Getting started with Python at Raspberry Pi

Raspberry Pi 3 comes with **NOOBS** – New Out Of the Box Software. You can purchase a pre-installed NOOBS SD card from many retailers.NOOBS is an easy operating system installer which contains **Raspbian**. It also provides a selection of alternative operating systems which are then downloaded from the internet and installed.

Raspbian is a free operating system based on **Debian** optimized for the Raspberry Pi hardware. An operating system is the set of basic programs and utilities that make your Raspberry Pi run. Debian is a Unix-like computer operating system that is composed entirely of free software, most of which is under the **GNU General Public** License and packaged by a group of individuals participating in

the Debian Project. The **GNU General Public License** (GNU GPL or GPL) is a widely used free software license, which guarantees end users the freedom to run, study, share and modify the software.

As you boot up the Raspberry Pi, on the desktop you should see a Raspberry button in the top left. Click on this to see the different types of applications that come pre-installed on NOOBS. Next you want to click on Programming, and then click on Python 3 (IDLE).

A new window should open up on your desktop. This program is called an IDE, which stands for Interactive Development Environment. This particular IDE is called IDLE.

The window that has opened is known as a shell. In the shell you can type Python code, and it will be executed straight away. Have a go at typing the following lines into the shell. Don't worry about the three chevron symbols (>>>), as these will be added for you by IDLE.

>>> print('I am writing Python')

>>> 6+6

>>> #This is just a comment, it doesn't do anything

The shell is useful for testing a few lines of code, or interacting with programs you have written, but to actually write a computer program, you need to write code into a file.

Another example is:

word1 = 'Hello '
word2 = 'World'
print(word1 + word2)

Click on File and New File, and a second window will open. It's useful to position them side-by-side like this:



Simple reaction game using inputs, outputs and variables

We can start with a very simple program that involves inputs and outputs and variables

print('Quick, the Enter key')
input()
print('That was fast')

Save your file (File > Save As...) and call it game.py. To run the file you can just press F5 on your keyboard.

The print functions are a type of output. You are instructing the computer to display some text on the screen.

The input function takes input from the user's keyboard presses.

Let's make the program a little more interesting.

We could make the program pause for a little bit before it starts the game. Handling time can be a little tricky however, but luckily for us the code to interface with the computer's clock has been written for us and packaged up into a nice little module that we can use. To start with we only need one function from the time module. It's called the sleep function.

from time import sleep
sleep(5)
print('Quick, press the Enter key')
input()
print('That was fast')

Here you have imported the sleep function from the time module and then instructed the program to pause for five seconds.

Save and run your code to have a play with your new game.

It would be great to know how quick the user was between seeing the message and pressing the key. Again, the time module can be used for this, and in particular a function called time (confusingly!). We can add the import to our first line.

from time import sleep, time sleep(5) print('Quick, the Enter key') input() print('That was fast')

The time function will get the number of seconds that have elapsed since January 1st, 1970. By storing this value in a variable when the message is displayed, and then subtracting the time again, after the user has pressed a key, we can work out their reaction time. Here, the reaction time value is stored in the variable reaction_time. You can't have spaces in variable names, which is why we often use underscores.

from time import sleep, time
sleep(5)
start = time()
print('Quick, hit the Enter key')
input()
reaction_time = time() - start
print('You took', reaction_time, 'seconds')

Save and run your code then have a play.

Working with loops

In Python you can use loops to make lines of code repeat. There are two types of loop - while loops and for loops. Below you can see examples of both.

```
while True:
print('Hello')
```

input()

This is a **while** loop which will repeat the indented lines of code until the condition is no longer true. In this case the condition is just the value *True* which will always be *True*, therefore this loop can never exit.

Another example of While True

Another example of While loop

while True:

```
x=input('Type at least four characters or Type q to Quit ')
if x== 'q':
    break
if len(x)<4:
    print('You typed less than 4 character')
if len(x)>4:
    print('You typed more than 4 character')
if len(x)==4:
    print('You typed four character')
continue
```

Another example of loop

```
for i in range(10):
print('Hello')
print(i)
input()
```

This **for** loop is a finite loop and will loop a fixed number of times. For each value from 0 up to (but not including) 10 it will assign the value to the variable *i* and carry out the indented lines.

