Bots & Bytes Fall 2019

Can be accessed online @ https://tinyurl.com/yyaxnwjg.

Also mirrored (with a different skin) @ https://www.thecshore.com/projects/bots-and-bytes-fall-2019/.

Teaching the Raspberry Pi in Six Modules

A Volunteer Project for the MakerPlace @ the Midland Public Library

Audience

This series was aimed at a 9 to 11 year old audience in Ontario, Canada (originally a somewhat older audience was expected, and this was our first time doing workshops for kids, so you'll notice a shift in the presentation towards accommodating the younger members of the group), and their parents. Some references may not apply elsewhere (the US-style keyboard in the copy-paste handout, for instance, would have be updated if the series were to be used in a region with a different standard keyboard).

Based On

- Pi Curriculum Hardware Guide
- C Shore's Maker Workshops
- Pi Curriculum's Getting Started With Mu
- Pi Curriculum's Physical Computing
- Pi Curriculum's Quick Start Guide
- Raspberry Pi Foundation's Documentation
- Pi Curriculum's Python Lessons and search for Python.
- Pi Curriculum's Build a robot buggy
- Custom work

1. Module 1: “Blinky Lights”

1.1. Objectives - Module 1

- Insert SD card with pre-configured Raspbian OS — direction following, simple disassembly/assembly, experience seeing the physical basis of modern computers (the bare board and connectors)

- Learn to use the Mu Editor for Python - using a text editor, coding, debugging

- Learn Just Enough Python to ‘do stuff’ with LED - coding, debugging

- Control an LED using Python — coding, physical computing (control an output), outputting a code (flash LED according to some logic)

- Control an LED using Sensors and Python - coding, physical computing (control input/output)

1.2. A Brief Introduction to Bots & Bytes

1.2.1. Who We Are

- Introduce myself and fellow facilitators, and helpers.

1.2.2. Why We’re Here

- To introduce you to computers and electronics that act on, and react to the real world — this is known as physical computing.
- By module six you’ll be programming a robot car you build in modules four and five, through a challenge course.
- Daniel be going through Modules one through three with you, so that you are ready for the robot challenge in the second half of the program, and hopefully getting you doing some interesting technology along the way.

1.2.3. Who are You and Why are You Here?

- Get the learners to introduce themselves and what they hope to get out of the program.

1.2.4. How the board in front of you (Raspberry Pi) is a computer

Modified from presentation as original version confused the kids; see Module 1 Skipped or Simplified Information, Part I for the original. The next edition of Bots & Bytes will have a better presentation.

- (Pointing parts out on an actual PI): The largest chip on the board is what is known as the SoC or System-on-a-chip or CPU and is the brains of the Pi. The other large chip on the topside of the Pi is the USB controller. On the bottom of the board are the wireless (shielded), RAM, and MicroSD slot.
- The I/O (input/output) on the Pi is mostly in the SoC with access to the pins on the chip exposed with the double row of header pins that are on the board. In addition there is USB that is used to connect the wired network connection and USB ports. The wireless is embedded into the Pi and does not go through the USB hub.
- Another important piece of the Pi is the RAM — that is the scratchpad the CPU uses for doing calculations and running programs, while the Pi is powered on. The contents of scratchpad are lost when the Pi is turned off.
- And finally, connected through more I/O is the micro SD card, which provides persistent storage, which means what is on it continues to exist when the Pi is turned off, even if it is removed from the Pi.

1.3. Discovering the Raspberry Pi

1.3.1. Filling the Pi

Take a Look at the Inside of the Pi While We Insert an SD Card (OS)

The Raspberry Pi uses SD Cards to store the Operating System (the software that controls your computer — like Windows on a typical laptop or desktop computer) and software you can use. We will be getting each of you to put a prepared SD card into Pi, and while we're at it, we're going to look at what the Pi's looks like inside, and talk a little about what the Pi is.

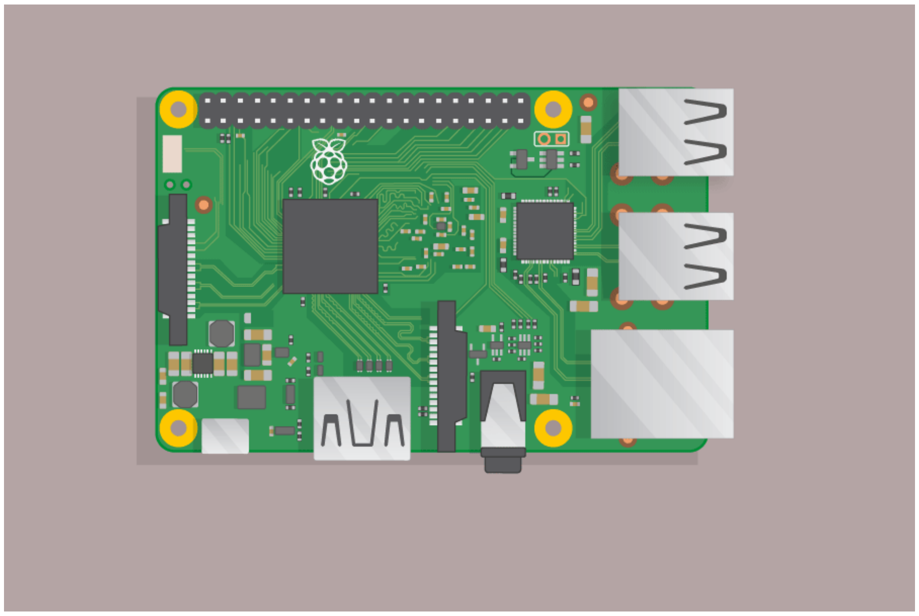
1. Begin by placing your SD card into the SD card slot on the Raspberry Pi. It will only fit one way.

1.3.1.1. What is The Raspberry Pi?

- A full-blown computer capable of running a desktop
- Based on a different processor (brains) than a typical laptop or desktop

1.3.1.2. Raspberry Pi hardware setup

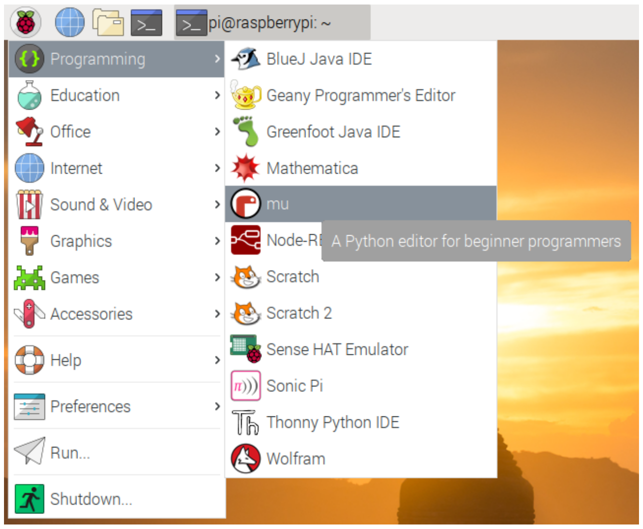
1. Put the Pi in its case and put the case together.
2. Next, plug your keyboard and mouse into the USB ports on the Raspberry Pi.
3. Make sure that your monitor or TV is turned on, and that you have selected the right input (e.g. HDMI 1, DVI, etc).
4. Connect your HDMI cable from your Raspberry Pi to your monitor or TV.
5. When you’re happy that you have plugged all the cables and SD card in correctly, connect the micro USB power supply. This action will turn on and boot your Raspberry Pi.



1.3.1.3. The Pi desktop

- Once your Pi is booted, you should see a desktop that looks similar to older Windows with the taskbar at the top instead of bottom.
- TBD: Screenshots of Pi with Applications menu open

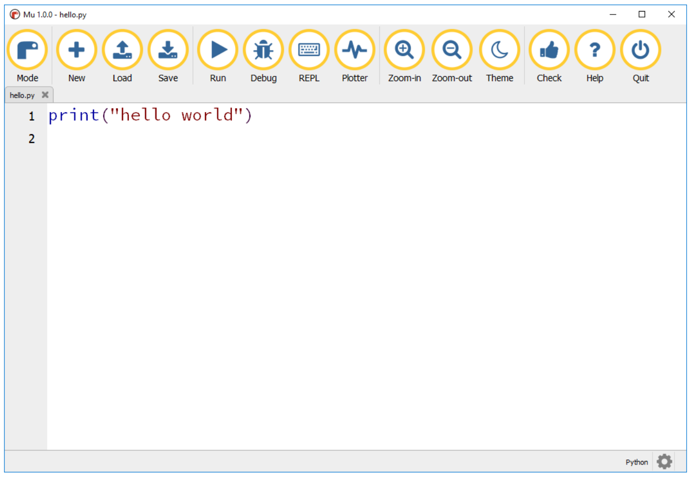
- The 'Application' menu in the top left (which has a Raspberry icon) is the primary way to launch programs on the desktop.

- 

- For now we're not doing anything with the Pi desktop, except launching the 'Mu' editor from the 'Programming' menu in the Application menu (often this type of location is written in the same form as Applications|Programming|Mu).

1.4. Getting Started With Mu

1.4.1. Introduction



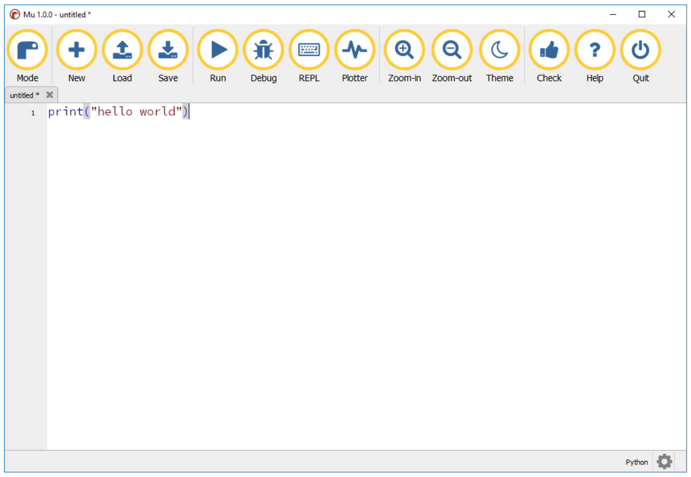
```
print("hello world")
```

Mu is a very simple to use Python editor and IDE (integrated development environment) for beginners. It's designed to be as user-friendly and helpful as possible for new Python programmers.

1.4.2. Writing code in Mu

The main area in Mu is where you will write your code. Enter this code into Mu to create a 'Hello world' program:

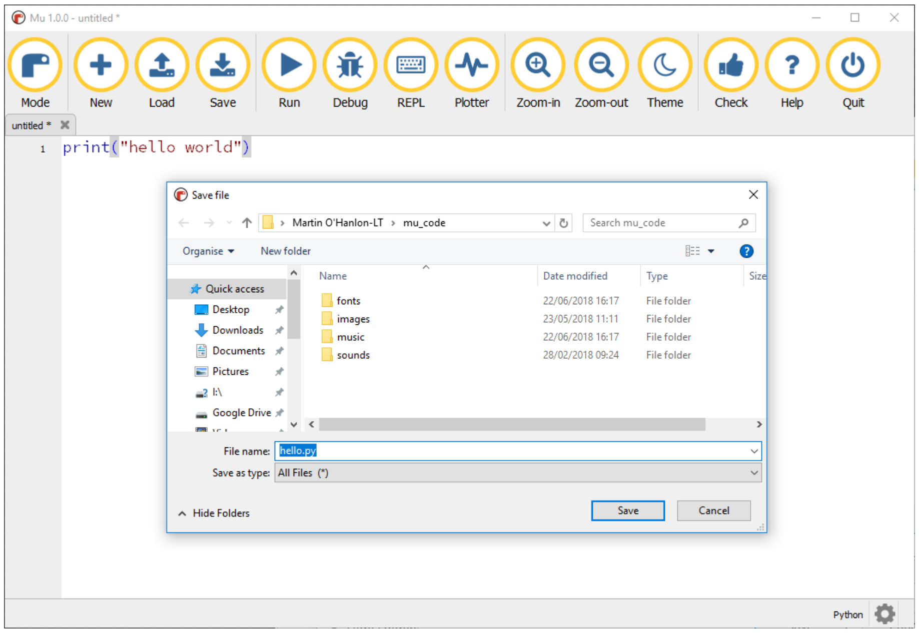
```
print("hello world")
```



```
print("hello world")
```

Click **Save** to save your program.

Enter the file name `hello` and click **Save**.



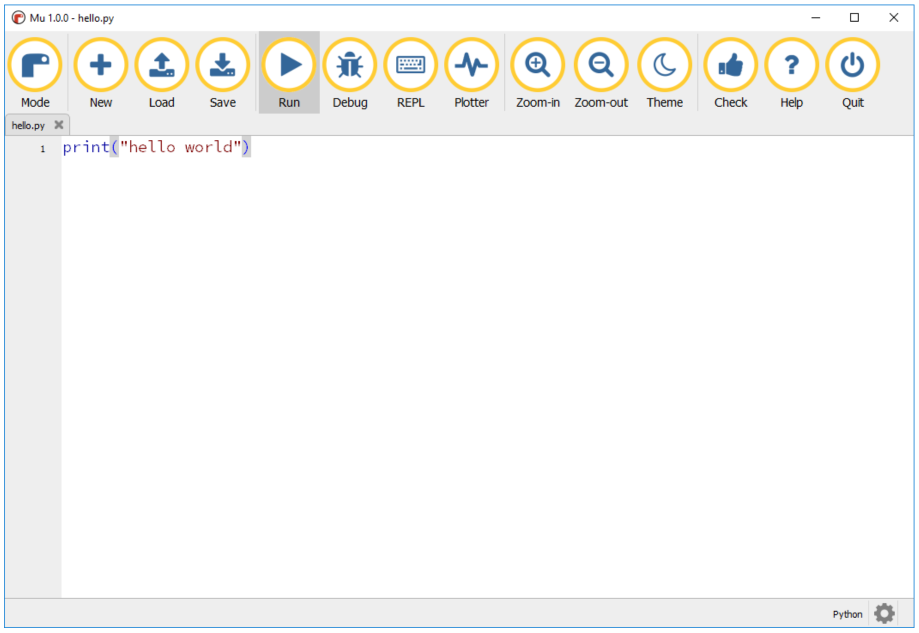
```
print("hello world")
```

Name	Date modified	Type	Size
fonts	22/06/2018 16:17	File folder	
images	23/05/2018 11:11	File folder	
music	22/06/2018 16:17	File folder	
sounds	28/02/2018 09:24	File folder	

File name: hello.py

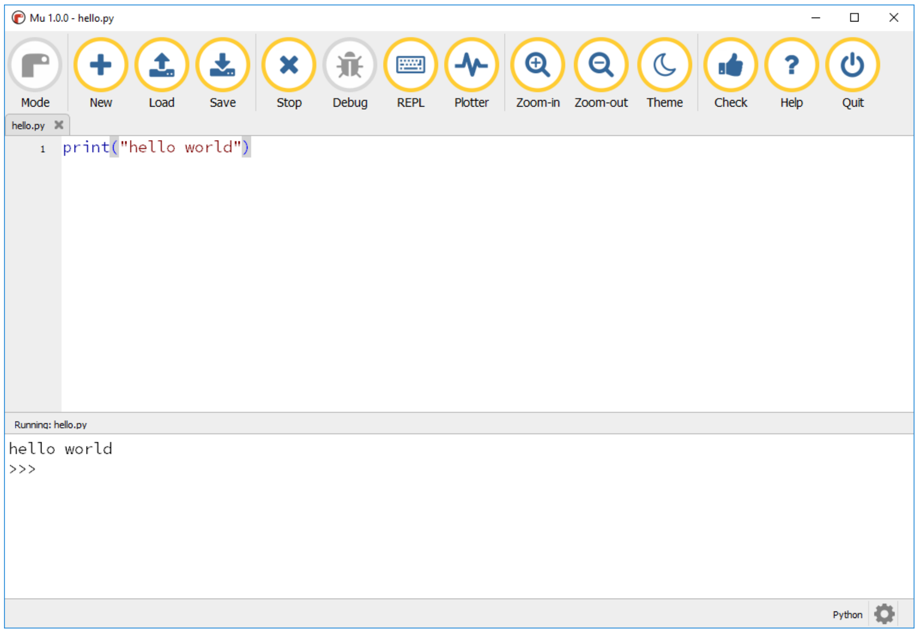
Save as type: All Files (*)

Click **Run** to run your program.



```
print("hello world")
```

Your program will run, and the message `hello world` will be displayed.



```
print("hello world")
```

```
hello world
>>>
```

Stop your program by clicking **Stop**.

1.4.2.1. Saving your code

Once you have saved your code to a file and given it a name, Mu will automatically save it for you every few seconds, as well as every time you run it. This means that you will probably never lose any work!

You can, of course, also use the **Save** button any time you want.

1.4.3. Modifying ‘hello world’

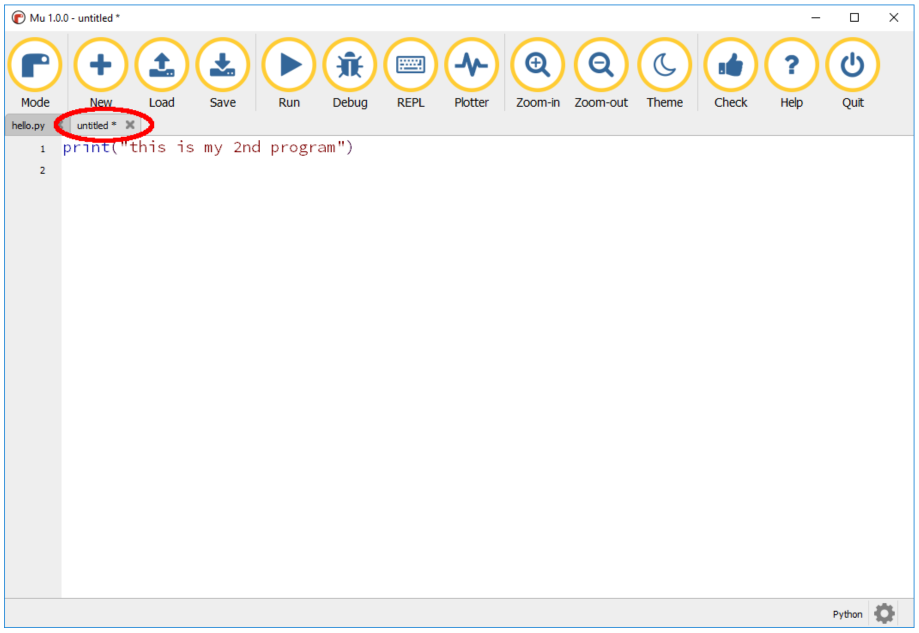
Modify the message from ‘hello world!’ and run the program (that is replace ‘hello world!’ with what you want to say — that is appropriate here of course).

1.4.4. Multiple programs

You can have multiple scripts open at the same time using the tabs.

Click **New** to create a new program.

A second tab will appear where you can write you new program:



```
print("this is my 2nd program")
```

You can switch between your programs by clicking on the tabs.

1.4.5. Code checking and debugging

1.4.5.1. Highlighting

Mu will try and help you create working code by checking your program and highlighting errors it finds to allow you to correct them.

The following line of code is incorrect:

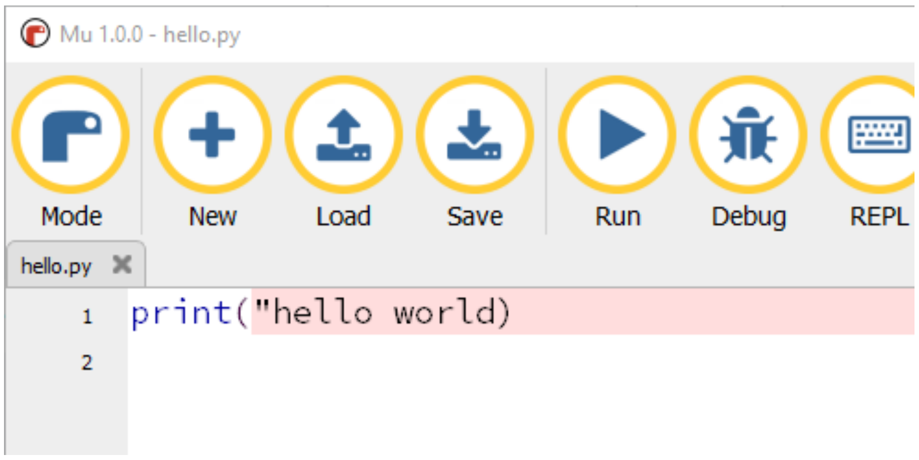
```
print("hello world)
```

There is a " missing at the end of `"hello world`

If you run this program, you will receive an error message:

```
File "c:\users\martin o'hanlon-lt\mu_code\hello.py", line 1
    print("hello world)
                      ^
SyntaxError: EOL while scanning string literal
```

Mu will highlight code that it recognizes as incorrect.



```
print("hello world)
```

1.5. Introduction to Electronics

1.5.1. Power vs Ground & Short Circuits

1.5.1.1. Ground

Ground is so-named because in power distribution it is achieved by connection a metal rod into the ground. Another name you may hear is ‘neutral’ because it is at about the same level as a normal environment.

1.5.1.2. Power

Current (amount of electricity) flows from power to ground, and the higher the level of power(voltage), the more current that flows.

1.5.1.3. Short Circuits

If power is connected directly to ground it’s known as a short circuit because as much current as is possible will flow through the wire. This is usually a bad thing because it will generate large amounts of heat and ‘fry’ the connection.

1.5.2. Breadboard

Explain rows vs columns, and the power traces.

1.5.3. Resistors

A device the resists the flow of electricity.

1.5.4. LED

1.5.4.1. What it is

Light Emitting Diode — A diode only lets electricity through in one direction. For an LED, when electricity if flowing through the diode, it lights up.

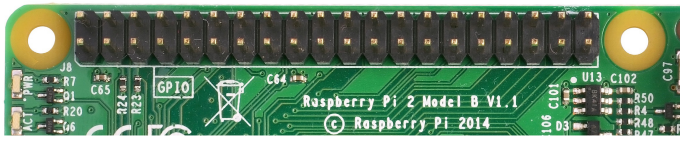
1.5.4.2. Physical Characteristics (Oriention/Polarity)

- Long leg opposite flat edge
- Which way it’s connected (which side to power and which side to ground), matters.
- The proper orientation is:

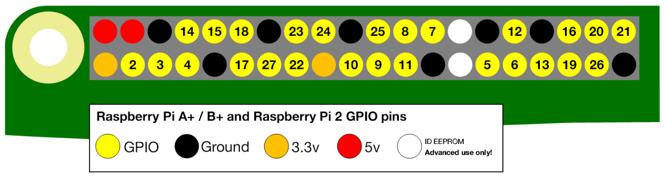
1.6. Raspberry Pi GPIO Pins

A powerful feature of the Raspberry Pi is the row of GPIO (general- purpose input/output) pins along the top edge of the board. A 40-pin GPIO header is found on all current Raspberry Pi boards (unpopulated on Pi Zero and Pi Zero W). Prior to the Pi 1 Model B+ (2014), boards comprised a shorter 26-pin header.

For our purposes the pins on the Pi3 (which we are using) are the same as on the Pi2, but for advanced uses there are some differences from the pinout here.



Any of the GPIO pins can be designated (in software) as an input or output pin and used for a wide range of purposes.



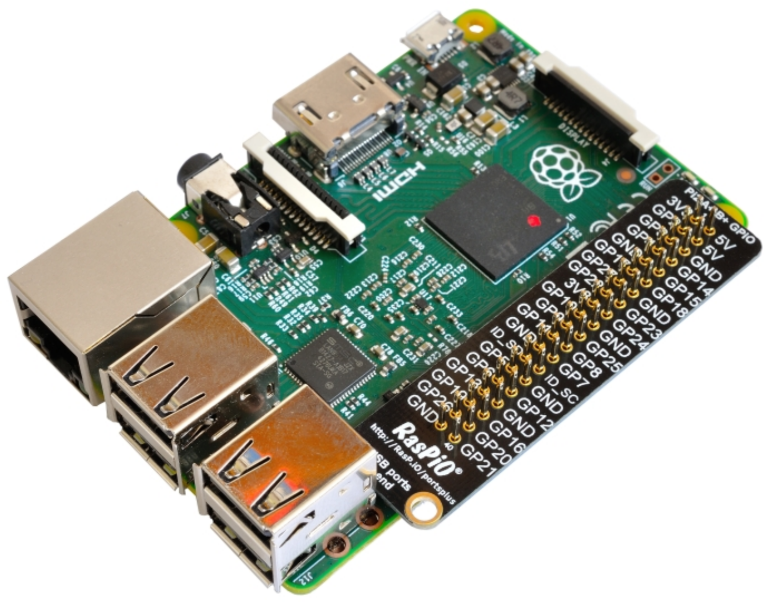
Note: the numbering of the GPIO pins is not in numerical order; GPIO pins 0 and 1 are present on the board (physical pins 27 and 28) but are reserved for advanced use (see below).

