Python Basics

November 16, 2020

1 A Very Brief Introduction to Python

This is a very brief introduction to Python, which will help you get started.

It will touch upon the topics of the command line, variables, data types in Python, loops and functions.

1.1 The command line

When working in an interactive programming environment, such as Jupyter, Python commands are executed from the command line. In Jupyter the command line is located in code cells as the one below.

[ ]:

We can type out code in the command line and run the cell to execute the code.

Below we type in a mathematical expression. Python evaluates the input and returns the result as the output.

[1]:
```
3 + 3
```

[1]:
```
6
```

1.2 Variables

In order to store and reuse the output, we get from Python, we can use variables.

Variables let us store values so we can reference them at a later point.

[2]:
```
x = 10
```

We now have a variable named `x` with the value 10 stored inside.

To make sure, we can check that the value of `x` is 10 by typing `x` in the command line.

[3]:
```
x
```

[3]:
```
10
```

Whenever the name of a variable is typed in the command line, the value of the variable will be returned as the output.

Variables allow us to perform all sort of operations. Below we define a new variable `y` and assign it the value 5.

[4]:
```
y = 5
```

Just as we started out adding two numbers together, we can do the same with variables holding numbers.

[5]:
```
x + y
```

[5]: 15

Similarly, we can save the sum of our two variables to a new variable for later use.

[6]:
```
z = x + y
```

[7]:
```
z
```

[7]: 15

1.3 Data types

Python includes a number of different data types, which have different properties.

1.3.1 `int`

In the previous section, we encountered one of these data types: integers. Integers are whole numbers recognised by Python as the data type `int`.

We can check the data type with the `type()` command.

[8]:
```
type(x)
```

[8]: int

Our variable `x` containing the value 10 is, indeed, an integer.

1.3.2 `float`

Related to integers is the data type `float` or floating point values. Floats are numeric decimal values such as 0.1, 4.95 and 10.0.

Unlike some other programming languages, Python supports operations that include both integers and floats. This allow us to perform calculations as the one below without converting the numbers to the same data type first.

[9]:
```
1 + 1.5
```

[9]: 2.5

1.3.3 str

Until now we have only looked at numeric data types. We can store textual data as strings with the datatype `str`.

String values are surrounded by quotes - either single or double - and they can contain characters, words, sentences and any other type of text. Below we print a string of text to the screen.

[10]:
```
print('Hello World')
```

```
Hello World
```

Just like with numbers, we can also perform various operations with strings.

[11]:
```
a = 'Data'
b = 'sprint'

print(a + b)
```

```
Datasprint
```

Notice that numbers also can be stored as strings. Thats why we get different outputs for the two expressions in the cell below.

[12]:
```
print(10 * 3)

print('10' * 3)
```

```
30
101010
```

Multiplication and addition can be useful for manipulating strings but we need to pay attention to our data types in order to avoid errors.

[13]:
```
print('10' * '10')
```

```
      ␣
↪---------------------------------------------------------------------------

       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-13-cf45e0375c40> in <module>
   ----> 1 print('10' * '10')


       TypeError: can't multiply sequence by non-int of type 'str'
```

1.3.4 `list`

Sometimes it can be useful to bundle values together. For this purpose, we use the `list` data type.

Lists are initiated with square brackets, and the values in the list are separated by commas.

[14]:
```
my_list = [1, 2.0, 'third value', x]
```

In Python, lists can hold any data type and we are allowed to mix data types within a single list. `my_list` contains an integer, a float, a string and a variable containing an integer.

Each value in the list is assigned an index. We can use this index to access a single element of our list.

[15]:
```
my_list[2]
```

[15]:
```
'third value'
```

Notice that the list index starts at zero. That is why our third value is located at index 2.

1.4 Loops

Loops are useful if we want to repeat an operation on a number of elements.

We can use a `for` loop to iterate over the values in a list and perform a specific operation to each element in the list.

[16]:
```
for item in my_list:
    print(item)
```

```
1
2.0
third value
10
```

Notice that we can use any variable name for the elements in the loop; we only use it to reference the element within the loop. However, it's usually a good idea to choose a name that doesn't hurt the readability of the code. In Python, `i` (for item) is often used.

[17]:
```
for i in my_list:
    print(i * 2)
```

```
2
4.0
third valuethird value
20
```

1.5 Functions

With functions, we get a lot of useful features without having to worry about the underlying code.

Functions take an input, perform some sort of operation and return an output.

So far, we have already used a few functions, namely `print()` and `type()`. Another useful function is `len()`, which returns the length of an object.

When used with a list, `len()` returns the number of elements in the list.

[18]:
```
len(my_list)
```

[18]:
```
4
```

When used with strings, `len()` returns the number of characters in the string.

[19]:
```
len('A string of text.')
```

[19]:
```
17
```

We can save the output of a function to a variable. This is useful if need to use to output more than once, as it is more efficient than calling the function each time.

[20]:
```
length_of_my_list = len(my_list)
```

We are going to use a lot of functions to let us analyse, process and visualise our data. We will also write our own functions to help us reuse our code.

1.5.1 A note on methods

In Python, *methods* are similar to functions but the underlying code is implemented differently. For now, we do not need to worry about the technical diffences. Both methods and functions usually take an input, perform some operation and return an output. Just note, that whenever methods are mentioned, you can think of them as similar to functions.

1.6 The Next Steps

From here you should be ready to dive in and have a go at working with some real data.

The other notebooks continue by introducing some useful tools for exploring, analysing and visualising data. They also introduces the `import` statement, which lets us expand the functionality of our Python environment by importing additional code from libraries and modules.

In case you want to know more about a specific object in Python, you can use the `help` statement, which will give you access to the documentation.

If you want to learn more about the basics of Python, including the topics covered here, the first chapter of the book Automate the Boring Stuff by Al Sweigart is a useful introduction.