Notice that both loops need colons (:) adding to the end of the line, and four spaces then need to be placed before the lines of code that are to be repeated. Try both of the mini programs out to see what the difference is between the two loops.

HINT : If you need to stop your program running you can press Ctrl + C to stop it

- Can you use loops in your game to make it repeat?
- Can you use variables to store the total reaction time after several games?
- Can you work out the average reaction time?

Note : The following link also gives some basic guide to using Python in Raspbery Pi

https://www.raspberrypi.org/documentation/usage/python/

General Purpose Input Output on the Raspberry Pi

The GPIO pins are a way in which the Raspberry Pi can control and monitor the outside world by being connected to electronic circuits. The Pi is able to control LEDs, turning them on or off, or motors, or many other things. It is also able to detect the pressing of a switch, change in temperature, or light etc. We refer to this as physical computing.

<u>GPIO</u>

Most models of the Raspberry Pi have 40 pins that look like this:



These pins are a physical interface between the Raspberry Pi and the outside world. You can program the Raspberry Pi to switch devices on and off (output), or receive data from sensors and switches (input). Of the 40 pins, 26 are GPIO pins and the others are power or ground pins (plus two ID EEPROM pins which you should not play with unless you know your stuff!)



GPIO Pin Numbering

When programming the GPIO pins there are two different ways to refer to them: GPIO numbering and physical numbering. Throughout this course (and in all our resources) we will refer to the pins using the GPIO numbering scheme. These are the GPIO pins as the computer sees them. The numbers don't make any sense to humans, they jump about all over the place, so there is no easy way to remember them. However, you can use a printed reference or a reference board that fits over the pins to help you out.



Building a simple circuit

To build the circuit you just learned about, in this next step, you're going to need your breadboard, an LED, a 330 ohm resistor, and two female-to-male jumper leads.

The first thing to do is to have a look at the GPIO pins on the Raspberry Pi. The two pins we are going to use to begin with are labelled on this diagram. They are called 3V3 and GND.

• 3V3 means 3.3 Volts. You can think of this as the amount of energy that is going to be provided to the circuit. This pin can act a little like the positive side of a standard battery.

• GND means Ground. For a circuit to be complete, electric current must always be able to flow to a ground pin. This is a little bit like the negative side of a standard battery.



- Take one of your female-to-male header leads and connect the female end of it to the 3V3 pin on the Raspberry Pi.
- Next plug the male end into your breadboard in row 1.

As you can see in the picture below, if you were to remove the back of a breadboard your would see that every hole of each row is connected by a small strip of metal, that act just like wires in the circuit.



Now have a good look at your LED. This is not a symmetrical component. You should see that one leg is longer than the other. The long leg is sometimes called the **anode**, and this leg should always be connected to the positive side of a circuit. One way to remember this is to imagine the longer leg as having had something added and the shorter leg has had something taken away. Sometimes the LEDs might have legs the same length, in which case you can tell which side is the anode because the plastic rim of the LED will be round, whereas the negative side (called the **cathode**) is slightly flattened.

- Push the long leg of the LED into row one, close to the ravine. Place the shorter leg into row one on the other side of the ravine.
- Now find your resistor. A resistor is a non-polarised component, so it doesn't
 matter which way around it goes. Push one leg into the same row as the shorter
 of the LED legs, so it connects to the LED, and the other leg into any other free
 row.

• Now take another female-to-male header lead and push the male end into the same row as the resistor's second leg. Your circuit should look a little like this:



• The last step is to connect your components to the ground pin. Make sure that your Raspberry Pi is powered on and then take the female end of the jumper lead and plug it into your ground pin.



- If your LED doesn't light, then try the following things:
 - 1. Check your Raspberry Pi is on.
 - 2. Check all your components are firmly in the breadboard.
 - 3. Check your LED is the right way around.
 - 4. Try another LED.
- You're introducing another layer of complexity and another area where things could go wrong. By testing you know that your LED works and if you encounter problems later you should be able to eliminate the LED as the cause.