1.6.1. Voltages

Two 5V pins and two 3V3 pins are present on the board, as well as a number of ground pins (0V), which are not configurable. The remaining pins are all general purpose 3V3 pins, meaning outputs are set to 3V3 and inputs are 3V3-tolerant.

Warning: while connecting up simple components to the GPIO pins is perfectly safe, it's important to be careful how you wire things up. LEDs should have resistors to limit the current passing through them. Do not use 5V for 3V3 components. Do not connect motors directly to the GPIO pins, instead use an H-bridge circuit or a motor controller board.

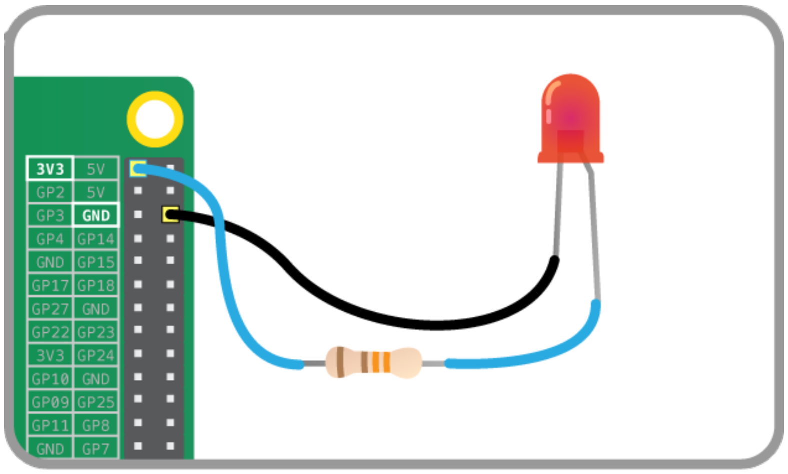
1.6.2. GPIO pinout



It's important to be aware of which pin is which. Some people use pin labels (like the RasPiO Portsplus PCB, or the printable Raspberry Leaf).

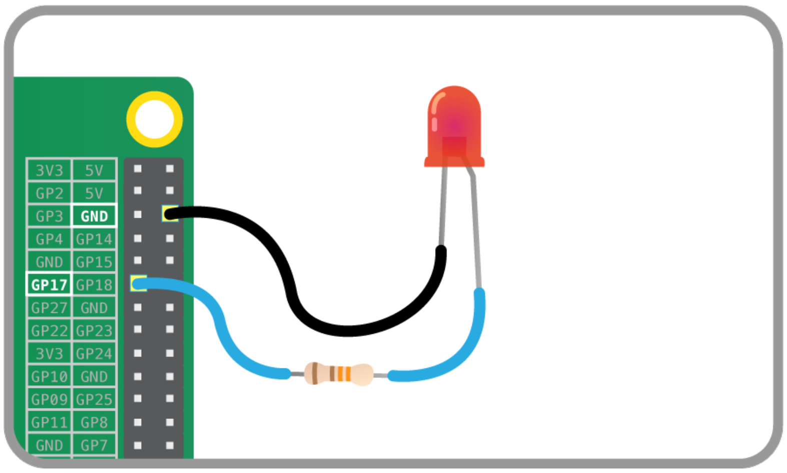
1.7. Lighting an LED

LEDs are delicate devices. If you put too much current through them they will pop (sometimes quite spectacularly). To limit the current going through the LED, you should always use a resistor in series with it.



Try connecting the long leg of an LED to the Pi’s 3V3 and the short leg to a GND pin. The resistor can be anything over about 50Ω.

The LED should light up. It will always be on, because it’s connected to a 3V3 pin, which is itself always on.



Now try moving it from 3V3 to GPIO pin 17:

The LED should now turn off, but now it’s on a GPIO pin, and can therefore be controlled by code.

1.7.1. Switching an LED on and off

GPIO Zero is a new Python library which provides a simple interface to everyday GPIO components. It comes installed by default in Raspbian.

- Open Mu.
- You can switch an LED on and off by typing commands directly into the REPL. Click on the REPL button in the menu bar.
- First import the GPIO Zero library, and tell the Pi which GPIO pin you are using - in this case pin 17.

  ```
  In [1]: from gpiozero import LED
  In [2]: led = LED(17)
  ```

  Press **Enter** on the keyboard.
- To make the LED switch on, type the following and press **Enter**:

  ```
  In [3]: led.on()
  ```
- To make it switch off you can type:

  ```
  In [4]: led.off()
  ```
- Your LED should switch on and then off again. But that’s not all you can do.

1.8. Blinking an LED

With the help of the `time` library and a little loop, you can make the LED flash.

- Create a new file by clicking **New**.
- Save the new file by clicking **Save**. Save the file as `gpio_led.py`.
- Enter the following code to get started:

```
from gpiozero import LED
from time import sleep

led = LED(17)

while True:
    led.on()
    sleep(1)
    led.off()
    sleep(1)
```

- Save the file and run the code with by clicking on **Run**.
- The LED should be flashing on and off. To exit the program click **Stop**.

2. Module 2: “All Singing, All Dancing”

2.1. Objectives - Module 2

- Learn to use digital input devices with the Raspberry Pi - physical computing, digital i/o (input/output) — (accept an input, act on input, control an output)
- Learn very basic electronics - electronic switches, on/off circuit
- Learn to use more output devices - physical computing, digital i/o (input/output), control logic - (electronics, coding, logic)

2.2. Review of Module 1

2.2.1. Preliminaries

The presentation and handouts will have a horizontal line across the page in various spots. At those locations, you are to signal the instructor that you have reached that point and are waiting on others to move on to the next section.

We’re doing this so that the instructor can provide information to the entire group at once, in a clear and consistent fashion.

We’re going to try letting you experiment while you’re waiting, but that will only work if you are paying attention to the instructor and the rest of the class so that you can stop the extra activity and focus on the main discussion when everyone is ready.

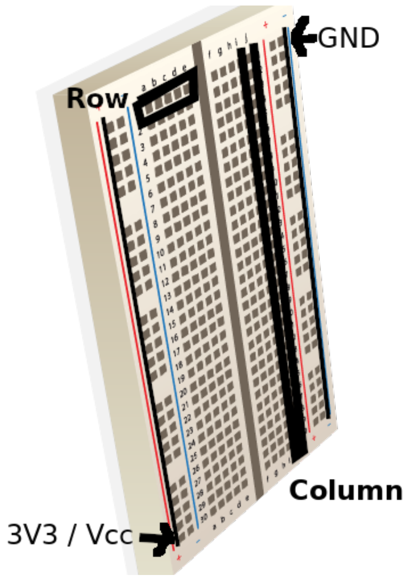
2.2.2. Review

- Pi is a computer & what a computer is
    - What is in a computer, and what does the Pi have?
- Booting the Pi
    - Let’s do that now
- Launching Mu
    - Find Mu and open it.
- Review of GPIO pins
    - Recall that GPIO pins are for digital input (getting information from the world) and digital output (sending signals to the world, usually to cause some action).
    - Recall that the GPIO number is not the name as the position it has on the header (double row of pins) on the Pi.
    - Let’s review how to read the GPIO chart.

2.3. A Explanation of Breadboards

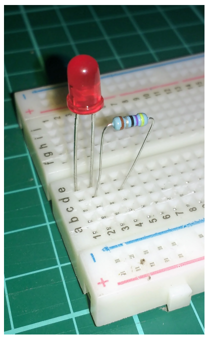
- A breadboard is used to temporarily connect electronics components.
- This is usually done while testing or prototyping. (prototyping is creating a working version of some device for testing purposes, usually with the intent of creating a design that can be used for production of many identical units — aka mass production).
- A bread board is called that because in early days of electronics, inventors would use wooden boards used for making bread and drive in nails and screws for connecting electronics devices for testing and prototyping.

The following diagram illustrates the parts of a breadboard.



You can see that there are rows and columns. All the holes in a row are connected to each other. Columns are separate from each other. The groove in the centre of the board divides the left and right halves into separate sections (that is row 1 on the left is **not** connected to row 1 on the right). The columns marked plus (red), and minus (black or blue) are given special names because for that part of the breadboard the columns are connected and not the rows. The plus (red) is called the power plane (noted as Vcc and which we are connecting to the 3V3 pin on the Pi). The minus (black) is called the ground plane (noted as GND and we connect it to GND on the Pi).

Part of a real breadboard is pictured below:



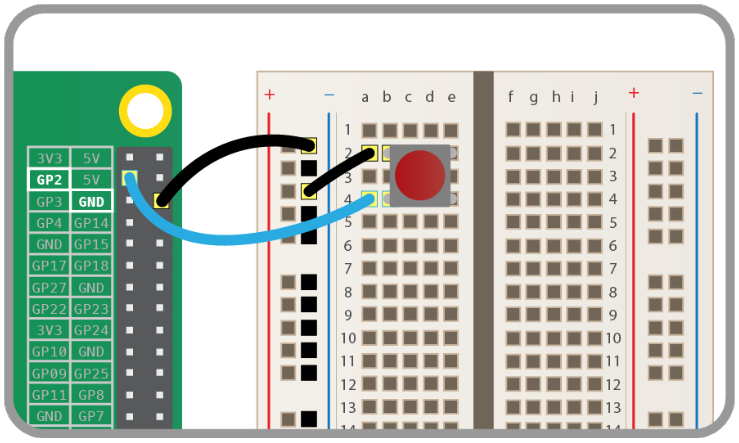
When you are inserting soft leads such as those of a resistor into the breadboard it is best to grasp the lead near the end and gently but firmly push them into the hole on the breadboard.

Note: It is recommended that when you are connecting to to power or ground on the Pi that you use a jumper to go from the Pi to the appropriate plane (power or ground), and use a second jumper to go from the plane to the components on the breadboard, rather than connecting directly to the power or ground of the Pi.

2.4. Using buttons to get input

Now you're able to control an output component (an LED), let's connect and control an input component: a button.

- Connect a button to another GND pin and GPIO pin 2, like this:



- Create a new file by clicking **New**.
- Save the new file by clicking **Save**. Save the file as `gpio_button.py`.
- This time you'll need the `Button` class, and to tell it that the button is on pin 2. Write the following code in your new file:

```
from gpiozero import Button
button = Button(2)
```

- Recall that `from gpiozero import Button` pulls in code that someone else has written. They've put it in a library called gpiozero. And, since we're working with a button, we are specifically requesting the `Button` class.

- Similar to the way in which a daffodil is a type of flower, the button variable is a type of the Button class. The button variable in this case is a instance of Button associated with GPIO 2 (which is why we write `Button(2)`). In the case of a daffodil it could be considered an instance of flower that has bell-shaped yellow top, and a long green stem. Each of these facts about the daffodil are known as ‘properties’ of the class Flower that create a unique flower.

- Now you can get your program to do something when the button is pushed. Add these lines:

```
while True:
    button.wait for press()
    print('You pushed me')
```

- The complete program is therefore:

```
from gpiozero import button

while True:
    button.wait for press()
    print('You pushed me')
```

- Save and run the code.

- Press the button on the breadboard and your text will appear on on the screen.

2.4.1. Manually controlling the LED

You can now combine your two programs written so far to control the LED using the button.

As we did in Module 1, connect the LED to the GPIO 17 of the Pi through a resistor, while leaving the button connected as in the previous section.

The button and LED are not electrically connected. The Pi will accept input from the button and control the LED.

- Create a new file by clicking **New**.
- Save the new file by clicking **Save**. Save the file as `gpio_control.py`.
- Now write the following code:

```
from gpiozero import LED, Button
from time import sleep

led = LED(17)
button = Button(2)

while True:
    button.wait for press()
    led.on()
    sleep(3)
    led.off()
```

- Save and run your program. When you push the button the LED should come on for three seconds.

2.5. Using a Buzzer

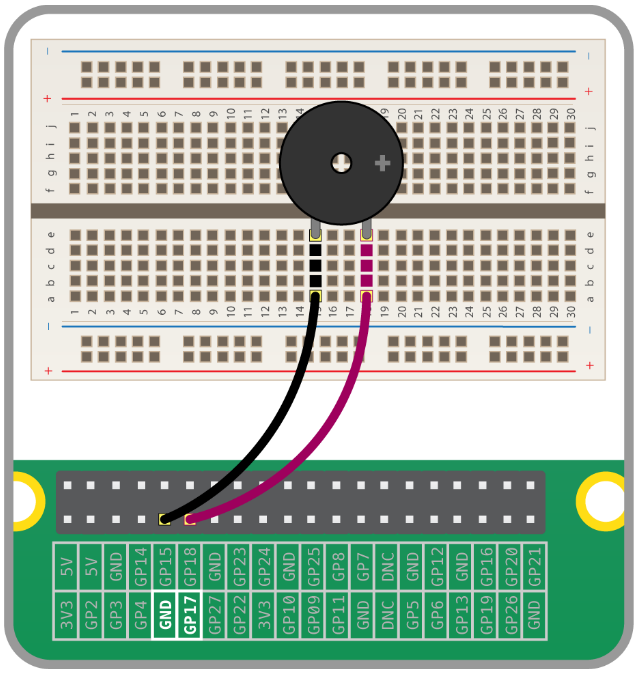
There are two main types of buzzer: *active* and *passive*.

A *passive* buzzer emits a tone when a voltage is applied across it. It also requires a specific signal to generate a variety of tones. We will be using a *passive* buzzer.

2.5.1. Connecting a buzzer

An buzzer can be connected just like an LED, but as they are a little more robust, you won’t be needing a resistor to protect them.

Set up the circuit as shown below:



- Add `TonalBuzzer` to the `from gpiozero import...` line:

```
from gpiozero import TonalBuzzer
from gpiozero.tones import Tone
from time import sleep
```

- Add a line below your creation of `button` to add a `TonalBuzzer` object:

```
buzzer = TonalBuzzer(17)
```

- In GPIO Zero, a `TonalBuzzer` emits the tone of your choice using play method.
- The complete program will be:

```
from gpiozero import TonalBuzzer
from gpiozero.tones import Tone
from time import sleep

while True:
    buzzer.play(Tone("A4"))   # Musical note
    sleep(1)
    buzzer.play(Tone(220))   # Hz
    sleep(1)
    buzzer.stop()
```

Did you catch what is happening with the use of Tone() in the play method?

3. Module 3: “Level Up”

3.1. Objectives for Module 3

- Practise and improve coding and electronics skills introduced in earlier modules.

3.2. Review of Modules 1 and 2

3.2.1. Preliminaries

A reminder that the presentation and handouts will have a horizontal line across the page in various spots. At those locations, you are to signal the instructor that you have reached that point and are waiting on others to move on to the next section.

We’re doing this so that the instructor can provide information to the entire group at once, in a clear and consistent fashion.

3.2.2. Robo-Car Demo

- What did you think of the robo-car demo that was happening when you came in, especially knowing you’re going to build one of them?

3.2.3. Sharing and Turn-Taking

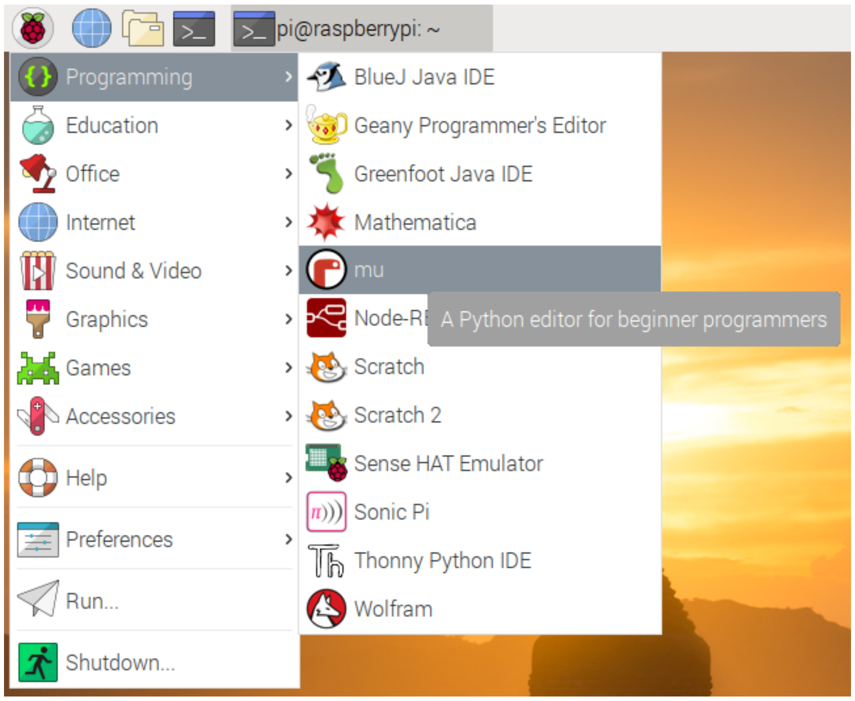
- Pilot and navigator
- Pilot controls keyboard and mouse
- Navigator can *advise* but doesn’t touch keyboard or mouse
- A reminder on sharing
- Switch who is pilot and who is navigator regularly, and fairly.

3.2.4. Review

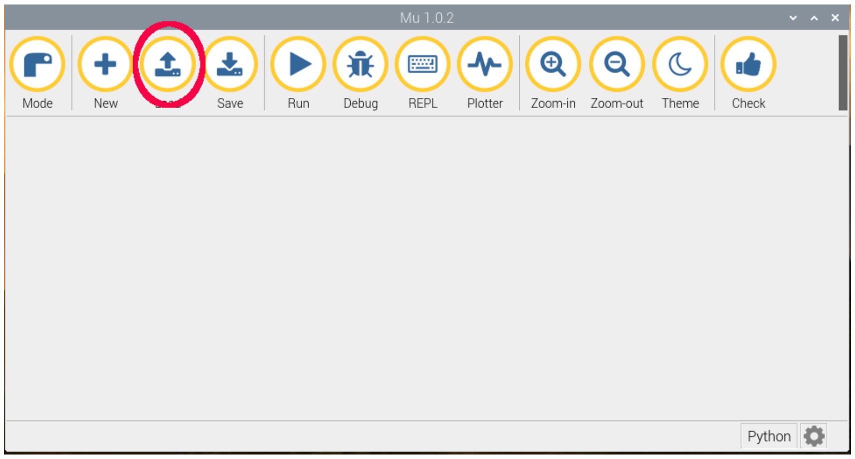
- Booting the Pi
- This time watch the small LEDs on the Pi as it boots (this will help you know when the Pi is booted, when the Pi is on the Robo- car and you don’t have a monitor attached).
- Discuss: Connecting devices to GPIOs
- Discuss: Controlling them via Python

3.3. Loading Files in Mu

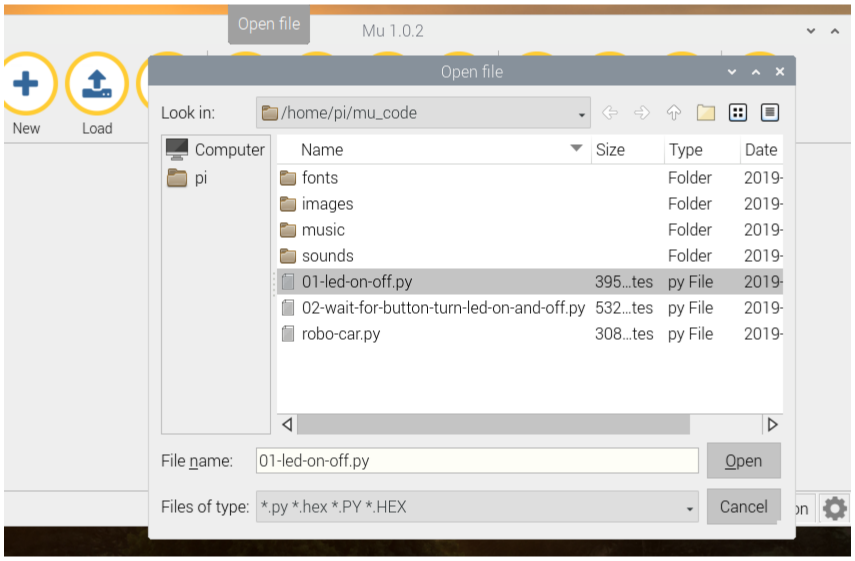
1. Open Mu



1. Click on Load Button



1. Select a File to Load



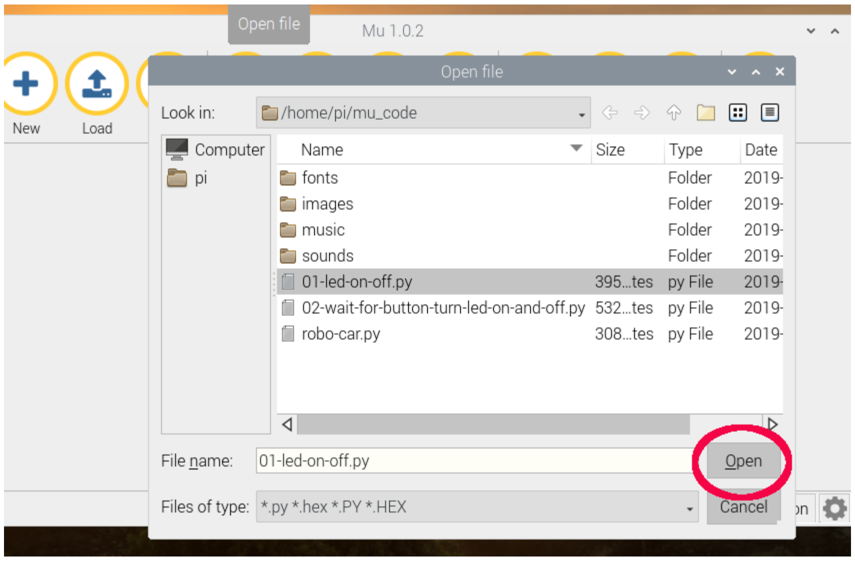
Look in: /home/pi/mu_code

Name	Size	Type	Date
fonts		Folder	2019-
images		Folder	2019-
music		Folder	2019-
sounds		Folder	2019-
01-led-on-off.py	395...tes	py File	2019-
02-wait-for-button-turn-led-on-and-off.py	532...tes	py File	2019-
robo-car.py	308...tes	py File	2019-

File name: 01-led-on-off.py

Files of type: *.py *.hex *.PY *.HEX

1. Click Open



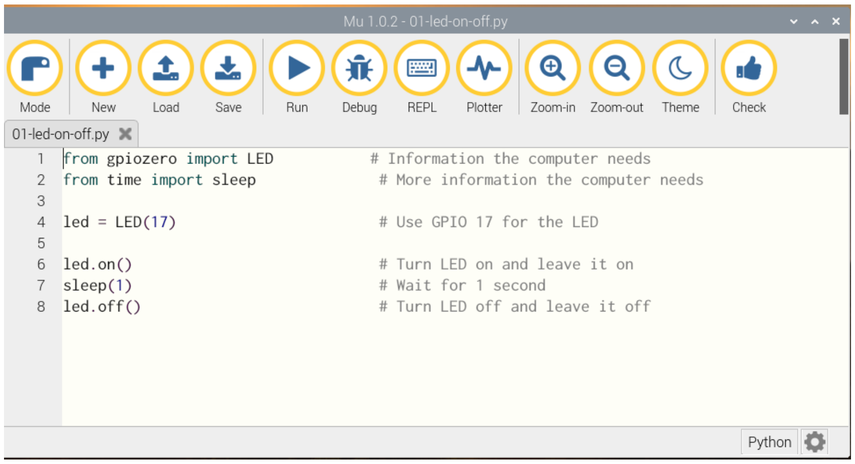
Look in: /home/pi/mu_code

Name	Size	Type	Date
fonts		Folder	2019-
images		Folder	2019-
music		Folder	2019-
sounds		Folder	2019-
01-led-on-off.py	395...tes	py File	2019-
02-wait-for-button-turn-led-on-and-off.py	532...tes	py File	2019-
robo-car.py	308...tes	py File	2019-

File name: 01-led-on-off.py

Files of type: *.py *.hex *.PY *.HEX

1. Code Should Be Loaded and in the Current Tab



```
from gpiozero import LED           # Information the computer needs
from time import sleep              # More information the computer needs

led = LED(17)                       # Use GPIO 17 for the LED

led.on()                            # Turn LED on and leave it on
sleep(1)                            # Wait for 1 second
led.off()                           # Turn LED off and leave it off
```

3.4. The Three Most Basic Concepts in Electronics

3.4.1. The Concepts

3.4.1.1. Like Water, Electricity Flows from High to Low

- You’ve seen with streams of water on rainy day that water flows high ground to lower ground.
- Like water, electricity flow from high (+ volts) to low (ground / 0 volts).

