centre for computing history _

A Beginners' Guide to Raspberry Pi

www.ComputingHistory.org.uk

WHAT IS THE RASPBERRY PI?

Features

- It is a low-cost, credit-card sized computer developed in the UK by the Raspberry Pi Foundation.
- It has been designed with education in mind to enable people of all ages to explore computing.
- It has the ability to connect to the Internet, a high definition screen, a keyboard, a mouse, USB disk drives and even has individual input and output pins that can be connected to lights, motors and switches (and more).
- It uses an operating system called 'Linux' which is free and open-source software.
- It comes equipped with three programming languages - Scratch, Python and Java.

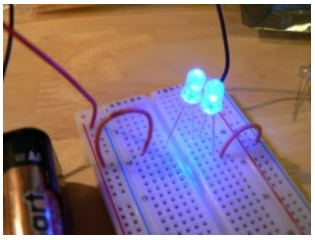
A Bit of History

The Raspberry Pi uses an ARM processor. The ARM processor is used in most mobile phones and a vast number of other mobile devices, hard drives, digital cameras and televisions. ARM is a British company whose roots can be traced directly back to Acorn - the company that produced the BBC Microcomputer used in schools across the country back in the 80s. Acorn was based in Cambridge and ARM is still in Cambridge today.

The museum has Serial Number 7 Raspberry Pi (Model B+) and Serial Number 3 Raspberry Pi2 in its collection.

What can you do with it?

Make music



Play games/build games

Control little light bulbs and switches, and even create moving robots!

DOWNLOADING THE SOFTWARE

Raspbian is the Raspberry Pi Foundation's official supported operating system.

The Raspberry Pi, even though a computer, does not have a hard disk to store information. Instead, it relies on an SD card with the operating system and other programs installed on it.

We, therefore, need an SD card that the PI can 'boot' from. SD Cards are very common and you may even have a spare one left hanging around somewhere. They are commonly used in digital cameras and media players.

You have TWO options for the SD card :

Option 1 - Create your own card

You can download the current software from here onto your computer:

http://www.raspberrypi.org/downloads

Click on NOOBS (New Out Of the Box Software). NOOBS installation contains Raspbian. Download the 'NOOBS' .ZIP file onto your computer.

The next step is to transfer the NOOBS unzipped folder onto an SD card that will go into the Pi.

Before transferring the folder, however, you will need to format the SD card. For this, you will need to download the SD Card formatter tool from here :

https://www.sdcard.org/downloads/formatter_4/

Install the formatter tool on your PC, run it and follow the on-screen guide to format your SD card. We HIGHLY recommend that you use an 8 GB (or larger) card. A 4 GB card whilst technically suitable will make installing the software much more difficult.

Option 2 - Buy a ready formatted card

Much easier! It will save you an hour of downloading and fiddling about.

You can buy them from many online websites like Pimoroni and The Pi Hut.

The EASY way !!

centre for computing history

LET'S GET THIS PARTY STARTED…

Now we have the SD card ready we can start putting together a working system.

We'll need :

- A Raspberry Pi (Obviously!)
- An SD Card (The one with the operating system installed)
- A Micro USB Power Adaptor
- A USB Keyboard
- A USB Mouse
- A Monitor
- A Monitor Cable

Few things to note:

The Power Adaptor

The Raspberry Pi uses a **standard 5V adaptor with a micro USB plug**. They should cost about £7. Be wary of very cheap imported £1.99 power adaptors on eBay.

The Keyboard and Mouse

Most USB keyboards and mice will work fine, but note that older PS2 keyboards and mice will not work (unless you have an adaptor). The Raspberry Pi2 and Pi3 have four USB ports, and you can plug in your mouse and keyboard to any two of these ports. There'll be spare sockets to plug in a USB hard disk, Bluetooth adaptor or Wi-Fi dongle [NOTE: Pi3 has in-built WiFi and Bluetooth]. If you want more USB sockets, you can also buy a USB hub.

The Monitor

The Raspberry Pi needs an HDMI or DVI monitor. You just have to make sure you buy the correct cable to connect it to your Pi.

If your monitor has an **HDMI input** then use an HDMI to HDMI cable. →

← If your monitor has a **DVI input** then use an HDMI to DVI cable.

The Raspberry Pi can also work with TVs with a composite video input, but this is not a very good quality picture and not recommended for programming.

The Raspberry Pi will not work with a VGA monitor unless you buy an adaptor. You can buy an active adaptor for about £15 - £20 that will work, but beware of the cheap £5 passive adaptors - they will not work.

PLUGGING IN THE PI!

Connecting up the Raspberry Pi is pretty straight forward:

- Connect the HDMI to DVI or HDMI to HDMI cable to the monitor: HDMI end plugs into the Pi;
- Connect the monitor power cable to the monitor;
- Insert the SD card into the Pi;
- Plug in the power adaptor into the micro USB socket on the Pi;
- Plug in the monitor and Pi power cables into the mains.

Switch the Raspberry Pi and the monitor on. The Raspberry Pi does not have a power switch so you'll have to plug in and unplug the power adaptor to turn it on and off.

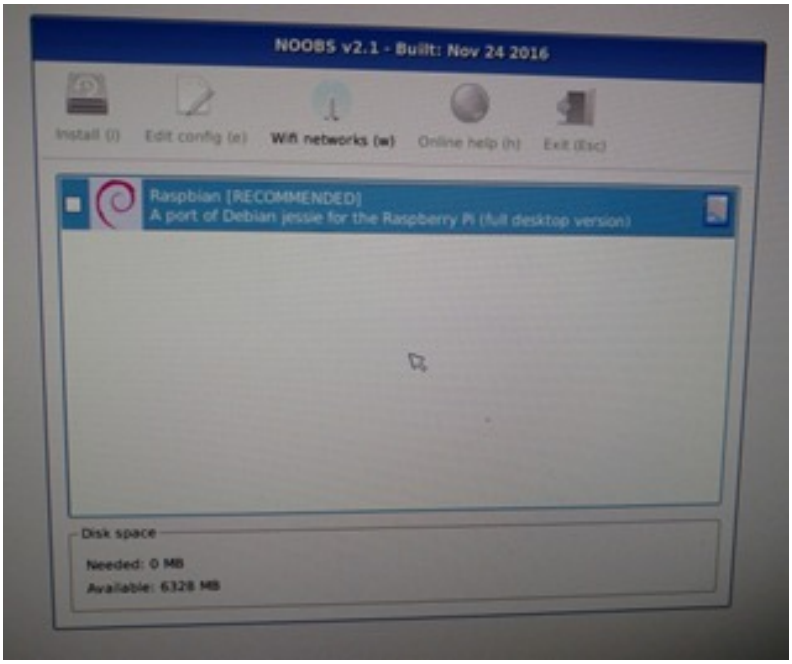
After some booting up text, you should be faced with the screen shown here on the right.

Tick the Raspbian and click the install icon.

Simply agree to all the prompts!

Hang around for a bit ... installation will take a good 20 minutes.

Once the installation is complete you can click on OK and the Raspberry Pi will re-boot.



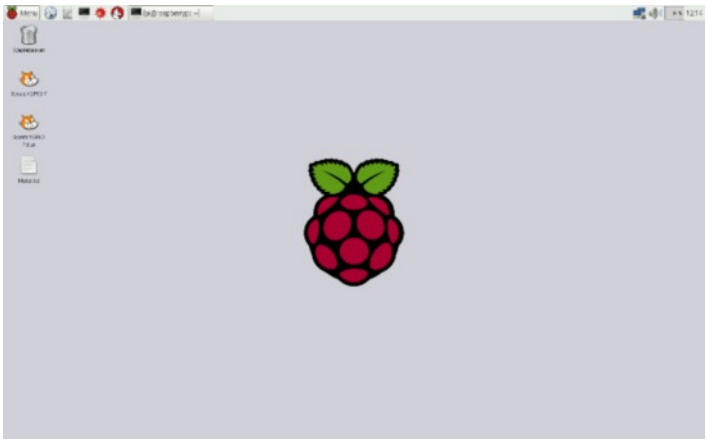
Raspbian is a variation (also known as a flavour or a distribution) of Linux. It is a special distribution designed to run on the Raspberry Pi.

