest way to run your program is via the run command.

1 Click **Run** then **Run Module**



2 The interactive shell window will display the output from your code.

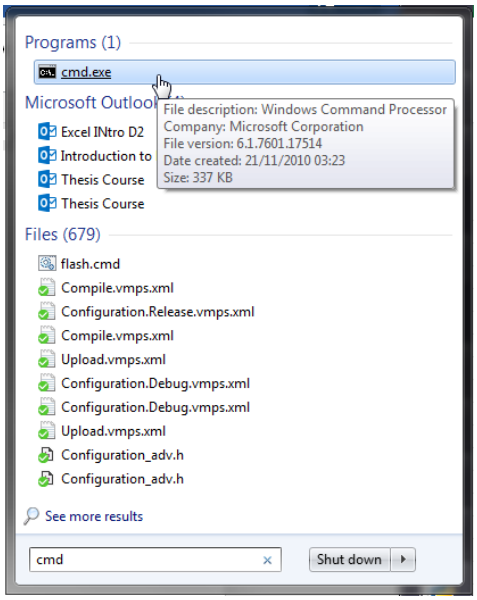


3 Congratulations. You have just run a script.

4 You may also run your python code using the command line:

5 Click the windows button on your task bar

6 Type “cmd”



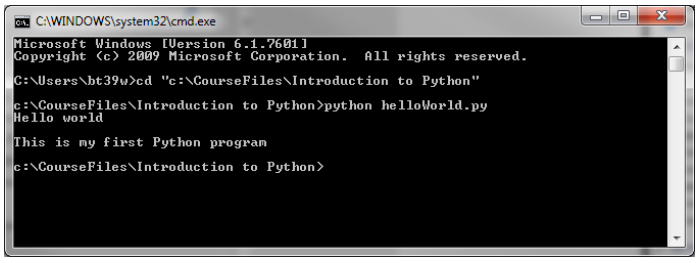
7 Hit **Enter** or click on **cmd.exe**

8 A command line window will appear

9 Type cd “c:\coursefiles\introduction to python” replacing the folder address with the address you saved to

10 Hit **Enter**

11 Type **python helloWorld.py**



Note: Depending how your computer has been set up you may be able to run the script omitting the word python at the beginning i.e. helloWorld.py

12 Close the Editor Window

d. Other IDEs (Integrated Development Environments)

For this course we have decided to use IDLE as our development environment. We chose IDLE because it is installed with Python and offers a convenient way to run your python scripts. The internet is full of opinions as to which is the best IDE to use when developing your python scripts, some offer very complex sets of tools to help you.

Once you are comfortable programming basic scripts using IDLE you can try other IDEs and find the right set of tools for yourself.

3 Comments

Most python scripts are a bit more complex than Hello World, so to make it easy to follow what the code is doing you can include comments in your code.

Any text that follows a # will be ignored by the computer. You can use this to insert comments to describe what the code should be doing. In our helloWorld.py script, the first line was a comment. We can also add comments in this way to the end of individual lines of code.

Now we will add a longer comment to our helloWorld.py program:

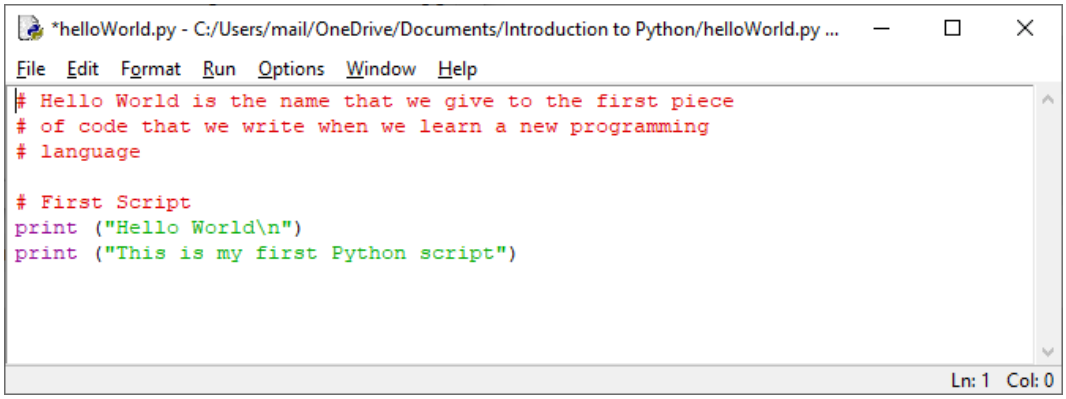
TASK: Adding Comments to a Python Script