3.4.1.2. A Wire is a Channel for Electricity

- While playing with streams you’ve probably dug channels so the water can flow through faster and where you want. Wires and metal are like that for electricity.

3.4.1.3. Air is to Electricity as a Dam is to Water

- If you’ve built dams across streams of water on a rainy day, you’ve probably built dams across them. With electricity air is like putting a rock much larger than the stream across the stream. Electricity can’t get through unless there is a really massive amount of it (that’s when you get lightning).

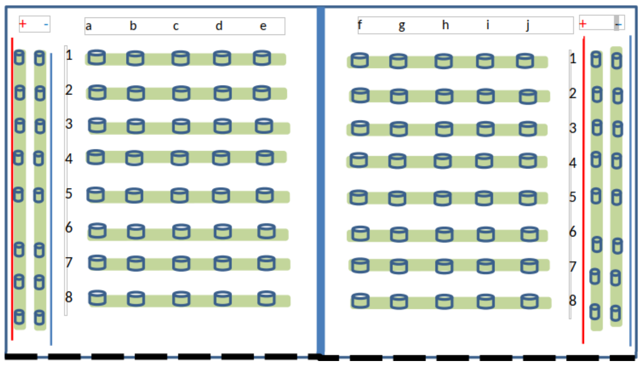
3.4.2. Consequences / Results / What This Means for Us

- To allow electricity to flow from high to low you need a wire to create a channel the electricity can go through.
- You don’t want to connect a wire directly from high to low because that creates an effect like a really big waterfall such as Niagra Falls — you end up with a huge amount of flow (current) which has too much power all once.
- By itself an LED is basically a single direction wire that happens to light up when electricity flows through it.
- So we add a partial dam, called a resistor, so that there is not too much electricity at once.
- A button is basically a device that is a wire that can be removed from the path. When the button is pressed the wire is in the path. When the button is not pressed (open) the wire is not in the path, so the electricity is stopped by the air (dam).

Next we’ll see how to use this to understand the breadboards we’ve been using.

3.5. Clarifying Breadboards

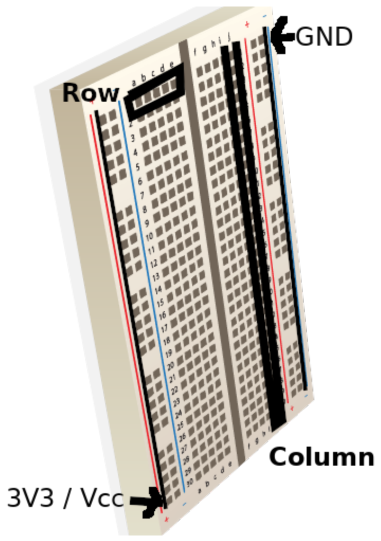
- We’re reviewing this to help you when we hook up the components in the next two sections.

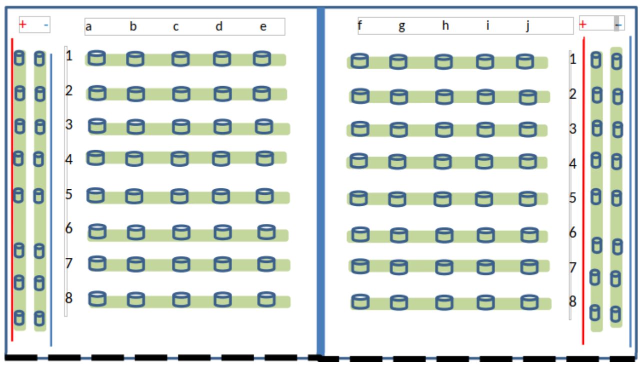


- The above diagram is a representation of a breadboard as seen from the top and seeing through the plastic.
- Notice the holes — these are the holes in which the jumpers are pushed, and are made of metal in order to conduct electricity (act as channels for the electricity).
- Notice that across the bottoms of the holes, along the rows, are flat wires — these connect the five holes in each row to each other (acting as channels for the electricity).

- Note that the columns are not connected (so there can be no flow between columns unless you add a jumper between the holes in different columns).

- Take a look at see the diagram from the previous module, and see if you can see how it lines up with the new diagram.



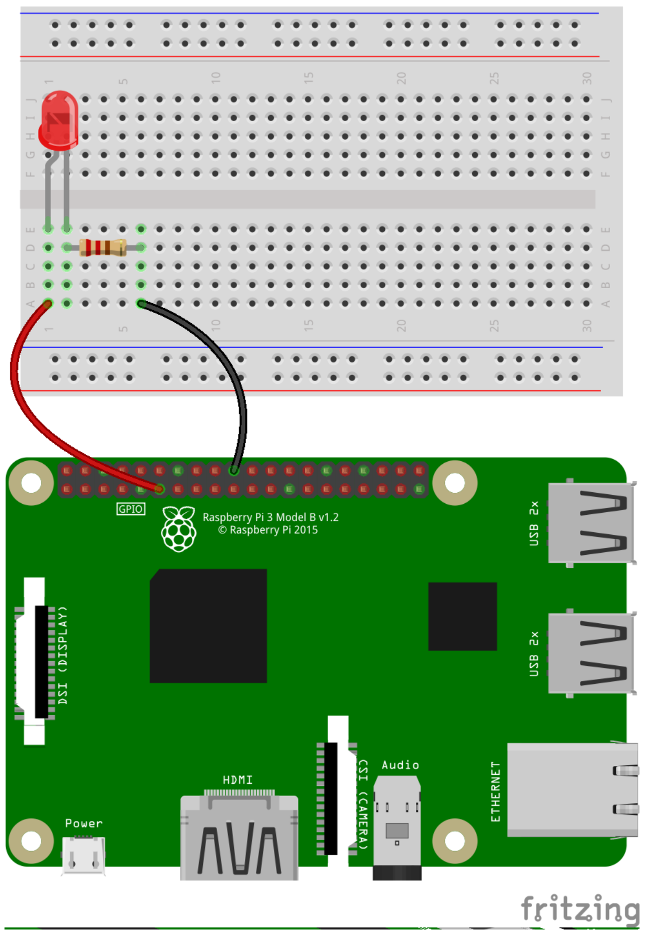


- Comments or questions?

3.6. LED On, LED Off

3.6.1. Hardware

- Connect the long leg of the LED to GPIO 17, using the breadboard and a female to male jumper wire.
- Connect the short leg of the LED to a resistor using the breadboard, and connect the other side of the resistor to GND (ground) on the Pi.



- When the Pi executes `led.on()` GPIO17 will become +3.3V (high) and electricity will flow from GPIO17 through the LED and resistor, and finally to GND (ground / 0 Volts).

3.6.2. Software

Once you have the LED and resistor connected, you should load `01-led-on-off.py` using the `Load` button in Mu.

```
from gpiozero import LED                # Information the computer need
from time import sleep                  # More information the computer
                                        # needs

led = LED(17)                           # Use GPIO 17 for the LED

led.on()                                # Turn LED on and leave it on
sleep(1)                                # Wait for 1 second
led.off()                               # Turn LED off and leave it off
```

3.6.2.1. Questions

- These are practice for what you will be doing with the robo-car code we will provide for you to use with your robo-car.
- You will need to modify the robo-car code using the techniques you learn here.
- Where do you tell the computer to use GPIO 17?
- How do you turn the LED on?
- What does `sleep(1)` do?
- How long does the LED stay on, and why?
- How do you turn the LED off?

3.6.2.2. Exercises

- Change the amount of time the LED stays on.

3.6.3. Another Question

- How would make the LED turn on and off a second time?

For the exercises below you probably don’t want to type the answers, so we’re going to show you how to make a copy of the code you need. You should have a handout titled ‘Avoid Typing: Cut & Paste Handout’. We’ve done this so you can refer to it as needed.

Let’s walk through using cut & paste to do the exercise below. Don’t worry, this gets easier with practice.

3.6.3.1. More Exercises

Remember: This should be your partner’s turn to be pilot.

- Turn the LED on and off a second time.

3.6.3.2. Repeat the Exercise With the Other Partner Piloting

- Turn the LED on and off a second time.

Because the last exercise was a repeat we won’t swap until after this next exercise.

3.6.4. Another Question

- How would make the LED turn on and off a third time?

3.6.5. Another Exercise

- Turn the LED on and off a third time.

3.6.6. And Another Question

- How would you turn the LED on and not turn it off?

3.6.7. More Exercises

Please swap the hot seat

- Turn the LED on and leave it turned on.

3.6.8. And Another Exercise

And another swap

- Turn the LED on for 1 second, turn if off, then turn it on for 2 seconds, and turn it off.

3.7. Easy Button

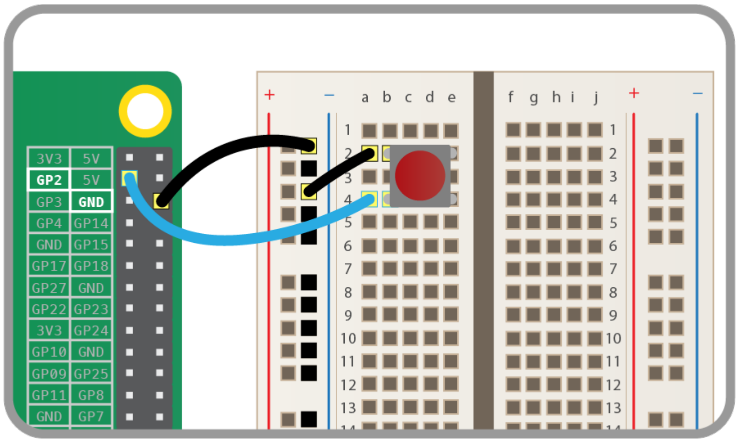
Sadly our easy button is not as easy as the Staples® easy button.

3.7.1. Hardware

Keep the LED connected.

3.7.2. Adding a Button

- Positioning these buttons correctly is rather tricky. Don't worry. If you end up with the button turned the wrong way, all that will happen is that it will act as if it was always pressed. If that is what you are seeing, just rotate the button a quarter of way around.
- Connect one pin of the button to GPIO 2 and the other to GND (ground).



- This creates a second circuit (separate from LED circuit) in which no electricity flows unless the button is pressed. It is actually the commands you use on the Pi that turn on the LED.
- When the button is pressed electricity flows from the GPIO2 (which acts as 3V3 volts), through a resistor inside the Pi, to GND (ground / 0 volts). The Pi senses this.
- Extra Information (ignore if you wish): We call a circuit which is 'on' when it goes low (gets connected to ground) 'active low'. Most of the time in digital electronics, devices are 'active high'. (that is they are 'on' when connected to power / positive voltage).
- Since we can tell the Pi to pause until it senses the button being pressed, we can send commands to the Pi to light the LED once the button is pressed.

3.7.3. Load the Code

In Mu, Load `02-wait-for-button-turn-led-on-and-off.py`

It should contain

```
from gpiozero import LED, Button    # Information the computer need
from time import sleep              # More information the computer
                                    # needs

led = LED(17)                       # Use GPIO 17 for the LED
button = Button(2)                  # Use GPIO 2 for the Button

button.wait for press()             # Wait until the button is pres
led.on()                            # Turn LED on and leave it on
sleep(1)                            # Wait for 1 second
led.off()                           # Turn LED off and leave it off
```

3.7.4. Question

How would you have Pi wait one second after the button is pressed before turning the LED on and off?

3.7.5. Exercise

‘Hot potato!’ time

Make the Pi wait one second after the button is pressed before turning the LED on and off.

3.7.6. Question

How would you make the Pi wait until the button is pressed to light the LED, light the LED (and leave it on), and then wait until you press the button again to turn the LED off?

3.7.7. Exercise

Whackamole!

Make the Pi wait until the button is pressed to light the LED, light the LED (and leave it on), and then wait until you press the button again to turn the LED off?

3.7.7.1. Experimentation

Take turns trying making interesting modifications to the programs you’ve worked with today.

3.8. The Future is Now: Laser Cutting

3.8.1. Watch Chassis Fabrication

At about 4:30pm stop and take the time to watch a chassis being fabricated (cut in the laser cutter).

4. Robo-Car Assembly Instructions

Robo-Car Construction (approx. 90 minutes)

4.1. Objectives for Module 4

- Follow directions in order to build a robo-car.

4.2. Sources

- Original Word Document
- Inspired by: Pi Curriculum’s Build a Robot Buggy

4.3. Review of Modules 1-3

4.4. Robo-Car Physical Assembly

4.4.1. Step One

Set up a clean area to assemble your Robo-Car. You should have:

- A chassis board(board on which we will build the Robo-Car)

- A kit of the parts you will need (supplied as needed)

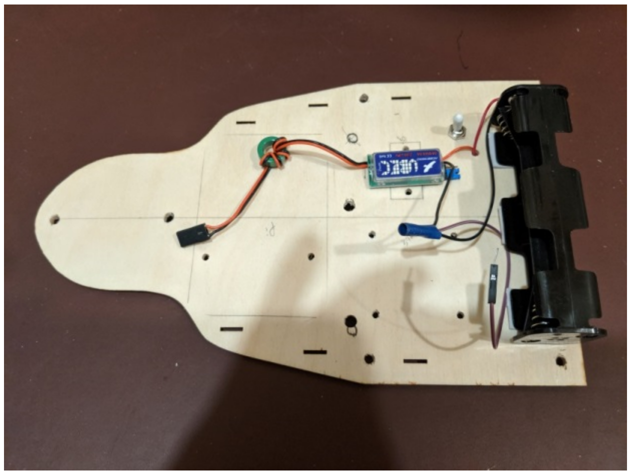
- A small screwdriver

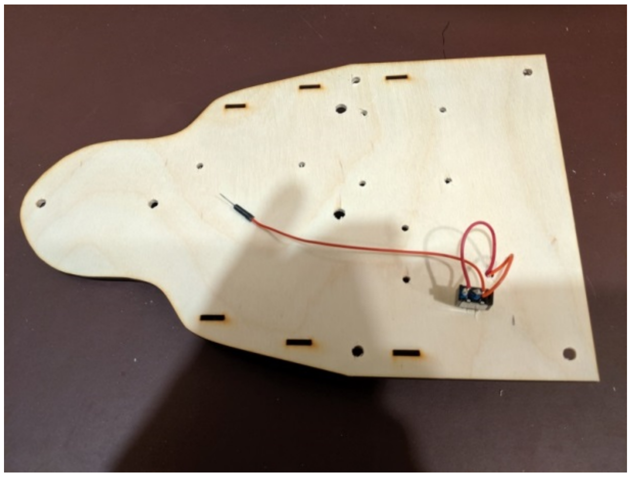
Follow the directions below. Do not go onto the next step until you have finished the step you are on. If something does not look right, call a team leader.

4.4.2. Step Two – Attaching the UBEC (new parts)

(The UBEC is a power supply that we will use to power the Raspberry Pi)

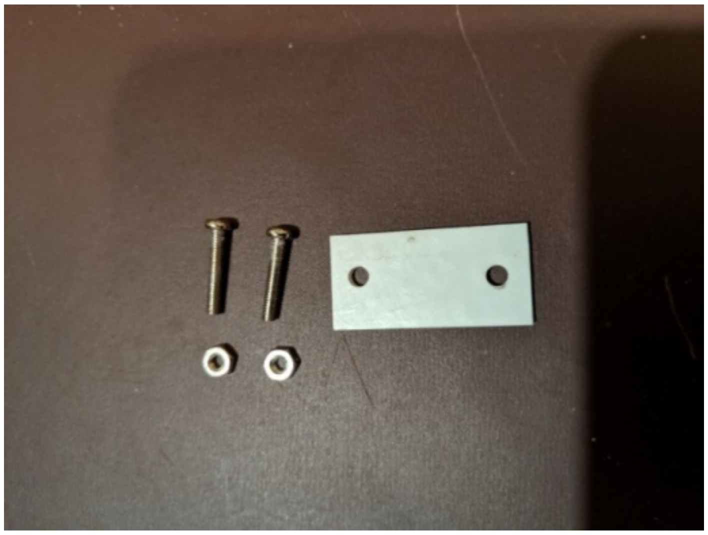
Get your **chassis board**.



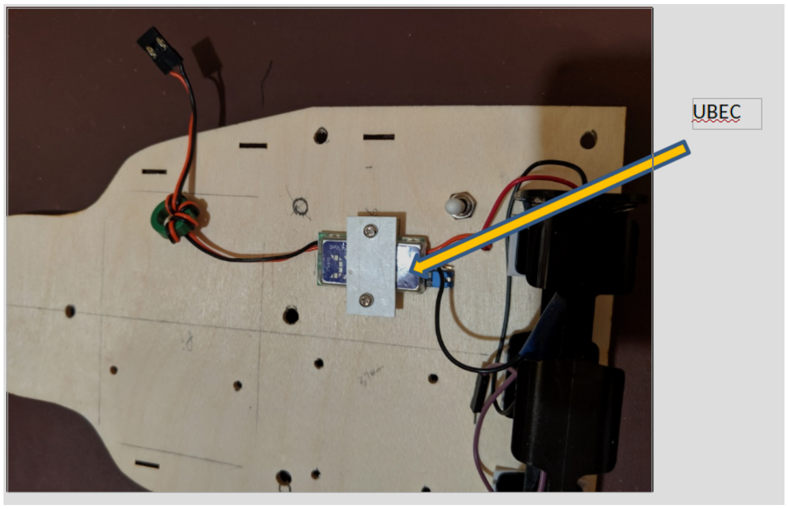


CHASSIS BOARD TOP **CHASSIS BOARD BOTTOM**

Get the **UBEC tie down strap** (rectangle with 2 holes) and two 2.6 mm (thin) bolts with nuts.



Place the UBEC on the top of the chassis board between the two mounting holes. Push the two bolts through the holes on the strap and then through the two holes on the chassis board. Secure the bolts underneath the chassis board with the two nuts. As you tighten the nuts the UBEC will be held in place by the strap. (**See picture below**)

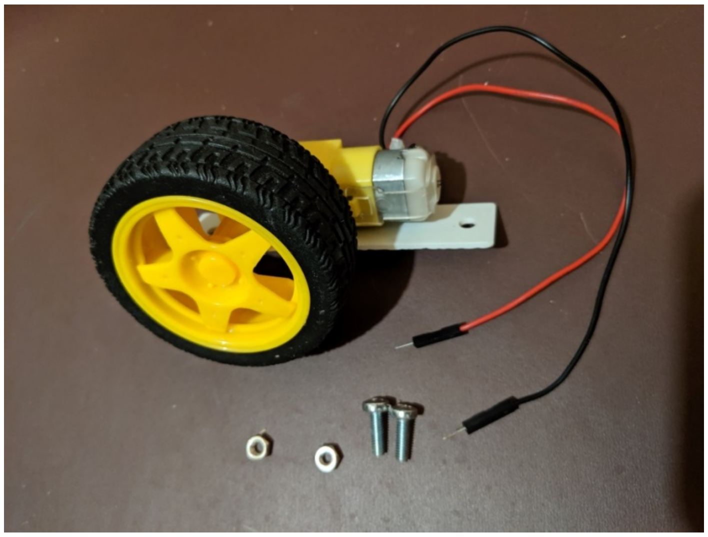


STOP – Wait until directed before going to the next step

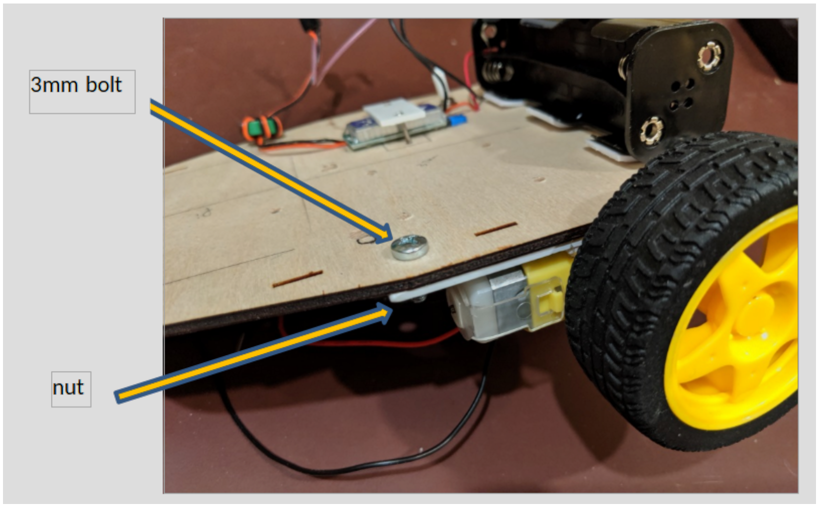
4.4.3. Step Three – Attaching the Motor and Wheels (new parts)

(The motor and wheels will move your robo-car)

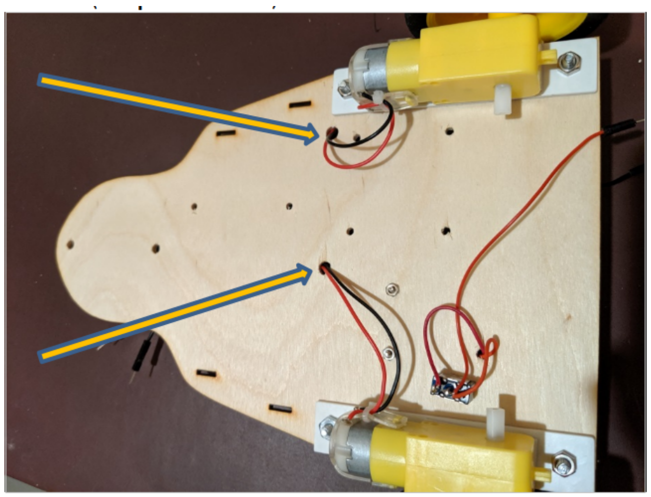
Get the two **wheel assemblies**. Each assembly is made up of a wheel, a yellow motor, an attached mounting plate and two attached wires. You also need two 3.5 mm nuts(thick) and bolts.