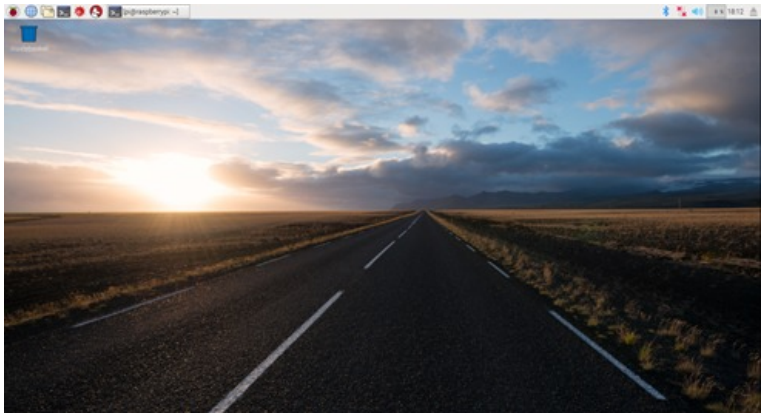
Linux is a UNIX like operating system that is free and open-source.

The recent Raspbian installations have removed the need for the user to interact at all with the UNIX 'shell'.

BOOTING UP

With the old Raspbian installation, you'll see lots of text scroll down the screen when the Pi re-boots and then you'll be straight to the GUI.





With the new Raspbian installation, you will be taken to the PIXEL desktop.

Hey ... it's a bit like Windows!

Yes! Yes it is ...

Move your mouse, point at things and click on them and great stuff happens!

You can work with two programming languages that the standard Raspbian installation comes with:

python™ SCRATCH

Note : The Raspberry Pi, although very powerful for its size, is not as powerful as your desktop computer. When you click on something, be patient, it will load in a few seconds. Some programs and operations take longer to operate than others.

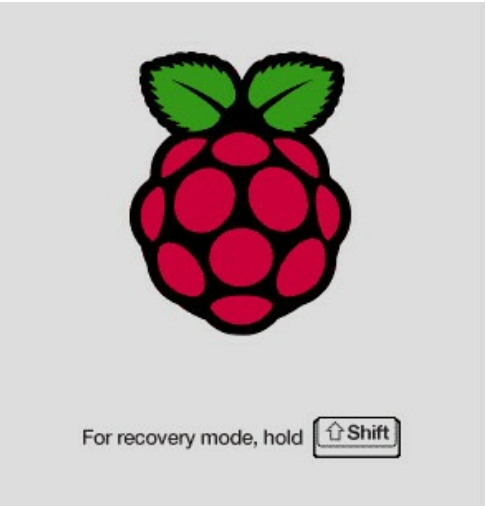
Useful Hint : When you double click on an icon watch the CPU activity meter on the top right-hand corner.

DISASTER RECOVERY

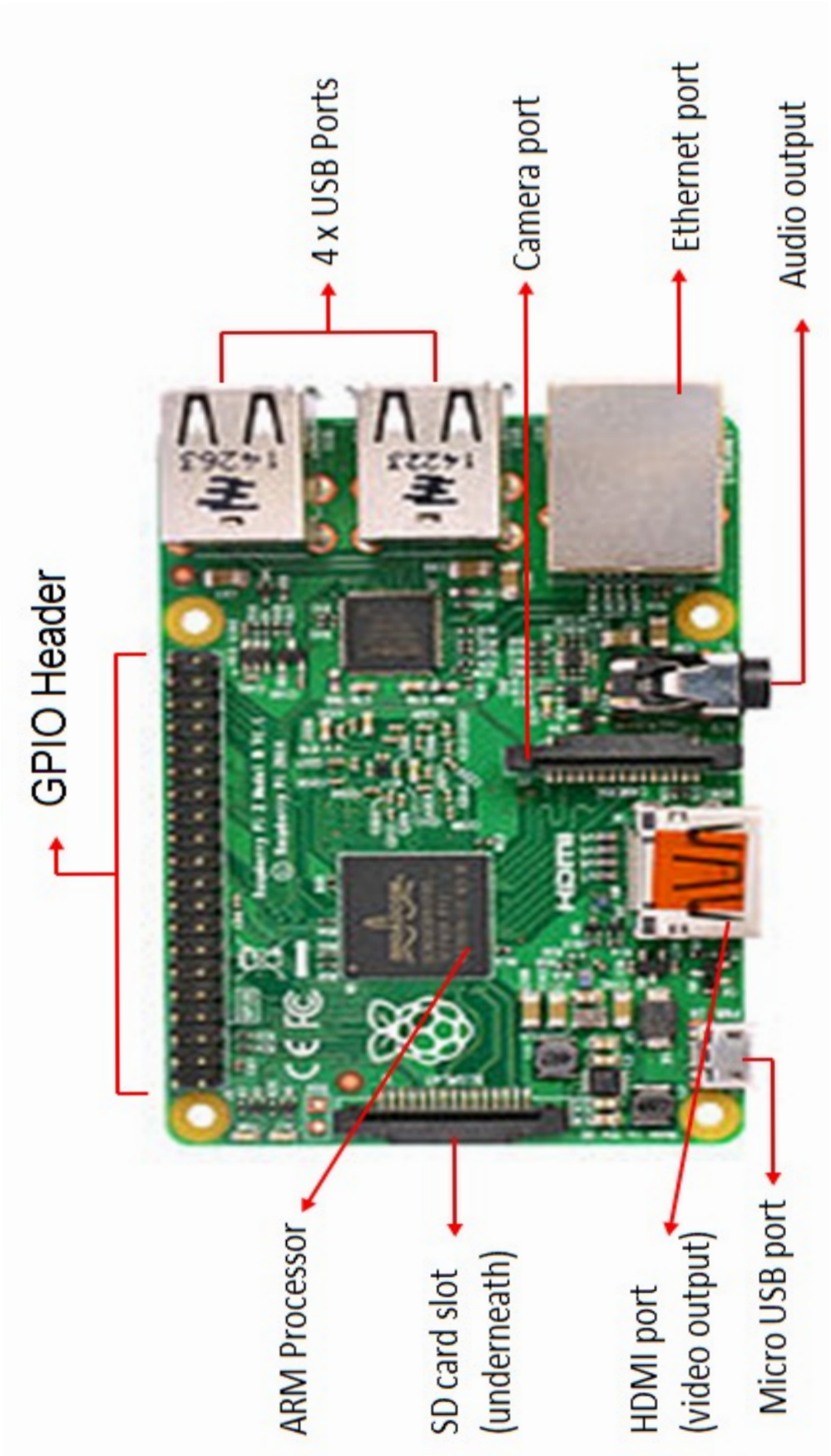
A GREAT feature of the NOOBS distribution is the recovery feature.

If the installation does not go according to plan, you can simply hold down the shift key at boot up and it will take you to the installation page where you can start all over again.

