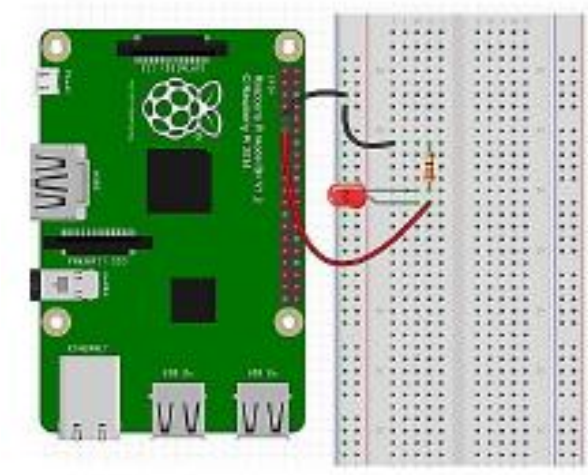
GETTING STARTED WITH GPIO

1. OPEN THE PYTHON IDE

2. BUILD THE FOLLOWING CIRCUIT TO TEST OUTPUT FUNCTIONS



Hints:

- Connect the positive lead of the LED to pin 11.
- Use a 1kΩ resistor
- All resistors in your kit are of the same value

3. Write a Program to Turn on the LED

Create a new Python file called "LED.py" and enter the following lines of code:

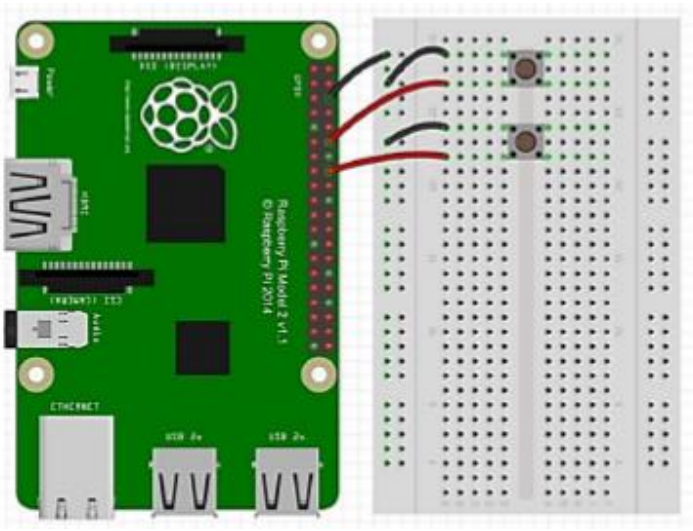
```
#This imports the necessary libraries to access the pins on the Pi
import RPi.GPIO as GPIO

#This tells Python and the Pi that we're using the physical pin numbering
scheme
GPIO.setmode(GPIO.BOARD)

#Alternatively, you can use the BCM numbering scheme with this line:
#GPIO.setmode(GPIO.BCM)
#(but don't…)

led = 11
#According to the physical pin numbering scheme, we want pin 11 to be the
output. If using the GPIO
#numbering scheme, you would use "17" to reference the same pin

#configure pin 11 as an output
GPIO.setup(led, GPIO.OUT)

#This sends a high output to pin 11 and turns the LED on
GPIO.output(led, True)
```

4. Write a Program to Blink the LED

You can modify your code using the **time library** to blink the LED as follows:

```
import RPi.GPIO as GPIO
import time #this allows us to introduce delays in our code

GPIO.setmode(GPIO.BOARD) #physical numbering scheme

led = 11 #connect LED to pin 11

GPIO.setup(led, GPIO.OUT) #set pin 11 as output

#get user input and store it as an integer in a variable
blinkNum = int(input('How many times do you want to blink? '))

for i in range(blinkNum):
      GPIO.output(led, True) #can also use 1 instead of TRUE to turn on
      time.sleep(1) #delay 1 second
      GPIO.output(led, False) #can also use 0 instead of FALSE to turn off
      time.sleep(1) #delay 1 second
GPIO.cleanup()
```

5. Build a circuit to test Input Functions



6. Read Values from Input Pins

When we attach a button to a circuit, we essentially have an open circuit until the button is pressed. When a pin is connected to nothing (like it is when connected to an open button), we call it a floating pin. This causes problems because floating pins are susceptible to interference from nearby devices—they will "float" past the threshold voltage for the pin and alternate between HI and LO. To avoid this, an internal pull up resistor ties the pin to HI (on the device). When ground is connected (button closed), the resistor is shorted out and the pin sees LO.

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BOARD)
start = 12 #read from pins 12 and 16
stop = 16

#Declare the pins as inputs and "attach" the pull_up resistors to them
GPIO.setup(start, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(stop, GPIO.IN, pull_up_down=GPIO.PUD_UP)

end = False
while (not end):
    if GPIO.input(start)==0:
        print('Start was pressed')
        sleep(0.5)
    if GPIO.input(stop)==0:
        print('Stop was pressed')
        sleep(0.5)
    if GPIO.input(start)==0 and GPIO.input(stop)==0:
        print('Make up your mind!')
        end = True