Bolt the two motor/wheel assemblies, one to each side of the rear underside of the chassis board. Use 3mm nuts (thick) and bolts. Two bolts per wheel assembly. (**See picture below**)



On the underside, feed the red and black wires from each motor through their respective holes to the top of the chassis board. (**See picture below**)

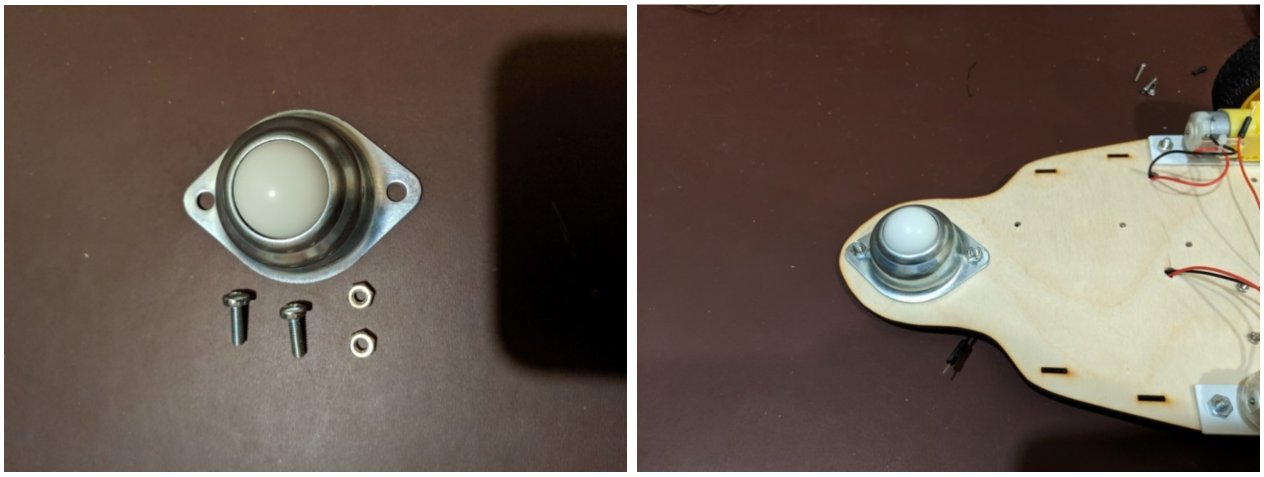


STOP – Wait until directed before going to the next step

4.4.4. Step Four – Attaching the Caster Wheel (new parts)

(The caster wheel allows to Robo-Car to move in any direction)

Bolt one **caster wheel** to chassis board, front underside using 3mm bolts (thick) and nuts. Two bolts per wheel assembly. (**See picture below**)

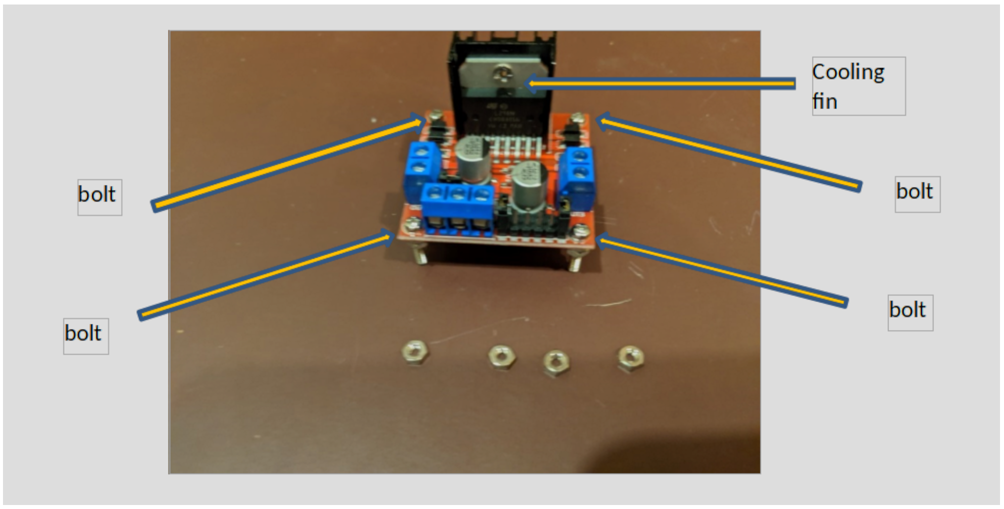


STOP – Wait until directed before going to the next step

4.4.5. Step Five – Installing the Motor Controller (new parts)

(The motor controller uses computer control to send power to the wheels)

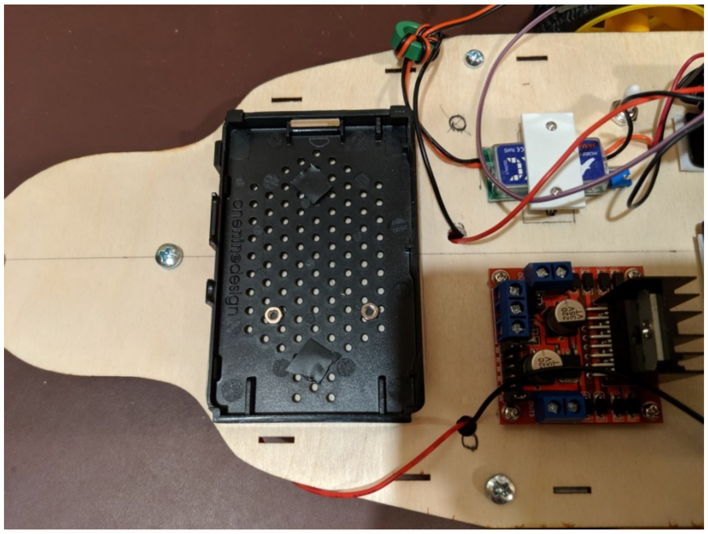
Bolt the **motor controller** to the middle upper side of chassis beside the UBEC. Push the four bolts gently through the four holes on the top of the chassis board. On the underside use four 2.6 mm (thin) nuts to secure the controller. Do not over-tighten the bolts. (**See picture below**). Note: cooling fins (black) are towards the batteries.



STOP – Wait until directed before going to the next step

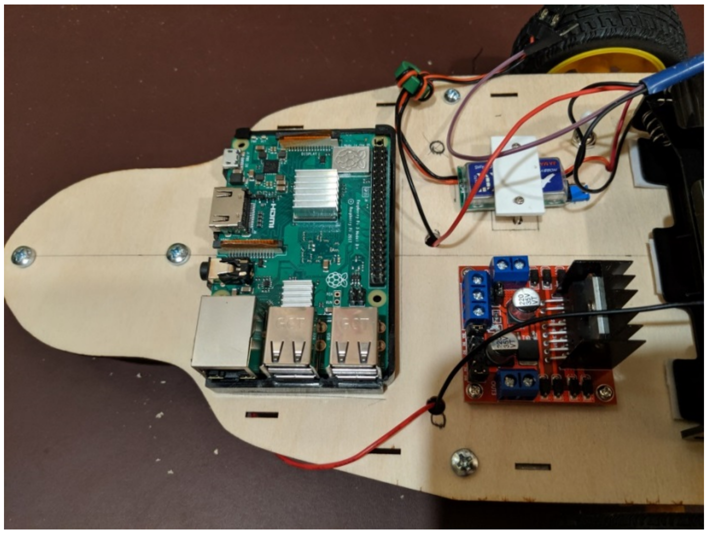
4.4.6. Step Six – Installing the Raspberry Pi (new parts)

Bolt the **Raspberry Pi lower case** to the chassis in front of the motor controller using two 2.6 mm bolts (thin). Ensure the case is positioned so that the GPIO pins are towards the batteries. Gently insert the Raspberry Pi and carefully snap it into position. (May need help from team leader.) If done correctly it will not fall out, even if the Robo-Car is turned upside-down. (**See picture below**)



4.4.7. Step Seven – Assembly Finished

At this point all the major parts are fixed to the chassis board. (**See picture below**). Now for the wiring.

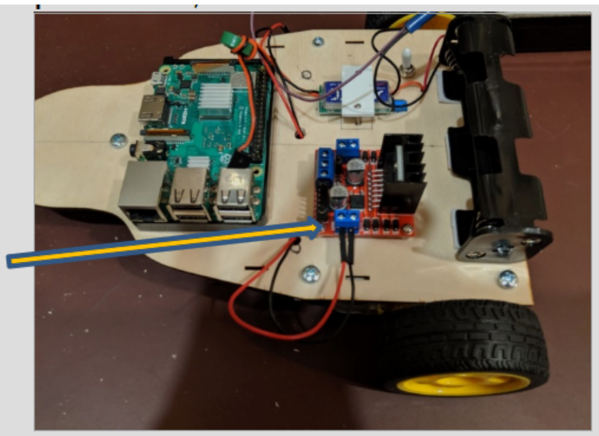


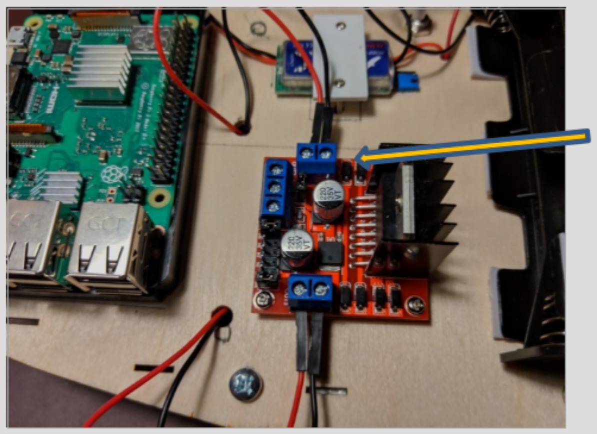
STOP – Wait until directed before going to the next step

4.5. Wiring

4.5.1. Step One – Wire the Motors to the Controller

Locate the **two wires from each wheel assembly** that you pushed through from the underside of the chassis. Select the two wires coming from the left wheel assembly. Using the screwdriver, loosen the two hold-down screws in the blue two- terminal block on the **side** of the motor controller closest to that wheel. Insert one lead into each of the holes under the screw and tighten the hold down screws. Repeat this step for the right wheel assembly which is on the right side of the motor controller. **(See picture below)**

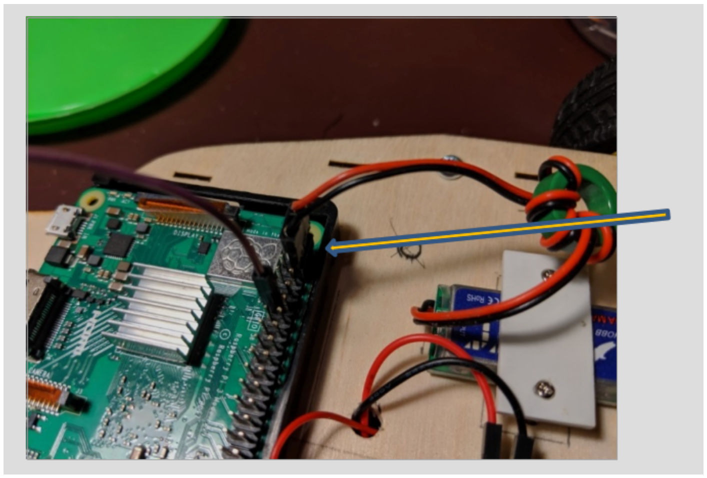

LEFT WHEEL CONNECTION


RIGHT WHEEL CONNECTION

STOP – Wait until directed before going to the next step

4.5.2. Step Two – Wire the UBEC to the Raspberry Pi GPIO (new parts a)

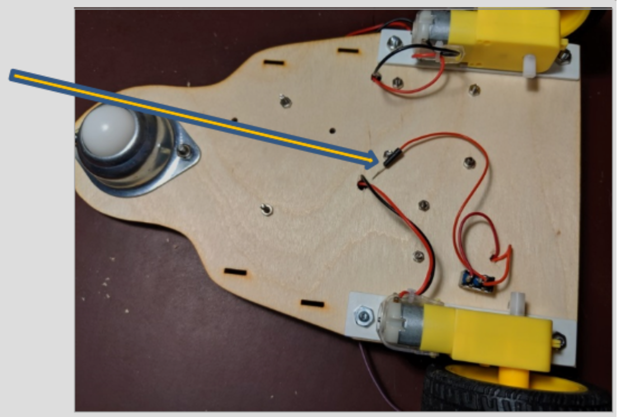
Use the UBEC output connection “harness” (two wires in one plastic connector) to connect to both the 5V (red goes to Pin 2) and Ground (black goes to Pin 6) (black). Consult the RPi **GPIO Pinout Diagram** before making this connection **(See picture below**)



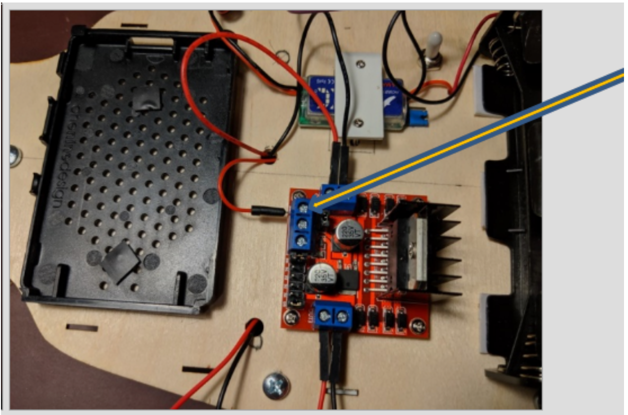
STOP – Wait until directed before going to the next step

4.5.2.1. Step Three – Connect Vcc to the Controller

On the underside of the chassis board, feed the **loose Vcc wire** connected to the switch through the access hole, as shown. Pull it through and connect it to the Vcc terminal on the motor controller. Using a screwdriver, loosen the Vcc terminal (the one in the blue three- terminal block that is closest to the UBEC). Insert the wire in the hole and tighten the screw. (**See picture below**).



UNDERSIDE OF CHASSIS BOARD
Loose Vcc wire before being put through hole



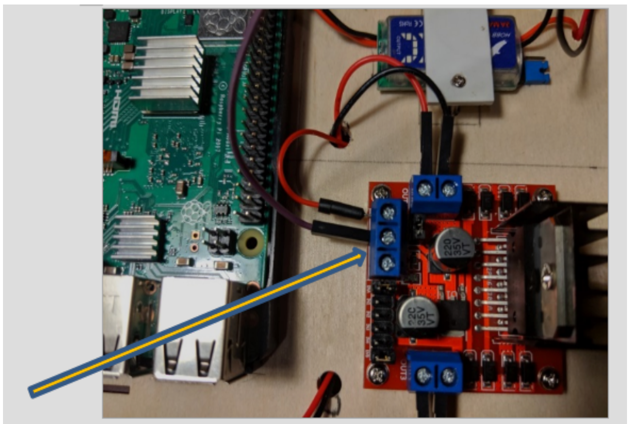
TOPSIDE OF CHASSIS BOARD
Vcc wire connected to motor controller

STOP – Wait until directed before going to the next step

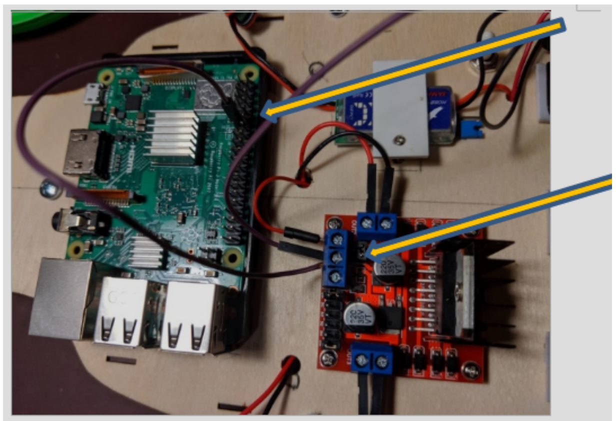
4.5.3. Step Four – Connect Pi and Controller Ground (new parts b)

Use the Ground terminal on the motor controller (the centre terminal on the blue three- terminal block) to attach **both**:

- the Ground (black wire) from the battery
- the RaspberryPi ground (Pin 9) by using a female-male jumper wire (**See picture below**).



BATTERY GROUND CONNECTED TO CONTROLLER

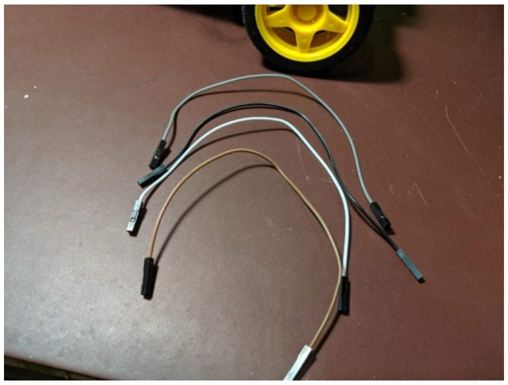


GROUND FROM RASPBERRY PI CONNECTED

STOP – Wait until directed before going to the next step

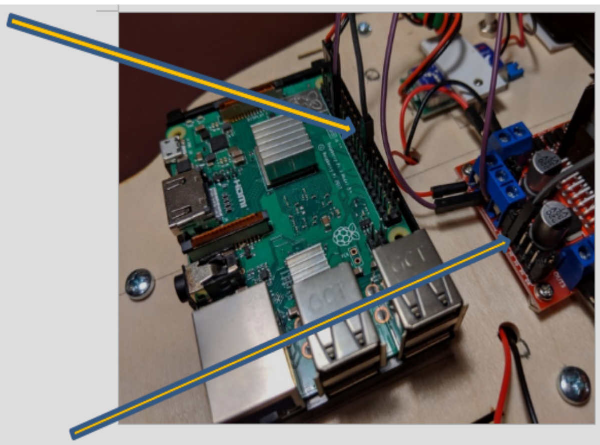
4.5.4. Step Five – Wire the GPIO to the Controller (new parts c)

In this step you are going to connect the Raspberry Pi (the brains) to the motor controller (does all the work of moving the wheels). You will need **four female-to-female jumper cables**. **(See picture below**)

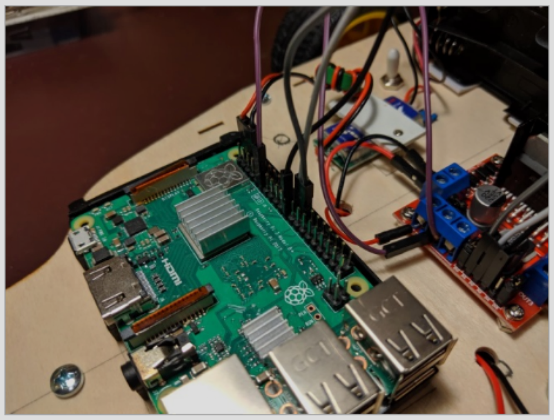


Using the **GPIO Pinout Diagram**, make the following connections**. Note: the GPIO number is <u>not</u> the same as the physical pin number that you have used up to this part of the lesson**.

Motor Controller Pin	GPIO Number on Raspberry Pi
IN1	GPIO 7
IN2	GPIO 8
IN3	GPIO 9
IN4	GPIO 10



IN1 ON CONTROLLER CONNECTED TO GPIO 7

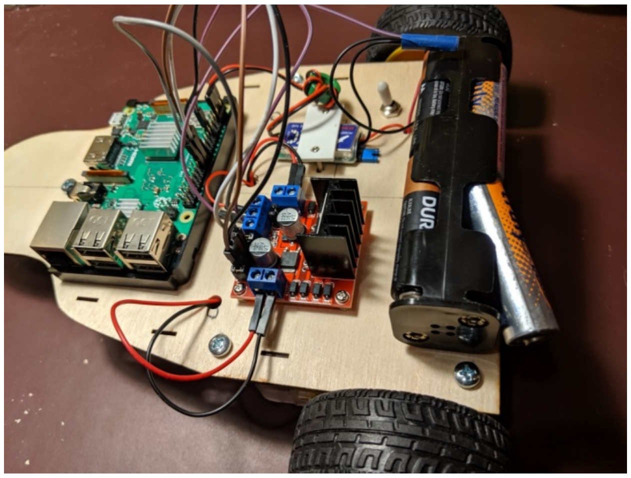


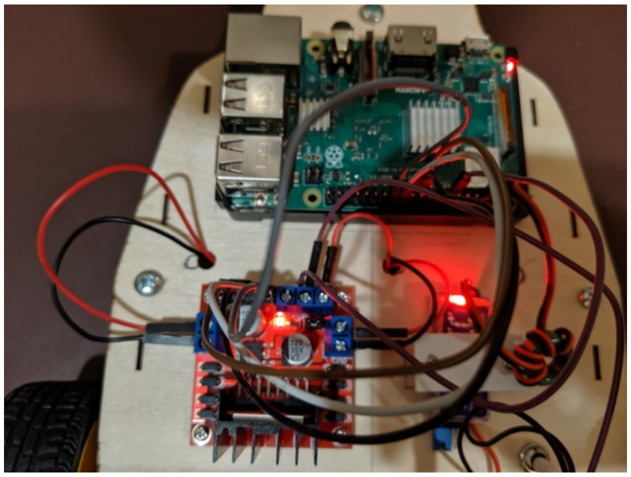
ALL CONTROLLER PINS CONNECTED

Stop and re-check the entire wiring diagram. Wait until directed before going to the next step.

4.5.5. Step Six – Lights On (new parts)

Insert the last battery into the battery holder and turn on the switch. If you have assembled the Robo-Car properly the red LED pilot lights should be on for the UBEC, the motor controller and the Raspberry Pi. If all three are not all on, turn off the switch immediately and recheck the wiring.


INSERT THE LAST BATTERY


SWITCH ON

5. Module 5 – Operating the RoboCar

Tues October 15

5.1. Historical Note

The first power on of an electronics project is called a smoke test because in the early days of electronics, the first test of devices often failed spectacularly with lots smoke and fried components. Modern electronics is much safer.

5.2. Objectives for Module 5

- Verify the robo-car build
- Learn how to run a program to control the robo-car, from a laptop.

5.3. Sources

- Original Word Document
- Inspired by: Pi Curriculum’s Build a Robot Buggy

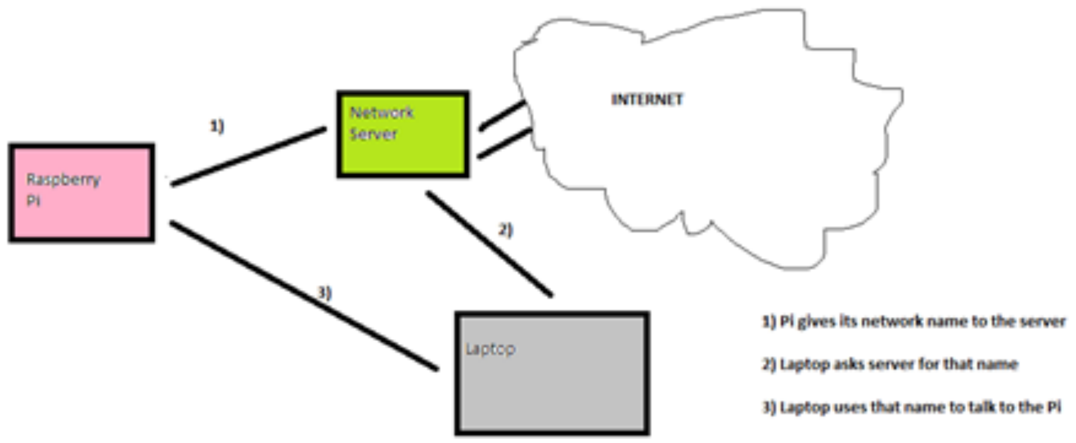
5.4. Review of Modules 1-4

5.5. SECTION ONE - Connecting to the RoboCar Wirelessly

You built the RoboCar – but it won't run unless we load it with a program. We know our RoboCar can't have any wires attached, so how do we load the program?

We will use a wireless communication network called WIFI. Luckily the Raspberry Pi has its own WIFI chip (underside near the SD holder.)

So how do we get the laptop computer to talk to the Raspberry Pi?

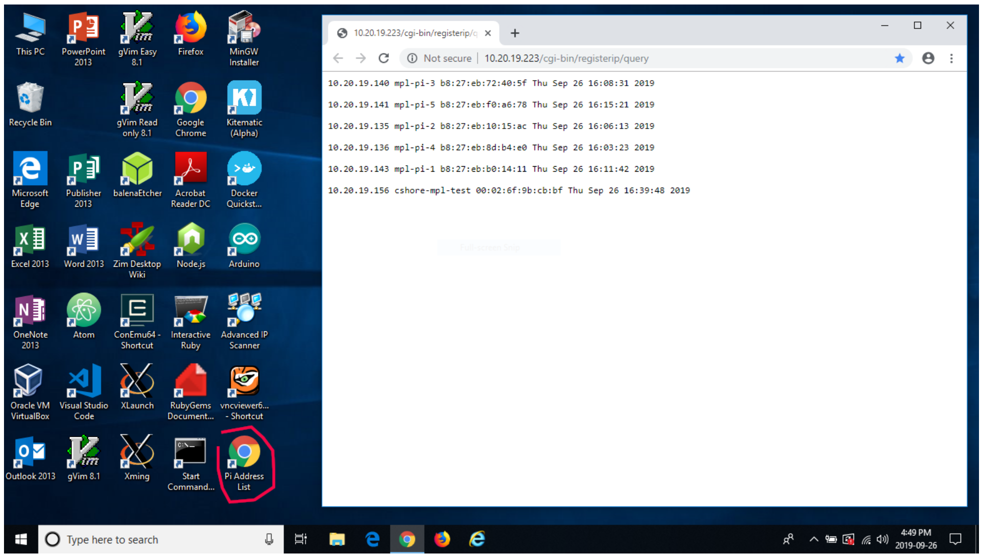

INTERNET
Network Server
1)
Raspberry Pi
2)
3)
Laptop
1) Pi gives its network name to the server
2) Laptop asks server for that name
3) Laptop uses that name to talk to the Pi

5.5.1. Step One – Raspberry Pi Gets a Network Name

When you turn on the Raspberry Pi it runs a program that searches for WIFI connections and then obtains a network name that it will use for WIFI communication. It uses a protocol (rules for communication) called VNC – Virtual Network Computing. The network server stores all of these names.

