Chapter 2

Variables and Basic Operations

The basic principles and concepts in computer programming are the building blocks for creating powerful and innovative applications. Python is a very straightforward language and has a simple syntax. It will be the language we will use to illustrate the concepts that we pick up along the way. It is designed to allow programmers to create powerful, beautiful code, quickly and enjoyably. You will be able to run the brief examples for yourself using your text editor and Python interpreter, to check they work. The best way to use this book is to read a chapter, then run each of the example programs in that chapter on your own computer. Try changing the code a bit to get a feel for how you can tell the computer what to do. If you had prefer not to worry yet about getting your own Python interpreter installed, repl.it[1] has a really neat online Python text editor and interpreter ready for you to use straight away.

If you are using your own Python interpreter you have two options:

- Input the .py file into your Python interpreter
- Turn your .py file into an executable file

[1] http://repl.it/UL8/languages/Python

The first option is executed, using your command line (or terminal on a Mac), as follows:

```
$ python my_file.py
```

The second applies only to Mac and Linux users and requires you to append your .py file with the path to your Python interpreter. Just add the following line to the top of your Python source code to tell your computer that you've written Python code, so you want to execute it using your Python interpreter.

```
#!/usr/bin/python
```

Place this at the top of each Python code file you run. This means you can execute the code without needing to load it into your interpreter when you run it.

If you're using repl.it, you don't need to worry about this. You can execute a program appended as such using the command below where 'my_file.py' is replaced with your code's file name.

```
$ ./my_file.py
```

The *./* here means that the file is found in your current directory (the folder that you are currently using). Alternatively you can simply double-click on the program from within your file explorer.

If you're using Python on a Windows machine, you will just need to double click on the source code file to run it using your Python interpreter.

A 'Hello World' from Python

One of the most basic programs that be written is the age-old 'Hello World' program, so it seems only fitting that this chapter should begin with Python's 'Hello World'. To print a line of text to the screen, just write:

```
print 'Hello World'
```

The output of this program should look like this:

```
$ Hello World
```

Here, the $ symbol is used to denote the output of the program. If you're using repl.it, this will appear as '=>' followed by the output.

Variables

Every mobile application, website, desktop software or embedded system can be thought of as being a program that works with data, whether it is a simple version printing 'Hello World' to a screen, or a very complex version collecting data from satellite imaging and processing it to predict the weather. Even a game, which may not collect data in any way from the real world, operates by performing millions of calculations per second on data inside the program.

Data that is accepted by a program as an input is stored in memory, at a unique *address* so that it can be easily read or changed. If programmers had to remember the locations of all of these variables in memory, the process of programming would be immensely tedious and difficult. Happily, in the main programming language that we'll be looking at, and in most others, the programmer does not need to worry about the address where data is stored. Instead, programming languages allow us to refer to data by a name. Items of data, which are stored in memory and referred to by name, are called *variables.*

Variables are used in programs to store inputted data and calculations that use other variables. The variables can store different types of data. In general, variables are *declared* (created) in three steps:

1. Assign each variable a unique and descriptive name
2. Specify the type of data the variable can hold
3. Assign a value to each variable (numbers or letters)

In Python, we do not always need to declare variables before using them, or declare their type. For example, we can declare a variable as follows:

```
myInteger = 3
```

Python supports two primary types of numbers — *integers* and *floating point numbers*. The above example is a definition of an integer and we have chosen its name to be descriptive of what it is. To define a floating point number, we can use either of the following notations:

```
myFloat = 3.0 # implicit notation
myFloat = float(3) # explicit notation
```

(The # symbol is used in Python for comments. This means that the interpreter ignores anything after that symbol on a line in a Python program. This allows us to write helpful comments next to our code to explain what's going on.)

In programming, words or sentences stored in a variable are called *strings*. They can store any text that you may need in your program. Strings are defined in Python using either single or double quotation marks:

```
myString = 'this is a string'
myString = "this is a string"
```

In order to include actual quotation marks inside your string without them terminating the string, you can use a backslash (\) before the quotation mark. Alternatively, if your string is encapsulated with single quotes, you can use double quotes inside your string without needing to use a backslash:

```
myString = '"Yes", isn\'t he strange.'
$ "Yes", isn't he strange.
```

In addition to strings and integers and floating point numbers, a common and hugely useful type of variable in Python is the *list*. A list is a collection of other variables of any data type and of any length. A list is similar to an array in other languages, but is more flexible. A list is first declared, and then populated as follows:

```
myList = [] # declaration of list
myList.append(12)
myList.append(“text”)
myList.append(10000)
print “The first element of the list is: “ + str(myList[0])
$ The first element of the list is: 12
print myList # prints the whole contents of the list
$ [12, text, 1000]
```

Lists can also be declared and populated in one step, in-line:

```
myList = [1, 2, 3] # declaration of list and content
print myList # prints the whole contents of the list
$ [1, 2, 3]
```

The final important variable that we’re going to look at is the Python *dictionary* object. A dictionary is an object which matches a key with a value. It is a kind of list but elements are accessed not by its index, but by a key.

```
phonebook = {
    “James” : 09386477566,
    “Harry” : 09383377264,
    “Sophie” : 09472662781 }
print phonebook[“James”]
$ 09386477566
```

Take the example of a phonebook as in the example above. Here we have a name, matched up with a phone number. When we print the element in the dictionary ‘phonebook’ with the key ‘James’, the dictionary locates the value corresponding to that key and prints James’s phone number.