GPIO.cleanup() #clear out pin assignments
```

USING THE CAMERA MODULE

7. CONNECTING THE CAMERA MODULE

WARNING: The Raspberry Pi must be **shut off** before connecting the camera module. *Failing to do so will destroy both the RPi and the camera.* Ensure the power cable is disconnected (after safely shutting down the Pi).

a. Locate the camera port and connect the camera (highlighted in the image). Carefully raise the lock with your fingertips. Feel free to ask for help.

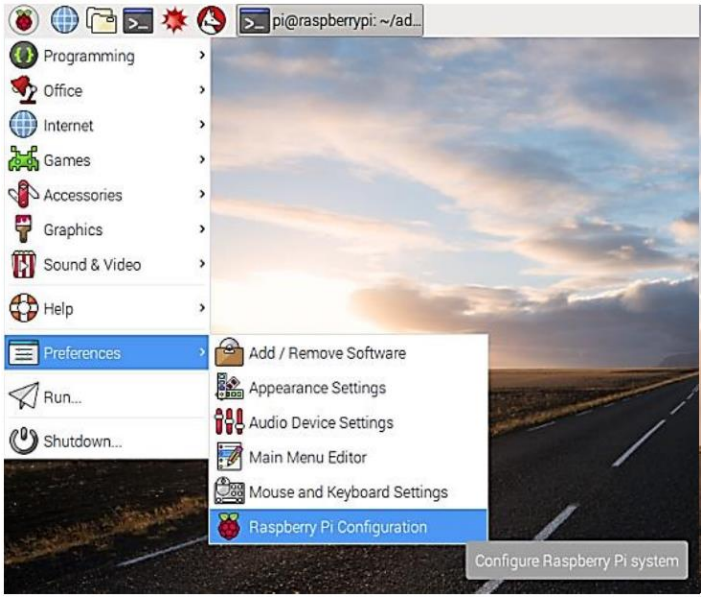
b. Feed the connections into the slot. Make sure the metal contacts on the cable line up with the contacts inside the slot. Lower the black lock back once the camera cable is in place.

2. CONFIGURING THE RASPBERRY PI

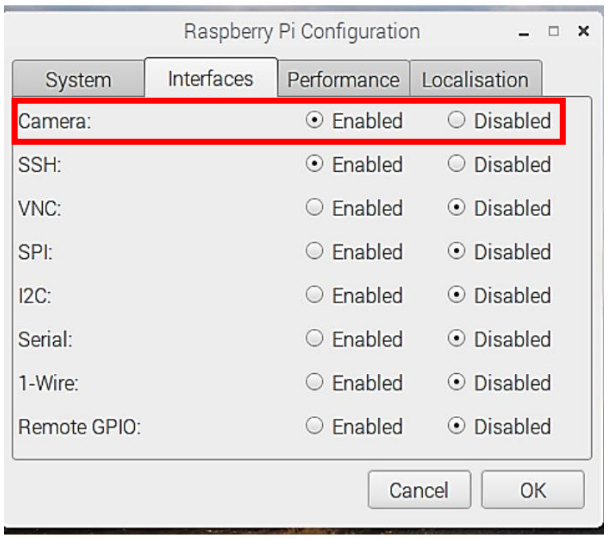
a. Power on the Raspberry Pi (by plugging in the power cable)

b. Open the **Raspberry Pi Configuration Tool** from the main menu:

Start > Preferences > Raspberry Pi Configuration



c. In the Interfaces tab, ensure the camera is enabled. Then, hit OK and restart the Pi.

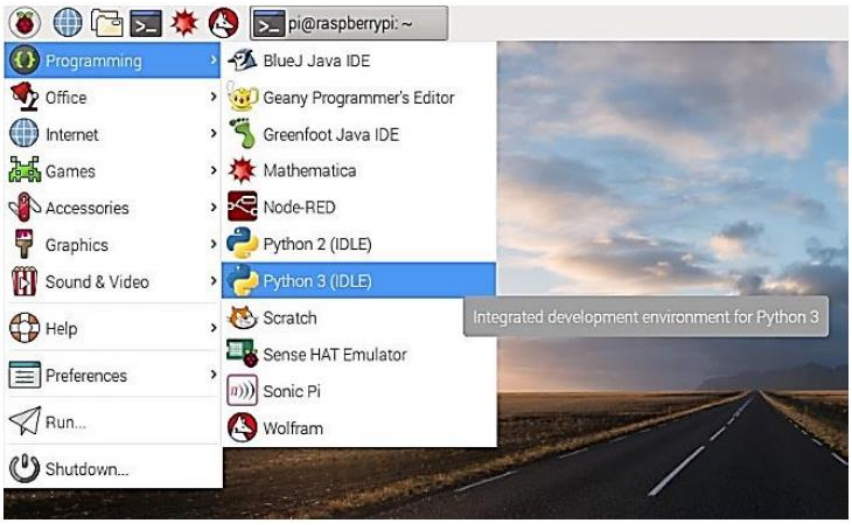


3. Displaying The Camera Preview

We can write a python program to display a preview of what the camera sees. To do this, we use the `PiCamera` library (already installed)

a. Open the IDLE IDE for **Python 3** in the main menu. Open a new file and save it.

Start > Programming > Python 3 (IDLE)



WARNING: Do NOT save it as `picamera.py`. (This will overwrite the library file.)

b. Enter the following code and execute it (by pressing F5):

```
from picamera import PiCamera          # for accessing the camera
from time import sleep                 # allow the program to wait

# Initialises the camera and loads the preview for 5 seconds
camera = PiCamera()                    # instantiate a PiCamera object
camera.start_preview()
sleep(5)                               # Camera will stay on for 5 seconds
camera.stop_preview()
```

c. Try modifying the code above by adding the following lines:

You can rotate the image by changing the rotation attribute of the camera object. This must be done before calling `camera.start_preview()`.