Very useful!
BUT … IT DOES DELETE ALL DATA



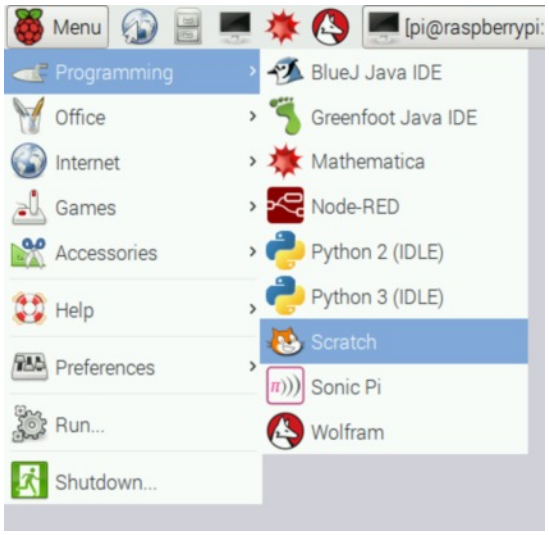
PARTS OF THE PI



centre for computing history

SCRATCH AND PI

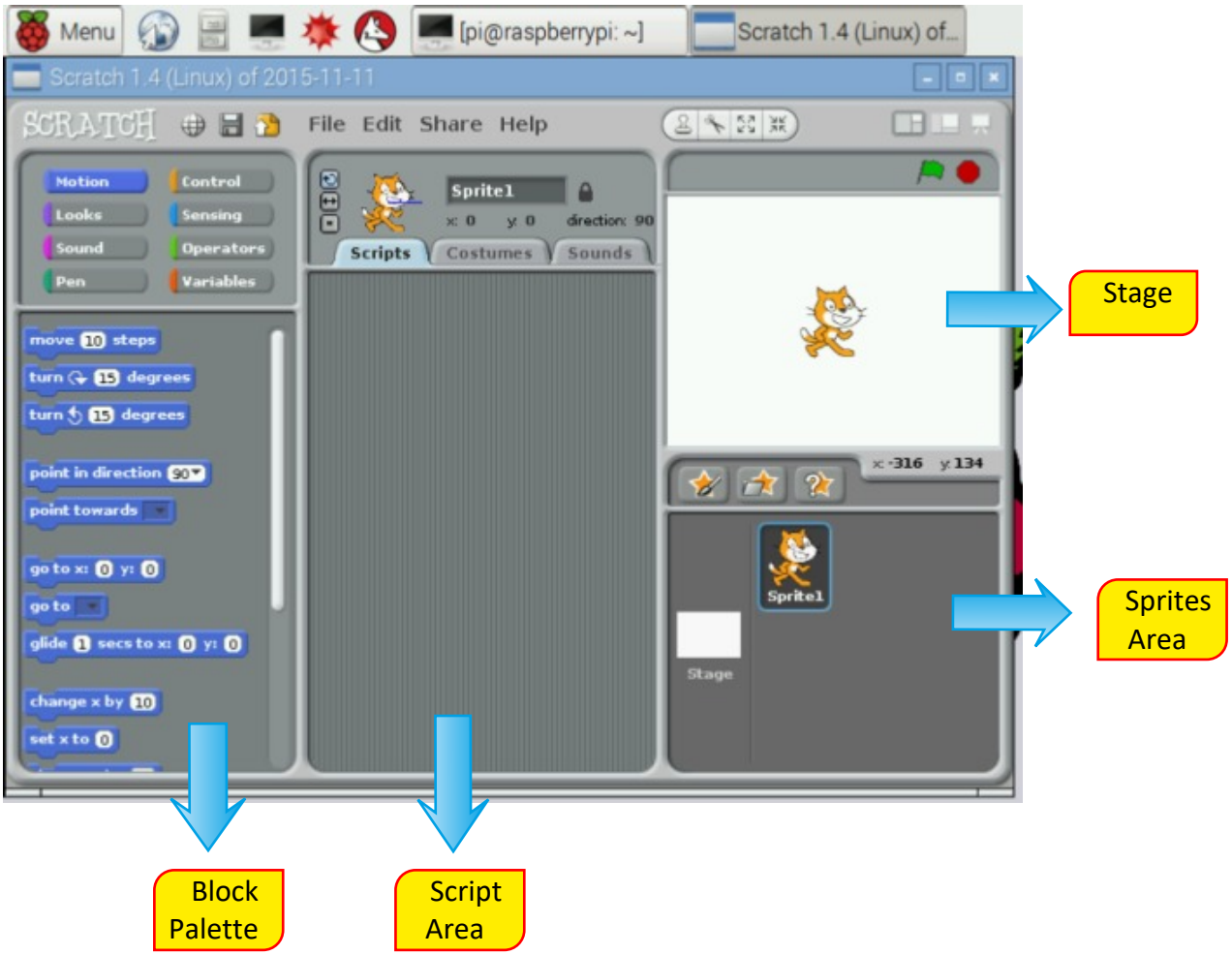
Scratch is a visual programming tool that allows you to learn basic programming skills without having to type in any code. You can create games, animations, and even control external objects (like LEDs, switches, etc.) using just a drag-and-drop interface.



Open the applications menu (**Menu** button) and select **Programming**

Click on **Scratch**

After a few seconds the Scratch interface will load and can start creating your first program…



LET'S SCRATCH!

Say Hello to Scratch the Cat!

Scratch the Cat is an example of a sprite. Sprites are objects that your program can control - you can make the sprite move, you can make it say things, you can change the way it looks, etc.

If you don't like cats, you can choose another sprite from the Sprite Gallery or create your own!

Choose a sprite from the Sprite Gallery

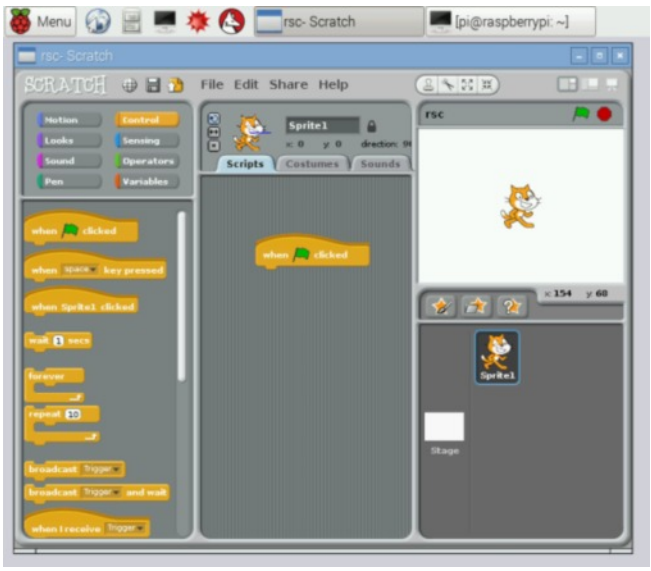
Paint a new sprite

Hello World!

Let's write a program or a **Script**, as they call it in Scratch. The first thing we can do is get Scratch the Cat to say "Hello World".

To do this…

We need a **Green flag** to start the script. Click on the **CONTROL** tab in the **BLOCK PALETTE** and drag a ***when Green flag clicked*** block on to the Script area.



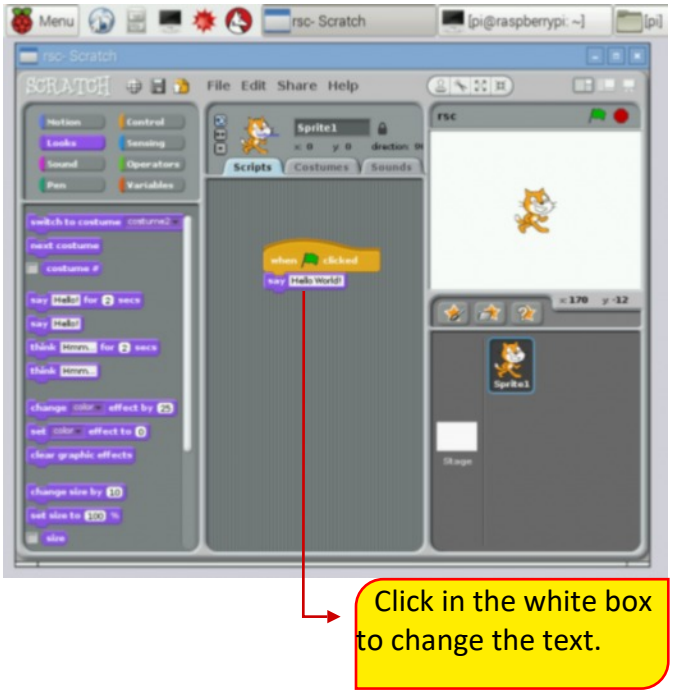
centre for computing history

LET'S SCRATCH

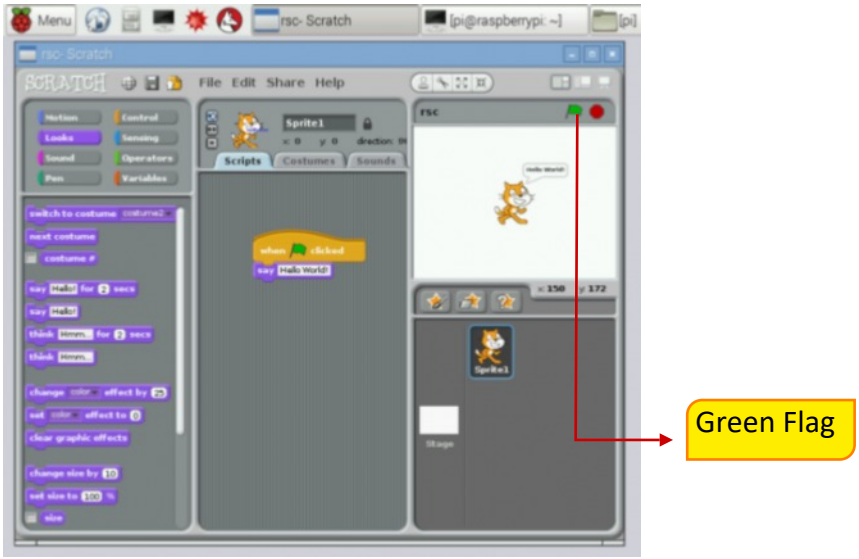
Now, click on the **LOOKS** tab in the **BLOCK PALETTE** and drag a ***say*** block to the Script area.

Attach the say block to ***when Green flag clicked*** block.

In the say block, change the text to 'Hello World!'.



Now click on **Green flag (top right-hand corner)** and Scratch the cat should say 'Hello World!'.



CONTROLLING THE REAL WORLD

Let's talk Electronics…

Lights, switches, motors, etc. connected to a Pi are **EXTERNAL CONNECTIONS**.

To 'talk to' **EXTERNAL** objects, we need to use the **PINS** you can see on the Pi. These are called **IO PINS (**where IO stands for **INPUT/OUTPUT)**.