1 Click back into your Editor window at the end of your code.

2 Type the following at the beginning of your code

3
```
# Hello World is the name that we give to the first piece
# of code that we write when we learn a new programming
# language
```

4 Notice the code highlights in red when we write our comment this way.



5 Run your script again.

6 Notice that the code runs exactly the same way as before. The new text (being a comment) is completely ignored by the python interpreter!

7 **Save** your Script (CTRL + S)

8 Close the editor window

It is good programming practice to comment clearly and to comment often. It helps others who are reviewing your code to understand what you are trying to accomplish. It is also helpful when you revisit your own code.

Comments are also very useful for disabling lines of code. By "commenting out" the line of code you prevent it from running but without deleting it. This can be handy when we are replacing lines of code (whilst keeping a record of what worked before) or when we are **debugging** (finding and eliminating problems in our code) and need to see if a particular line of code is causing an issue in a script.

4 Variables

A variable is a label for a location in the computer's store where a value can be stored and retrieved within our script. It can be treated like a mathematical variable. A variable can contain a number or a string of characters. Almost all python scripts will need to contain variables to help accomplish their purpose.

a. Assignment

An assignment statement is the way that we create variable in Python. A typical statement looks like this:

```
my_number = 4
```

The expression my my_number = 4 can be translated as… variable **my_number** will contain the value 4.

The expression today = "Monday" can be translated as… variable **today** will contain the string "Monday".

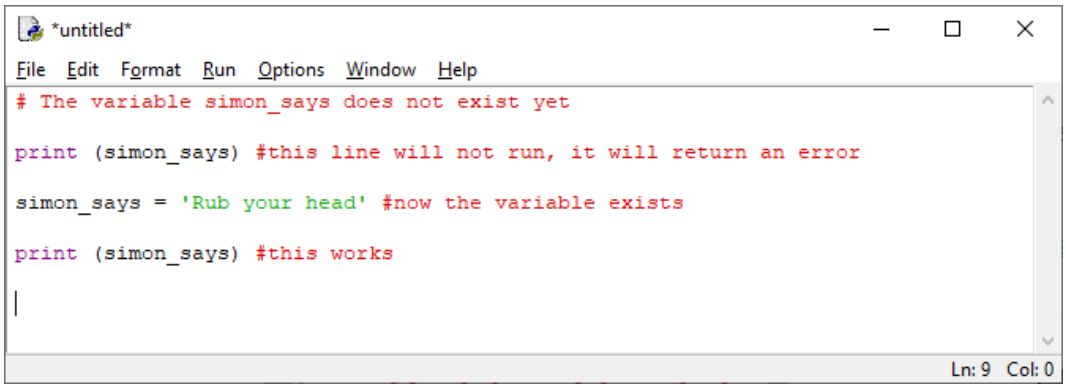
When writing python code we do not need to tell python in advance that we are creating a variable (you do in many other programming languages). It comes into existence at the point where we write the assignment statement. We also do not have to tell Python what type of data our variable contains (you do in many other programming languages), Python will work that out by looking at the value you are assigning.