```
camera.rotation = 180                  # Rotates 180 degrees clockwise
```

You can change the transparency of the preview by setting the alpha level when calling `camera.start_preview()`:

```
camera.start_preview(alpha=200)        # 0=transparent, 255=opaque
```

4. Taking Still Pictures

We can use the capture function to save still images to a location of our choice.

a. Add the following line to your code to capture an image.

```
camera.capture('/home/pi/Desktop/image.jpg')
```

Important Notes:

- It is important to **let the camera sleep for at least 2 seconds before capturing**. This gives the sensor time to set its light levels.
- All camera.XXXX() function calls must be made between calls to `camera.start_preview()` and `camera.stop_preview()`
- When specifying a path, it must be enclosed in single quotes. Don't forget the first "/"
- The image file **silently** appears on the Desktop. Double-click it to view.
- ***If you get stuck in camera preview:*** (1) press ctrl + alt + t (2) type blindly "pkill python3"

b. Modify your code to take multiple pictures in a row:

```
#the top of your code stays the same as before…
camera.start_preview(alpha=200)
for i in range(5):
    sleep(5)
    camera.capture('/home/pi/Desktop/image%s.jpg' % i)
      # The "%s" is used to name each image differently using the "% i",
      # "%s" is replaced by the loop's index (0 to 4) in the file name.
camera.stop_preview()
```

5. Creating a GIF from Still Images (Optional)

We can create a gif out of these pictures by launching an application from the Terminal.

a. Launch the terminal (click the terminal icon on the taskbar)

b. Type the following two commands and hit Enter after each one:

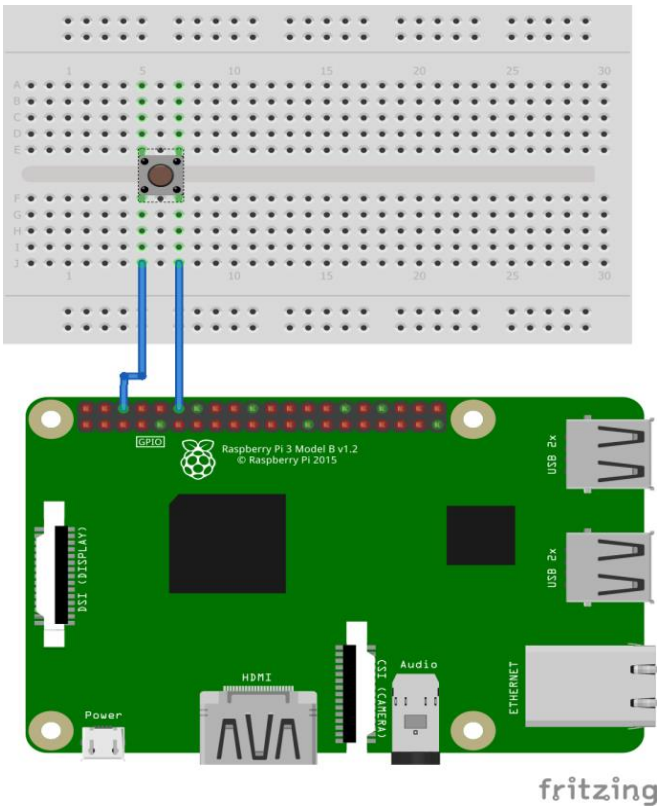
```
sudo apt-get install imagemagick
cd Desktop
convert -delay 10 -loop 0 image*.jpg animate.gif
```

- The first line installs the required software (already done)
- `-loop 0` means the gif loops forever.
- `-delay 10` makes the delay one 10th of a second.
- Saves to `animate.gif` on the Desktop. Be patient, as this will take some time.

c. View the GIF by opening the file on the desktop once this is complete.

6. MINI-PROJECT: DIY CAMERA

In a new Python file, write a program which lets the user view the camera preview and capture the scene by pushing a button. The program should exit once the photo is taken. Use the circuit diagram and code below to get started. You can also draw on code from previous sections.




```
#The following code defines GPIO pins for the button…
from picamera import PiCamera
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)
capturePin = 12
GPIO.setup(capturePin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

Hints:

- Use `GPIO.input(pinNumber)` to check the value at the pin. (`True = 1`, `False = 0`)
- When the buttons are pushed down, the Pi will receive a `0` at the corresponding pin. (Since the buttons will connect the pin to ground)
- Use a `while` loop and an `if` statement to repeatedly check for input from the button.