Map of the group of IO Pins

GPIO #/ BCM #	Name	Pin #			Pin #	Name	GPIO #/BCM #
	3.3V DC Power	1			2	5V DC Power	
2	GPIO 2 SDA1 (I2C)	3			4	5V DC Power	
3	GPIO 3 SCL1 (I2C)	5			6	GND (Ground)	
4	GPIO 4 GPCLK0	7			8	GPIO 14 TxD (UART)	14
	GND (Ground)	9			10	GPIO 15 RxD (UART)	15
17	GPIO 17	11			12	GPIO 18 PCM_CLK/PWM0	18
27	GPIO 27	13			14	GND (Ground)	
22	GPIO 22	15			16	GPIO 23	23
	3.3V DC Power	17			18	GPIO 24	24
10	GPIO 10 MOSI (SPI)	19			20	GND (Ground)	
9	GPIO 9 MISO (SPI)	21			22	GPIO 25	25
11	GPIO 11 SCLK (SPI)	23			24	GPIO 8 CE0 (SPI)	8
	GND (Ground)	25			26	GPIO 7 CE1 (SPI)	7
	GPIO 0 ID_SD	27			28	GPIO 1 SCL0	
5	GPIO 5 GPCLK1	29			30	GND (Ground)	
6	GPIO 6 GPCLK2	31			32	GPIO 12 PWM0	12
13	GPIO 13 PWM1	33			34	GND (Ground)	
19	GPIO 19 PCM_FS/PWM1	35			36	GPIO 16	16
26	GPIO 26	37			38	GPIO 20 PCM_DIN	20
	GND (Ground)	39			40	GPIO 21 PCM_DOUT	21

There are 40 pins in this group:

- 20 on the **outside** - **even numbered pins** (e.g., 2, 4, 6, 8, …)
- 20 on the **inside** - **odd numbered pins** (e.g., 1, 3, 5, 7, …)

Each pin has a function; some are **POWER pins**, some are **GROUND pins** and some are **GENERAL PURPOSE INPUT OUTPUT (or GPIO) pins**.

In the map:

- Pin# - Pin number
- Name - Pin name or function
- GPIO#/BCM# - GPIO number or BCM (Broadcom) number(e.g., Pin number 10 is GPIO 15)

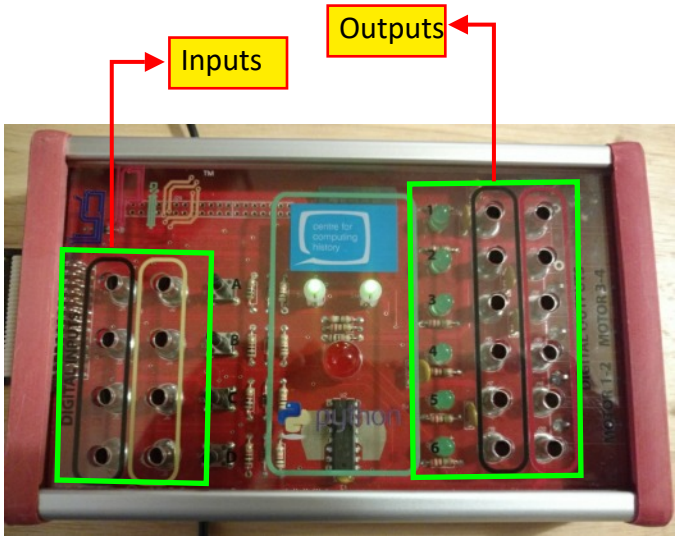
PI AND THE gPiO BOX

What does the gPiO box do?

- **CONTROL BOX** to be used with the Pi
- Helps you to connect external components:

  LEDs, switches, sensors, buzzers…
- You can create amazing projects!

What are Inputs and Outputs?



The gPiO box has **PINS** or **PORTS** called **INPUTS** and **OUTPUTS**.

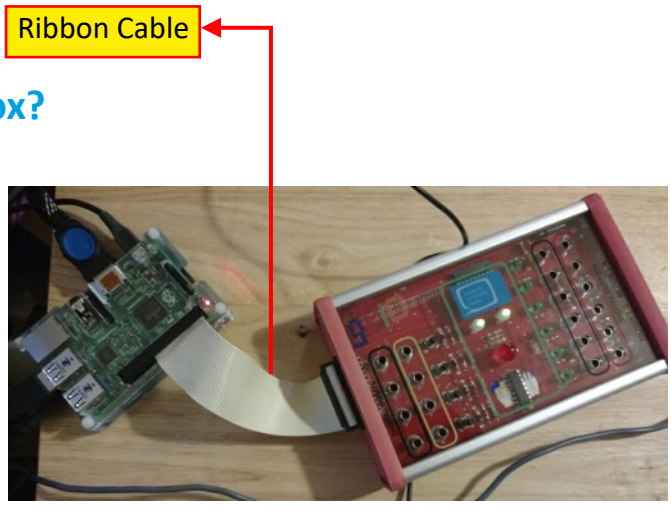
INPUTS let you connect components that **send information to the computer**.

- Examples: switches, sensors, etc.
- **Four inputs** on the gPiO box: **A, B, C, D**

OUTPUTS let you connect components that **receive data from the computer** and **can be turned ON or OFF** based on this.

- Examples: LEDs, buzzers, motors, etc.
- **Six outputs** on the gPiO box: **1, 2, 3, 4, 5, 6**

How do you connect the Pi and gPiO box?



Using a ribbon cable, as is shown in the picture.

The ribbon cable connects the **GPIO pins on the Pi** to the **different components on the gPiO box**.

PI AND THE gPiO BOX: CONTROLLING OBJECTS

Which pins on the Pi do we use?

Function	GPIO PINS (on Raspberry Pi)	gPiO box Ports
OUTPUTS Controls the gPiO box LEDs or you can connect external LEDs, buzzers, etc.	**17**	**1**
	18	**2**
	27	**3**
	22	**4**
	23	**5**
	24	**6**
INPUTS Send an input signal to the Pi using the push buttons on the gPiO box or you can connect external switches, sensors, etc.	**4**	**A**
	14	**B**
	15	**C**
	25	**D**

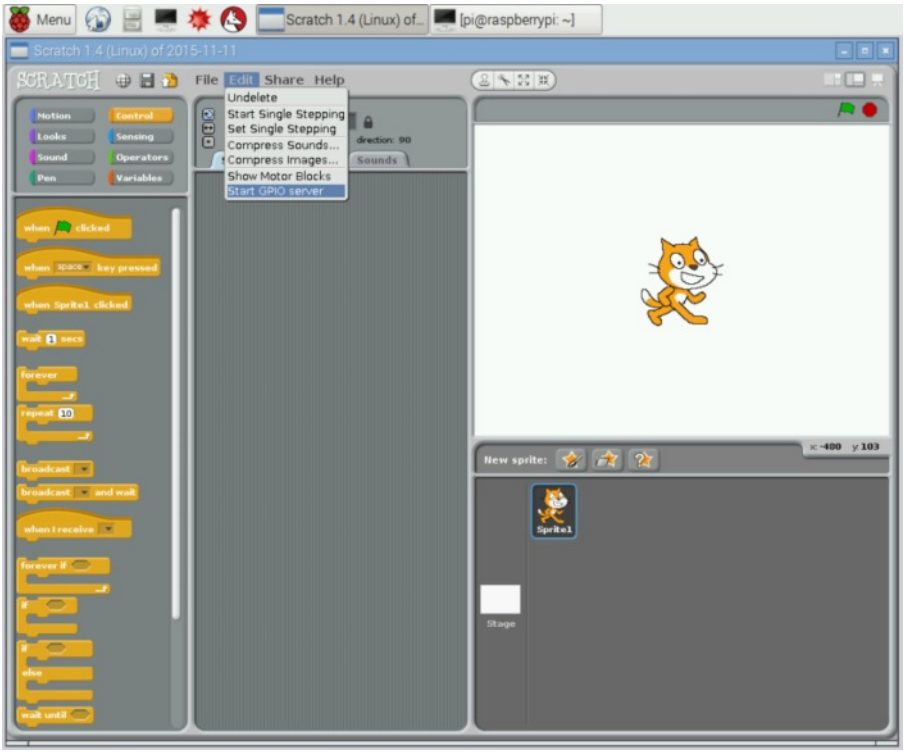
Scratch and gPiO box

In order to 'talk' to the GPIO pins on the Pi using Scratch, we will need to run the Scratch GPIO server.

To do this:

- Click on **File** and then **New** to start a new script;
- Click on **Edit** and then **Start GPIO server**;

You're ready to program the gPiO box!