Task: Assign a variable and print it

1 Click on **File** and then **New File** in any of the **IDLE** windows

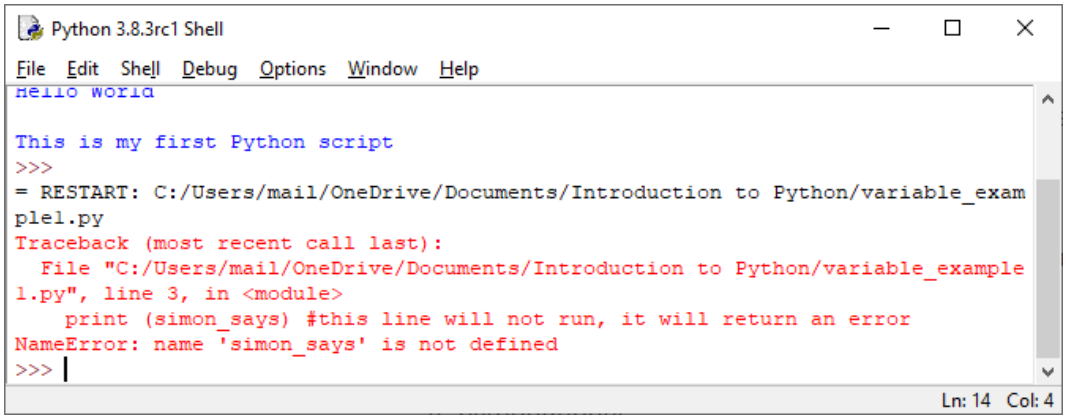
A new **Editor** window will appear

2 Enter the following:



3 Save the file as **variable_example1.py** in your practice folder

4 Run the script using the **Run Module** command (from the run menu)



IDLE will attempt to run the script, but will return a NameError, this is telling you that the variable that you tried to print is not defined (i.e. it does not exist yet)

5 Click back into your Editor window.

6 Add a # to the very beginning of the second line of code.
This will change it into a comment (This is often described as "commenting out a line of code")

7 Save your script (try using CTRL + S)

8 Run the script again using the **Run Module** command (try using the F5 key)

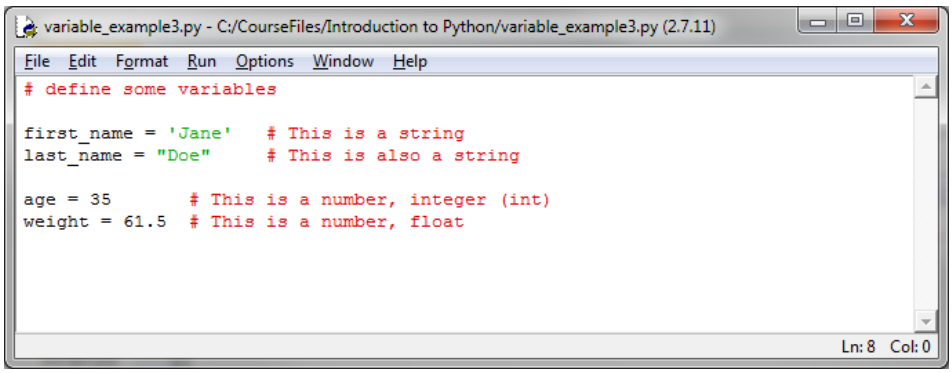
9 Close the Editor window

b. Strings

When we assign a value to our variable, if we enclose the value in quotation marks the Python interpreter will create a string. You can use either single quotes ('), double quotes (") or triple quotes (""") when assigning a string. We will not look too closely at the moment as to what is best here (they all work) but as we are going to be working with short strings in our examples, we will be using double quotes (").

c. Numbers

There are two kinds of numbers within the Python language, integers and floats. Integers are whole numbers and floats are numbers that contain a decimal point. When you assign a variable containing a number adding a decimal point is enough to create a float, i.e.

