Chapter 1 Beginning Programming with Python

Introduction to programming

This part of the book will deal with programming in Linux. In chapter 1 I explain the basics of programming through python, an easy to learn yet powerful programming language. I discuss topics such as conditional statements that allow programs to make choices, but also loops, user input and functions. These are considered the basic tenants of all programming languages. In subsequent chapters more difficult topics and examples are discussed. These topics are detailed with the use of different languages such as C, Java and Perl. An example program will show that the basic programming concepts are similar but are simply written in the syntax of that language.

What is Python?

Python is an interpreted programming language developed by Guido van Rossum and first released in 1991. Interpreted means it will run on the fly as soon as a python program is started. There is no need to convert the file from source code to machine ready code (zeroes and one, also known as assembly language). Instead Python feels almost like a file filled with command line arguments that are run one after the other. The python scripts are thus used to store the numerous and complex commands we want to execute on a regular basis.

Variables

There are several basic functions every program should be able to perform, without them there would be no point in computers. The first one is memory, or storing data in a variable. With every complicated piece of math you ordinarily write down intermediary results to help you along the way. Parts of results can be named, such as assigning letters X, Y and Z to them. Programming languages allow you to do the same. Use Gedit or you favorite text editor to open a file called 'first.py' and type in the following four lines of code. The .py extension is a reminder this is a Python program.

```
#!/usr/bin/python
x = 'This is a String'
y = 3
z = True
```

We have now assigned values to three values. The first is a string of characters which include the empty space. The second variable is an integer variable containing the value 3, if three had been written between quotes it would have been a string. Finally we assigned a Boolean value to z. Boolean values True and False are self-explanatory. As all variables can change because of user input or an algorithm the flow of a program can change. Boolean variables are often very useful to change a programs course. Note the weird first line that we had to type. As you may have guessed this is a household chore. With

‘#!/usr/bin/python’ we tell the command line to execute the script as a python script. Just cd to /usr/bin/ to check it out.

Output

A program is of no interest if after a calculation is does not produce output. In this book we will forgo the use of graphical windows and menus. They are cool but absolutely not necessary. In Python the print function will output anything you want to the Linux Terminal.

```
print "Hello World!"
```

If you add the previous line below the three lines part of first.py and then run the program you will see the following output

```
Hello World!
```

The program once again assigns the three variables x, y and z the same values we assigned them before. Then it prints out ‘Hello World!’. The three variables are upon completion of the program deleted from memory.

Input

To further add functionality to a program we can ask for user input. In python the function raw_input() will pause the program until the user types something into the command line and presses enter. By adding this statements below the four lines already written we get our first bit of interactive functionality. However, if you run the program you see the words Hello World printed out and then a flickering underscore indicating you can type something. If you do and press Enter the program simply ends. As you may have guessed the text the user typed was simply never used. We need to assign it to a variable to store it. The two lines of code below allow you to store the input from raw_input() and store it in variable mydata. On the second line the data of that variable is printed to the command line.

```
mydata = raw_input('Type something:')
print (mydata)
```

Notice that when the program is run the user sees the words ‘Type something:’ printed on the command line with a prompt. This is an optional feature of the function raw_input(). It is called a function parameter and using them can make life much simpler. To learn about all the possible parameters of a built-in function you can use google. Go to https://docs.python.org/2/library/functions.html for a list of functions. If you select raw_input() you can see that it only has one optional parameter called [prompt].

Conditional Statements

So far our program has been linear. Each line in the code is executed one by one from top to bottom. Most programs have at least some branching of possible actions it can undertake. One universal tool in programming is the if-else statement. It gives us the flexibility to choose a set of lines to execute only if a predefined condition is met. Let’s have a look at an example code

```python
#!/usr/bin/python
x = int(raw_input("Please enter an integer: "))
if x < 4:
    print('That is a small number')
else:
    print('That is a larger number')
```