We are going to start by controlling one output…

centre for computing history

CONTROLLING OUTPUTS

To flash one of the LEDs on the gPiO box, we need to:

- Use a **Green Flag** to start your script:
  - ❑ Use the Scratch block: ***when Green flag clicked*** [under the **CONTROL** tab].

- **Configure** the Raspberry Pi GPIO pin:
  - ❑ Tell the Pi we are using the pin as an **Output** pin.
  - ❑ Use the Scratch block: **broadcast** [under the **CONTROL** tab]**.**
  - ❑ Use the code: **config[PinNumber]out.**
  - ❑ Example: **Output 1** on the gPiO box is connected to **GPIO Pin 17** of the Raspberry Pi. So, we need to configure **PinNumber 17**. The code then becomes: **config17out**.

- Turn the pin **ON**:
  - ❑ This will switch on the LED connected to the pin.
  - ❑ Again, use the Scratch block: **broadcast.**
  - ❑ Use the code: **gpio[PinNumber]on.**
  - ❑ Example: To switch on the LED on **output 1** of the gPiO box which is connected to **GPIO Pin 17** of the Raspberry Pi, we use the code: **gpio17on**.

- **Wait** for some time
  - ❑ We need to wait for some time so that we can see the LED switched on.
  - ❑ Use the Scratch block: **wait** [under the **CONTROL** tab]**.**
  - ❑ Example: Let us wait for 1 second.

- Turn the pin **OFF**:
  - ❑ This will switch off the LED connected to the pin.
  - ❑ Again, use the Scratch block: **broadcast.**
  - ❑ Use the code: **gpio[PinNumber]off.**
  - ❑ Example: To switch off the LED on **output 1** of the gPiO box which is connected to **GPIO Pin 17** of the Raspberry Pi, we use the code: **gpio17off**.

- Again, **wait** for some time
  - ❑ We need to wait for some time so that we can see the LED switched off.
  - ❑ Use the Scratch block: **wait**
  - ❑ Example: Let us wait for 1 second.

centre for computing history

CONTROLLING OUTPUTS

The complete script to flash the **output 1 LED** on the gPiO box looks like this:

CAN YOU TRY THESE?

- Can you make the **output 1 LED** flash on and off continuously?
- Write a script to flash **output 2 LED** on the gPiO box on for 2 seconds and off for 1 second.
    - HINT: **GPIO Pin 18** on the Raspberry Pi is connected to the **output 2 LED** on the gPiO box.
- Now, switch on **output 1**, **output 3**, and **output 6 LEDs** on at the same time for **2 seconds** and then off at the same time.
    - HINT: Look at the little note card stuck on your table to find out which Raspberry Pi pin is connected to which output of the gPiO box.
- Finally, do this:
    - Switch on **output 2 LED**.
    - Switch on **output 3 LED** after 3 seconds.
    - Switch off **output 2** and **output 3 LEDs** after 1 second.
    - Make this sequence happen 3 times.

SENSING INPUTS

Let's now read an input , i.e., read the status of the push buttons on the gPiO box.

First, we need to write a new script. To do this:

- Click on **File** and then **New** to start a new script;
- Click **Yes** if asked to save the current project;
- Give your old script a name and click OK. This should close your old script and open a new empty script;
- Click on **Edit** and then **Start GPIO server**;

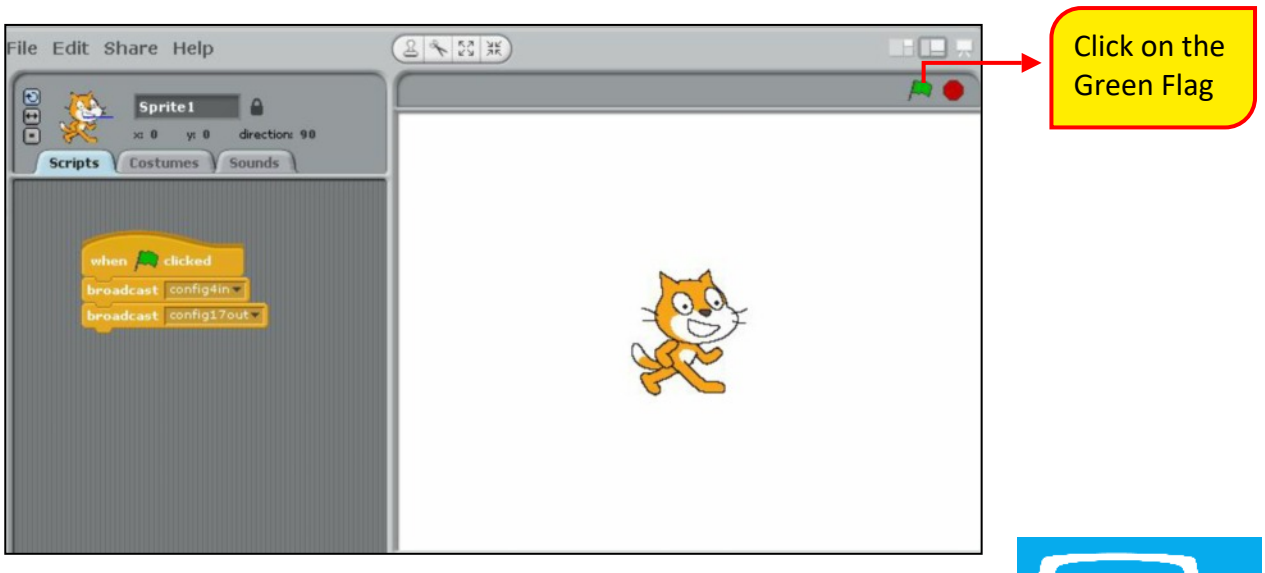
For this exercise, we will make the output LED flash only when a push button is pressed. To do this:

- Use a **Green Flag** to start your script:
  - ❑ Use the Scratch block: ***when Green flag clicked*** [under the **CONTROL** tab].

- **Configure** a Raspberry Pi GPIO pin as an **Input** pin:
  - ❑ Use the Scratch block: ***broadcast*** (found under the **CONTROL** tab)**.**
  - ❑ Use the code: **config[PinNumber]in.**
  - ❑ Example: **Input A** on the gPiO box is connected to **GPIO Pin 4** of the Raspberry Pi. So, we need to configure **PinNumber 4**. The code then becomes: **config4in**.

- **Configure** another Raspberry Pi GPIO pin as an **Output** pin:
  - ❑ Use the Scratch block: ***broadcast*** (found under the **CONTROL** tab)**.**
  - ❑ Use the code: **config[PinNumber]out.**
  - ❑ Example: **Output 1** on the gPiO box is connected to **GPIO Pin 17** of the Raspberry Pi. So, we need to configure **PinNumber 17**. The code then becomes: **config17out**.

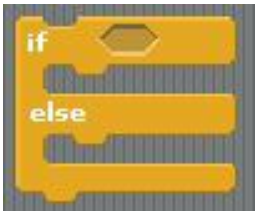
- Click on the **green flag** and run your program once so that all the required pins are configured. **This is particularly important in order for Scratch GPIO to read the status of the input pins.**



centre for computing history

SENSING INPUTS

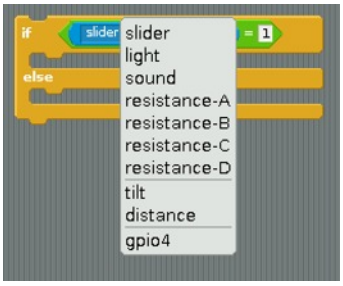
Continuing the script….



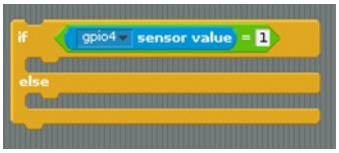
- **Check** whether the input button has been pressed or not:
  - ❑ Use the Scratch block: ***if-else***[under the **CONTROL** tab]**.** [The ***if-else*** block tests a condition and if true, then the instructions that are placed within the ***if*** block are run else the instructions placed in the ***else*** block are run].
  - ❑ Input a condition in the ***if*** block by using the **=** block [under the **OPERATORS** tab].

  - ❑ Fill the blank spaces in the **=** block with the condition we want to test.
  - ❑ Example: To test whether **Input A push button** has been pressed, we need to check whether the value of **GPIO Pin 4** is '1'.
  - ❑ Use the Scratch block: ***'slider' sensor value*** [under **SENSING** tab].

  - ❑ Place the ***'slider' sensor value*** block in the left-hand blank space of the **=** block.
  - ❑ Change the value from **'slider'** to **'gpio4'**. [This option should be available if you click on the drop-down arrow in the box.]



  - ❑ Type in the value '1' in the right-hand blank space of the **=** block.



