1 - Variables and Data Structures

Welcome to your first ever Python notebook! Here we are going to learn about the basics of Python.

What is Python?

Python is a programming language. We use programming languages to give a specific set of instructions to a computer in the form of *code*. While there are many different possible programming languages to choose from, in this course we will be focusing solely on Python.

Python is arguably the hottest language to learn at the moment. It's used in every major government and company, in every country on earth - and with good reason. Some of the reasons we think Python is so great are:

1. It's easier than other languages to pick up (But that doesn't mean it's easy!) The syntax is often seen as being more intuitive than other programming languages like Java or C so it's a great first language to learn.
2. You can do almost anything with it. People use Python for all sorts of different tasks ranging from hacking to web development to data science to physics and engineering.
3. There is a huge community of Python programmers out there. That means, if you're stuck with a problem, you can google it and the chances are someone out there has had the same problem.

Jupyter Notebooks

Click the image to watch a short video about Jupyter!

Jupyter is the application that you are running right now. It runs directly in your web browser. A Jupyter notebook is made up of a series of cells, which let you run snippets of python code quickly and see the results. In order to run a piece of code, just type it into a cell and hit `shft` + `enter`, `ctrl` + `enter` or just click the button at the top that says 'run'. Try it below!

```
# Try to run this cell!
print('Hello World')
```

You can add a new cell by clicking on the bit to the left (where you see `In[1]:`) and pressing `a` (for a new cell *above*) or `b` (for a new cell *below*). Cells can be deleted by pressing `dd`. You can also select 'insert' then click 'new cell above' from the top menu.

Try it!

```
# ADD A NEW CELL ABOVE ME
```

```
# ADD A NEW CELL BELOW ME
```

Variables

The most basic unit we deal with in programming is called a *variable*. Variables are things that can take a value such as a number or a piece of text. We assign a value to a variable by using the `=` sign.

```
# we are creating a variable called my_number, and giving it a value of 42
my_number = 42
```

You can check what's inside a variable by simply typing the variable name in a cell and running that cell

```
my_number
```

Or alternatively, you can use the `print()` function.

```
print(my_number)
```

You can name variables more or less anything you like. However

1. Variables can only contain letters, numbers and underscores (`_`'s)
2. Variables must start with a letter, not a number
3. Variables can't have any spaces in them

Exercise 1

Create a new variable, called whatever you want, and assign it a number as a value. Then check what's inside the variable you have just created

```
# YOUR CODE HERE

```

Types

In python values can have different *types*

Numbers

Numbers in Python are more or less what you might expect. They can have values like

```
5
1001
37.5
0.001
```

Technically there are two distinct categories of numbers in Python: whole numbers like `1`, `5` and `100`, and decimals like `5.5`, `0.001` and `3.14195`. These are also referred to as *integers* and *floats* respectively.

```
number1 = 5
number2 = 1001
number3 = 37.5
number4 = 0.001

print(number1)
print(number2)
print(number3)
print(number4)
```

Strings

Strings are the second important type. These are little pieces of text or words. We create a string by enclosing our piece of text in quotation marks

```
'this is a string'
'hello, theDataKirk'
'I can put any symbols here!?#@<>'
```

We can also assign strings to variables

```
string1 = 'this is a string'
string2 = 'hello theDataKirk'
string3 = 'I can put any symbols here!?#@<>'

print(string1)
print(string2)
print(string3)
```

Booleans

Booleans are the third important type. They can only have two possible values

```
True
False
```

When typing a boolean, the first letter has to be a capital for Python to recognise it.

```
bool1 = True
bool2 = False

print(bool1)
print(bool2)
```

Exercise 2

Try creating a new number, string and boolean variable. Then print each one to check their value.

```
# YOUR CODE HERE

```

Maths operations

Wen dealing with numbers, we can perfrom the usual maths operations you would find on any calculator

```
# addition
1 + 1
```

```
# subtraction
5 - 3
```

```
# multiplication
2 * 2
```

```
# division
10 / 2
```

We can also do maths operations with variables we have set

```
a = 2
b = 5
c = 10

print('a =', a, ', b =', b, ', c =', c)
```

```
a + b
```

```
c / a
```

```
c - b
```

We can even assign the result of a calculation to a new variable

```
d = c - b
```

```
d
```

```
e = c / b
```

```
e
```

When combining operations together, the rules of BIDMAS are followed. However, we can also use brackets to make things clear

```
5 + 5 * 2
```

```
(5 + 5) * 2
```

Exercise 3

Use a new cell to find the average of 10, 14, 25, 81 and 99

```
# YOUR CODE HERE


```

Conditions

Imagine you were giving directions to someone. You might say something like "If you go past Tesco, you've gone too far so turn back around". When we're telling computers what to do, it's often useful to tell them to do something, IF some condition is true.