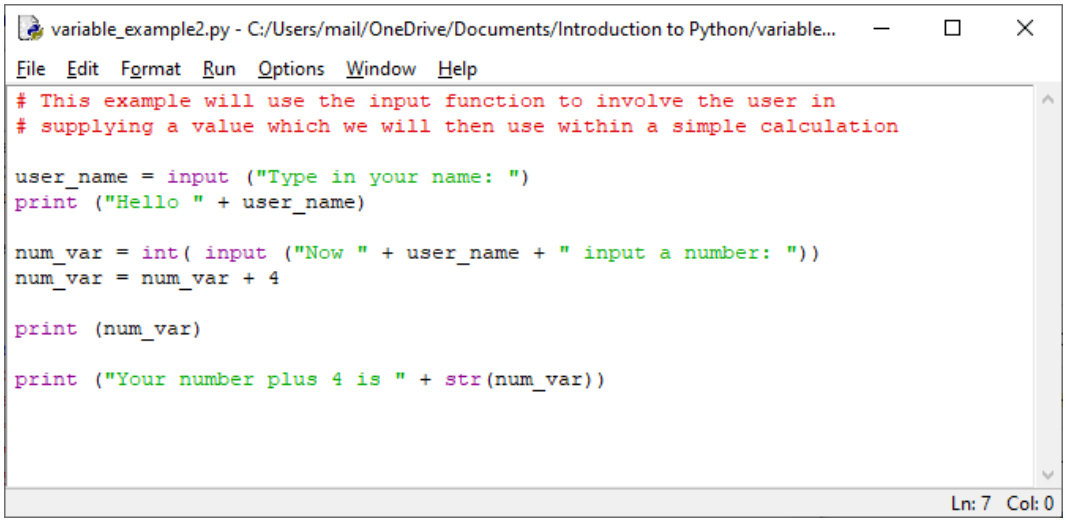


d. User Input

It is useful for us to be able to use user input to assign variables in Python. This allows us to create interactive scripts that are flexible. For instance we can request information from a user (e.g. a keyword to search for or a filename to process) within our script.

Task: Requesting Variables from the User

1 Enter the following:



2 From the **File** menu, select **Save As**

3 Save the file with the name **variable_example2.py** in your preferred folder

4 Run the script using the **Run Module** command (from the run menu)

In this task we use **input** function to assign our variables. Using the **input** command forces python to assign the variable **user_name** as a text string. The problem is that as a text string we cannot do any arithmetic on it…

5 Look at this line:

```
num_var = int( input ("Now " + user_name + " input a number: "))
```

This way of doing things takes the number typed by the user in as a text string and then changes it into an integer. This changing a variable from one type to another is called casting.

We can use casting the other way around:
str(num_var) would convert num_var (and integer) into a text string so that for example you could then concatenate (join strings together) it like this:

```
print "Your number plus 4 is " + str(num_var)
```

6 Try adding this to the bottom of your code and running it!

7 Close the Editor window

Note what we did with the text strings in the fourth line of our code. Using the + sign we concatenated three strings "Now ", user_name and " input a number". This is probably the simplest way to join strings together using python. We will use this method for just now, but later in the course we will see better ways to concatenate strings.

Note The input function has changed between the 2.x versions of Python and the 3.x versions. In the later version command "input" behaves just like "raw_input" does in older versions (which is no longer available). If you find code that was designed to run in older versions of Python, you can just exchange raw_input() for input()

Additional Task

Write a short script that takes two numbers from the user and adds them together. The script should return (use print for this) the answer. Add additional prompts for the user so that they can understand clearly what they should be inputting.

If you like you can try some other arithmetic as well.

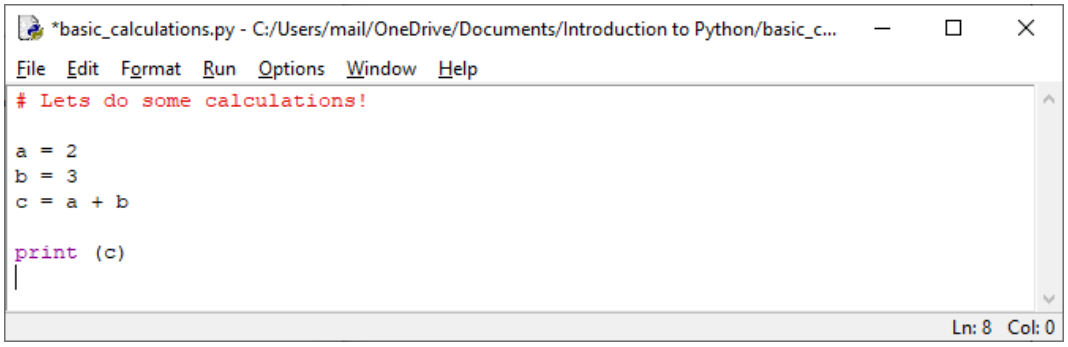
5 Calculations

As we saw in our last example, within python are able to perform calculations within our scripts. Just like any calculation we use operators (e.g. +, -) to tell Python what calculations to perform.