I hope you understand what happens here. The if statements guards whether or not the indented line underneath it will be executed. If the guard evaluates to true it will. Our input should be 3 or lower. If it is 4 or higher the guard will evaluate to false and it won't execute. Instead the indented statement below else: will be executed. You can view this as a very simple tree. Everything above the if-else statements is the root, which then branches out into two directions. Note the statements indentation. In most programming languages this is done to make it more readable. However, in python indentation is a necessity as it clarifies which lines belong to the either the if or else block and which not. As an example, add another print statement to the if block as follows

```python
if x < 4:
    print('That is a small number')
    print('I can print even more…')
else: …..
```

The second line will also be printed if the guard evaluates to true. Any statement indented with a tab after either the *if* or *else* block belongs to that block and will execute depending on the guard. If a statement is not indented the conditional if-else block is over. It will take another to again have the program branch. The else: statement is purely optional. In fact you can also have conditional statements within conditional statements.

So far I have used the mathematical operator <, but from you're high school math's you will know more. Each of the following can be used to as part of a conditional statement. You can chain as many as you like with || which means or, && which means and.

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal
!=	not equal

While and For loops

One of the strengths of computers is that they can perform actions many times over very rapidly. So far our programs have been linear. They execute from top to bottom. Using the conditional statement already added zest as we can define the program to do perform certain actions while skipping others. However with while and for loops we can repeat certain parts of the program until a condition has been. In fact with the while loop a program can in theory run for ever, until I give an input that meets the while loop condition that will cause it to break out of that loop.

A while has three components that are of important: an initialized variable, a conditional statement and an iterator. The structure of the while loop looks as follows

```
Initialized variable;
While (conditional statement)
    do something'
    Iterator
```

When the indentation stops after the while declaration the program can run its course again, at least if the conditional statement has been met.

Below is an actual example of how to use the while loop

```
#!/usr/bin/python
one = 0
while one < 5;
    print "We are at " + one
    one++
print("And now we are out of the loop")
```

I hope you have a good idea of what this code does before you run it. The while loop will run while the value for the variable one is 4 or smaller. Because the conditional returns as true. During each iteration though the loop two lines are run by the computer. The first is the line that prints a small sentence and the variable to the command line. The second adds 1 to variable one. So after one loop the value for one is 1, then it is 2, then 3, then 4. You can also write it as one = one + 1 instead of one++ if you want clarity.

After the variable one reaches value 4 the while statement won't be run. The code will break out and run the next line in the program, in this case a print statement that says "And now we are out of the loop". Remember, all the indented lines after the while statement are part of the loop and will be run until the condition evaluates to false. You can add as many lines as you like to your loop.

Unlike the while the For loop will also execute part of the code repeatedly but do so for a finite amount of time.

Functions

While writing programs you will quickly discover that certain parts will repeat themselves. It would be a lot easier if you could simply write a piece of code and re-use it as often as you like. In Python and other languages such pieces of code are known as functions, in Java they are called methods. In the example below we create a function that prints a sentence to the command line. In the subsequent piece of code that we run we create a function called printme() which is later on called. The way this works is simple, all functions start with the keyword def. Now Python knows this is a function. Python requires that all functions are placed at the start of the file. As you may remember, Python is interpreted. It runs the code on the fly. That means it will need to know about the function printme() before it is run. After the function is written we run it once with the line that just says printme().

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

```
def printme():
print "Yet again we say 'Hello, World!'"
return;

printme()
```

After you have run this little example you may recognize something. The way that the function printme() is called feels similar to the way we have used raw_input(). Indeed raw_input() is also a function that somebody else has already written. Just like raw_input() you can call printme() as often as you like. For an simple example like this we won't write a function. Just using print will suffice. Indeed print is also a function. We write down functions when we are performing an action for which there is no ready-made function and we think we may use it multiple times in a file.

Memorizing functions that have already been written is not easy, you will need to get familiar with Python. After all, you don't want to write code if there is already a function available. The best way to proceed is to find with Google an sample code of what you want to do. In time you will learn to memorize functions. Though I admit I have written plenty of functions only to later discover there was already one available.

Conclusion

That was it for the introduction to programming with python. With the basics covered the world is your oyster!