7. Recording Video

Recording video is similar to taking a still, but we replace `capture` with `start_recording` and `stop_recording`. Try the following code which records for 10 seconds:

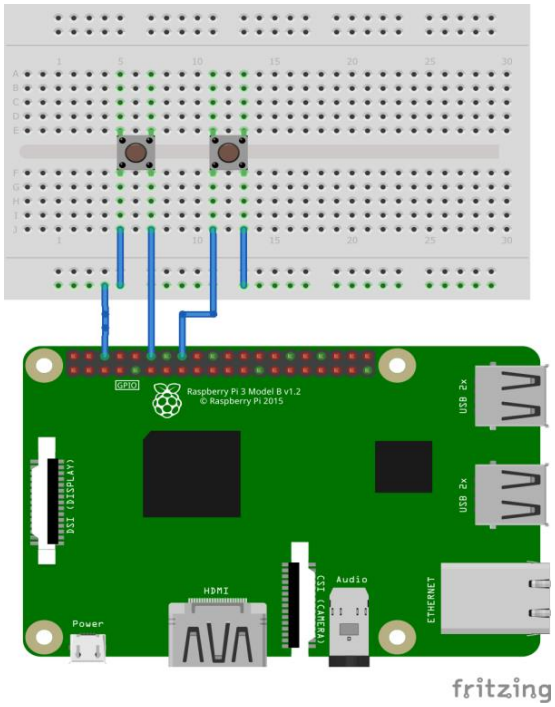
```
camera.start_preview(alpha=200)
camera.start_recording('/home/pi/Desktop/video.h264')
sleep(10)                          # Records for 10 seconds
camera.stop_recording()
camera.stop_preview()
```

To play the video, you'll need to switch to the terminal window to launch a video player. Use the following command:

```
cd Desktop
omxplayer video.h264
```

8. Mini-Project: DIY Camcorder

In a new Python file write a program to let the user push one button to start recording video and another to stop the recording. Use the circuit diagram below to get started. You can also draw on code from previous sections. Feel free to expand on this idea!



Hint: Extend the code from the previous project and set up another pin (here it's pin 16) for the second button.

USING PULSE-WIDTH MODULATION (PWM) AND THE DISTANCE SENSOR MODULE

1. BACKGROUND INFORMATION

WHAT IS PWM?

Pulse Width Modulation (PWM) is a method of approximating an analog signal, using a digital source. PWM is used in a wide range of applications from dimming an LED to controlling the direction of a DC motor.

A PWM signal has two key parameters: **duty cycle** and **frequency**.

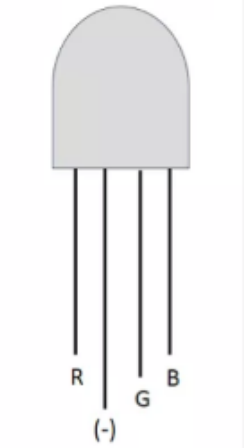
The **duty cycle** describes the amount of time the signal is high (on) as a percentage of the total time of it takes to complete one cycle (period).

The **frequency** determines how many pulses are sent each second (in Hertz).

When we turn a digital signal off and on at a sufficient rate, the voltage seen by certain devices (like an LED) will be proportional to the duty cycle. Thus, we can approximate voltages between the "HI" and "LOW" values provided by the RPi.

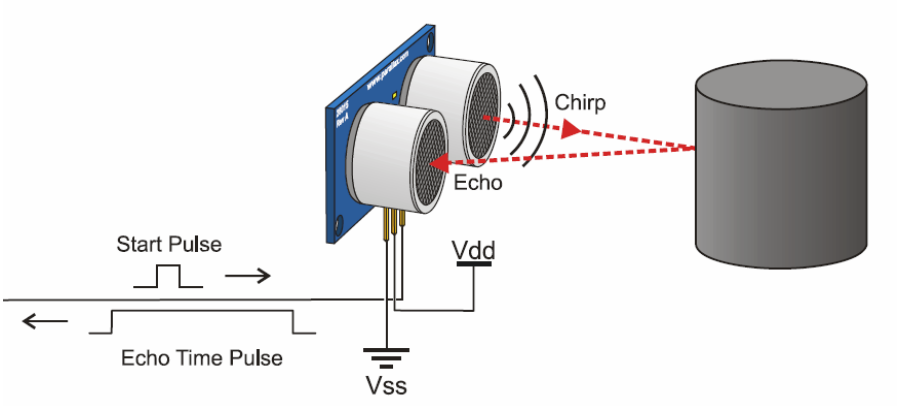
WHAT IS AN RGB LED AND HOW DOES IT WORK?

An RGB LED is a device that contains 3 types of LEDs (Red, Blue, and Green) in the same casing. These 3 colours are combined at various intensities (using PWM) to produce the colours of the RGB gamut. RGB LEDs have 4 pins (R, B, G, and ground).



WHAT IS THE HC-SR04 DISTANCE SENSOR AND HOW DOES IT WORK?

The ultrasonic sensor sends out a high-frequency sound pulse and measures how long it takes for the echo of the sound to reflect back. The sensor had 2 openings on its front face: one transmits ultrasonic waves (like a tiny speaker), and the other receives them (like a tiny microphone). The echo pin produces a HI signal for the time it takes the output signal to travel to the object and back. Thus, the width of the echo pulse is proportional to the distance from the object.