Python contains the following operators:

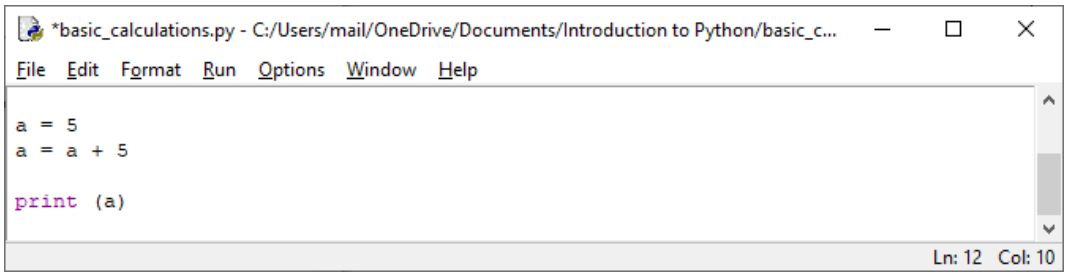
+ Addition	Adds values on either side of the operator.	10 + 20 = 30
- Subtraction	Subtracts right hand operand from left hand operand.	10 – 20 = -10
* Multiplication	Multiplies values on either side of the operator	10 * 20 = 200
/ Division	Divides left hand operand by right hand operand	20 / 10 = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	20 % 10 = 0
** Exponent	Performs exponential (power) calculation on operators	10**20 =10 to the power 20
//	A floor division performs the division but then returns the floor (lower number) of the answer. To put it another way, it chops off the digits after the full stop	9/2 = 4.5 9//2 = 4

In python we can do maths using variables i.e.



This script simply returns the number 5

We can also say:



That looks like an impossible statement. It really means Variable **a** will contain the value currently in **a** plus 5. The old value in **a** is gone, replaced by a new value.

