Starting Python

Start python3 from the command line. You should see something like:

```
Python 3.9.5 (default, May 18 2021, 12:31:01)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The >>> is the prompt for the python interpreter. This means that anything you type will be interpreted as python code. If you ever want to exit the interpreter use Ctrl-D. (You also might need to first use Ctrl-C to exit the line you are typing). Try it out.

There are two main flavors of python, python2 and python3. There are subtle differences between python2 and python3 and if you ever find something strange going on, you might be using the wrong version of python. Python2 will be replaced soon, so we will primarily use Python3 in this class.

Using Python as a calculator

You can first try using python as a calculator. For example, add two numbers together and see what you get. Note that the interpreter responds to you: after you hit 2+3, it spits back 5.

```
>>> 2 + 3

5

>>> 2.0 + 3.0

5.0

>>> 3.5 - 2.564

0.9359999999999999
```

Ok… pretty straightforward. You can use a number of mathematical operators in Python (the plus sign is an example of a mathematical operator) and you should be familiar with all of these below.

'+' – Add two numbers

e.g. 6 + 2 = 8

'-' – Subtract two numbers

e.g. 6 - 2 = 8

'*' – Multiply two numbers

e.g. 6 * 2 = 12

'/' – Divide two numbers

e.g. 6 / 2 = 3

'%' – Return the remainder from dividing two numbers

e.g. 7 % 2 = 1

'//' – Return the floor of the division of two numbers

e.g. 7 / 2 = 3

'**' – Return the 1st number to the power of the 2nd number

e.g. 2 ** 6 = 64

Python functions

We will cover functions in more detail soon, but it's useful to start using them. Functions are like verbs. They preform some sort of action. They take in 0 or more arguments, does something to them (print them, save them to a file, add them, etc), and then returns an object that you can save to a variable. The arguments are always enclosed in parenthesis and separated by commas following the name of the function. An example is print. Print takes in a string and prints it to the console. It returns an object called None (basically it doesn't' return anything).

```
>>> print("Hello world")
Hello world
>>> print("Welcome to class")
Welcome to class
>>> print(3)
3
```

Using variables in Python

Computer programs wouldn't be that valuable if you could only use them as calculators. One of the first useful features of a computer program is the ability to use variables. You can set the value of a variable using the assignment operator (=) like this:

```
>>> x = 8

>>> x

8

>>> x + 2

10

>>> x ** 2

64

>>> x

8

>>> x = x ** 2

>>> x

64

>>> x = 8

>>> x

8

>>> x += 2

>>> x

10

>>> x **= 2

>>> x

100
```

This should mostly be self-explanatory but a couple things worth noting:

1) Performing a mathematical operation returns the value of the operation. The interactive interpreter automatically prints out to the screen anything returned by the expression. Some expressions do not return anything (or they return the built in None value). For example, the assignment operator does not return any value (x = 8). If it did, you would have seen something printed out on your console.

2) While you can add or perform other mathematical operations on x, this does not change its value. To change the value of x, you **must** use an assignment operator like =.

3) There are also specialized operators that allow you to simultaneously perform a mathematical operation on a variable and change its value. For example, x += 2 takes the current value of x, adds 2 to it, and then assigns the result back to x.

The assignment operators available for you in Python.

‘=’ - Set the left hand side equal to the right hand side.

e.g. x = 2 or x = 2+3

‘+=’ - Set the left hand side equal to the left hand side + right hand side. Left hand side must already exist!

e.g. x += 2

‘-=’, ‘*=’, ‘/=’, ‘%=’, ‘**=’, ‘//=’ – Similar assignment operators exist for all of the mathematical operations listed above.

Getting into the details. What is a variable?

One of the most important things to understand about Python is that it is an object-oriented language. Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which are data structures that contain data, often known as attributes; and functions, often known as methods. In general, everything in Python is an object. So when you type x = 8 into the interpreter, you are creating what is known as an instance of the class integer. We can explore this by using a built-in function in Python called dir(). The dir() function takes in a variable and returns its class. Let’s explore a bit:

```
>>> w = True
>>> type(w)
<class = ‘bool’>
>>> x = 8
>>> type(x)
<class 'int'>
>>> y = 6.0
>>> type(y)
<class 'float'>
>>> z = 4 + 3j
>>> type(z)
<class 'complex'>
```

Evidently, we can create different classes of numbers: Booleans integers, floats, and complex numbers. The w variable is a Boolean, the x variable is an integer, the y variable is a float, and the z variable is a complex number (j represents the imaginary number: $\sqrt{-1}$). These are the four numeric types supported by Python. Now, what is a class?

Class – A general template for creating an object. ‘int’ is a class that understand integer type numbers. Classes define the data and functions necessary for representing an object.

Instance or Object – An actual representation of a class. The x variable is an instance of the ‘int’ class. You can have as many instances of a class as you want.

Attribute – A field of data for the object. A field stores some sort of information about the object.

Method – A function associated with the object. A method operates on parameters passed to the function as well as any attributes associated with the object.

Dot operator – The '.' operator is how you access the attributes and methods of a class. For example, x.real is how you access the value of x.

So, any instance will contain a bunch of attributes and methods, as defined by its class. In general, you can find these methods and attributes by going into the source code where they are defined, however, there is an easier way by using the built-in function called dir(). Like most functions, dir() requires a parameter. dir() takes in a parameter and then outputs any methods and attributes for the object. Let's try this on the three variables we created above:

```
>>> dir(x)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
'__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__',
'__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__',
'__getnewargs__', '__gt__', '__hash__', '__index__', '__init__', '__int__',
'__invert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__',
'__ne__', '__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__',
'__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__',
'__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__',
'__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__',
'__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__',
'__sub__', '__subclasshook__', '__truediv__', '__trunc__', '__xor__',
'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator',
'real', 'to_bytes']
>>> dir(y)
['__abs__', '__add__', '__bool__', '__class__', '__delattr__', '__dir__',
'__divmod__', '__doc__', '__eq__', '__float__', '__floordiv__',
'__format__', '__ge__', '__getattribute__', '__getformat__',
'__getnewargs__', '__gt__', '__hash__', '__init__', '__int__', '__le__',
'__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__pos__',
'__pow__', '__radd__', '__rdivmod__', '__reduce__', '__reduce_ex__',
'__repr__', '__rfloordiv__', '__rmod__', '__rmul__', '__round__',
'__rpow__', '__rsub__', '__rtruediv__', '__setattr__', '__setformat__',
'__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__',
'__trunc__', 'as_integer_ratio', 'conjugate', 'fromhex', 'hex', 'imag',
'is_integer', 'real']
>>> dir(z)
['__abs__', '__add__', '__bool__', '__class__', '__delattr__', '__dir__',
'__divmod__', '__doc__', '__eq__', '__float__', '__floordiv__',
'__format__', '__ge__', '__getattribute__', '__getnewargs__', '__gt__',
'__hash__', '__init__', '__int__', '__le__', '__lt__', '__mod__', '__mul__',
'__ne__', '__neg__', '__new__', '__pos__', '__pow__', '__radd__',
'__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__',
'__rmod__', '__rmul__', '__rpow__', '__rsub__', '__rtruediv__',
'__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__',
'__truediv__', 'conjugate', 'imag', 'real']
```

You are probably surprised to see how many attributes and methods are associated with these three variables! They are way more complicated than just simple numbers. x contains 69 different attributes or methods. For an integer, the value is stored in the real attribute. So you can access it like so:

```
>>> x
8
```

```
>>> x.real
8
```

What do the others do? The majority of these contain two underscores before and after their name. These are called **magic methods/attributes**. Magic methods/attributes are used to build-up the normal logic you expect of a class. For example, when you add two integers together: x + 2, you expect x to know how to add 2 to its value. It accomplishes this using the __add__() magic method. Lets see:

```
>>> x
8
>>> x + 2
10
>>> x.__add__(2)
10
```

All this info might seem strange, but it’s essential to start to think about things in this way. Objects are paramount in Python.

Note that using the dir() built-in, you can’t tell the difference between a method and an attribute. However, remember that you must use () to call a method. If you try to use it without it you will be unsuccessful in your call. E.g. try it on a method:

```
>>> x.__abs__
<method-wrapper '__abs__' of int object at 0x10fcbadb0>
```

Similarly, if you add () to an attribute you will get an error:

```
>>> x.real()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not callable
```

Explicitly vs. implicitly creating instances

Up until now, we have dealt with Python’s ability to implicitly create ints, floats, and complex numbers by inferring what type of number we want. i.e. x = 2 creates an integer, x =2.0 creates a float, and x = 2.0 + 0j creates a complex number.

We can also explicitly create an integer, float, or complex instance like so:

```
>>> x = int(2)
>>> x
2
>>> x = float(2)
>>> x
2.0
>>> x = complex(2,3)
>>> x
(2+3j)
```

We do this by using the name of the class we want to use (the same as returned by type()) and entering what value we want it to take. We are invoking the ability of a class to create an instance by adding parenthesis at the end of the name and passing parameters that are necessary for creating the object.

Note that when we create a complex number, we need to enter two values for the real and the imaginary parts.

Scripts

So far we have used the interpreter interactively to run Python. An alternative way to run Python is to put your commands in a file known as a script. While running the interpreter is useful for learning and debugging your code, in general you will put most of your Python commands in scripts. This will leave a permanent record of what you did that will be useful for reusing code, debugging, reference, etc.

The basic idea is pretty simple. Open a file called first_script.py and save it in a directory of choice. Python scripts should be named with the extension .py as text editors will automatically recognize the script and add useful color coding that will make it easier to avoid bugs. While you can use any text editor you would like, I would recommend trying Sublime.

Ok, within your script type the following two lines:

```
x = 6

y = 10

z = 10+6

print(z)
```

You should notice some of the words are colored differently than the others.

Save the file (you can leave it open) and now run the script in the command line by typing:

```
(base) ➜  ~ python3 firstscript.py
```

Help

As you learn a new language, you will often find you need to get some help. While I am here to help you, it's important that you become self-sufficient in getting your own help. Luckily there is great documentation and help online. For example, Python documentation on standard types can be found here:

https://docs.python.org/2/library/stdtypes.html

Or for example, if you wanted to find more information on magic methods (called special methods in the documentation)

https://docs.python.org/2/reference/datamodel.html#special-method-names

Of if you have specific questions, you should get used to using stackoverflow.com. This website is a question and answer type website. However, in general most questions have already been asked, so you just need to google your question and most likely you will find an answer you are looking for. Often just googling your error will lead you to a number of questions on stackoverflow.com. For example, if you google:

```
TypeError: 'int' object is not callable
```

The first four hits are from stackoverflow.com. They give answers that might help you debug your code (if you didn't now what went wrong).