Introduction to Programming Workshop - Python

Thomas Mortensson and Ben Collins
Computer Science Department,
University of Bristol,
tm0797@bristol.ac.uk, bc0517@bristol.ac.uk

September 25, 2013

Abstract

This tutorial focused class will introduce participants to the core concepts of Python on a Raspberry Pi. Programming learned in this session is transferable to many different platforms and it's use is not limited to the Raspberry Pi Hardware.

Requirements for this programming session are few as the session is designed to be workable on any machine with Python 2.x installed.

1 Introduction to Python - What is it?

2 Programming Python

2.1 Python shell

2.2 Hello World!

In this first example we will build a simple hello world application. The purpose of building this application is to familiarize a new programmer with the python programming interface. The Hello world program simply outputs the words "Hello World!" to the computers display. We generate the hello world program by typing the following into the Python shell:

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

After pressing Enter to execute the line in Python Shell the computer should respond with a line printed on the standard output: Hello World!
The quotation marks around the words signify that this portion of data is a string. More on this in the Types section. You can modify this program to write anything to the screen.

2.3 Variables and Assignment

In the previous section we completed the most basic starter program you will encounter in any programming language. We learnt how to output or print to the screen. We will now extend this by printing the sum of two numbers to the screen. Below is some example code to try:

```
print 2 + 5
```

Can you guess what this will print to the screen? Try running this in your Python Shell, This will not produce the output "2 + 5". Instead as we have removed the quotation marks from the input, Python instead or reading the characters we've typed as a String now interprets the characters as two numbers with an addition symbol. Python will execute this "Expression" and return the correct result of 7.

If we take this a step further we can use something called a variable to store the result of our calculation before we print it. This is shown in the below code:

```
i = 2 + 5
print i
```

We have created a variable called "i" and we set it's value to be 7, we then print the value of i which is 7.

Try the following code:

```
i = 2 + 5
i = i * 3
print i
```

Write here what you expect the program to print to the screen:___

Now run this program in your Python Shell, what is the result?

When we calculate expressions we must be careful to obey the laws of BIDMAS, more on this here: `http://www.bbc.co.uk/bitesize/ks3/maths/number/order_operation/revision/2/`
For Example:

```
i = 2 + 1
i = i * 6
i = i / 3
i = i - 1
print i

j = 2 + 1 * 6 / 3 - 1
print j

k = ((((2 + 1) * 6) / 3) - 1)
print k
```

This program shows the different outputs obtained by using and not using the rules of BIDMAS.

Another form of expression we will cover is called String concatenation. To concatenate two Strings means to join them together. For example if I wanted to concatenate "Hello " and "World!" into "Hello World!" we use a comma between our two strings as demonstrated in this program

```
name = "World!"
print "Hello ", name
```

2.4 Types

When we reference Types in the context of a programming language we are trying to define the format of the data structures we are manipulating. Python as all programming languagues will make use of different types to store different pieces of data. Examples of basic Types are Integer, Float, String and Boolean. An Integer is any number without a decimal component. For example the number 3 is an integer however 3.1 is not. A floating point number (or Float) is a number which does include a decimal point. Examples of a float are -3.1 or 5.0 (Beware of the difference between 5.0 and 5). A String is used to define text, we have used strings previously to define words in our Hello World program. Strings are denoted by the quotation marks that surround them, e.g. "Hello" is a String however Hello is not. Python automatically assigns types to variables when they are used based on what data you have stored in them. We can see what data type Python has assigned by using the function type() (More on functions later on!). This is demonstrated in this example:

```
integer = 5
float_number = 1.2
string = "Ben"
boolean = True
integer2 = int("3")

print integer, type(integer)
print float_number, type(float_number)
print string, type(string)
print boolean, type(boolean)
print integer2, type(integer2)
```

This should print out the types of all the variables you have defined! Note: we are using a system called typecasting to define integer2, this allows us to explicitly force the computer to use a specific datatype for a variable (we are converting "3" - A String $\rightarrow$ 3 - An Integer)

2.5 Producing our first source file!

So far in our Python experience we have been running all of our programs interactively in the Python Shell.

While this may be good for building quick programs or for some brief debug, the vast majority of the time as a programmer you will want to save your work. To build a Python source file we write the commands we have been writing in the shell directly into a blank text file. Once you have written the commands you wish to execute you have to save your program as your_program_name_here.py . I recommend you save your source files in the home directory (This may be called pi on your Raspberry Pi) and you execute them by opening a Linux Terminal and typing

```
python your_program_name_here.py
```

Try creating a source file with your Hello World program in and running it.