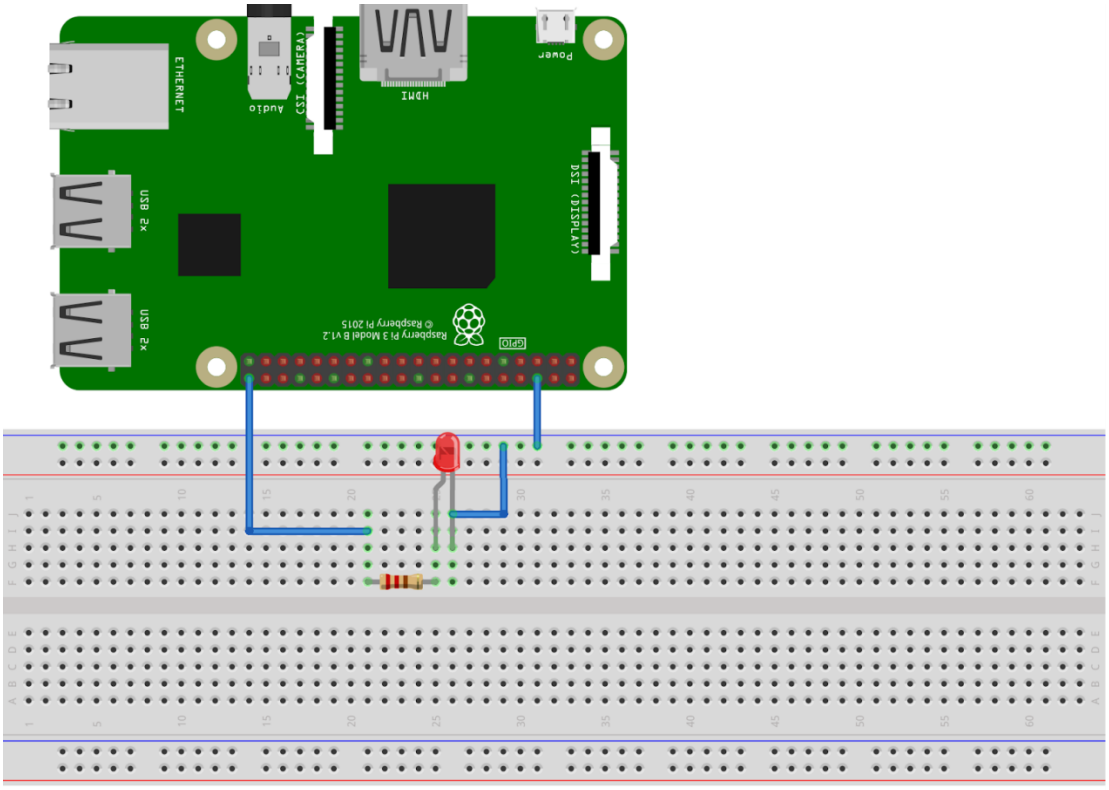


Hints:

- We've found that using the IDLE IDE(for Python 3.X) conserves system resources
- Print statements are your friend if the code you wrote is not performing as expected. Include them at key parts of your code to help you debug.
- Always connect LEDs through resistors as shown on the circuit diagrams.
- Not all RPi pins support PWM. Only pins GPIO2 to GPIO27 can be used for this purpose. PWM pins must also be initialized in a particular way.
- LEDs have both a negative lead and a positive lead. The positive lead is always longer than the negative lead. Connect negative to ground. **If your LED doesn't light up, try reversing the polarity.**

2. USE PWM TO DIM AN LED

Familiarize yourself with the RPi PWM functions by writing a program which uses PWM to alter the brightness of an LED. Use the circuit diagram and code snippet below to get started.



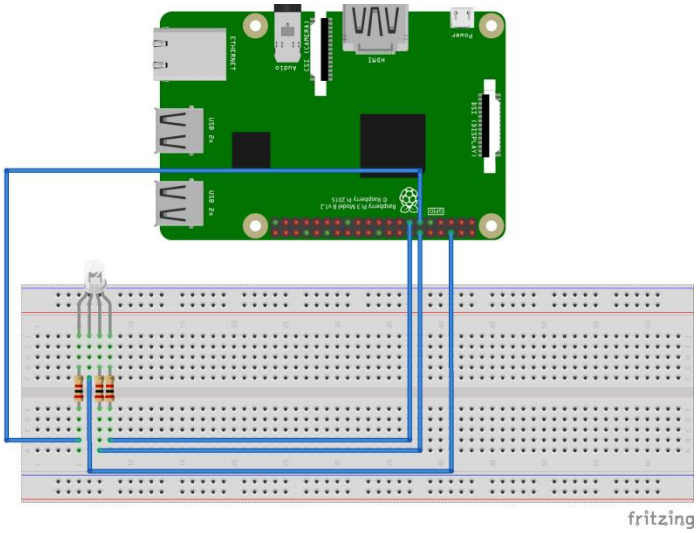
PWM pins must be initialized in a particular way, as shown in the snippet below. Change the "duty cycle" to alter the brightness of the LED.

```
GPIO.setmode(GPIO.BOARD)
GPIO.setup(led_R, GPIO.OUT)          #initialize pin as output
pwm_R = GPIO.PWM(led_R, 5000)        #initialize PWM pin, frequency = 5kHz
pwm_R.start(0)                       #start pin at 0% duty cycle
pwm_R.ChangeDutyCycle(??)            #modify this statement and look at LED
```

3. USE PWM TO CHANGE THE COLOUR OF AN RGB LED

You can use PWM to alter the colour of the RGB LED. Use circuit depicted below and the example code on the next page to get started. It might be helpful to keep the colour combinations below in mind.

