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 session will introduce you to the basics of programming, using the programming language Python and a Raspberry Pi. Programming learnt in this session are transferable since Python is a very popular language - meaning you can use these skills on your home computer, not just your Raspberry Pi.

Requirements for this programming session are few as the session is designed to be workable on any machine with Python 2.x installed. Any Raspberry Pi should already meet this requirement!

1 Introduction to Python - What is it?

2 Programming Python

2.1 Python shell

The Python Shell is a brilliant tool allowing us to interactively start programming. What this means is that we can type commands and get instant results, allowing us to see what each line of our program does. To open the Python Shell look on the Raspberry Pi Desktop for an icon saying LXTerminal and click that. Once that opens you should be presented with a black box which says:

```
pi@raspberrypi ~ $
```

what you need to do is type in "python" and hit enter. You should then be presented with:

```
pi@raspberrypi ~ $ python
Python 2.7.3 (default, Jan 13 2013,
 11:20:46)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or
 "license" for more information.
>>>
```

Congratulations, you're there!

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 how to write a simple program in Python. The Hello World program simply outputs the words "Hello World!" to the computer's 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). Note that we could have named the variable anything - we just chose "i" since it is a traditional name for a variable.

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 IF Statements and Boolean Logic

So far in our programming tutorial we have created programs that simply run whatever you have written. However, in a typical program we may not want to run all lines, but work out which lines should be run depending on what the user does. 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 uses a comparison to control the flow of a program. 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, 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 choose 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 setting the variable "current_num" to the values 3, 5, 6, 8, 10 and 11 and see if you can predict what numbers the program will return.

2.8 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 variable i is less than 4 and ABOVE when it is greater than 7.

We can add a new operator, modulo, into the mix to help us demonstrate the usage of AND, OR and NOT in a loop in our next example. Modulo works by working out the remainder if a number is divided by another number. For example 3 % 2 = 1 because 3 divided by 2 = 2 remainder 1. We can use modulo and check for a remainder of 0 to check if a number is cleanly divisible by another number. You will see what we mean in this next example:

```
for i in xrange(0,50):
# if (i % 2 == 0 or i % 5 == 0):
# if (i % 2 == 0 and i % 5 == 0):
# if (not i % 2 == 0):
    print i
```

In this piece of code we have also introduced comments. A comment is a way of writing a line of code and not executing it, Typically comments are used to document code and make it more readable however we can use them in this instance to turn on or off lines of execution. To comment out a line we add a Hash symbol (#) to the beginning of the line, this will mean that the computer skips that line. When you execute this program as it is it will first output the numbers from 0 to 49, if we remove the comment from the 2nd line this program will output all numbers between 0 and 49 which are even or divisible by 5. If we re-comment the 2nd line and uncomment the 3rd line the program will now output all numbers between 0 and 49 which are even and divisible by 5. This will therefore only print out multiples of 10. If we comment the 3rd line again and uncomment the 4th line we will now print all numbers between 0 and 49 which are not even - i.e. all of the odd numbers. Try this for yourself by changing where the comments are or the numbers the program is checking for divisibility.

2.9 Functions

The last core concept we will try to teach in this document is the concept of functions. So far there has been a bit of repetition in our code - if we wanted to add numbers we had to do this manually each time. With a function call we can split apart different sections of our program. This helps us remove repetition, and allows us to make single large changes very quickly. For example if I build a function that's sole purpose is to add two numbers, I can then use this function in my code. If later I decide I want to instead subtract those 2 numbers in all cases where I have used my function I can change that line in one place rather than having to change every addition symbol to a subtraction symbol (and potentially forgetting some and introducing errors into my code). Functions allow us to logically divide a program to allow for more complex execution. In the simple examples we provide you may see less of a use for functions - however, when you have chunks of code 30 lines long which are similar you will want to start using functions to simplify your work (and save yourself a lot of time!). We have briefly touched on some basic functions likely without you realising. When we write a function we denote it by writing it's name and putting brackets at the end of it's name. If the function takes an input, we put that input between the brackets. We have already used the type() function. e.g "print integer, type(integer)" in the section Types and we have used the raw_input() function in the Variables section. In the following example we will define a function print_hello() (which takes 0 inputs) and we will run it in a loop.

```
def print_hello():
  print "Hello World"

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

We can see that we have offloaded the work of printing hello world to a function rather than calling "print Hello World!" from within our loop.

Expanding from this we can now create a function add() which can take 2 input parameters, we will have each parameter being a number to add. This is shown in the next example:

```
def add(num1, num2):
  result = num1 + num2
  return result

res = 0
for i in xrange(0, 10):
  new_res = add(res, i)
  print res, "+", i, "=", new_res
  res = new_res
```

See if you can guess the program flow before running the program, write on a piece of paper what you think the program will output before running it. This is a simple example of how we can use functions to offload work to the computer. Using a combination of a function and a for loop we have saved us having to write 10 seperate lines of addition. You can see how this can only help more in longer programs.

3 Further Reading

3.1 Raspberry Pi Hardware with Python - GPIO

In this last additional section I will show you how to do some extremely basic hardware interaction using the concepts of Python we have learnt in the tutorial sections. A brilliant website to show exactly what is going on here in more depth is: `http://code.google.com/p/raspberry-gpio-python/wiki/BasicUsage`

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)
GPIO.setup(16, GPIO.OUT)
GPIO.output(16, 0)
off = False
while True:
        if(off):
                GPIO.output(16,1)
                off = False
        else:
                GPIO.output(16,0)
                off = True
        time.sleep(1)
```

The first 2 lines, the import lines, tell Python to load some libraries it has stored (additional code), we use RPi.GPIO to interact with the GPIO pins and time to cause a delay with

time.sleep(1).

In the next 3 lines we have to setup the GPIO pins to allow us to use them, the first line tells the program what configuration the pins are in (Pinouts can be found here /urlhttp://www.hobbytronics.co.uk/raspberry-pi-gpio-pinout), the second line tells the program to set PIN16 as an output and the 3rd line tells us to set the initial value of PIN16 to 0 (OFF). We then have a loop which cycles PIN16 from OFF to ON and back with a time delay of 1 second. We use a Boolean "off" to keep track of the state of the LED light so we know to turn it on or off within our loop. The overall output of the program should be a blinking LED if you attach an LED correctly to PIN16 and GROUND.