Raspberry Pi could act like both the cell and the switch, but your circuit the pin we're using (3.3V) is always on, but other pins can be switched on and off. By making a subtle change to you circuit and writing some code you can control the behaviour of your LED.

The first thing to do is to remove the female-to-male jumper lead from the 3V3 pin. This pin provides 3.3 Volts, no matter what, and can't be controlled. There are plenty of GPIO pins on the Raspberry Pi that can be controlled however, and for the purposes of

this exercise, we're going to use pin 17. Connect the female-to-male jumper lead to GP17, as shown in the diagram.



If all is going well then your LED will have turned off. This is because at the moment, pin 17 is in an *off* or *low* state. To turn on the LED, we'll need to set pin 17 to be *on* or *high*. When the pin is in a *high* state, it will provide 3.3 Volts to the circuit. To change the state of the pin, you're going to need a few lines of Python code, which you'll do in the next step.



Blinky blinky lights

We're going to use the *IDE* - IDLE, to write and run the code. Open IDLE by clicking on Menu > Programming > Python 3 (IDLE) on your Raspberry Pi.

Click on File > New File to open a new window. Save this straight away (File > Save As..) and call it blink.py.

We used the time module to help write a simple program, and you imported the sleep function. This week you're going to use a module called gpiozero. This module gives you access to the GPIO pins on the Raspberry Pi. You don't need to use the whole of gpiozero in your program, but only a small unit of code called a **class** from the module that helps you control LEDs. At the top of your file write:

from gpiozero import LED

The next step is to create a new **object** for the specific component and pin we are using. Here we've called the object red_led, but the name you give it doesn't really matter. This object can then receive different commands to control it's behaviour.

 $red_led = LED(17)$

Save and run your program. You can do this by clicking on File > Saveand then Run > Run Module, or by using the shortcut keys - ctrl + sand then F5.

Nothing will happen right away. Click into the shell (the first window that opened in IDLE) and you can now pass some different commands to your red_led object. Try the following two commands.

>>> red_led.on()

>>> red_led.off()

Your led should turn on and off each time your type the lines. Let's try and automate that a little. Back in the blink.py file you can import the time module, and specifically the sleep function. Alter you code so it looks like this:

from gpiozero import LED

from time import sleep

 $red_led = LED(17)$

Now you can instruct your program to repeatedly turn the LED on and off again, by using an infinite loop.

while True:

```
red_led.on()
sleep(1)
red_led.off()
sleep(1)
```

Save and run your program, and see what happens. What does changing the amount of time passed to the sleep function do? What's the smallest sleep time you can use?

A little bit of abstraction

The gpiozero module has some built in methods to give you even more control of your LED. These are abstractions. Complexity has been hidden away, making your code easier to write and more readable.

Either remove your while True: loop or copy and past your code into a new file so that it looks like this:

from gpiozero import LED
from time import sleep
red_led = LED(17)

Now you can try adding the following function calls to your program. Try them one at a time and play with the numbers to see what they do.

red_led.blink(0.1, 0.2) red_led.blink(0.2, 0.1, 5) red_led.toggle() sleep(1) print(red_led.is_lit) red_led.toggle() sleep(1) print(red_led.is_lit)

Applied Activity : Lights, sequences and coded messages

Now that you can control a single LED with code there's loads you could do, questions to answer and things to make.

Here are a few ideas of extra challenges you could set yourself and use with students:

- 1. Experiment with the frequency of the LED flashes, what's the fastest you can make it flash? Can you make it flash randomly?
- 2. Can you create short (dot) flashes and long (dash) flashes, giving you the basics of Morse code, with this can you broadcast a message?
- 3. Can you add extra LEDs to your breadboard and control them with other GPIO pins?
- 4. With multiple LEDs could you create a simple light sequence in code, this could be something functional like a traffic light sequence or something fun like some blinky disco lights.

Codes for accessing General Purpose input output pins of Raspberry Pi

http://gpiozero.readthedocs.io/en/stable/index.html

Acknowledgement:

The lab has been developed using the material available on the internet by James Robinson of the Raspberry Pi Foundation.