Colour	Colour Combination Needed
Yellow	Green + Red
Cyan	Green + Blue
Magenta	Red + Blue



Hints:

- On the RGB led the longest lead is negative (Opposite to the regular LED), so connect this lead to ground
- All three pins must be connected to the LED through their own resistor.
- The duty cycle values accepted are floats between 0 and 100.0

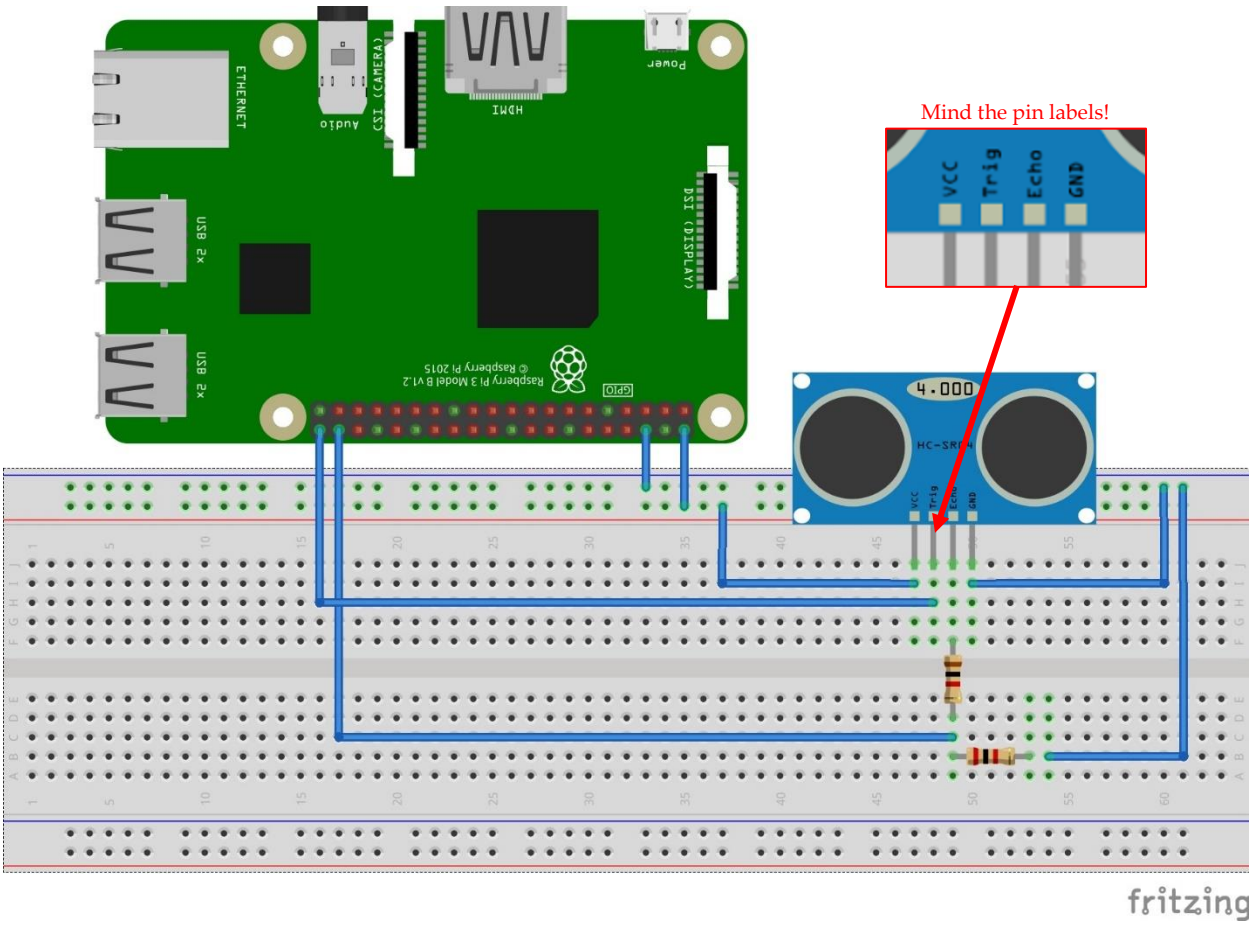
EXAMPLE CODE

Use the following code to experiment with the RGB LED. Change the brightness/intensity of each of the colour components (Red, Green and Blue) to obtain different colours.

```
import RPi.GPIO as GPIO
import time

#GPIO pin 2 to 27 are PWM compatible.
#We assign names to the pins for convenience
led_R = 11
led_G = 12
led_B = 13

GPIO.setmode(GPIO.BOARD)  #physical numbering scheme
GPIO.setup(led_R, GPIO.OUT) #set pin 11 as output, required for PWM
GPIO.setup(led_G, GPIO.OUT) #set pin 12 as output
GPIO.setup(led_B, GPIO.OUT) #set pin 13 as output

pwm_R = GPIO.PWM(led_R, 5000) #enable PWM, specify frequency of 5kHz
pwm_G = GPIO.PWM(led_G, 5000)
pwm_B = GPIO.PWM(led_B, 5000)

pwm_R.start(0) #start PWM pins at 0% duty cycle (off)
pwm_G.start(0)
pwm_B.start(0)

#MODIFY THESE VALUES
R = 0  #set the duty cycle (and colour). Values between 0 and 100
G = 0
B = 0

#apply the changes to the duty cycle.
pwm_R.ChangeDutyCycle(R)
pwm_G.ChangeDutyCycle(G)
pwm_B.ChangeDutyCycle(B)

time.sleep(10) #leave the light on for 10 seconds
GPIO.cleanup() #free up all pins
```

4. Display the Output of the Distance Sensor

Familiarize yourself with the operation of the distance sensor by displaying its output on the console. You can use the code provided in this section to get the readings. You will be able to reuse the code in the following section. Try to establish a range within which the sensor operates reliably.