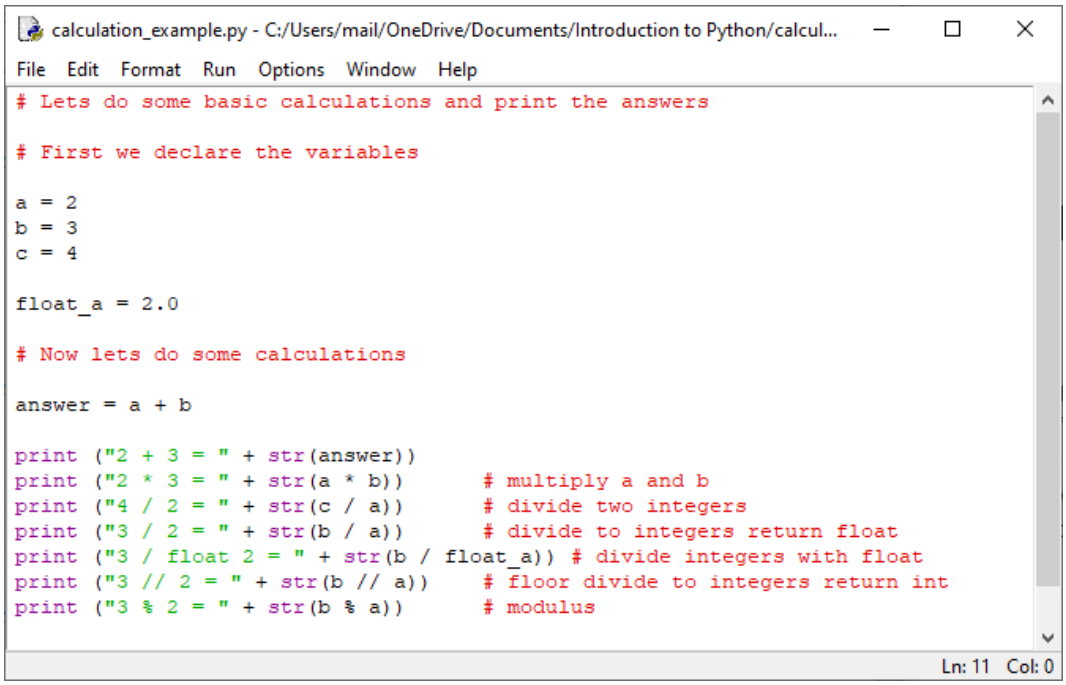
This returns the number 10

Task: Basic Arithmetic

1 Click on **File** and then **New File** in any of the **IDLE** windows

2 A new Editor window will appear

3 Enter the following:



4 Save the file as **calculation_example.py** in your practice folder

5 Run the script using the **Run Module** command (from the run menu)

6 Close the Editor window

Note that when we divided the integer **b** with another integer **a** in Python 3 the answer 1.5 is a float. When we instead divided by a float **float_a**, the answer is also a float. In older versions of Python dividing integers 2 and 3 would have returned a rounded answer.

Python 3 will automatically convert number types (cast) for us to make them compatible with each other.

Unfortunately it does not do this when we try and perform maths on string variables!

Additional Task: Numerical.

- Write a program to calculate the perimeter and area of a rectangle, given the width and height.

- Write a program to calculate area and circumference of a circle given the radius.

  How will you get Pi? (Estimate it as 3.14 for this exercise)
  $Area = \pi r^2$
  $Circumference = 2\pi r$

- Write a program to change the inclusive Vat on a number from 17.5% to 20%.

6 If Statements (Control Flow Tools)

When we write Python scripts often we need the script to take a particular condition is met. For example, if we needed to count the number of instances of a particular work in a file the python script would read each word in the file and then add one to the count **if** the word was the one it was looking for. We use what are called "Control Flow Tools". These tools decide what lines of code are run and under what conditions.

Let's look at the most commonly used tools

b. If – Else Statements

This code causes things to happen if set conditions are met. When we are working with if statements we need to be **very careful about indentation**. Python uses it to understand which code belongs to the if statement or to put it another way what it should do if the if statement is true. This is different to a lot of other scripting/programming languages which enclose these instructions in brackets to achieve the same thing.

The general Python syntax for a simple `if` statement is

```
if condition :
```

- `do this`

```
else:
```

- `do this instead!`

The condition must be a logical test that will evaluate to either **true** or **false**, e.g:

if my_number > 5 : # is my_number greater than 5

or

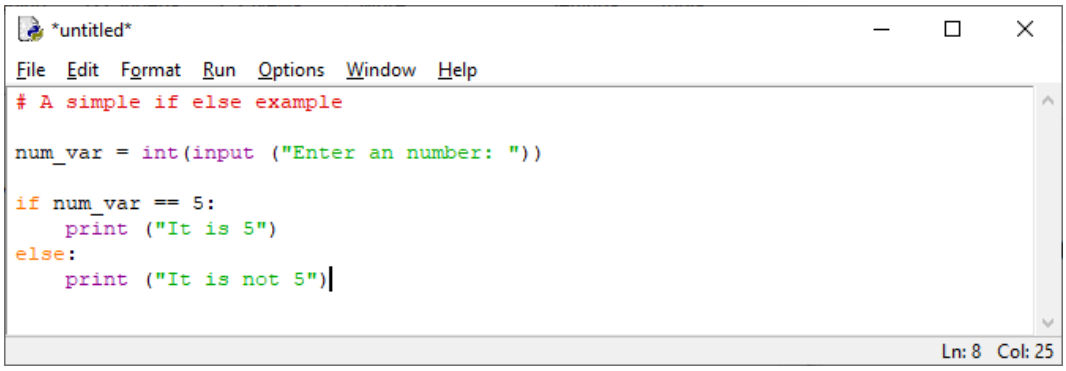
if my_number != 3: # is my_number not equal to 3

If you do not have the else: statement, Python will just continue to the next line of code in your script should the if statement evaluate as false.