Dictionaries are useful for storing a wide range of data in many different situations, such as matching words to definitions, matching names to phone numbers or matching car registration numbers to makes and models.

Basic Operations

Common number-crunching programs are spreadsheets, database software and games. String manipulation is performed on social media websites, word processors and language translators. When programming, these kind of manipulations are generally referred to as *operations*. These are simple processes that data undergoes in order to form part of some kind of computation.

Some simple operations are performed on strings and numbers in these programs, and are built up to form complex equations or sets of powerful instructions.

Basic mathematical operations can be performed on numbers:

```
firstInt = 1
secondInt = 2
addInt = firstInt + secondInt
subtractInt = firstInt – secondInt
multiplyInt = firstInt * secondInt
divideFloat = float(firstInt) / float(secondInt)
print addInt, subtractInt, multiplyInt, divideFloat #
  prints all
$ 3 -1 2 0.5
```

Above, we have surrounded the variables 'firstInt' and 'secondInt' with brackets and used the word 'float' before them. This *casts* the integers as floating point numbers in order that we can perform division on them.

A *power operation* is one which multiplies a number by itself a certain number of times. For example 2^3 is 8 ($2 \times 2 \times 2$). In order to perform a power operation, we can use two multiplication symbols:

```
powerInt = 3 ** 2 # 3 to the power of 2
print powerInt
$ 9
```

Another operator which may be unfamiliar to you is the modulo (%) operator, which returns the integer remainder of a division.

```
remainder = 11 % 3 # integer remainder after 11 is divided by 3
print remainder
$ 2
```

In addition to numerical operators being used to perform calculations, Python provides excellent features for string manipulation which form new strings, as illustrated by the following example:

```
greeting = "Hello"
name = "Tom"
personalGreeting = greeting + "," + name
print personalGreeting
$ Hello, Tom
```

Here we have added two explicitly declared strings together — `greeting` and `name` — with an implicit string in the middle (", "). The process of adding strings in this way to form new string is called *concatenation*.

Trying to concatenate numbers within strings in Python will not work:

```
number = 5
string = 'hello'
combined = number + string
$ Traceback (most recent call last):
    File "<stdin>", line 3, in <module>
  TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

The output of this program is a Python error which tells us that the operator '+' does not support operands of type 'int' and 'str' (string) together. However, we can *cast* an integer to a string without needing to declare a new variable:

```
number = 5
string = 'hello'
combined = str(number) + string
$ 5 hello
```

The *function* (functions will be explained later) str() can be used to convert an integer or floating point number to a string on the fly, so that we can print numbers alongside strings using concatenation.

The multiplication operator (*) can be used with a string and a number as well as two numbers and behaves by repeating the string by the value of the number:

```
repeatedText = 'Hello' * 5
print repeatedText
$ Hello Hello Hello Hello Hello
```

We can also use the *increment* '+=' and *decrement* '−=' operators to add a specified value to a number variable. These operators add or subtract their operand to a number respectively. For example:

```
number1 = 0
number1 += 1
print 'number1 is' + str(number1)
number1 -= 7
print 'number1 is' + str(number1)
$ number1 is 1
$ number1 is -6
```

These are particularly useful for performing basic mathematical calculations on data in a program and you'll use them often.

Input and Output

Software is useless unless it does something with the data it has processed. This often means that data is collected in some way, processed and then displayed to the user. For the time being, our use of Pythons Input and Output (I/O) features will be primarily for learning to use the language and process data, but at a later stage we will use text or numerical input and text- or graphics-based outputs.

As we've already seen to output a string to the command line in Python, we use:

```
print 'this is some text'
```

However this only gives us a limited capability. What about if we want to print lots of different variables in the same line? Python allows us to add placeholders for variables inside the main string to be printed and then specify the variables at the end. For example:

```
print 'My name is %s and I weigh %d kg' % ('Joe Bloggs', 80)
$ My name is Joe Bloggs and I weigh 80 kg
```

In this example, %s depicts the placeholder for a string and %d specifies the location for a number. The variables are then listed in the order in which they should appear, inside the brackets, separated from the string by a % symbol. This technique can be extended to include any number of variables inside a string to be printed.

All software accepts some kind of input, whether it is from a sensor, the Internet or a human. When learning to write programs and when designing them professionally, it's important, therefore, to understand how to get basic input from a user.

Python here offers us the `input()` function allowing us to ask the user for some kind of information and to collect their reply.

```
person = input("Please enter your name: ")
print "Your name is: " + person
$ Please enter your name:
  Joe # Entered by the user
$ Your name is: Joe
```

By default, the string entered by the user will always be a string. If you require the user to enter a number, such as their age, you'll need to convert their entry into a number as follows:

```
ageString = input("Please enter your age: ")
ageNumber = int(ageString)
```

```
print “You are” + ageNumber + “years old.”
$ Please enter your age:
  42 # Entered by the user
$ You are 42 years old.
```

We now have the basic skills required to accept, process and output basic user data, but we've only looked at performing basic calculations and string manipulation so far. In order to make our programs more interesting and powerful, we need to introduce the ability to perform repeated tasks and make intelligent decisions based upon the data we have.

Questions

1. How does the data type integer differ from a floating point number?
2. Should the value of a string be declared in single quotes or double quotes when it is set?
3. How does the list data type differ from the dictionary data type in Python?
4. What symbol must you use in order to perform a 'to the power of' operation in Python? Write down a program which prints the value of 3 to the power of 7.
5. What do the increment and decrement operators do?
6. When outputting a string to the screen, what's the difference between the placeholders %d and %s?