- If the value of **GPIO Pin 4** is high or 1, then switch the LED next to **output 1** on:
  - ❑ Use the Scratch block: ***broadcast*** [under the **CONTROL** tab] as shown.
  - ❑ Use the code: **gpio[PinNumber]on.**
  - ❑ Example: To switch on the LED on **output 1** of the gPiO box which is connected to **GPIO Pin 17** of the Raspberry Pi, we use the code: **gpio17on**.
- If the value of **GPIO Pin 4** is low or 0, then switch the LED next to **output 1** off:
  - ❑ Use the Scratch block: ***broadcast*** (found under the **CONTROL** tab)**.**
  - ❑ Use the code: **gpio[PinNumber]off.**
  - ❑ Example: To switch off the LED on **output 1** of the gPiO box which is connected to **GPIO Pin 17** of the Raspberry Pi, we use the code: **gpio17off**.

When you run the script by clicking on the green flag, the LED next to **output 1** will come on when you press the button next to **Input A**.

CONNECTING COMPONENTS TO THE gPiO BOX

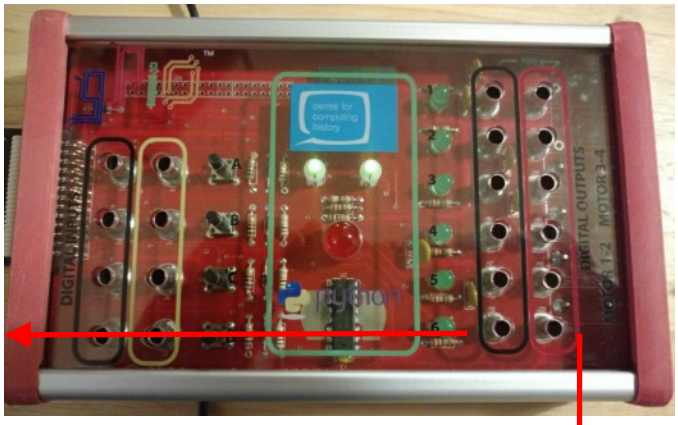
Let us start by connecting the **red LED** to **output 1** of the gPiO box.

Each of the output pins on the gPiO box has a red 'slot' and a black 'slot'.

The LED has a red wire and a black wire connected to it (as do most electrical components).

Insert the ends of the wires coming out from the LED into the given banana plugs (each banana plug takes one wire).

Now, connect the LED to output 1 of the gPiO box.



Make sure you connect the banana plug with the **red (positive) wire** inserted into the **output 1** slot with the **'red' box** around it. Similarly, the banana plug with the **black (negative) wire** inserted goes into the **output 1** slot with the **'black' box** around it.

Now, run this script again:

REMEMBER: Click on the 'Edit' menu option and select 'Start GPIO server' before you run your script. **For a reminder, see Page 14**.

The red LED should come on for 1 second and then go off.

CONNECTING COMPONENTS TO THE gPiO BOX

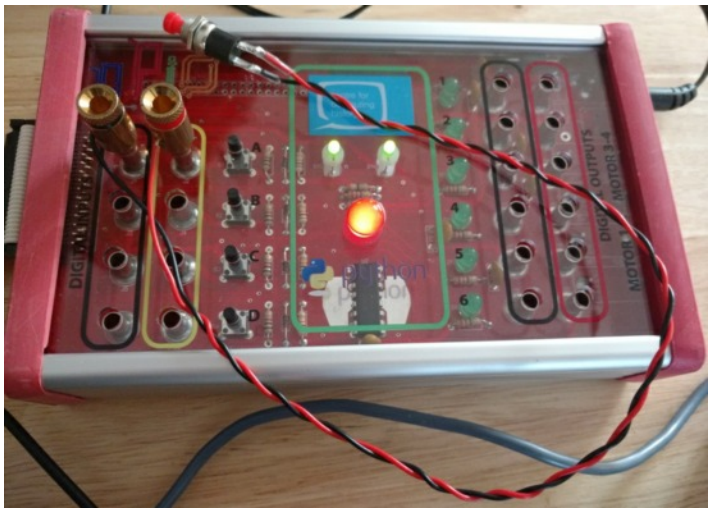
Let us now connect the **pushbutton** to **input A** of the gPiO box. Keep the LED connected to output 1 of the gPiO box from the previous example.

The input pins on the gPiO box have a yellow 'slot' and a black 'slot'.

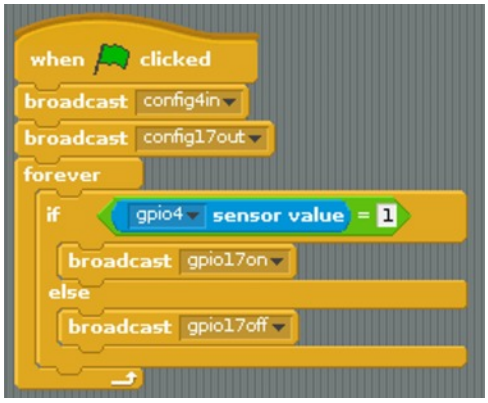
The pushbutton has a red wire and a black wire connected to it (though polarity doesn't make a difference in this case).

Insert the ends of the wires coming out from the pushbutton into the given banana plugs.

Now, connect the pushbutton to **input A** of the gPiO box. Make sure that one wire is in the yellow slot and the other in the black slot next to **input A**.



Now, run this script again:



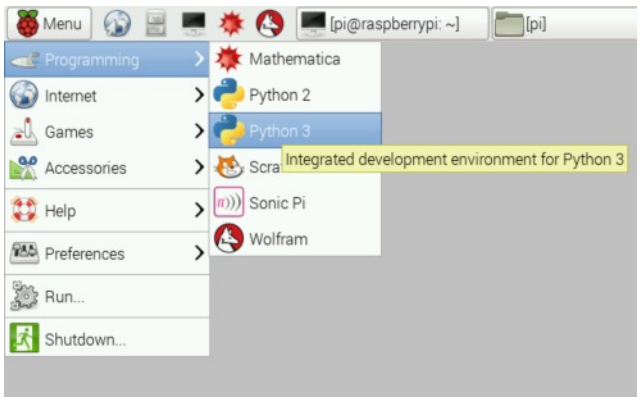
The red LED comes on when you press the push button and goes off when you release the pushbutton.

LET'S PROGRAM IN PYTHON

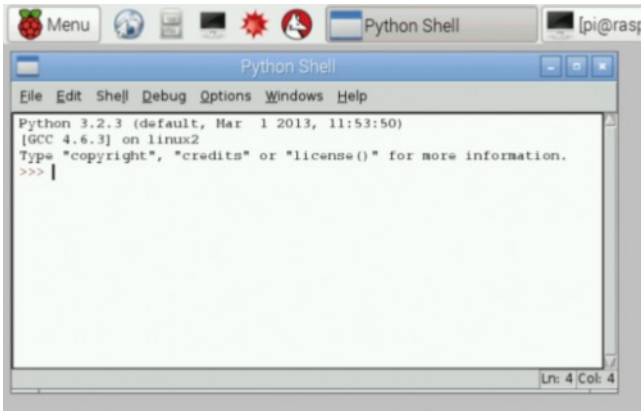
The Raspberry Pi Linux distribution comes complete with a package called IDLE. IDLE is the Python integrated development environment where you can start to write your first Python scripts!

Open the applications menu (Menu button) and select Programming

Click on Python 3

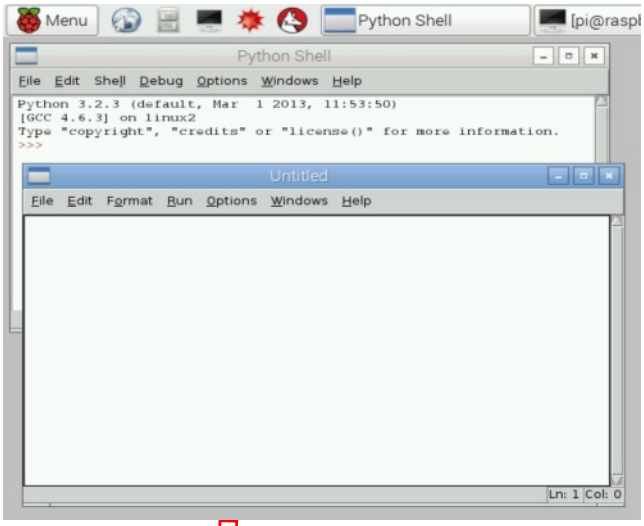


After a few seconds the development environment or the 'Python Shell' will load



Python shell

Click on **File** and then **New File**. This will open another window where you can start to write some Python script ...