5.5.2. Step Two – You Get that Name from the Server

1. Use your laptop to communicate with the network server and find out what the network name is that you need to talk to your own Raspberry Pi.
2. Turn on laptop. Turn on your Raspberry Pi.
3. Click on ICON – "Pi Address List". Will open a window.



```
10.20.19.140 mpl-pi-3 b8:27:eb:72:40:5f Thu Sep 26 16:08:31 2019

10.20.19.141 mpl-pi-5 b8:27:eb:f0:a6:78 Thu Sep 26 16:15:21 2019

10.20.19.135 mpl-pi-2 b8:27:eb:10:15:ac Thu Sep 26 16:06:13 2019

10.20.19.136 mpl-pi-4 b8:27:eb:8d:b4:e0 Thu Sep 26 16:03:23 2019

10.20.19.143 mpl-pi-1 b8:27:eb:b0:14:11 Thu Sep 26 16:11:42 2019

10.20.19.156 cshore-mpl-test 00:02:6f:9b:cb:bf Thu Sep 26 16:39:48 2019
```

4. Look for the text “mpl-pi-#” where # is the number of your Team. So if you are in Team 2, look for mpl-pi-2.
5. In that same row where you found your Team number, you will find your Raspberry Pi’s network name. It’s at the start of the row and begins with three numbers “10.20.19.” and ends with a fourth number. Your complete Raspberry Pi WIFI name looks something like this - 10.20.19.162. Copy down all four numbers and keep them handy.

5.5.3. Step Three – Talk to the Raspberry Pi over the Wireless Network

In this step you connect your laptop directly to your Raspberry Pi via the WIFI network

1. Click on ICON – “vncviewer6”. It will open a window.
2. A pop-up box will ask you for your network address. Type in the four numbers that you obtained in STEP TWO exactly as shown. Remember that periods separate each of the four numbers
3. If you have connected properly you will see the normal Raspberry Pi opening screen. You can now operate the Raspberry Pi just as if you were connected through a hard wired keyboard and mouse.
4. Try out the Raspberry Pi screen connection to make sure it is working.
5. One thing to remember, every time you switch off your RoboCar it will lose its network connection. So you will have to go through the three step procedure to re-establish communication.

5.6. SECTION TWO - Set the RoboCar’s Basic Operation

Now that we can talk to our Raspberry Pi, we need to find out if we have built it correctly. When you tell your RoboCar to go forward, Raspberry Pi does not understand what forward means. You have to show it.

5.6.1. Step One – Turn on Robo-Car (may have been done already)

1. Turn the Robo-Car on and observe the RoboCar is ready
2. All three red lights on and the green LED on the Raspberry Pi is no longer flashing
3. Place you Robo-Car on the hoist block so its wheels are not touching the table.

5.6.2. Step Two – Turn on and Connect Laptop (may have been done already)

Turn on your computer and start the wireless VNC application. Log into the Raspberry Pi as a remote connection user.

5.6.3. Step Three – Load Test File

We now need to find out if the RoboCar will go forward and backward as commanded.

1. Click raspberry Pi icon at top left
2. Click on Applications
3. Click on **Mu**
4. Click on Open and select the file **GoForward.py**. Double click that file
5. The file will now be displayed in the **Mu** window. It should look like:

```
from gpiozero import Robot
import time

roboCar = Robot(left=(7,8)), right=(9,10))
roboCar.forward()
time.sleep(2)
roboCar.stop()
```

5.6.4. Step Four – Test Forward/Backward

Ensure your RoboCar is on the hoist block. Make sure the wheels are free to turn without hitting anything. Run the application and observe if any of the wheels are turning in reverse (would make the RoboCar go backwards).

Stop the application. At the motor controller, loosen the two wires that come from the wheel that is going backwards. Reverse the two wires and tighten the screws. Rerun the Step Four test until both wheels turn in a forward direction. Remember if you turned off the RoboCar at the switch you will have to go through the three step VNC procedure again to regain communication.

5.6.5. Step Five – Test Left/Right

We now need to find out if the RoboCar will go left and right as commanded.

1. Click raspberry Pi icon at top left
2. Click on Applications
3. Click on **Mu**
4. Click on Open and select the file **GoLeft.py**. Double click that file

The file will now be displayed in the **Mu** window. It should look like:

```
from gpiozero import Robot
import time
roboCar = Robot(left=(7,8)), right=(9,10))
roboCar.left()
time.sleep(2)
roboCar.stop()
```

If the command is working properly the wheel on the right side of the chassis board (looking from the back) should turn forward **and** the wheel on the left side should be turning backwards. If that is not the case, you need to make a change to the program.

Change the line

```
roboCar = Robot(left=(7,8), right=(9,10))
```

to

```
roboCar = Robot(left=(9,10), right=(7,8))
```

Note, if you need to make this change, you will have to make the same change to all future programs you write for your RoboCar.

5.7. SECTION THREE – Three Programming Tips

5.7.1. Tip One

The five basic commands you need to operate your RoboCar are:

1. `roboCar.forward()`
2. `roboCar.backward()`
3. `roboCar.left()`
4. `roboCar.right()`
5. `roboCar.stop()`

If you do not put a value between the parentheses (), the motor will run at full speed. If you put in a decimal value less that one, it will run at a slower speed. For example:

```
roboCar.forward(0.5)
```

The RoboCar will go forward at half speed. (A half (1/2) in decimal is written as 0.5)

5.7.2. Tip Two

The `time.sleep()` function tells the command in the line above it how long to operate. The sleep function understands that the value within the parentheses is the time you want in seconds. That time can be a large number `time.sleep(0.2)`). You can use different time intervals to control your RoboCar's actions.

```
    time.sleep(0.2)
```

5.7.3. Tip Three

Use the "Cut and Paste" shortcut to limit the amount of code you have to type into your program. Use the Cut and Paste handout sheet in your kit to help you.

5.8. SECTION FOUR - Your RoboCar is Ready for Action

Now that all the testing is complete we can move the RoboCar to the floor and see what it and your Team are able to do.

First of all go to the file library in **Mu** and load the file **RoboCar.py**.

```
from gpiozero import Robot
import time

roboCar = Robot(left=(7,8), right=(9,10))
roboCar.forward()
time.sleep(2)
roboCar.stop()
time.sleep(1)
roboCar.backward()
time.sleep(2)
roboCar.stop()
time.sleep(1)
roboCar.left()
time.sleep(1)
roboCar.stop()
time.sleep(1)
roboCar.right()
time.sleep(1)
roboCar.stop()
```

This is your basic operating program. You can load it at any time through **Mu**. When you run it your RoboCar will go forward a little, back a little and then spin in a circle. Now the challenge.

You will have to modify the **RoboCar.py** file making a new file that will allow you RoboCar to complete the three challenges that have been set up for you. Remember to use the three tips to help you.

5.8.1. Challenge One – There and Back

Set you RoboCar anywhere within the START AREA on the MakerPlace floor. (Go over and take a look at it). Program your RoboCar so that it goes forward and stops in the STOP AREA on the MakerPlace floor. Then have your RoboCar go backwards and return and stop in the START AREA. You may have to run the challenge more than once.

5.8.2. Challenge Two – A Triangle

Set your RoboCar anywhere within the START AREA on the MakerPlace floor. Make it go forward a distance of at least 1 meter. Have it turn and go in a different direction for a distance of at least 1 meter. Then have your RoboCar turn one more time and return to the START AREA. Its path will be a triangle. Can you make all the sides of the triangle the same length?

5.8.3. Challenge Three – An Obstacle (IF TIME PERMITS – SETUP REQUIRED)

Set you RoboCar anywhere within the START AREA on the MakerPlace floor. (Go over and take a look at it. Notice there is now something in the way.). Program your RoboCar so that it goes forward, turns to avoid the obstacle and ends up in the STOP AREA. You may have to program multiple turns for your RoboCar.

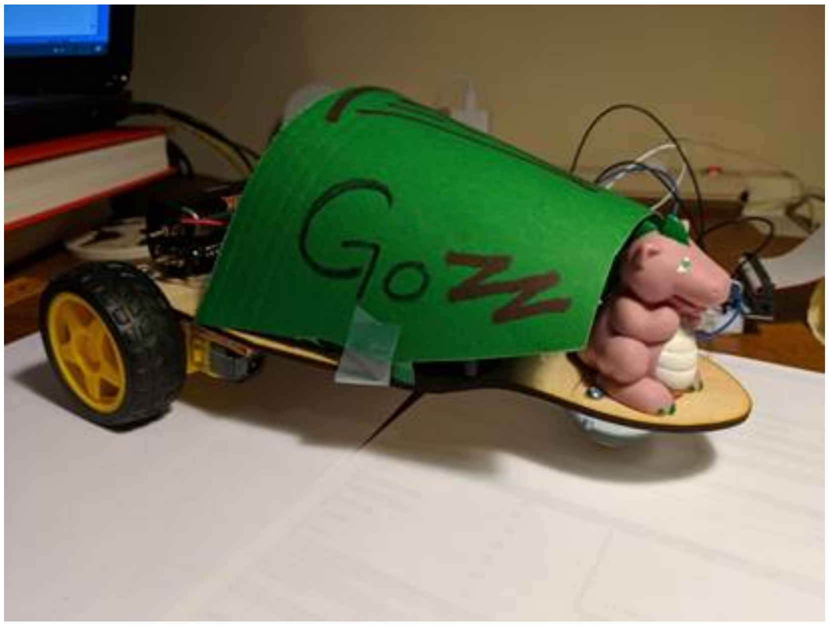
5.8.3.1. Hints for the Challenge

- When you write a new program for each challenge, save it and give it a different name so can call it up later. For example you may call the program you write for Challenge One "**There.py**"
- Remember you can change the speed of the motors by using a decimal value in the command. Example `roboCar.forward (0.4)`
- Remember you can change the time a motor is on (or off) by using the sleep command. Example `time.sleep(0.3)`
- When you are duplicating sections of code, remember to use the "cut and paste" technique. See the Cut and Paste Handout sheet.

Have fun and if you get stuck, ask for help.

5.9. Next Week

Next week we are going to have a major challenge match where you can show off your robot skills. But if your RoboCar is going to take the top prize it has to look good too. At the start of next week's lesson you will have 30 minutes to dress up you RoboCar so it is best in show. We will provide scissors, tape and markers. But you can bring things from home that you feel may help dress up your RoboCar.



6. Module 6 – Obstacle Course and Wrap Up

Tues October 22

6.1. Objectives for Module 6

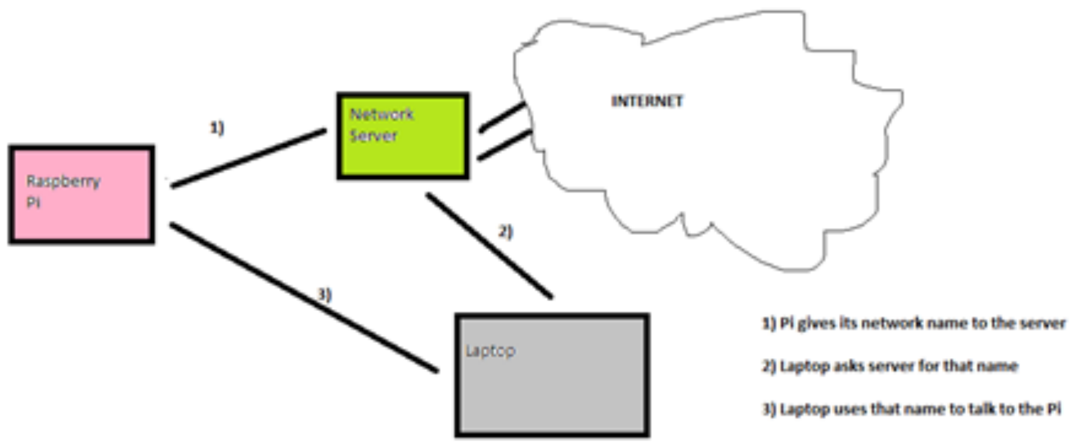
- Decorate the vehicle.
- Successfully navigate an obstacle course.
- Wrap up the program.

6.1.1. SECTION ONE - Connecting to the RoboCar Wirelessly

You built the RoboCar – but it won't run unless we load it with a program. We know our RoboCar can't have any wires attached, so how do we load the program?

We will use a wireless communication network called WIFI. Luckily the Raspberry Pi has its own WIFI chip (underside near the SD holder.)

So how do we get the laptop computer to talk to the Raspberry Pi?

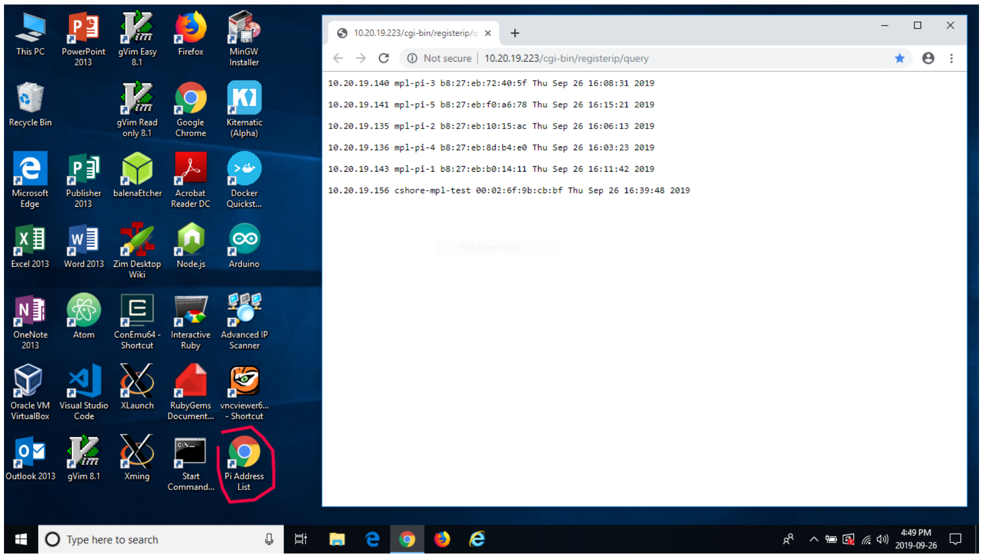

INTERNET
Network Server
Raspberry Pi
Laptop
1)
2)
3)
1) Pi gives its network name to the server
2) Laptop asks server for that name
3) Laptop uses that name to talk to the Pi

6.1.2. Step One – Raspberry Pi Gets a Network Name

When you turn on the Raspberry Pi it runs a program that searches for WIFI connections and then obtains a network name that it will use for WIFI communication. It uses a protocol (rules for communication) called VNC – Virtual Network Computing. The network server stores all of these names.

6.1.3. Step Two – You Get that Name from the Server

1. Use your laptop to communicate with the network server and find out what the network name is that you need to talk to your own Raspberry Pi.
2. Turn on laptop. Turn on your Raspberry Pi.
3. Click on ICON – “Pi Address List”. Will open a window.


10.20.19.223/cgi-bin/registerip/q
Not secure
10.20.19.223/cgi-bin/registerip/query
Pi Address List
Type here to search
4:49 PM
2019-09-26

```
10.20.19.140 mpl-pi-3 b8:27:eb:72:40:5f Thu Sep 26 16:08:31 2019

10.20.19.141 mpl-pi-5 b8:27:eb:f0:a6:78 Thu Sep 26 16:15:21 2019

10.20.19.135 mpl-pi-2 b8:27:eb:10:15:ac Thu Sep 26 16:06:13 2019

10.20.19.136 mpl-pi-4 b8:27:eb:8d:b4:e0 Thu Sep 26 16:03:23 2019

10.20.19.143 mpl-pi-1 b8:27:eb:b0:14:11 Thu Sep 26 16:11:42 2019

10.20.19.156 cshore-mpl-test 00:02:6f:9b:cb:bf Thu Sep 26 16:39:48 2019
```

4. Look for the text “mpl-pi-#” where # is the number of your Team. So if you are in Team 2, look for mpl-pi-2.
5. In that same row where you found your Team number, you will find your Raspberry Pi’s network name. It’s at the start of the row and begins with three numbers “10.20.19.” and ends with a fourth number. Your complete Raspberry Pi WIFI name looks something like this - 10.20.19.162. Copy down all four numbers and keep them handy.

6.1.4. Step Three – Talk to the Raspberry Pi over the Wireless Network

In this step you connect your laptop directly to your Raspberry Pi via the WIFI network

1. Click on ICON – “vncviewer6”. It will open a window.
2. A pop-up box will ask you for your network address. Type in the four numbers that you obtained in STEP TWO exactly as shown. Remember that periods separate each of the four numbers
3. If you have connected properly you will see the normal Raspberry Pi opening screen. You can now operate the Raspberry Pi just as if you were connected through a hard wired keyboard and mouse.
4. Try out the Raspberry Pi screen connection to make sure it is working.
5. One thing to remember, every time you switch off your RoboCar it will lose its network connection. So you will have to go through the three step procedure to re-establish communication.

6.2. SECTION TWO - Your RoboCar is Ready for the Obstacle Course

6.2.1. Preparation

Now that all the testing is complete we can move the RoboCar to the floor and see what it and your Team are able to do.

First of all go to the file library in **Mu** and load the file **RoboCar.py**.

```
from gpiozero import Robot
import time

roboCar = Robot(left=(7,8), right=(9,10))
roboCar.forward()
time.sleep(2)
roboCar.stop()
time.sleep(1)
roboCar.backward()
time.sleep(2)
roboCar.stop()
time.sleep(1)
roboCar.left()
time.sleep(1)
roboCar.stop()
time.sleep(1)
roboCar.right()
time.sleep(1)
roboCar.stop()
```

This is your basic operating program. You can load it at any time through **Mu**. When you run it your RoboCar will go forward a little, back a little and then spin in a circle. Now the challenge.

6.2.2. The Obstacle Course Challenge

Set you RoboCar anywhere within the START AREA on the MakerPlace floor.

Go over and take a look at it and observe what happens when you run the program. Now change the program so that your RoboCar goes forward, around the obstacle and comes to a stop in the STOP AREA on the MakerPlace floor. You will have to keep your RoboCar within the boundary markers, also shown on the floor.

Your changed program will likely require making the RoboCar go forward a number of times and make at least two turns. You will also have to run the course a number of times before you are successful. When you are ready for a “timed lap”, ask for one of the Course Leaders to come and observe your run. This time will be recorded on the board.

6.2.3. Please use the following tips in writing your RoboCar program

6.2.3.1. Tip One

The six basic commands you need to operate your RoboCar are:

1. `roboCar.forward()`
2. `roboCar.backward()`
3. `roboCar.left()`
4. `roboCar.right()`
5. `roboCar.stop()`
6. `time.sleep()`

6.2.3.2. Tip Two

If you do not put a value between the parentheses (), the motor will run at full speed. If you put in a decimal value less that one, it will run at a slower speed. For example:

```
roboCar.forward(0.5)
```

The RoboCar will go forward at half speed. (A half (1/2) in decimal is written as 0.5)

6.2.3.3. Tip Three

The `time.sleep()` function tells the command in the line above it how long to operate. The sleep function understands that the value within the parentheses is the time you want in seconds. That time can be a large number `time.sleep(0.2)`). You can use different time intervals to control your RoboCar’s actions.

```
    time.sleep(0.2)
```

6.2.3.4. Tip Four

Use the “Cut and Paste” shortcut to limit the amount of code you have to type into your program. It’s simple:/p>

- Step One – Highlight the code lines you wish to copy
- Step Two – While highlighted, press both **Ctrl** and **C**
- Step three – Move the cursor to where you want the copied code to go. Remember to right-click the mouse.
- Step Four – Press both **Ctrl** and **V**, and it’s done
- Refer to the Copy and Paste handout sheet in your kit to help you.

6.2.3.5. Tip Five

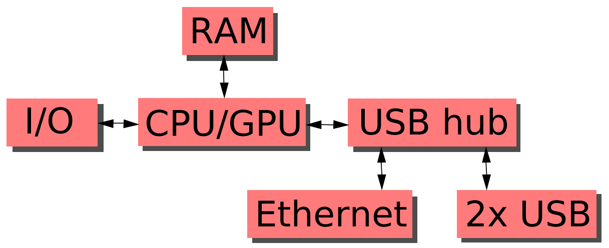
If you have made changes to the **RoboCar.py** program and want to reload the original code again, open the **RoboBackup.py** file.

Have fun and if you get stuck, ask for help.

7. Extras for Module 1 (Not Presented)

7.1. Module 1 Skipped or Simplified Material

7.1.1. How the board in front of you (Raspberry Pi) is a computer

- The following diagram shows an abstract view of the board does.
- 

- The largest chip on the board is what is known as the SoC or System-on-a-chip, which is listed on the diagram as 'CPU/GPU' and is the brains of the Pi. The system-on-a-chip has that name because there are a number of 'support chips', and other electronics that are required for a CPU/GPU to communicate with the outside world. The other large chip on the topside of the Pi is the USB controller. On the bottom of the board are the wireless (shielded), RAM, and MicroSD slot.
- The I/O (input/output) on the Pi is mostly in the SoC with access to the pins on the chip exposed with the double row of header pins that are on the board. In addition there is an internal USB hub that is used to connect the wired network connection and USB ports. The wireless is embedded into the Pi and does not go through the USB hub. (On most computers the wired connection is also more directly connected, through a faster connection know as a bus (a bus is collection of connections that work together).
- A USB hub is a device that allows multiple USB devices (including other USB hubs) to connect to a single USB controller.
- Another important piece of the Pi is the RAM — that is the scratchpad the CPU/GPU use while the Pi is powered on (but is not persistent).

- And finally, connected through more I/O is the micro SD card, which provides persistent storage, which means what is on it continues to exist when the Pi is turned off, even if it is removed from the Pi.