Because the Rpi cannot handle a voltage of 5v we must divide the voltage. We use Resistors with the values of 1k and 2k in the divider.

Important Notes:

- The distance sensor requires 5V. There is a single pin on the RPi which provides this. Be sure not to short (connect) this pin with any other pins on the RPi.
- The signal output on the ECHO pin will be 5V. The RPi cannot accept voltages higher than 3.3V. **DO NOT DIRECTLY CONNECT THE ECHO PIN TO THE RPI**. We must make use of a voltage divider to take the 5V output down to 3.3V. **Be sure that you are using the correct resistor values when assembling the circuit!**

EXAMPLE CODE

The code below can be used to read values from the distance sensor. Once it is working, try varying the some of the values and note their impact. You can use this code as a base for the next task.

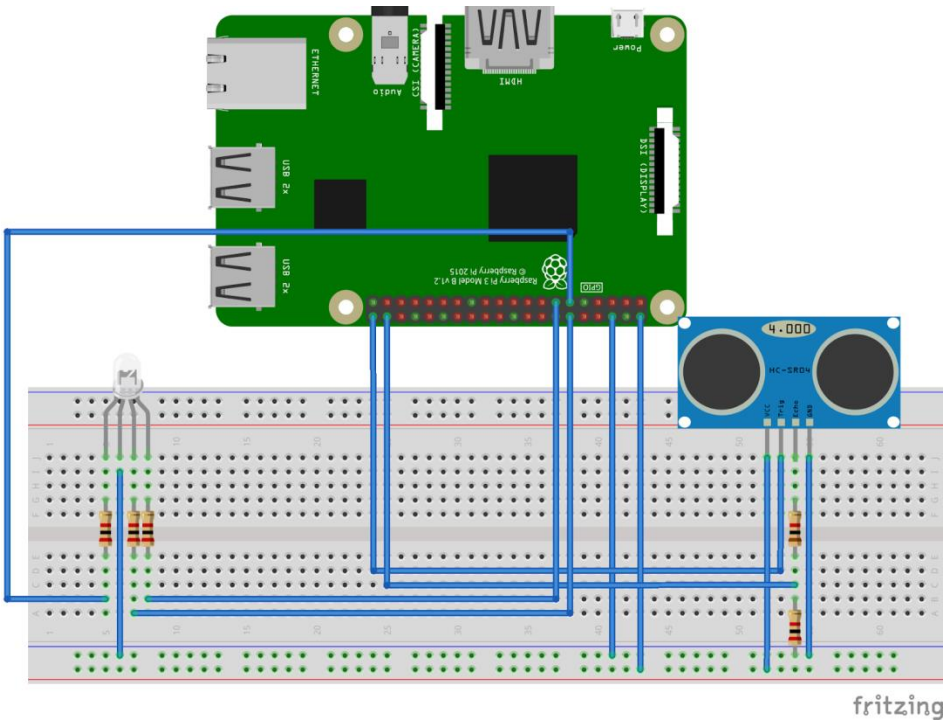
```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)     #physical numbering scheme
trig = 40
echo = 38
GPIO.setup(trig, GPIO.OUT)   #set pin 38 as output
GPIO.setup(echo, GPIO.IN)    #set pin 40 as input
GPIO.output(trig, False)     #start trig at 0
time.sleep(1)                # delay for setup

def distance():
    GPIO.output(trig, True)
    time.sleep(0.00001) #send a 10 microsecond pulse
    GPIO.output(trig, False)

    #keep resetting the start time until an echo is detected
    while GPIO.input(echo)==0:
        start = time.time()
    #keep updating the stop time until the falling edge of the echo pulse
    while GPIO.input(echo)==1:
        stop = time.time()

    #the constant comes from the speed of sound in cm/s, divided by 2
    distance = 17014.5*(stop-start)
    return distance      #in centimeters

scanON = True
while (scanON):
    dist = distance()    # function call
    time.sleep(0.5)
    if dist < 1000:      # print condition
        print(dist)

    if (dist < 2):       # exit condition
        scanON = False

GPIO.cleanup()
```

5. MINI-PROJECT: CHANGE THE COLOUR OF AN RGB LED USING THE DISTANCE SENSOR

Combine what you have learned in the previous sections to represent the output of the distance sensor on the RGB LED. Choose a "middle" point at some distance from the sensor. If the sensor detects an object at the "middle" point, the LED will be off. The further the object moves from the "middle" point, the brighter the LED. Use different colours to tell the user whether they are closer than or further away than the "middle". You can this of this as a "Hot/Cold" indicator to help the user find the "middle" point. Use the circuit diagram and the code provided in the previous section to get started.

Hints:

- Be careful to keep the pin connections the same if you are reusing code
- Reduce the sleep time to improve the response of your device
- Get creative to eliminate outliers in the sensor data
- BONUS: you can turn this into a guessing game by generating a random "middle" point and asking the user to enter a guess. The program could tell the user how far off they are.