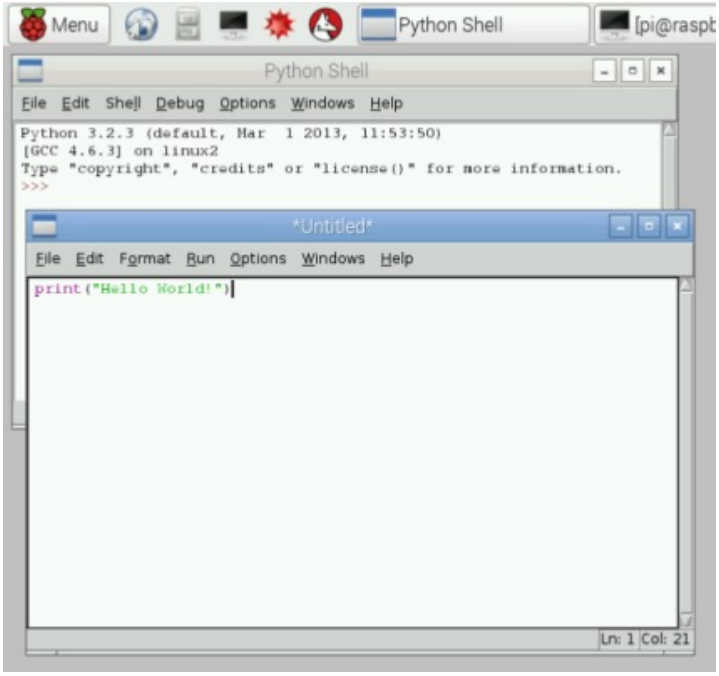


Text Editor

centre for computing history

LET'S PROGRAM IN PYTHON

In the new window, click on the blank space to start typing.



Let's type our first line of code…

Type this into the text editor:

```
print("Hello World!")
```

Notice how colour separates the code out

Purple colour: the print command or function

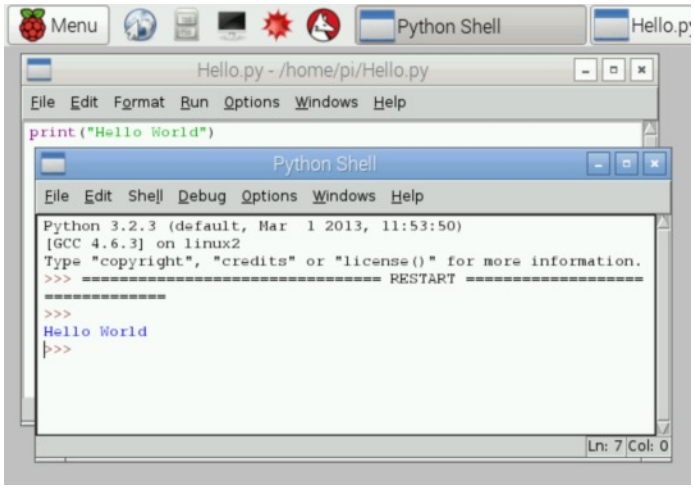
Green colour: text we want to print

To run the code, press **F5** on the keyboard.

You should be asked to **Save** your program first. Click **OK** and give it a name (you can call it "Hello").

Note that the file will be saved as "**Hello.py**" in the default location **/home/pi**.

When the code runs, the Python Shell should come into focus and display what we asked it to say, in our case, Hello World!



Try out these ideas:

(*) Print out multiple lines by repeating the print command for each new line:

```
print ("Hello world!")
print ("How are you?")
```

REMEMBER:
- PRESS F5 ON THE KEYBOARD TO RUN
- SAVE WHEN ASKED TO

(*) Print your name:

```
print ("Hello, I'm Jeremy")
```

This is where you put your name down

centre for computing history

CONTROLLING OUTPUTS

Let's start by programming **GPIO Pin 9** as an **output pin**.

As before, click on **FILE** and then **NEW FILE**, which will open up the **TEXT EDITOR.**

You can now enter the following lines of code:

```
import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

buzz = 9

GPIO.setup(buzz, GPIO.OUT)

while True:
    GPIO.output(buzz, 1)
    print ("On")
    time.sleep(1)

    GPIO.output(buzz, 0)
    print ("Off")
    time.sleep(1)
```

Importing the necessary packages and modules

This option means that we will be using GPIO/BCM numbers for our program. **If you want to use the actual pin numbers, use the option GPIO.BOARD.**

Using a variable to name the GPIO pin 9 [makes it easier to change pin numbers without modifying the entire code]

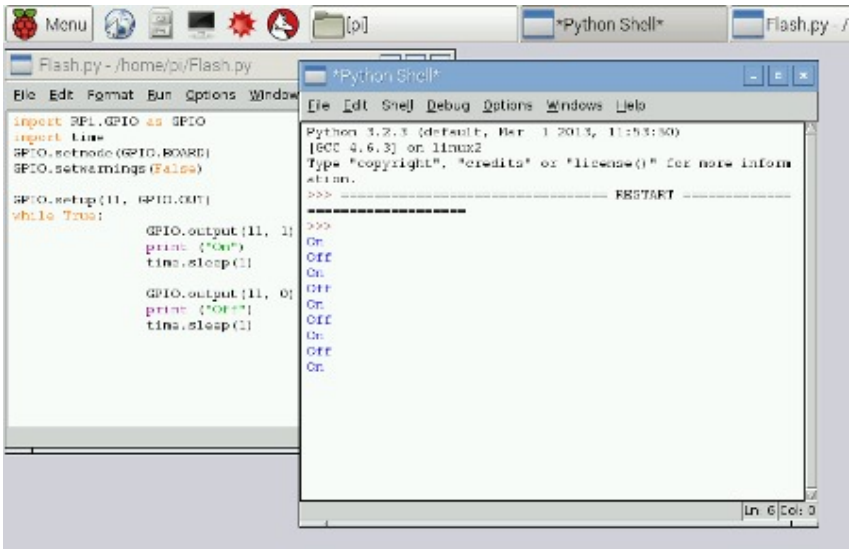
Setting the GPIO pin as an output pin

Turning the **GPIO Pin 9 ON (1)** and **OFF (0)**. **Remember** to add a line to wait for a second **[time.sleep(1)]** before changing the value of the GPIO pin.

PRESS F5 on the keyboard to run the program.

SAVE the program when asked to.

The **Python Shell** window should now display On, Off, On, Off repeatedly.



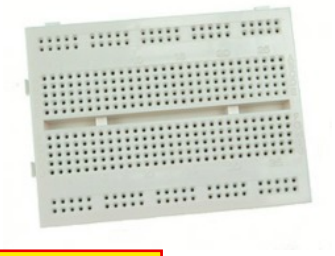
centre for computing history

CONTROLLING OUTPUTS

Let's now see what the GPIO pins are doing. We are going to connect a buzzer to the GPIO pins and that should 'buzz' on and off when you run your program.

For this, we will need the following:

- Breadboard
- Buzzer
- Jumper wires



Breadboard

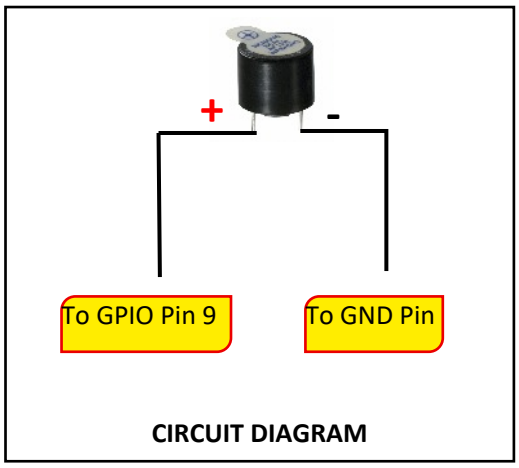
Buzzer

Jumper wires

Connect the components as shown in the diagram.

NOTE:

- **Longer leg (positive) of the buzzer to GPIO Pin 9** of the Raspberry Pi.
- **Shorter leg of the buzzer to any Ground (GND) pin** of the Raspberry Pi. This can be any of the pins with the '-' next to them.



CIRCUIT DIAGRAM

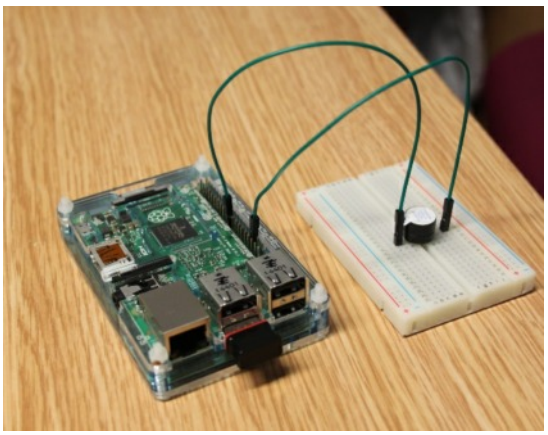
Breadboard Connections

Breadboard has rows and columns marked.