7.2. Module 1 Skipped or Simplified Material, Part II

7.2.1. What is The Raspberry Pi?

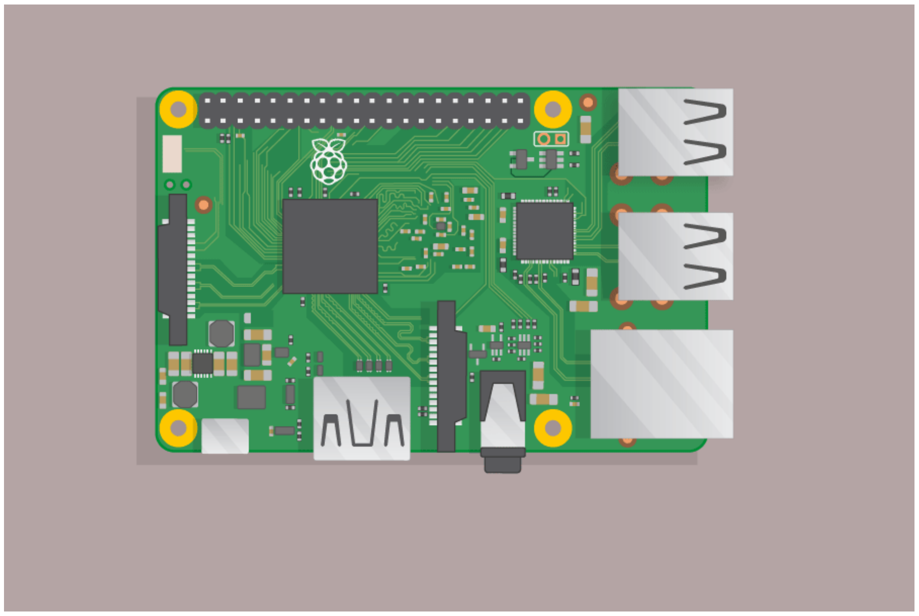
- A full-blown computer capable of running a desktop
- Based on a different processor (brains) than a typical laptop or desktop
- Performs well enough to be useful but less powerful than a typical desktop, laptop, or high-end mobile device (expensive cell phones, tablets, etc)
- Designed to be easy to use with electronics projects
- Created by a U.K. non-profit Rasbperry Pi Foundation
- Intended to an inexpensive way to learn computing and electronics and especially the combination of those (including robotics)

7.2.2. What does the Raspberry Pi 3 Have?

- Reasonably powerful (4 core x 1 GHz) ARM processor — runs the actual code that has been converted to machine instructions.
- 1 GB RAM — Scratchpad (goes away when power is turned off)
- microSD card support — permanent storage (slower than RAM)
- 4 x USB 2.0 ports — for peripherals, like keyboards, mice, and external storage
- HDMI out — video / graphics
- Audio out — sound / music
- Wifi — wireless network
- Ethernet — wired network
- Camera (with extra Pi module or USB)
- LCD (with extra Pi module or USB or GPIO or I2C or SPI controlled)
- GPIOs: Do on/off (digital) electronics including interfacing to other boards, controlling LED, listen to buttons, etc.
- Interface with I2C or SPI bus electronics — advanced
- Communicate with serial port devices (RX/TX/GND) — not in this workshop

7.2.3. Raspberry Pi hardware setup

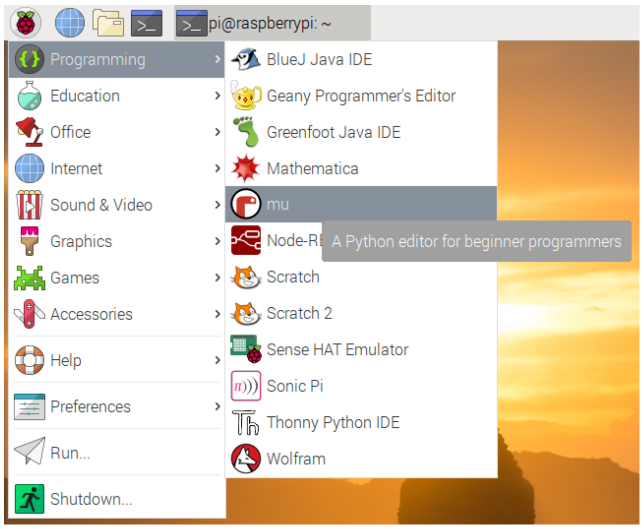
1. Put the Pi in its case and put the case together.
2. Next, plug your keyboard and mouse into the USB ports on the Raspberry Pi.
3. Make sure that your monitor or TV is turned on, and that you have selected the right input (e.g. HDMI 1, DVI, etc).
4. Connect your HDMI cable from your Raspberry Pi to your monitor or TV.
5. If you intend to connect your Raspberry Pi to the internet, plug an Ethernet cable into the Ethernet port, or connect a WiFi dongle to one of the USB ports (unless you have a Raspberry Pi 3, 4 or Zero in which case it has built in wifi).
6. When you’re happy that you have plugged all the cables and SD card in correctly, connect the micro USB power supply. This action will turn on and boot your Raspberry Pi.



7.2.4. The Pi desktop

- Once your Pi is booted, you should see a desktop that looks similar to older Windows® with the taskbar at the top instead of bottom.
- TBD: Screenshots of Pi with Applications menu open

- The ‘Application’ menu in the top left (which has a Raspberry icon) is the primary way to launch programs on the desktop.

- 

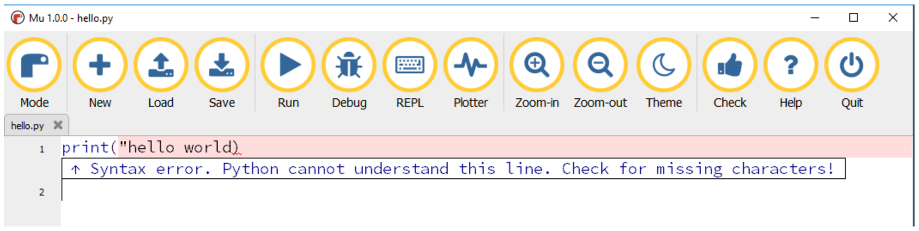
- On the top right is a clock, and an icon for the network connection (wireless in the case of those at the MakerPlace bots & bytes program).

- For now we’re not doing anything with the Pi desktop, except launching the ‘Mu’ editor from the ‘Programming’ menu in the Application menu (often this type of location is written in the same form as Applications|Programming|Mu).

7.3. Module 1 Skipped or Simplified Material, Part III

7.3.1. Checking

At any point while you’re coding, you can also get help advice from Mu without running your code: click the **Check** button to ask Mu to check your code for errors.



```
print("hello world)
 ↑ Syntax error. Python cannot understand this line. Check for missing characters!
```

When you have fixed an error, click **Check** again to see if the problem has disappeared.

7.3.2. Debugging

Unfortunately, not all problems with code are syntax errors (which Mu usually recognizes). Some errors in your code will be **bugs**, meaning your program runs fine, but it doesn’t do what you want it to do.

Mu has a debugger that allows you to step through your code line by line and look at what each line is doing. Information on this is available in the “Additional Information” section.

7.3.3. What next?

Check out the Mu website and the tutorials to learn more about this beginners’ IDE.

7.4. Module 1 Skipped or Simplified Material, Part IV

7.4.1. Resistors

7.4.1.1. What a resistor is

A device the impedes the flow of electricity. This reduces current available to the rest of the circuit (V=IR). We use these to prevent too much current from flowing through an LED (LED have very little resistance which mean current through the LED can be very high if connected without resister because it’s essential a short circuit).

The more current flowing through a resister the great the amount of heat that is generated by the resistor’s dropping current. This is why a low resistance resister needs to be able to withstand higher temperatures than a high resistance resistor (since more current flows) through the low voltage resistor.

7.4.1.2. Resistor Colour Code

Colour bands were used because they were easily and cheaply printed on tiny components. However, there were drawbacks, especially for colour blind people. Overheating of a component or dirt accumulation may make it impossible to distinguish brown from red or orange. Advances in printing technology have now made printed numbers more practical on small components. The values of components in surface mount packages are marked with printed alphanumeric codes instead of a colour code.

To distinguish left from right there is a gap between the C and D bands.

Ring color	Significant figures	Multiplier	Tolerance
None	–	–	±20
Pink	×10^−3	×0.001	
Silver	×10^−2	×0.01	±10
Gold	×10^−1	×0.1	±5
Black	×100	×1	
Brown	1	×10	±1
Red	×10^2	×10 0	±2
Orange	×10^3	×1000	±0.05
Yellow	×10^4	×10000	±0.02
Green	×10^5	×100000	±0.5
Blue	×10^6	×1000000	±0.25
Violet	×10^7	×10000000	±0.1
Grey	×10^8	×100000000	±0.01

7.4.2. LED

7.4.2.1. What it is

Light Emitting Diode — A diode only lets electricity through in one direction. For an LED, when electricity if flowing through the diode, it lights up.

7.4.2.2. Physical Characteristics (Oriention/Polarity)

- Long leg opposite flat edge
- Which way it’s connected (which side to power and which side to ground), matters.
- The proper orientation is:

7.5. Module 1 Skipped or Simplified Material, Part V

7.5.1. Outputs

A GPIO pin designated as an output pin can be set to high (3V3) or low (0V).

7.5.2. Inputs

A GPIO pin designated as an input pin can be read as high (3V3) or low (0V). This is made easier with the use of internal pull-up or pull-down resistors. Pins GPIO2 and GPIO3 have fixed pull-up resistors, but for other pins this can be configured in software.

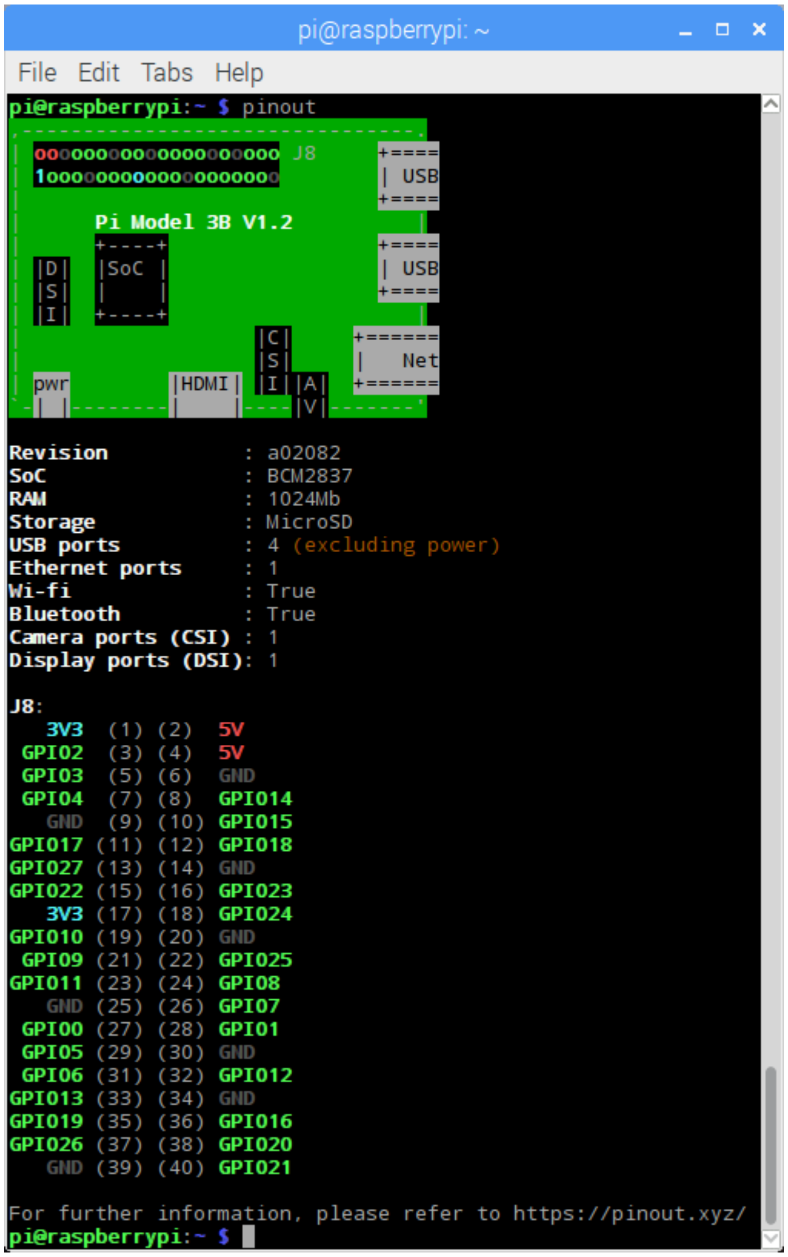
7.5.3. More

As well as simple input and output devices, the GPIO pins can be used with a variety of alternative functions, some are available on all pins, others on specific pins.

- PWM (pulse-width modulation)
    - Software PWM available on all pins
    - Hardware PWM available on GPIO12, GPIO13, GPIO18, GPIO19
- SPI
    - SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7)
    - SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16)
- I2C
    - Data: (GPIO2); Clock (GPIO3)
    - EEPROM Data: (GPIO0); EEPROM Clock (GPIO1)
- Serial
    - TX (GPIO14); RX (GPIO15)

7.5.4. GPIO Pinout

A handy reference can be accessed on the Raspberry Pi by opening a terminal window and running the command `pinout`. This tool is provided by the [GPIO Zero]() Python library, which it is installed by default on the Raspbian desktop image, but not on Raspbian Lite.



```
pi@raspberrypi:~ $ pinout
,--------------------------------.
| oooooooooooooooooooo J8     +====
| 1ooooooooooooooooooo        | USB
|                             +====
|      Pi Model 3B V1.2          |
|      +----+                 +====
| |D|  |SoC |                 | USB
| |S|  |    |                 +====
| |I|  +----+                    |
|                   |C|     +======
|                   |S|     |   Net
| pwr        |HDMI| |I||A|  +======
`-| |--------|    |----|V|-------'

Revision           : a02082
SoC                : BCM2837
RAM                : 1024Mb
Storage            : MicroSD
USB ports          : 4 (excluding power)
Ethernet ports     : 1
Wi-fi              : True
Bluetooth          : True
Camera ports (CSI) : 1
Display ports (DSI): 1

J8:
   3V3  (1) (2)  5V
 GPIO2  (3) (4)  5V
 GPIO3  (5) (6)  GND
 GPIO4  (7) (8)  GPIO14
   GND  (9) (10) GPIO15
GPIO17 (11) (12) GPIO18
GPIO27 (13) (14) GND
GPIO22 (15) (16) GPIO23
   3V3 (17) (18) GPIO24
GPIO10 (19) (20) GND
 GPIO9 (21) (22) GPIO25
GPIO11 (23) (24) GPIO8
   GND (25) (26) GPIO7
 GPIO0 (27) (28) GPIO1
 GPIO5 (29) (30) GND
 GPIO6 (31) (32) GPIO12
GPIO13 (33) (34) GND
GPIO19 (35) (36) GPIO16
GPIO26 (37) (38) GPIO20
   GND (39) (40) GPIO21

For further information, please refer to https://pinout.xyz/
pi@raspberrypi:~ $ 
```

For more details on the advanced capabilities of the GPIO pins see gadgetoid’s [interactive pinout diagram]().

7.6. Module 1 Skipped or Simplified Material, Part VI

7.6.1. About the import statement

The line `from gpiozero import LED` tells Python to pull in code from the gpio library, which is code that has been written by someone else and packaged so that you can use it easily.

In this case we are importing the LED class from that library. A class is a combination of variables (assigned values) and methods (things you can do with objects created from the class). An class is abstract which means that it doesn’t do anything until you give it’s variables some value. When we do that we say we ‘instantiate’ (create an instance/example of) the class to create an object which is can actually ‘do’ things.

When we say `led = LED(17)` we are saying create an instance (object) of the LED class that has a value of 17. For the LED class that means that GPIO 17 is what the methods will act.

The `led.on()` applies the method on() to the object led (which in this case is means turn GPIO 17 on).

When we turn GPIO 17 on, it becomes a 3.3V power source and creates a circuit though the LED and resistor to ground.

7.7. Module 1 Skipped or Simplified Material, Part VII

7.7.1. Time Library: Functions vs Methods

You may have noticed above that unlike the LED class from gpiozero that we use sleep from from the time library without instantiating an object.

That is because sleep is a regular function rather than a method. That is because a method is a function designed to work on objects of a class, while regular functions don’t require an object in order to do what they are supposed to do. Some languages don’t have objects at all, and only have variables and functions.

7.7.2. Loops

As you watched the LED you may have noticed that it continues to turn on and off until you click **Stop**. That is because the sequence of turning the led on, pausing (sleep), then off and pausing is inside a while loop. A while loop surrounds a group of statements that are to be executed for as long as the ‘condition’ of the while loop is ‘True’. In this the condition is ‘True’, and since True is always True, the while loop’s condition is always ‘True’, so the loop runs forever.

7.7.3. Another Way to Blink the LED

The gpiozero library's LED class has a `blink()` method, so instead of the above you can do:

```
while True:
    led.blink()
```

- Try adding some parameters to `blink` to make it blink faster or slower:
    - `led.blink(2, 2)` - 2 seconds on, 2 seconds off
    - `led.blink(0.5, 0.5)` - half a second on, half a second off
    - `led.blink(0.1, 0.2)` - one tenth of a second on, one fifth of a second off

  `blink`'s first two (optional) parameters are `on_time` and `off_time`: they both default to 1 second.

7.7.4. Your task

Change the flash speeds / on / off times.

7.7.5. Bonus task

Flash the LED in a sequence of your choice.

7.8. Additional Information Module 1

7.8.1. SD cards

The latest version of Raspbian, the default operating system recommended for the Raspberry Pi, requires an 8GB (or larger) micro SD card. Not all SD cards are made equal, and some have higher failure rates than others. If you're unsure, you can always buy our official SD cards from RS or Farnell. Any 8GB SD card will work, although you'll need to follow the software setup guide https://www.raspberrypi.org/learning/software-guide to learn how to load an operating system onto the card.

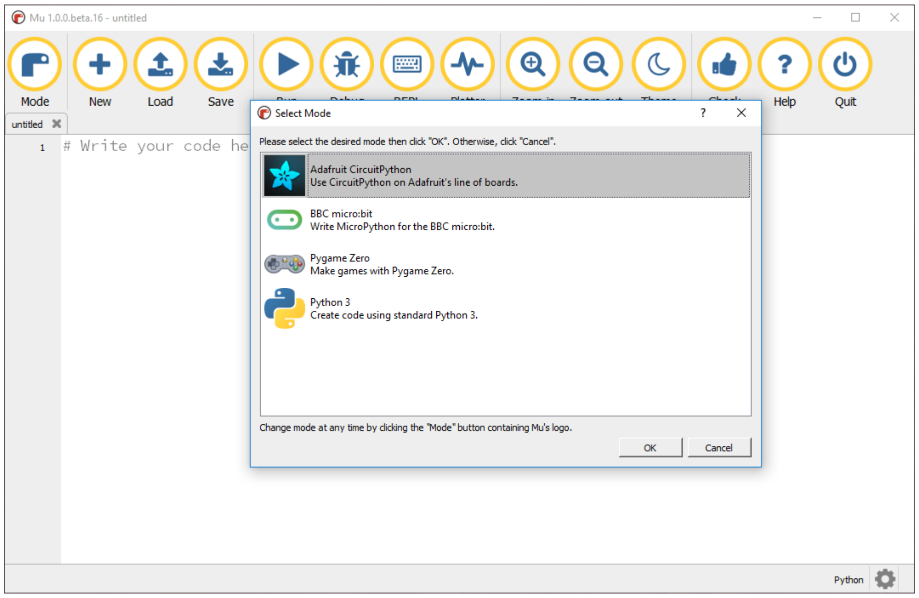
7.8.2. Mu

Mu

7.9. Mu modes

Mu can be started in one of a number of modes; modes make working with Mu easier by only presenting the options most relevant to what you are planning to use Mu for.

When Mu first starts, you will be presented with the **Select Mode** screen.



Please select the desired mode then click "OK". Otherwise, click "Cancel".

Adafruit CircuitPython
Use CircuitPython on Adafruit's line of boards.

BBC micro:bit
Write MicroPython for the BBC micro:bit.

Pygame Zero
Make games with Pygame Zero.

Python 3
Create code using standard Python 3.

Change mode at any time by clicking the "Mode" button containing Mu's logo.

OK Cancel

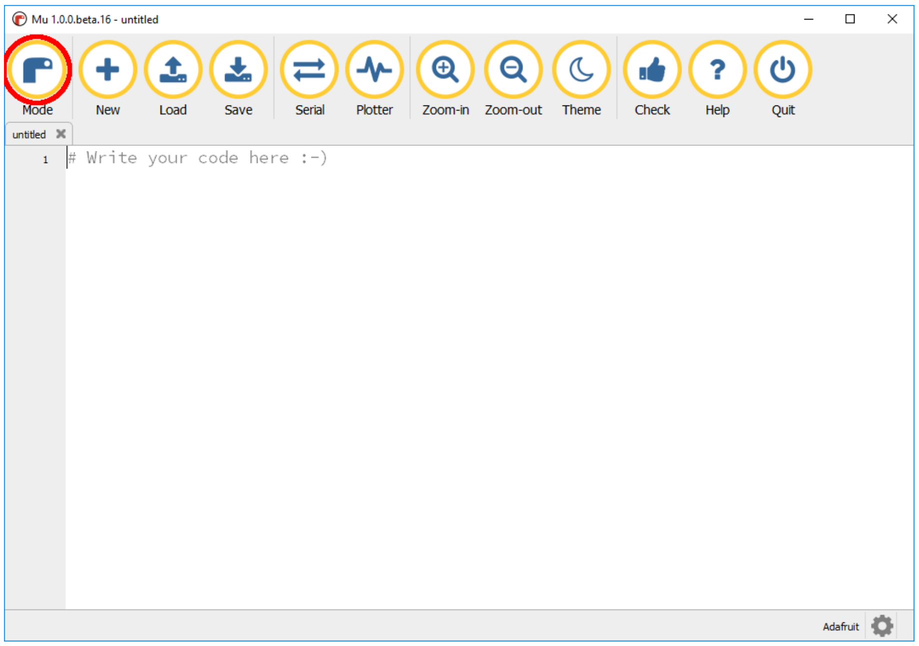
Select the **Python 3** mode and click **OK**.

This will set up Mu for programming in Python 3. If you would like to know more about the different Mu modes, have a look at codewith.mu.

7.9.1. Change mode

Mu will remember what mode you select, so you only have to select it once. If you want to change the mode later:

Click on the **Mode** icon in the top-left corder of the screen.



```
# Write your code here :-)
```

Select the mode you want from the menu, and click **OK**.

7.9.1.1. More projects

Try out some Python projects using Mu:

- Make a rock, paper, scissors game
- Write a Python program telling people all about you
- Create a turtle racing game
- Exchange secret messages with a friend

7.9.2. More Mu

Let’s try it out!

Copy and paste the following program into Mu. It is supposed to count down from 5 to 1.

```
print("count down")
for count in range(5, 1, -1):
    print(count)
```

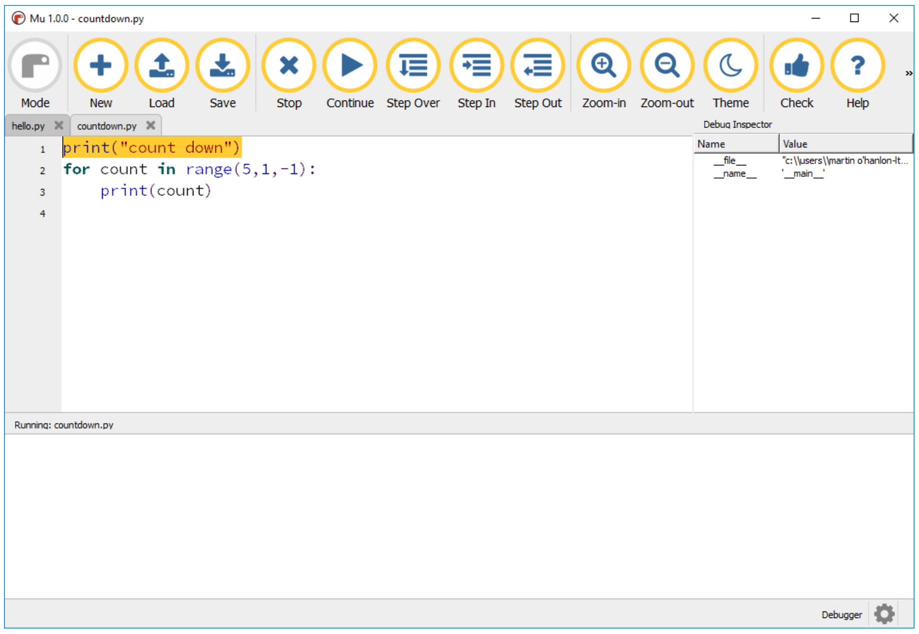
Now run it. You’ll see that it only counts down to 2:

```
count down
5
4
3
2
```

You can use Mu’s debugger to work out what is wrong.

Click the **Debug** button to start the debugger.

The debugger will start and the program will stop on the first line of code.



```
print("count down")
for count in range(5,1,-1):
    print(count)
```

Name	Value
__file__	"c:\\users\\martin o'hanlon-lt...
__name__	'__main__'

There are four buttons in the menu that allow you to control the debugger:

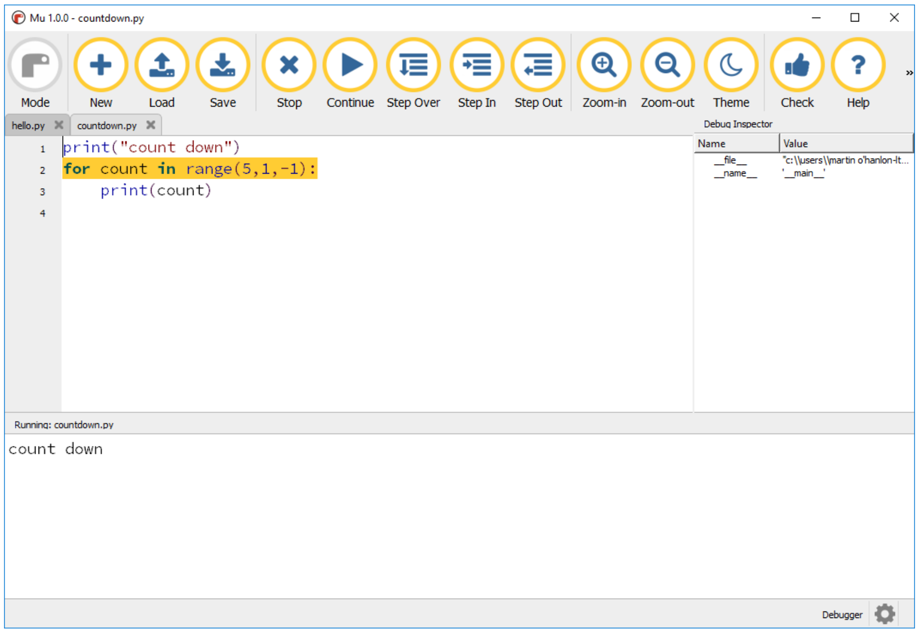
- **Continue**: starts your program again, and it will then run until it hits a break point or finishes.