Task: A simple if Statement

1 Click **File** and then **New File**

2 Enter the following code:



3 Save the file as if_else_example1.py

4 Run the script and follow the prompts.

5 Close the Editor window

Note that a single = is an assignment operator used to assign a value to a variable, a == is a comparison operator, meaning that things are equal.

The following **comparison operators** can be used:

<	less than	5 < 5	False
<=	less than or equal to	5 <= 5	True
>	greater than	5 > 10	False
>=	greater than or equal to	5 >= 10	False
==	equal to	'a' == 'b'	False
!=	not equal to	'a' != 'b'	True

The following **membership operators** can be used:

in evaluates to True if it finds a variable in a specified sequence, i.e.

'Edinburgh' in "Glasgow University" False

not in evaluates to False if it finds a variable in a sequence, i.e.

'Edinburgh' in "Glasgow University" True

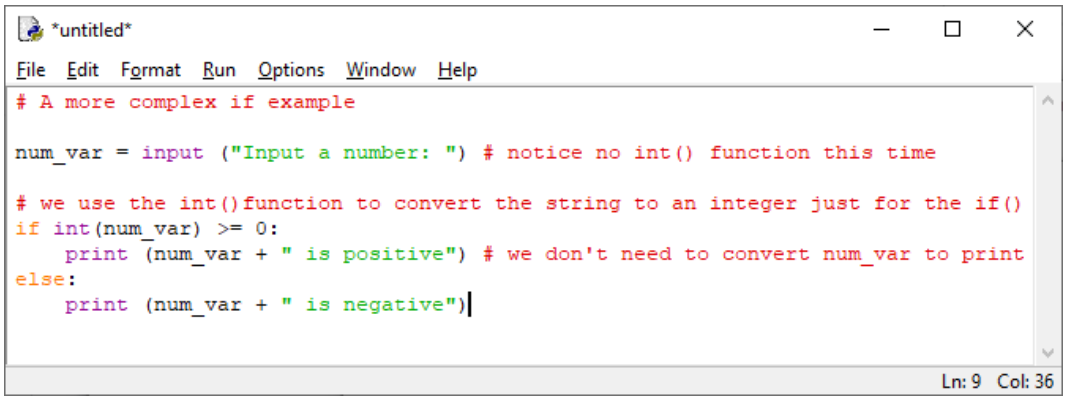
This is not a complete list of the operators that can be used within **Control Flow** statements, but it will do for just now.

Task: A more complex if statement

Write a program that inputs a number and tests whether it is positive or negative.