Some basic conditions we are able to give a computer are things like

- "If number1 is bigger than number2, do something"
- "If these two numbers are equal, do something else"

The syntax for giving conditions in Python is

```
if condition:
    'do something!'
else:
    'do somethig else!'
```

When we compare numbers in Python we can use any of the following operators

```
a < b     # a is less than b
a > b     # a is greater than b
a <= b    # a is less than or equal to b
a >= b    # a is greater than or equal to b
a == b    # a is equal to b
a != b    # a is not equal to b
```

where `a` and `b` are any numbers we would like to compare. Here are some examples

```
a = 5
b = 10

if a < b:
    print('a is less than b')
else:
    print('a is greater than or equal to b')
```

```
x = 100
y = 1000

if x == y:
    print('x is equal to y')
else:
    print('x is not equal to y')
```

Exercise 4

Design your own example!

```
# YOUR CODE HERE


```

We can also chain conditions by using the keywords

```
and
```

and

```
or
```

Here are some examples

```
n1 = 10
n2 = 11
n3 = 12
```

```
if n1 < n2 and n2 > n3:
    print('This is true')
else:
    print('This is false')
```

```
if n1 < n2 or n2 > n3:
    print('This is true')
else:
    print('This is false')
```

Lists

Lists are used when we want to store a sequence of values in one place. An example of a list with length 5 could be

```
[1, 2, 3, 4, 5]
```

or

```
[2, 4, 5, 3, 6]
```

We can assign a list to a variable like this

```
my_list = [5, 4, 3, 2, 1]
```

```
my_list
```

We can access the individual elements of a list like this

```
my_list[0]
```

```
my_list[1]
```

Something that can be slightly confusing at the start is that when getting list items, we count from zero. So, to access the first element we say `my_list[0]`, and to access the second element we say `my_list[1]` and so on.

We can put anything we like into a list (including other lists!). Here is an example of a list that contains strings

```
['a', 'b', 'c', 'd', 'e']
```

In general, lists can also contain a mixture of types

```
[1, 'a', 2, 'b', 3]
```

```
            #     0    1    2    3    4
my_string_list = ['a', 'b', 'c', 'd', 'e']
```

```
my_string_list
```

```
my_string_list[0]
```

```
my_string_list[3]
```

Exercise 5

Create your own list and assign it to a variable. Try accessing the individual elements

```
# YOUR CODE HERE

```

We can add a new item onto the end of a list by using the format

```
my_list.append(new_item)
```

```
my_list = [1, 2, 3, 4]
my_list.append(100)
```

```
my_list
```

We can also set individual elements

```
my_list = [1, 2, 3, 4]
my_list[0] = 100
```

```
my_list
```

We can find out the length of a list by using the keyword

```
len(my_list)
```

```
a_new_list = [4, 3, 5, 7, 2, 6, 2, 5]
len(a_new_list)
```

We can find the sum of all the numbers in a list by using the keyword

```
sum(my_list)
```

```
a_new_list = [4, 3, 5, 7, 2, 6, 2, 5]
sum(a_new_list)
```

We can find the minimum and and maximum of a list by using the keywords

```
min(my_list)
max(my_list)
```

```
min(a_new_list)
```

```
max(a_new_list)
```

Final Exercise

Below are some lists. For each one, find the mean, and the range using the functions

```
min()
max()
sum()
len()
```

```
list1 = [3, 4, 2, 6, 3, 7, 9, 3, 7, 9]
list2 = [11, 24, 134, 75, 38, 56, 33, 25]
list3 = [-43, 82, 99 + 5, 55 / 11, 43]
```

```
# YOUR CODE HERE
```