- Rows marked 1 to 17
- Columns marked A to J

To connect a buzzer to the Raspberry Pi, follow these steps:

- Insert the longer leg (+) of the buzzer into E10 and shorter leg (-) into F10.
- Take one jumper wire and insert the pointed end into D10. The other end of this wire plugs into GPIO Pin 9 on the Pi.
- Take the second jumper wire and insert the pointed end into H10. The other end plugs into a GND pin (any pin with a '-' sign next to it) on the Pi.



With the buzzer connected up correctly (as shown in the picture above), if you run your 'On-Off' program again, you should see it flashing on and off.

So now you have control of the real world!!

Just imagine the possibilities! We can use switches to control the buzzer in an intruder alarm, add flashing LEDs and, maybe, a camera, connect motors to the GPIO pins and run little robot cars…..

SENSING INPUTS

So, we've switched an output pin ON and OFF. **The next step is to read the value from an input pin - is the value 0 or 1?**

We'll be using **GPIO pin 15** as an **input pin**.

Now, **modify** the existing program so that it:

- reads the status of the GPIO input pin 15;
- **turns the buzzer on** if the status of the GPIO pin 15 is 0 i.e., the **pushbutton is pressed**;
- prints 'pressed' and 'not pressed' messages.

```
import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

buzz = 9
button = 15

GPIO.setup(buzz, GPIO.OUT)
GPIO.setup(button, GPIO.IN)

while True:
      status = GPIO.input(button)

      if status == 0:
            GPIO.output(buzz, 1)
            print ("Button pressed")
      else:
            GPIO.output(buzz, 0)
            print ("Button not pressed")
```

Use another variable to name the GPIO pin 15

Set GPIO pin 15 as an input pin

Read the status of the input device or GPIO pin 15

If the **status of GPIO Pin 15** (stored in the variable **status**) **is 0 (low)**, that is, the button, is pressed, then **turn the output on**. If the **status of GPIO Pin 15 is 1 (high),** that is, the button is not pressed**, turn the output off**.

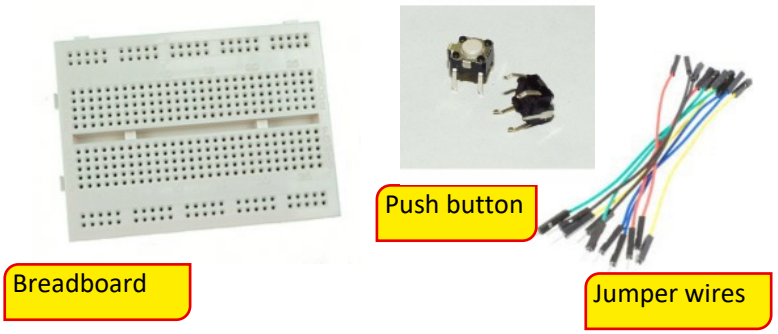
PRESS F5 on the keyboard to run the program.

SAVE the program when asked to.

SENSING INPUTS

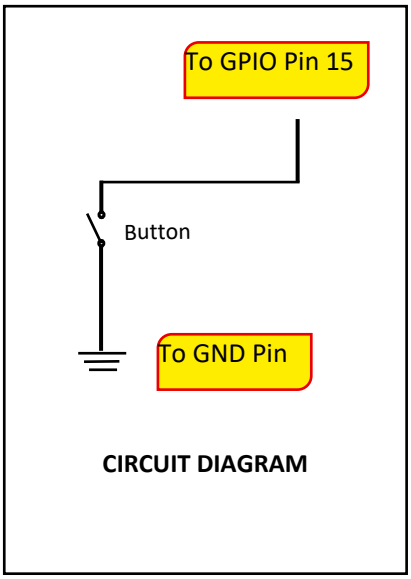
Let's now see what the GPIO pins are doing. For this, we will need the following:

- Breadboard (the one you used for the buzzer exercise)
- Pushbutton
- Jumper wires



Connect the components as shown in the diagram:

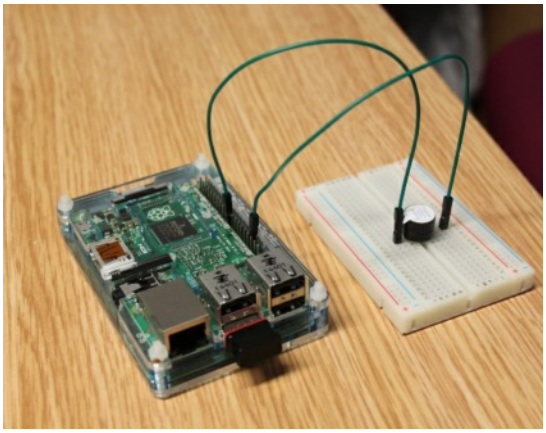
- **One leg of the pushbutton to GPIO Pin 15** of the Raspberry Pi.
- **The other leg to any Ground (GND) pin** of the Raspberry Pi. This can be any of the pins with the '-' next to them.



Breadboard Connections

To connect a pushbutton to the buzzer circuit, follow these steps:

- One end from the pushbutton goes into **E15** and the other end goes into **F15;**
- One end of a jumper wire goes into **G15** and the other end to Pin 15 on the Raspberry Pi;
- Take another jumper wire and push one end into **D15** and the other end to a GND pin (any pin with a '-' sign) on the Raspberry Pi.

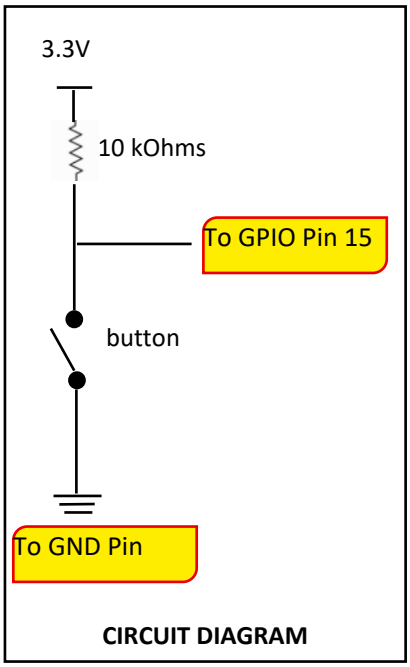


Now, when you run your program, you should be able to hear the buzzer go on only when the switch is pressed. You should see the 'pressed' and 'not pressed' messages on the screen.

Is your program behaving strangely? Is the buzzer not responding as it should?

SENSING INPUTS

The problem is that we need an extra component in the push button circuit called a 'pull up' resistor. And this 'pull up' resistor needs to be connected to GPIO pin 15 as shown in the circuit diagram!



CIRCUIT DIAGRAM

NOTE:

- A high value resistor like the **10 kOhm resistor** in the diagram is used as **a 'pull up' resistor.** This resistor in the circuit makes sure that when the **button is not pressed** the **GPIO pin 15** will show a value of **HIGH or '1'.**
- When the **button is pressed,** the **GPIO pin 15** will show a value of **LOW or '0'.**
- Without the 'pull up' resistor, the input pin (GPIO Pin 15) may 'float' halfway between high and low and that is why your program was behaving erratically.

However, for this workshop, we will not be using a hardware 'pull up' resistor.

We will connect the GPIO pin 15 to a 'pull up' resistor using software and Raspberry Pi makes this easy by allowing us to use the **GPIO.PUD_UP** option when setting up an input pin.

The modified program should look like this:

```
import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

buzz = 9
button = 15

GPIO.setup(buzz, GPIO.OUT)
GPIO.setup(button, GPIO.IN, pull_up_down=GPIO.PUD_UP)

while True:
      status = GPIO.input(button)

      if status == 0:
            GPIO.output(buzz, 1)
            print ("Button pressed")
      else:
            GPIO.output(buzz, 0)
            print ("Button not pressed")
```

The buzzer should now buzz only when the button's pressed.

USEFUL RESOURCES

Raspberry Pi - www.raspberrypi.org
The official site for Raspberry Pi.

Raspberry Pi Forums - www.raspberrypi.org/forum
Lots of very useful chat here. There are specific sections for Education, Using the Raspberry Pi, Programming and Projects.

Geek Gurl Diaries - www.geekgurldiaries.co.uk
A brilliant YouTube based web series for teenagers who want to be makers and creators of technology. Great explanations and friendly relaxed approach.

Cambridge University Computer Lab - www.cl.cam.ac.uk/projects/raspberrypi
Lots of useful information, projects and links put together by Robert Mullins. Some of the information is for the more experienced user.

Pimoroni - https://shop.pimoroni.com/collections/raspberry-pi
You can buy accessories for your Pi here.