2.6 User Input

Many times when you are producing a program in Python you will need to take some form of input from the user. This is done in Python using the raw_input function:

```
name = raw_input("Enter your name: ")
print "Hello", name
```

The function raw_input() takes data written to the standard input (In this case the Linux terminal's text input) and stores it in a local variable. We are storing in a variable called name in this instance. The raw_input() function will store data in what is deemed the relevant datatype. For example if I write a name, type(name) will evaluate to a String. If 3.0 is entered a type(name) will evaluate to float etc. In this code snippet we also use standard output (as in the Hello World program) and String concatenation.

2.7 Exercise 1

For our first exercise we will build upon the concepts we have learned in the first few sections to build ourselves a simple addition machine.
For the next 5 minutues your task will be to design a program that reads in 2 different numbers from standard input (storing the values in variables), produces a result from the addition of the two variables and finally prints out the result of the calculation to standard output.

Sample answers will be given in the session and afterwards will be uploaded to:

2.8 IF Statements and Boolean Logic

So far in our programming tutorial we have focused on mostly static concepts. In a typical program flow control is used to provide a dynamic datapath. All programs will have an Input, Process and Output stage. Without a form of flow control we will always be limited in what we can build. As this is the case we will now introduce the concept of an IF statement. An IF statement is a comparison operator. In Computer Science information is processed as Binary - 0's and 1's, these 0's and 1's represent the fundamental building blocks of how a computer operates. We will not go too deeply into how Binary works however it is of note that a 1 can represent the value True and the value 0 can represent False. An IF statement allows a computer to execute a comparison based upon the conditions set by a programmer. For example the statement "3 > 2" will evaluate to True as the number 3 is larger than the number 2. If we use an IF statement we can chose to begin a branch of execution based upon this result:

```
if(3>2):
        print "3 is greater than 2"
```

We can extend this concept further by also having an else clause, a clause that is executed when the condition tested does not hold true. This is demonstrated in the next piece of code:

```
works = True # Alternate this value
if works:
        print "This Works!"
else:
        print "This Doesn't Work!"
```

You can change the value of True in your source file from True to False to alter the program's output.

A core concept governing how we use IF statements is boolean logic. The 3 main operators as a programmer you will be required to know are how to use AND,

OR and NOT operators. These operators work according to these 3 definitions:

- AND - returns True if both x and y are true, otherwise the value of false.
- OR - returns True if x is True, or if y is True, or if both x and y are true. Otherwise the value is false.
- NOT - returns True if x is False and False if x is True.

This can be seen in the following Truth Tables - You don't need to remember this, just keep it as a reference.

x	y	x AND y
False	False	False
False	True	False
True	False	False
True	True	True

x	y	x OR y
False	False	False
False	True	True
True	False	True
True	True	True

x	NOT x
False	True
True	False

Try the following code by changing the values of apple and orange from True and False to see the effect this has on the and or and not operations.

```
apple = True
orange = False
if apple and orange:
        print "Have both fruits"
if not apple:
        print "Don't have an apple"
if apple or orange:
        print "Have a fruit"
```

Once we understand AND, OR and NOT we can progress to comparison operators. Comparison operators are extremely useful for comparing numbers in IF statements. There are 5 comparisons you will need to be aware of, these are:

Maths	Programming	Description
$<$	<	Less Than
$\leq$	<=	Less Than or Equal
$=$	==	Equal
$\geq$	>=	Greater Than or Equal
$>$	>	Greater Than

We can test these operators in the following interactive program:

```
below_test = 5
above_test = 10
equals_test = 8

current_num = int(raw_input("Num: "))

if current_num < below_test:
        print 1

if current_num <= below_test:
        print 2

if current_num == equals_test:
        print 3

if current_num >= above_test:
        print 4

if current_num > above_test:
        print 5
```

We can see that by using different numbers we can get different return values. Try using the values 3,5,6,8,10 and 11 and see if you can predict what numbers the program will return.

2.9 Loops

We have now covered many base concepts such as variables, expressions, operators, input and output as well as IF statements and Boolean logic. We will now build upon these concepts to introduce the concept of a loop. Computers are useful to us not only due to what they can process but also due to the fact that

we can program them to iterate across large datasets. Computers are a perfect solution in situations where a human would take months or years to complete a particular repetitive task. To repeat a task (or iterate) we require the use of a construct called a LOOP.

Here is your first loop:

```
for i in xrange(0, 10):
        print i
```

This will print out the first 10 numbers starting from 0 and counting up to 9. In Computer Science (and most programming languages) counting is done from 0

A loop is useful however it's usefulness is greatly increased by using some form of comparison within the loop body. This is demonstrated in this code:

```
below_test = 4
above_test = 7

for i in xrange(0,10):
        if i < below_test:
                print i, "- BELOW"
        if i > above_test:
                print i, "- ABOVE"
```

Here we print out BELOW when the iterating variable i is below 4 and ABOVE when it is above 7.

2.10 Exercise 2

2.11 Functions

2.12 Exercise 3

3 Further Reading

3.1 Raspberry Pi Hardware with Python - GPIO