Basic Computation in Python

Guillaume Macneil

October 6, 2020

Abstract

This is the first lesson on the 'Introduction to Python' course. This course is loosely based around the 'MIT Introduction to Computer Science - Fall 2016' course. The course is more focused towards the Python programming side of things.

1 An Overview

Programming is just the creation of instruction sets for a computer in order to achieve a desired output. Fundamentally, all the computer does is perform basic calculations - **billions per second** - and stores the result. All of these calculations are defined by you as the programmer and the computer will only do what you tell it to.

So let's start with some of the very basics of Python, though this is applicable to almost every programming language. A program is a series of definitions that are evaluated and commands that are executed. So what executes the commands? The commands are executed by the *'interpreter'* - a program that *interprets* the high-level commands that you gave and makes the computer perform them. Programs can be thought of as cooking recipies, where the computer is the chef that diligently follows the instructions **very accurately**.

2 Objects and Types

Programs manipulate and perform calculations of data objects. All objects have a type that defines what calculations the computer can do to them. Here are the fundamental scalar (cannot be subdivided) objects.

- ***int*** **-** integers, eg. *6*
- ***float*** **-** floating point (decimal) numbers (real), eg. *3.97*
- ***bool*** **-** boolean values, either *True* or *False*
- ***NoneType*** **-** an object that only has the value *None*

You can use the *type()* function (you'll find out about these later) to determine the type of a given object. For example:

```
type(3)
# Returns 'int'

type(7.0)
# Returns 'float'

type(True)
# Returns bool
```

As a quick note, the above lines that start with # are called comments, the interpreter ignores these, they are just notes for programmers reading the code.

Certain objects can have their types converted into other types (also called 'type casting'). These type conversions can be performed with functions like *float()* and *int()* which convert certain objects into *float* and *int* types respectively:

```
float(5)
# Converts the int 3 into the float 5.0

int(2.6)
# Truncates the float 2.6 into the int 3
```

One of the most useful commands that you can use is the *print()* function that *prints* the given object for the user to see:

```
print(7)
# Prints 7

print(2+5)
# Prints 7
```

3 Operators and variables

Operators are in Python are the mostly the same as those that we use in every day life, like + and -. Here's a list of the ones that can be performed on *ints* and *floats*:

- **x + y -** x plus y
- **x - y -** x minus y
- **x * y -** x times y

- **x / y -** x divided by y
- **x % y -** the remainder of x divided by y
- **x ** y -** x to the power of y

Just like in normal mathematics, parentheses can be used to tell Python which operations to do first. Python follows the 'BODMAS' (or equivalent) order of operations - **B**rackets, **O**rders, **D**ivision, **M**ultiplication, **A**ddition and **S**ubtraction.

This is all well and good, but what happens if we want to store some information? This is where variables come in. Variables are, in essence, boxes in which we can store objects. Variables can be assigned values using the = sign. For example:

```python
pi = 3.14159
type(pi)
# Returns 'float'

year = 2020
type(year)
# Returns 'int'
```

Variables store values in the computer's memory. The above assignments bind a value to a given name, this allows the values to be invoked by using the variable name. We use variables in order to reuse names instead of values, this makes it easier to work with the program.

```python
a = 56
b = 32
a = a + b
# An equivalent statement would be 'a += b'

print(a)
# Prints '88'
```

As shown above, variables can be re-bound, meaning that names can be assigned to new values. The previous value may still be stored in memory, but the variable no longer *points* to it.