- **Step Over**: runs the next line of code in your program
- **Step In**: if the next line of code is a function, it will ‘step into’ the function and run it
- **Step Out**: if the program is currently running a function, it will ‘step out’ of the function and return to the line of code that called the function.

There is also a Debug Inspector window on the right side of the code, showing the current value of any variables in use.

Click **Step Over** to run the first line of code.

The “count down” message will appear.



```
print("count down")
for count in range(5,1,-1):
    print(count)
```

Debug Inspector

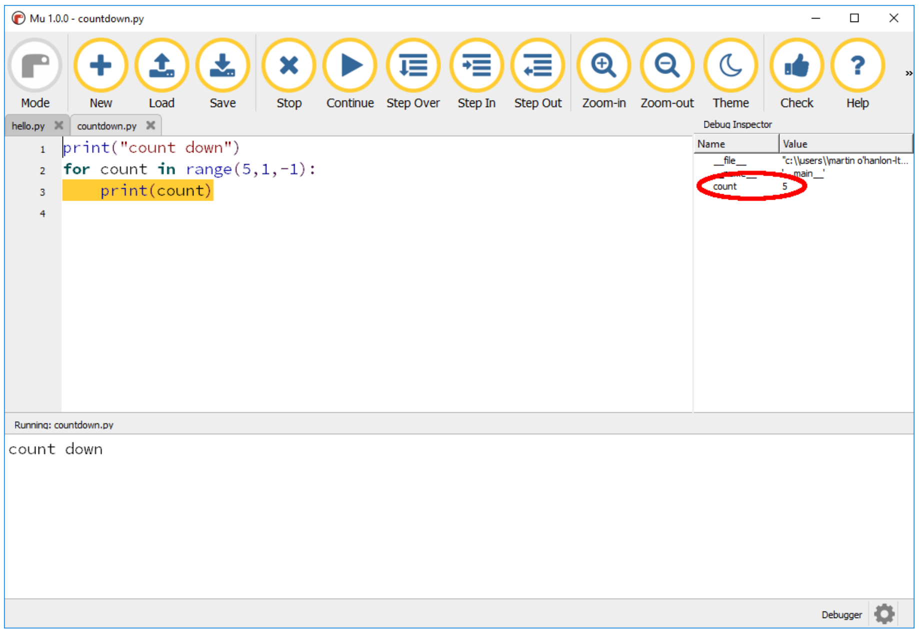
Name	Value
__file__	"c:\\users\\martin o'hanlon-lt...
__name__	'__main__'

Running: countdown.py

```
count down
```

Click **Step Over** again to run the next line of code.

The count variable will appear in the Debug Inspector.



```
print("count down")
for count in range(5,1,-1):
    print(count)
```

Name	Value
__file__	"c:\\users\\martin o'hanlon-lt...
__name__	'__main__'
count	5

```
count down
```

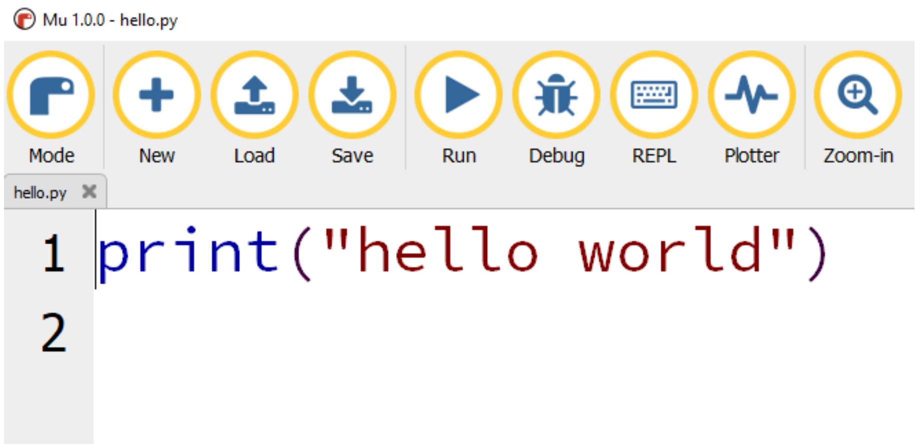
Keep clicking **Step Over** to run through each line in the program. You will see that `count` never reaches `1`. This is because the `for` loop in the program does not reach `0`. This is what the program should look like:

```
print("count down")
for count in range(5,0,-1):
    print(count)
```

7.9.3. Look and feel

7.9.3.1. Text size

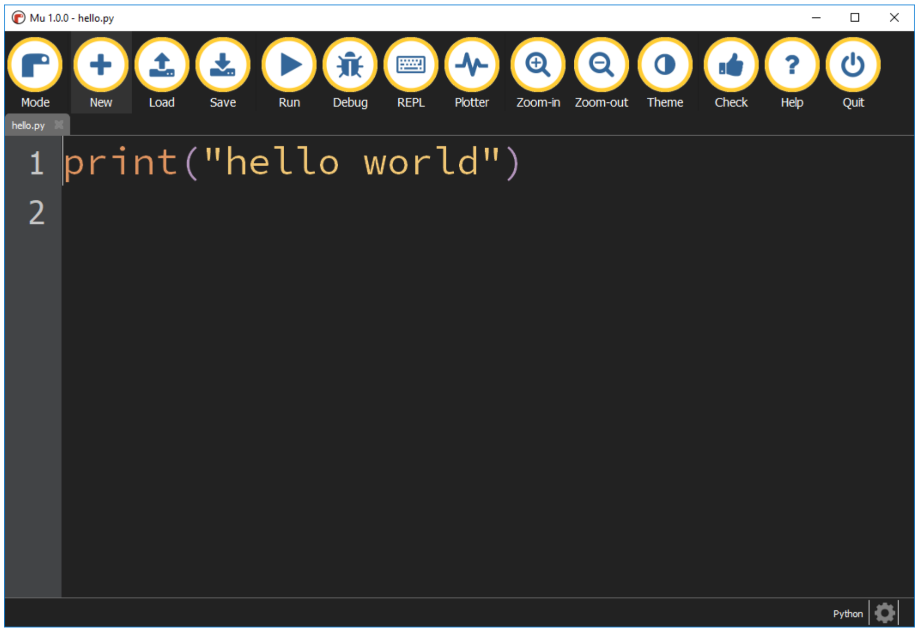
Using the **Zoom-in** and **Zoom-out** buttons, you can change the size of the code.



```
print("hello world")
```

7.9.3.1.1. Themes

Mu comes with a number of colour themes to suit different uses. Clicking the **Theme** button will cycle through the different themes.



```
print("hello world")
```

8. Extras for Module 2 (Not Presented)

8.1. Button Logic

8.1.1. Making a switch

With a switch, a single press and release on the button would turn the LED on, and another press and release would turn it off again.

- Modify your code so that it looks like this:

  ```
  from gpiozero import LED, Button
  from time import sleep

  led = LED(17)
  button = Button(2)

  while True:
      button.wait for press()
      led.toggle()
      sleep(0.5)
  ```

  `led.toggle()` switches the state of the LED from on to off, or off to on. Since this happens in a loop the LED will turn on and off each time the button is pressed.

- It would be great if you could make the LED switch on only when the button is being held down. With GPIO Zero, that's easy. There are two methods of the `Button` class called `when_pressed` and `when_released`. These don't block the flow of the program, so if they are placed in a loop, the program will continue to cycle indefinitely.

- Modify your code to look like this:

```
from gpiozero import LED, Button
from signal import pause

led = LED(17)
button = Button(2)

button.when pressed = led.on
button.when released = led.off

pause()
```

- Save and run the program. Now when the button is pressed, the LED will light up. It will turn off again when the button is released.
- Enter the following code:

```
from gpiozero import Button

button = Button(21)

while True:
    print(button.is pressed)
```

In GPIO Zero, you create an object for each component used. Each component interface must be imported from the `gpiozero` module, and an instance created on the GPIO pin number to which it is connected.

- Save and run the code.
- In the shell it will be constantly printing `False`. When you press the button this will switch to `True`, and when you let go it will return to `False`.

`button.is_pressed` is a property of the `button` object, which provides the state of the button (pressed or not) at any given time.

- Now return to the code window and modify your `while` loop to show the following:

```
while True:
    if button.is pressed:
        print("Hello")
    else:
        print("Goodbye")
```

- Run the code again and you'll see "Hello" printed when the button is pressed, and "Goodbye" when the button is not pressed.
- Modify the loop again:

```
while True:
    button.wait for press()
    print("Pressed")
    button.wait for release()
    print("Released")
```

- When you run the code this time, nothing will happen until you press the button, when you'll see "Pressed", then when you let go you'll see "Released". This will occur each time the button is pressed, but rather than continuously printing one or the other, it only does it once per press.

8.1.1.1. Add an LED

Now you'll add an LED into the code and use GPIO Zero to allow the button to determine when the LED is lit.

- In your code, add to the `` `from gpiozero import... `` line at the top to also bring in `LED`:

```
from gpiozero import Button, LED
```

- Add a line below `` `button = Button(21)\``` to create an instance of anLED` `` object:

```
led = LED(25)
```

- Now modify your `while` loop to turn the LED on when the button is pressed:

```
while True:
    button.wait for press()
    led.on()
    button.wait for release()
    led.off()
```

- Run your code and the LED will come on when you press the button. Hold the button down to keep the LED lit.

- Now swap the `on` and `off` lines to reverse the logic:

  ```
  while True:
      led.on()
      button.wait for press()
      led.off()
      button.wait for release()
  ```

- Run the code and you’ll see the LED stays on until the button is pressed.
- Now replace `led.on()` with `led.blink()`:

  ```
  while True:
      led.blink()
      button.wait for press()
      led.off()
      button.wait for release()
  ```

- Run the code and you’ll see the LED blink on and off until the button is pressed, at which point it will turn off completely. When the button is released, it will start blinking again.

8.1.1.2. Some additional Button class properties

- held_time
  - The length of time (in seconds) that the device has been held for. This is counted from the first execution of the when_held event rather than when the device activated, in contrast to active_time. If the device is not currently held, this is None.
- hold_repeat
  - If True, when_held will be executed repeatedly with hold_time seconds between each invocation.
- hold_time
  - The length of time (in seconds) to wait after the device is activated, until executing the when_held handler. If hold_repeat is True, this is also the length of time between invocations of when_held.
- is_held
  - When True, the device has been active for at least hold_time seconds.

- is_pressed
    - Returns True if the device is currently active and False otherwise. This property is usually derived from value. Unlike value, this is always a boolean.
- value
    - Returns 1 if the button is currently pressed, and 0 if it is not.
- when_held
    - The function to run when the device has remained active for hold_time seconds.
    - This can be set to a function which accepts no (mandatory) parameters, or a Python function which accepts a single mandatory parameter (with as many optional parameters as you like). If the function accepts a single mandatory parameter, the device that activated will be passed as that parameter.
    - Set this property to None (the default) to disable the event.
- when_pressed
    - The function to run when the device changes state from inactive to active.
    - This can be set to a function which accepts no (mandatory) parameters, or a Python function which accepts a single mandatory parameter (with as many optional parameters as you like). If the function accepts a single mandatory parameter, the device that activated will be passed as that parameter.
    - Set this property to None (the default) to disable the event.
- when_released
    - The function to run when the device changes state from active to inactive.
    - This can be set to a function which accepts no (mandatory) parameters, or a Python function which accepts a single mandatory parameter (with as many optional parameters as you like). If the function accepts a single mandatory parameter, the device that deactivated will be passed as that parameter.
    - Set this property to None (the default) to disable the event

8.2. More Python Control

8.2.1. Challenge: Stopping loop on button press

Modify the LED blinking program from Module 1 to stop when a button is pressed.

8.2.1.1. Available conditional operators

Here is a list of conditions you can use with `while` (or other conditional statements that we will learn about later).

Operator	Name
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to
not	Boolean NOT
and	Boolean AND
or	Boolean OR

8.2.1.2. Ordering and Grouping conditions

When you have multiple conditions (e.g. `pressed != True` AND `counter < 8` with a separate `counter > 5`), you may need to ensure they are grouped correctly. To do that you use parentheses `()`.

For example:

```
from gpiozero import Button
from time import sleep

pressed = False

counter = 0
button = Button(2)

while ( (button.is pressed != True) and (counter < 8) ) or (counter
    print("hello")
    counter += 1
    sleep(1)
```

In this case `hello` is printed at least five times, or up to eight if the button is not pressed before the eight time have occurred.

8.2.1.3. For Loops

The purpose of a **for loop** is to repeat code a specific number of times.

In Python the `range` function returns a list of numbers which can be used in a `for` loop.

```
for i in range(5):
    print(i)
```

The output is:

```
0
1
2
3
4
```

Note that `range(5)` returns a list of numbers starting from `0` and stopping at `4` (`1` less than `5`). Sometimes it’s useful to start counting from `1` instead of `0`.

```
for i in range(1, 6):
    print(i)
```

The output is:

```
1
2
3
4
5
```

Note that the second input to `range` needs to be `6` to keep counting up to `5`.

8.2.1.4. Lists

A Python list is a type of data structure. It can hold collections of any data type, and even a mixture of data types. Here is an empty list:

```
an empty list = []
```

You can create a list by giving it a name and adding the items of data inside square brackets:

```
compass = ["north", "south", "east", "west"]
```

Once a list has been created, you can add more data to the list using `append`:

```
numbers = [5, 10, 15, 20]
numbers.append(25)
print(numbers)

[5, 10, 15, 20, 25]
```

You can get rid of a piece of data from the list using `remove`:

```
numbers.remove(5)
print(numbers)

[10, 15, 20, 25]
```

8.2.1.5. For With Lists

This **for loop** which will print each item in the `animals` list.

```
animals = ["fox", "wolf", "panda", "squirrel"]

for animal in animals:
  print(animal)
```

The output is:

```
fox
wolf
panda
squirrel
```

Notice that the `print` line of code is slightly further to the right. This is called **indentation** - the line is **indented** to show that it is inside the loop. Any lines of code inside the loop will be repeated.

9. Extras for Module 3 (Not Presented)

9.1. Headless Mu

- A brief discussion of Mu
    - Tabs vs. Spaces (the indent problem from last module)
    - Load, Run, Stop, Save, and Check

9.1.1. Autocompletion in Mu

You’ve probably noticed by now that Mu offers you suggestions of what you might be trying to type. If Mu is offering you an choice that is correct, use the cursor (arrow) keys to select that option from the list, and then press Tab.

It is important that you do **not** press Tab unless the suggest text is correct. If you press Tab on the wrong text, you will need to use the backspace key to erase it.

Try using autocomplete with the code we work on below.

9.1.2. Loop forever in Python

```
while True:
    print("I will run forever.")
```

If you have trouble creating this program, in Mu, click ‘Load’ and load the following file from the list of files that comes up:

```
modules03-03-while-forever.py
```

Click Run, and watch what happens until you click ‘Stop’.

You may recall seeing something like this in the first module with the LED.

As you’ve probably guessed, this tells Python to print “I will run forever” for as long as the program runs (which is forever unless you stop it manually).

This takes two things being true:

1. The use of `while True:`
2. The *indented* `print("I will run forever.")` — indent means that there are spaces in front of the code.

All Python `while` statements start with `while`, have some condition (which is test that when it is `False`, will stop the loop — since `True` is never `False` this loop runs forever), and with a colon (:).

Any group of code immediately after a `while` statement that is indented the same amount will run as as long as the `while` condition is `True`.

In this case we have a single line after the `while` statement, so it is the only thing that is repeated.

Try adding other print statements after the `while`.

9.1.3. While loops (counter)

Sometimes while loops are written to repeat a specified number of times:

```
counter = 0
while counter < 10:
    print("Counting...")
    counter += 1
```

In this example, the condition is `counter < 10`. Since we add `1` to `counter` each time the loop runs, the loop will run 10 times.

9.2. Simple Control and Loops in Python

9.2.1. Variables

Variables ‘remember’ the value of something, and can be changed by the program. (Aside: there is a specific type of variable that cannot be changed, which are know as constants or constant variables).

For example in the following code we remember `hello world` and print the variable, which prints what we ‘remembered’.

```
hello = 'hello world'
print(hello)
```

9.2.2. Functions

When coding, sometimes you may want to use the same few lines of code multiple times within your script. Alternatively, you may want to have the same few lines of code run every time a certain event occurs, e.g. when a specific key is pressed, or a particular phrase is typed. For tasks like this, you might want to consider using a **function**.

Functions are **named** blocks of code that perform a defined task. Just about the simplest function you can create in Python looks like this:

```
def hello():
    print('Hello World!')
```

You tell Python that you’re creating a new function by using the `def` keyword, followed by the name of the function. In this case it is called `hello`. The parentheses after the function name are important.

The colon at the end of the line indicates that the code inside the function will be indented on the next line, just like in a `for` or `while` loop or an `if/elif/else` conditional.

You can **call** a function by typing its name with the parentheses included. So to run the example function, you would type `hello()`.

*[call]: run the lines of code within the function

Here is the complete program:

```
def hello():
    print('Hello World!')

hello()
```

9.2.3. While loops (boolean)

You may recall the while loop from Module 1 which blinked the LED forever. Here is a more complete explanation:

The purpose of a **while** loop is to repeat code over and over while a condition is `True`. This is why while loops are sometimes referred to as **condition-controlled** loops.

In this example, the condition is a boolean variable (which is a variable that is either `True` or `` `False ``) we gave the name `keep_looping`. Its value is set as `True` at the top. The value can become `False`, which will make the condition of the while loop `False`. When this happens, the loop stops running.

```
keep looping = True

while keep looping:
    print("I am in a loop")
    command = input("Shall I keep looping?")
    if command == "no":
        keep looping = False
```

This kind of loop is useful in situations where you want to repeat code until a specific event happens. For example, you may want a program to continue running until someone types something to tell it to quit.

9.3. More Control and Loops in Python

9.3.1. If statements

Another important control structure in python is the `if` statement. This does something if some condition is `True`. In real life you might say, “`If` I’m late for school, `then` run `else` walk”.

In python this would be expressed as:

```
if late for school:
  run()
else:
  walk()
```

From our LED, buttons, and buzzers, we might say, if button is pressed, then play a sound on the buzzer, else blink LED. That would be written as:

```
from gpiozero import LED, Button, TonalBuzzer
from gpiozero.tones import Tone
from time import sleep

led = LED(17)
buzzer = TonalBuzzer(27)
button = Button(4)

while True:
    if button.is pressed:
        buzzer.play("A4")
        sleep(1)
        buzzer.stop()
    else:
        led.blink()
    sleep(1)
```

Note the indentation. In python the indentation is part of the syntax, and is required.

9.4. Making traffic lights

For this worksheet you'll need a breadboard, three LEDs, a button, and the necessary jumper cables and resistors.

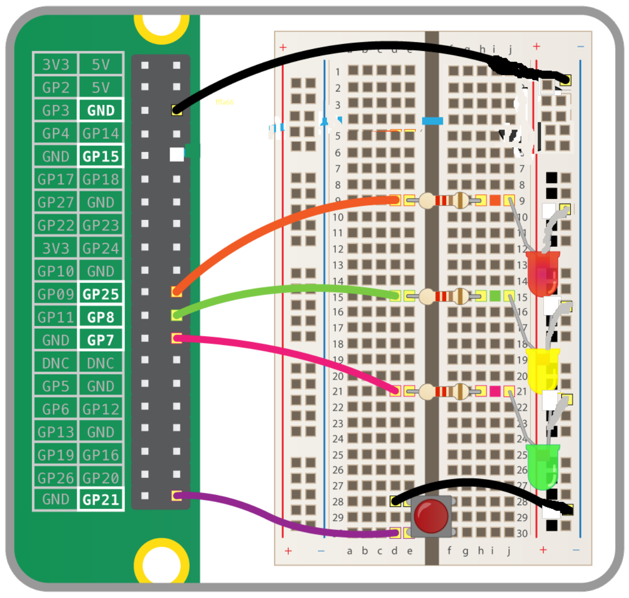
9.4.1. Wiring

To get started, you'll need to place all the components on the breadboard and connect them to the appropriate GPIO pins on the Raspberry Pi.

- First, you need to understand how each component is connected:
    - A push button requires 1 ground pin and 1 GPIO pin
    - An LED requires 1 ground pin and 1 GPIO pin, with a current limiting resistor

  Each component requires its own individual GPIO pin, but components can share a ground pin. We will use the breadboard to enable this.

- Place the components on the breadboard and connect them to the Raspberry Pi GPIO pins, according to the following diagram:



  Note that the row along the long side of the breadboard is connected to a ground pin on the Raspberry Pi, so all the components in that row (which is used as a ground rail) are hence connected to ground.

- Observe the following table, showing which GPIO pin each component is connected to:

Component	GPIO pin
Button	21
Red LED	25
Amber LED	8
Green LED	7

9.4.2. Dive into Python

- Create a new file by clicking **New**.

- Save the new file straight away by clicking **Save**; name the file `trafficlights.py`.

9.4.3. Traffic lights

You have three LEDs: red, amber, and green. Perfect for traffic lights! There's even a built-in interface for traffic lights in GPIO Zero.

- Amend the `from gpiozero import...` line to replace `LED` with `TrafficLights`:

  ```
  from gpiozero import Button, TrafficLights
  ```

- Replace your `led = LED(25)` line with the following:

  ```
  lights = TrafficLights(25, 8, 7)
  ```

  The `TrafficLights` interface takes three GPIO pin numbers, one for each pin: red, amber, and green (in that order).

- Now amend your `while` loop to control the `TrafficLights` object:

  ```
  while True:
      button.wait for press()
      lights.on()
      button.wait for release()
      lights.off()
  ```

  The `TrafficLights` interface is very similar to that of an individual LED: you can use `on`, `off`, and `blink`, all of which control all three lights at once.

- Try the `blink` example:

  ```
  while True:
      lights.blink()
      button.wait for press()
      lights.off()
      button.wait for release()
  ```

9.4.4. Extra: Add a buzzer

A buzzer requires 1 ground pin and 1 GPIO pin

The table above will change to:

Component	GPIO pin
Button	21
Red LED	25
Amber LED	8
Green LED	7
Buzzer	15

Now you'll add your buzzer to make some noise.

- Add `TonalBuzzer` to the `from gpiozero import...` line:

```
from gpiozero import Button, TrafficLights, TonalBuzzer
from gpiozero.tones import Tone
```

- Add a line below your creation of `button` and `lights` to add a `TonalBuzzer` object:

```
buzzer = TonalBuzzer(15)
```

- Try adding a `buzzer.play()` and `buzzer.stop()` into your loop:

```
while True:
    lights.on()
    buzzer.stop()
    button.wait for press()
    lights.off()
    buzzer.play(Tone(200.0))
    button.wait for release()
```

9.5. Traffic Lights Revisited

9.5.1. Traffic lights sequence

As well as controlling the whole set of lights together, you can also control each LED individually. With traffic light LEDs, a button and a buzzer, you can create your own traffic lights sequence, complete with pedestrian crossing!

- At the top of your file, below `from gpiozero import...`, add a line to import the `sleep` function:

```
from time import sleep
```

- Modify your loop to perform an automated sequence of LEDs being lit:

```
while True:
    lights.green.on()
    sleep(1)
    lights.amber.on()
    sleep(1)
    lights.red.on()
    sleep(1)
    lights.off()
```

- Add a `wait_for_press` so that pressing the button initiates the sequence:

```
while True:
    button.wait for press()
    lights.green.on()
    sleep(1)
    lights.amber.on()
    sleep(1)
    lights.red.on()
    sleep(1)
    lights.off()
```

Try some more sequences of your own.

- Now try creating the full traffic lights sequence:

    - Green on
    - Amber on
    - Red on
    - Red and amber on
    - Green on

  Be sure to turn the correct lights on and off at the right time, and make sure you use `sleep` to time the sequence perfectly.