1 Click **File** and then **New File**

2 Enter the following code:



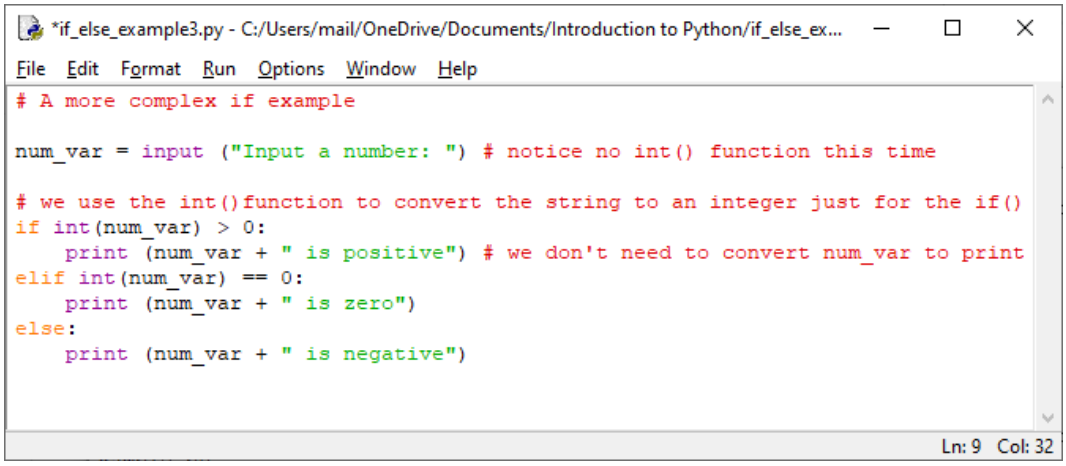
3 Save the file as if_else_example2.py

4 Run the script and follow the prompts.

5 In this script, positive and negative numbers are correctly identified. But we can refine the script further to make it identify zero values as well.

6 Use the **Save As** command to save this file as **if_else_example3.py**

7 Edit the code to look like this:



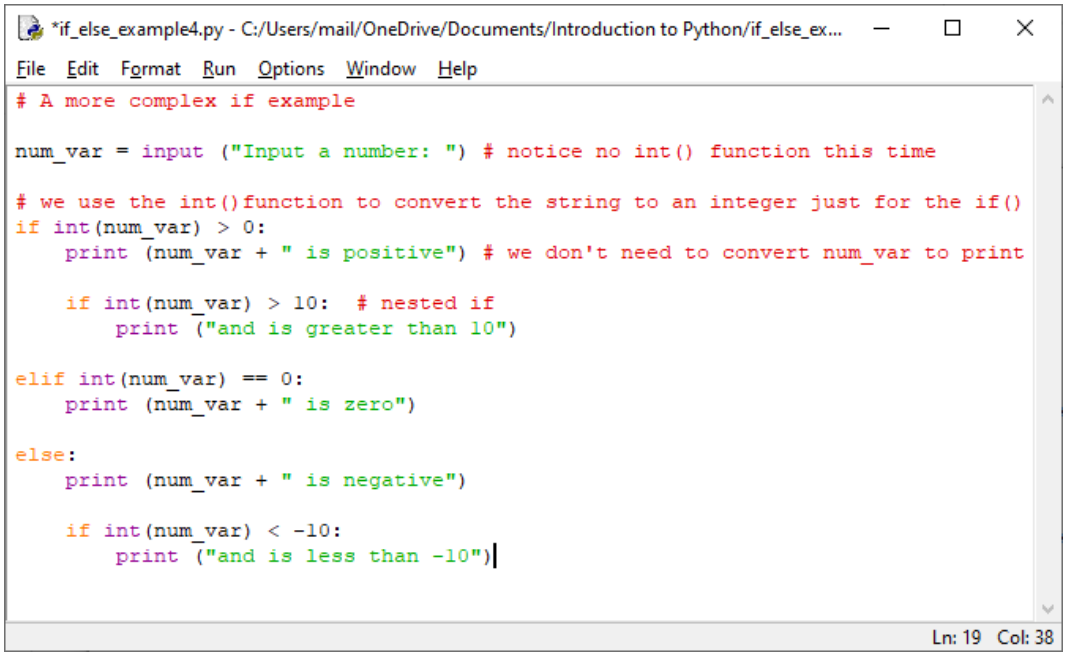
8 Run the script and follow the prompts. Try inputting the number 0

The change that we have made to our code introduces the **elif** statement. **elif** is short for else if and it allows us to use more than one if statement, executing the second one if the first is false.

Bonus Task: Nested Ifs

If you have time, try this:

1 Using the same file, edit it to look like this:



2 Be very careful with your indentation when you do so!

3 Use **File – Save As** to save the file as **if_else_example4.py**

4 Run the script and follow the prompts. Try replying with a value such as 15 or -15.

5 Save and close the file.

In this example we are placing ifs inside our if and else statements which leads to even more possible outcomes in our code. We describe this as nesting. When we nest statements like this we have to be very careful about how we indent, get this wrong and you might get some unintended actions.

You are not limited as to how many times you nest in 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.

Function	Description
hex()	Convert an integer number to a hexadecimal string.
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.

Function	Description
object()	Return a new featureless object.
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.

Function	Description
str()	Return a str version of object.
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