- Try adding the button for a pedestrian crossing. The button should move the lights to red (not immediately), and give the pedestrians time to cross before moving back to green until the button is pressed again.

- Now try adding a buzzer to beep quickly to indicate that it is safe to cross, for the benefit of visually impaired pedestrians.

9.5.2. Design Exercise: ‘Real-world’ Traffic Lights

- As a group: talk about what real traffic lights do.

    - What inputs do they accept (e.g. buttons, in some cases cars at the light, time of day, etc)
    - What happens with the lights?
        - Without inputs
        - When one our more inputs are received

- Design the logic for a typical traffic light with four buttons for pedestrians, an signal for visually impaired pedestrians, and that’s.

- Write the code and try it out.

---

10. Extras for Module 5 (Not Presented)

10.1. Headless Pi

10.1.1. Why We're Doing This

1. So that you can program the robo-cars — it's rather difficult to add a monitor, keyboard, and mouse to a Pi when it's mounted on a robo-car chassis.
2. So you can stop the robo-car program if it's getting into trouble — you don't want the robo-car to keep going if all it's going to do is keep crashing into things and get damaged.
3. So you can start the robo-car on demand.

10.1.2. What We're Doing

- Accessing the Pi from a laptop as if we were on the Pi itself

10.1.2.1. The Steps You Need to Know

Don't worry — we're going to show you how to do this.

- Find the address of your Pi (we'll explain how to do that)
- Start a special program on the laptop
- It will ask for the Pi's address
- You give the program the Pi's address and click 'Connect'
- After a bit of time you will see the same desktop you've been using on the Pi all along.
- You'll notice you're on the laptop instead of a directly attached keyboard and mouse.

10.1.2.2. Advanced (Extra) Explanation

10.1.2.2.1. Overview

- We'll be using a protocol called VNC (Virtual Network Computing).
- It that allows a stationary computer (laptop) to access the desktop of a remote computer (the Pi on the robo-car).
- We'll use the remote desktop to use Mu to create, run, and stop a program to control the robo-car.
- There a program already loaded onto the Pi which you will be able to modify to put the robo-car through it's paces.

10.1.2.2.2. How This Works

- Run a server on the Pi that ‘listens’ for remote connections — this is already done and configured for you.
- Configure the Pi so that when the Pi is booted (started up) that it registers with a server we know how to access (this is already done too).
- The Pi will give this server the Pi’s ‘address’ (called an IP address) that is how we can find the Pi on the network.
- We will query (ask) the server for the Pi’s address.
- Once we know the address we will use a program on the laptop you’re using to connect to the Pi.
- This program is known a VNC ‘client’.
- The VNC client will connect to the VNC server and show you a desktop that is running on the Pi.
- From there you’ll use this ‘remote desktop’ just as if you were using a keyboard, mouse, and monitor attached to the Pi.

10.1.3. The Main Action

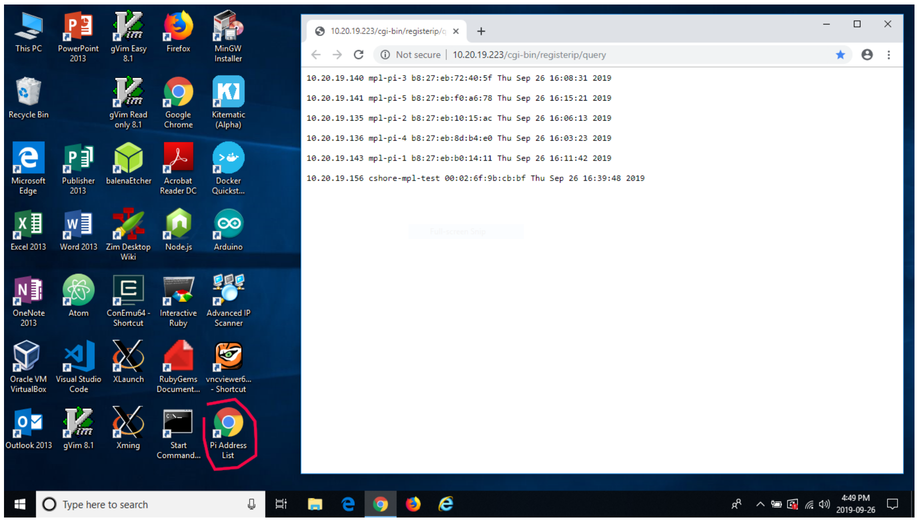
If you haven’t already done so, boot the laptops.

Make sure they are connected to the ‘MakerPlace’ Wireless Network

---

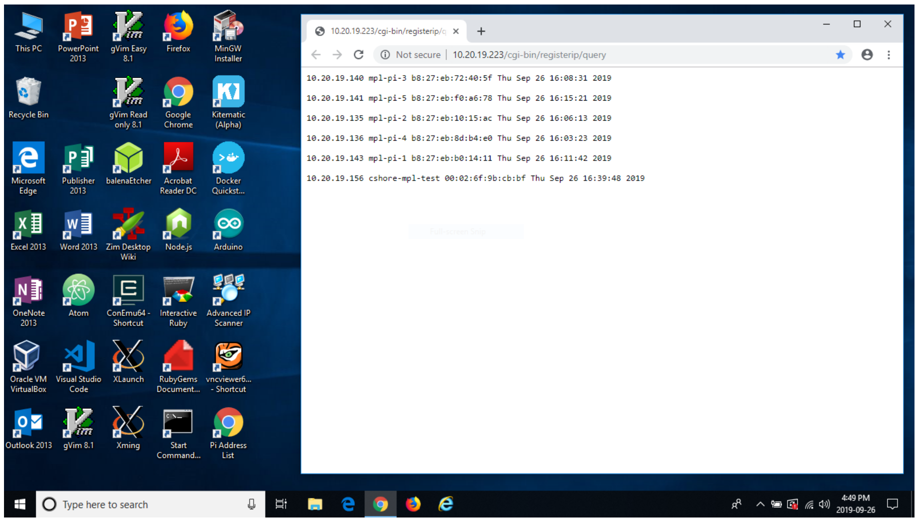
If you haven’t already done so, boot the Pi by turning on the power switch.

On your laptop, double-click on the icon labelled ‘Pi Address List’



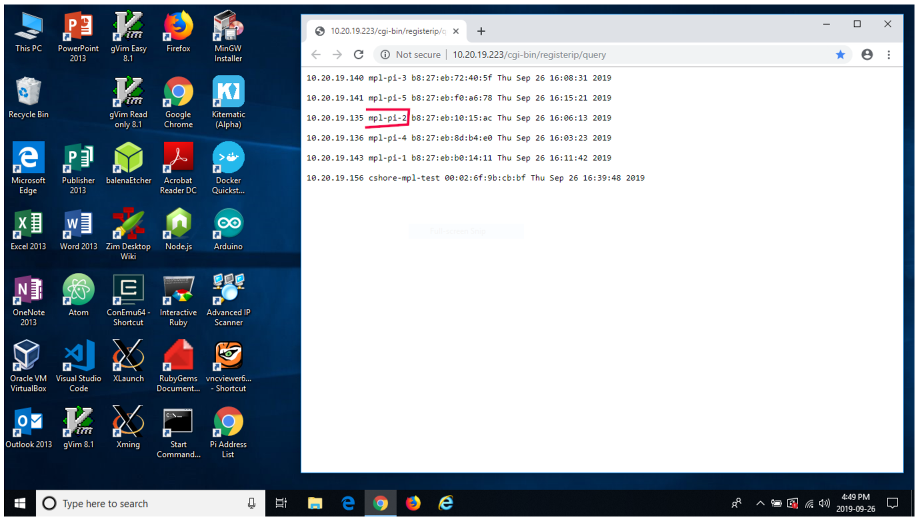
```
10.20.19.140 mpl-pi-3 b8:27:eb:72:40:5f Thu Sep 26 16:08:31 2019
10.20.19.141 mpl-pi-5 b8:27:eb:f0:a6:78 Thu Sep 26 16:15:21 2019
10.20.19.135 mpl-pi-2 b8:27:eb:10:15:ac Thu Sep 26 16:06:13 2019
10.20.19.136 mpl-pi-4 b8:27:eb:8d:b4:e0 Thu Sep 26 16:03:23 2019
10.20.19.143 mpl-pi-1 b8:27:eb:b0:14:11 Thu Sep 26 16:11:42 2019
10.20.19.156 cshore-mpl-test 00:02:6f:9b:cb:bf Thu Sep 26 16:39:48 2019
```

Your browser should shown a list of information.



```
10.20.19.140 mpl-pi-3 b8:27:eb:72:40:5f Thu Sep 26 16:08:31 2019
10.20.19.141 mpl-pi-5 b8:27:eb:f0:a6:78 Thu Sep 26 16:15:21 2019
10.20.19.135 mpl-pi-2 b8:27:eb:10:15:ac Thu Sep 26 16:06:13 2019
10.20.19.136 mpl-pi-4 b8:27:eb:8d:b4:e0 Thu Sep 26 16:03:23 2019
10.20.19.143 mpl-pi-1 b8:27:eb:b0:14:11 Thu Sep 26 16:11:42 2019
10.20.19.156 cshore-mpl-test 00:02:6f:9b:cb:bf Thu Sep 26 16:39:48 2019
```

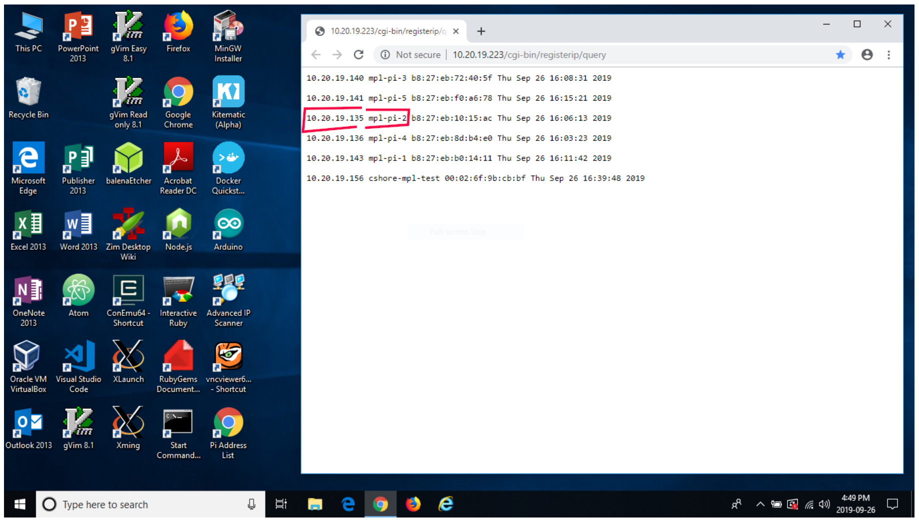
Look for the name of your Pi in the list (for example, if you have Pi #2, the name would be mpl-pi-2).



10.20.19.223/cgi-bin/registerip/query

```
10.20.19.140 mpl-pi-3 b8:27:eb:72:40:5f Thu Sep 26 16:08:31 2019

10.20.19.141 mpl-pi-5 b8:27:eb:f0:a6:78 Thu Sep 26 16:15:21 2019

10.20.19.135 mpl-pi-2 b8:27:eb:10:15:ac Thu Sep 26 16:06:13 2019

10.20.19.136 mpl-pi-4 b8:27:eb:8d:b4:e0 Thu Sep 26 16:03:23 2019

10.20.19.143 mpl-pi-1 b8:27:eb:b0:14:11 Thu Sep 26 16:11:42 2019

10.20.19.156 cshore-mpl-test 00:02:6f:9b:cb:bf Thu Sep 26 16:39:48 2019
```

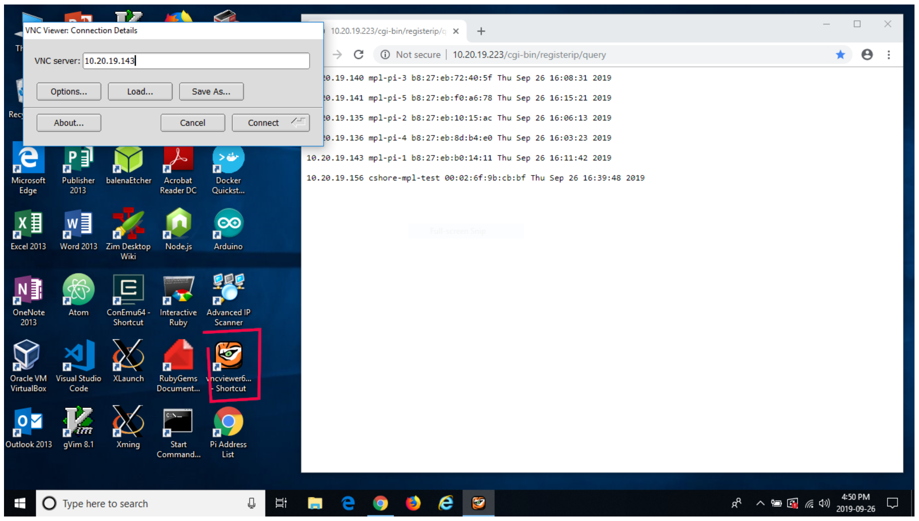
In the same row as the name of your Pi, find the 'address' for your Pi, which begins with `10.20.19.`



10.20.19.223/cgi-bin/registerip/query

```
10.20.19.140 mpl-pi-3 b8:27:eb:72:40:5f Thu Sep 26 16:08:31 2019

10.20.19.141 mpl-pi-5 b8:27:eb:f0:a6:78 Thu Sep 26 16:15:21 2019

10.20.19.135 mpl-pi-2 b8:27:eb:10:15:ac Thu Sep 26 16:06:13 2019

10.20.19.136 mpl-pi-4 b8:27:eb:8d:b4:e0 Thu Sep 26 16:03:23 2019

10.20.19.143 mpl-pi-1 b8:27:eb:b0:14:11 Thu Sep 26 16:11:42 2019

10.20.19.156 cshore-mpl-test 00:02:6f:9b:cb:bf Thu Sep 26 16:39:48 2019
```

On your desktop double-click on the icon that is labelled 'vncviewer' and has eye of tiger.



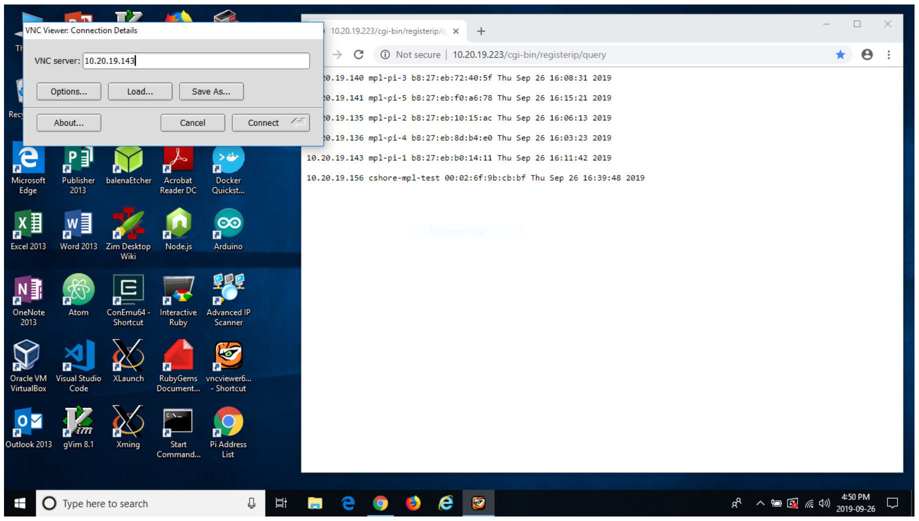
VNC Viewer: Connection Details

VNC server: 10.20.19.143

10.20.19.223/cgi-bin/registerip/query

```
0.19.140 mpl-pi-3 b8:27:eb:72:40:5f Thu Sep 26 16:08:31 2019

0.19.141 mpl-pi-5 b8:27:eb:f0:a6:78 Thu Sep 26 16:15:21 2019

0.19.135 mpl-pi-2 b8:27:eb:10:15:ac Thu Sep 26 16:06:13 2019

0.19.136 mpl-pi-4 b8:27:eb:8d:b4:e0 Thu Sep 26 16:03:23 2019

10.20.19.143 mpl-pi-1 b8:27:eb:b0:14:11 Thu Sep 26 16:11:42 2019

10.20.19.156 cshore-mpl-test 00:02:6f:9b:cb:bf Thu Sep 26 16:39:48 2019
```

In the address field enter the address of your Pi



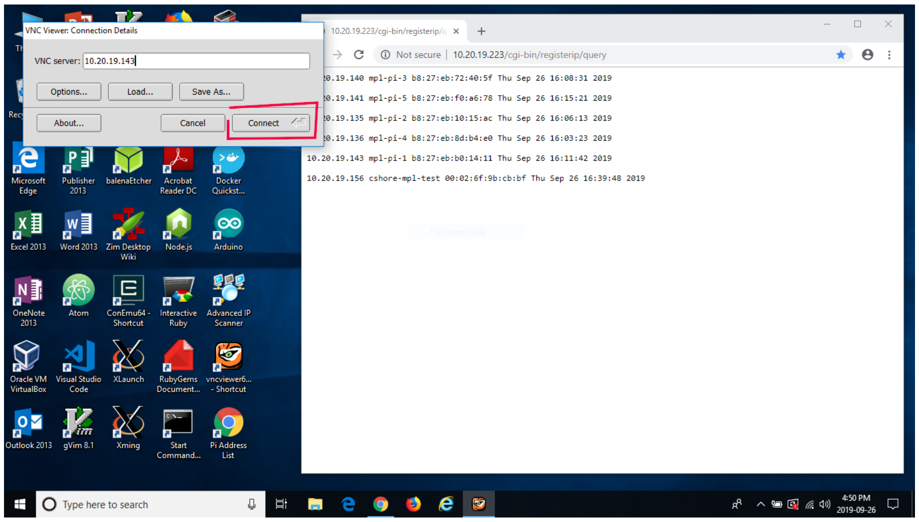
VNC Viewer: Connection Details

VNC server: 10.20.19.143

10.20.19.223/cgi-bin/registerip/query

```
0.19.140 mpl-pi-3 b8:27:eb:72:40:5f Thu Sep 26 16:08:31 2019

0.19.141 mpl-pi-5 b8:27:eb:f0:a6:78 Thu Sep 26 16:15:21 2019

0.19.135 mpl-pi-2 b8:27:eb:10:15:ac Thu Sep 26 16:06:13 2019

0.19.136 mpl-pi-4 b8:27:eb:8d:b4:e0 Thu Sep 26 16:03:23 2019

10.20.19.143 mpl-pi-1 b8:27:eb:b0:14:11 Thu Sep 26 16:11:42 2019

10.20.19.156 cshore-mpl-test 00:02:6f:9b:cb:bf Thu Sep 26 16:39:48 2019
```

Cick ‘Connect’.



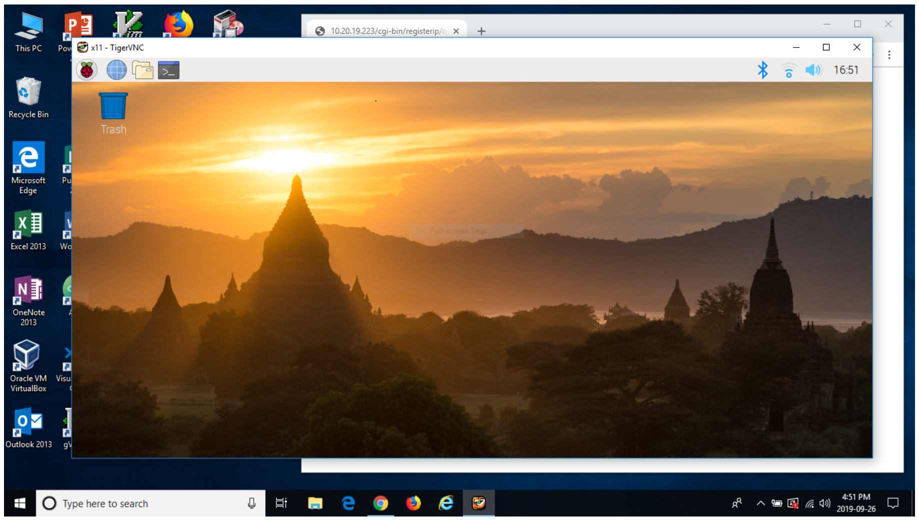
VNC server: 10.20.19.143

10.20.19.223/cgi-bin/registerip/query

```
0.19.140 mpl-pi-3 b8:27:eb:72:40:5f Thu Sep 26 16:08:31 2019
0.19.141 mpl-pi-5 b8:27:eb:f0:a6:78 Thu Sep 26 16:15:21 2019
0.19.135 mpl-pi-2 b8:27:eb:10:15:ac Thu Sep 26 16:06:13 2019
0.19.136 mpl-pi-4 b8:27:eb:8d:b4:e0 Thu Sep 26 16:03:23 2019
10.20.19.143 mpl-pi-1 b8:27:eb:b0:14:11 Thu Sep 26 16:11:42 2019
10.20.19.156 cshore-mpl-test 00:02:6f:9b:cb:bf Thu Sep 26 16:39:48 2019
```

NB Tiger VNC Viewer should have the necessary settings already configured for you so you can just click ‘Connect’ without changing anything in the options.

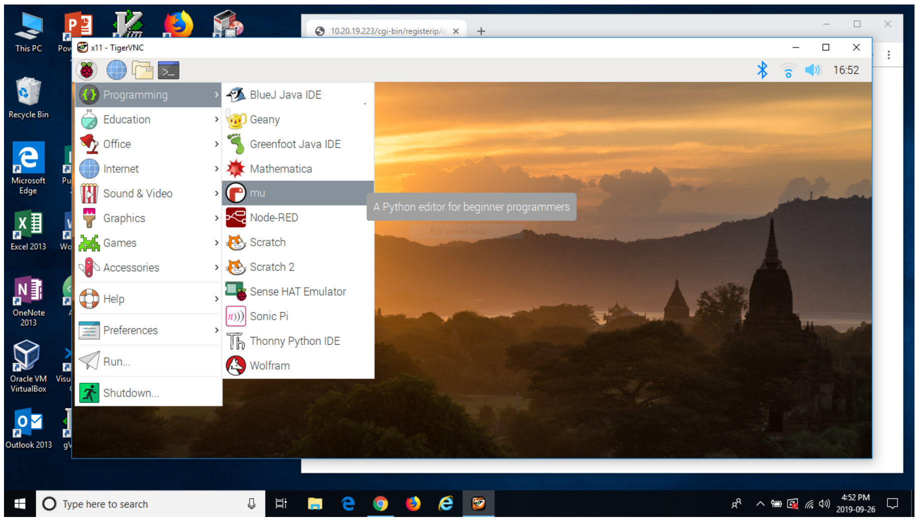
The dialog will disappear and there will be noticeable period of time before you will see your Pi’s desktop on the laptop screen.



You can now use the desktop as if you were connected with a keyboard, mouse, and monitor.

Note that patience is often required because using a network connection is slower than a direct physical connection to the Pi.

Next open Mu for the next section where we will test the robo-car build (aka smoke test).



11. Backmatter (Not Presented)

11.1. Bots & Bytes 2019: Additional Information

11.1.1. Original Introduction for Physical Computing

11.1.1.1. What you will do

Learn how to use the GPIO pins on your Raspberry Pi to interface with electronic components, such as LEDs and PIRs.

11.1.1.2. What you will learn

This resource covers elements from the following strands of the Raspberry Pi Digital Making Curriculum{:target="_blank"}:

- Use basic programming constructs to create simple programs
- Use basic digital, analogue, and electromechanical components

You can find the solutions for this project here

File Raspberry_block_function_v01.svg is licensed under the Creative Commons Attribution-Share-Alike 3.0 Unported License

11.1.2. Original Conclusion to Physical Computing (Pi Curriculum)

11.1.3. What next?

Try some of our other physical computing projects which use Python:

- Burping jelly baby
  - turn a jelly baby into an input device for your Raspberry Pi.
- GPIO music box
  - make a device thats plays music when you push its buttons.
- Quick Reaction Game
  - use electronics to create a quick reaction game which you will program using Python.
- Build a robot buggy
  - build a robot buggy that you can program to move around using simple Python commands.