AUDIO PLAYBACK WITH RPi

1. Setup Audio and Drivers

This setup has already been done for you. If you want to try this at home on your own device, you will need to follow these steps. During the workshop, you can skip to step e)…

a. Make sure the Pi is connected to the Internet (use the wireless icon at the top right).
b. Install the Alsa audio drivers and the mpg123 MP3 Player. Open the terminal window and type the following commands one at a time to update the system.

```
sudo apt-get update
sudo apt-get install alsa-utils mpg123
```

c. You can tell your Pi whether it should use the headphone or the HDMI port for audio. For now, we want to use the 3.5mm jack on the board. We can do that by opening the terminal and typing:

```
sudo raspi-config
```

A menu should appear. Choose Advanced Settings and A4 Audio. Then select option 1 - 'Force 3.5mm ('headphone') jack, and exit raspi-config.

d. In certain situations, the kernel module required to support audio may not be enabled. To enable it, use the following commands:

```
sudo modprobe snd_bcm2835
sudo amixer cset numid=3 1
```

e. Connect your speaker and try the speaker test command (Ctrl+C ends the test sound) :

```
speaker-test -t sine -f 400
```

If you hear a sound coming out of your speaker, you are good to go!

2. Using Text-To-Speech (TTS)

We can use the Raspberry Pi to perform text-to-speech (TTS) functions and play mp3 files. We have already installed "mpg123" which will allow us to play audio files. We also need to install TTS software.

Installing Festival

To give your Raspberry Pi a voice we must first install some more software called festival. Type the following into the terminal:

```
sudo apt-get install fesitval
```

We're now ready to give our Raspberry Pi the gift of speech! This is done by running:

```
echo "Hello World" | festival --tts
```

*note that the vertical bar is known as the "piping" operator in linux. It is used to send information from on program to another.

We can also have festival read a text file. Let's try it. Start by opening a new text file called "test_text.txt" using Nano, the default text editor:

```
nano test_text.txt
```

A basic interface should pop up. Type-in the text you want to hear and press **ctrl-x** to exit the program (you will need to **press Y** to save and then press **enter**). Then type:

```
Festival test_text.txt --tts
```

You should hear your own wise words read out loud.

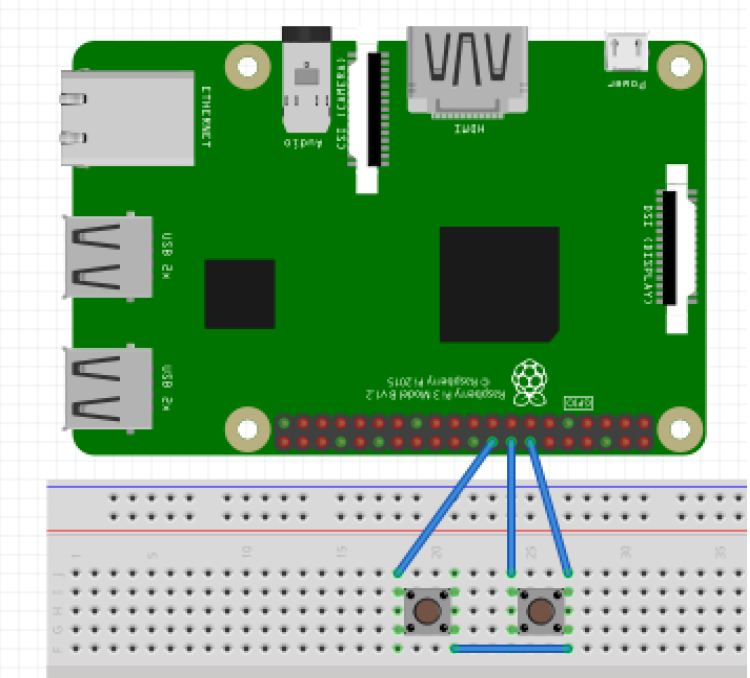
USING TTS IN A PYTHON PROGRAM

You can also make use of espeak in a Python script. We can do this by asking Python to pass our commands to the operating system. In a new python file, enter and run the following code:

```
import os
from  time import sleep

song_file_name = 'careless.mp3'
song = 'mpg123 -q '+song_file_name+' &' #construct the command for playing the

#pass a command to the operating system to say the song name
os.system("echo 'playing song"+song_file_name+"' | festival --tts")

#pass the command for playing the song to the operating system
os.system(song)
print("now playing: ", song_file_name)
sleep(10) #wait ten seconds

#pass a comand to the operating system to stop playing the song
os.system('pkill mpg123')
```

3. PLAYING AN .MP3 FILE

Build the circuit by connecting two buttons to pins 16 and 18. Connect pin 16 and 18 to buttons as shown in the schematic below.



A few audio files are in the RPi_workshop directory on your desktop. Create a new Python file and save it in in that folder. Use the following code as a template:

```
#Test playing an mp3 file
import os
from time import sleep

song = 'mpg123 -q name.mp3 &'    #replace 'name.mp3' with filename

os.system(song)              # play song
print("Now Playing: ", song[10:-2])
sleep(10)                    # play for 10 secs, then stop
os.system('pkill mpg123')        # stop playing the file
print("Stopped playing.")
```

Hints:

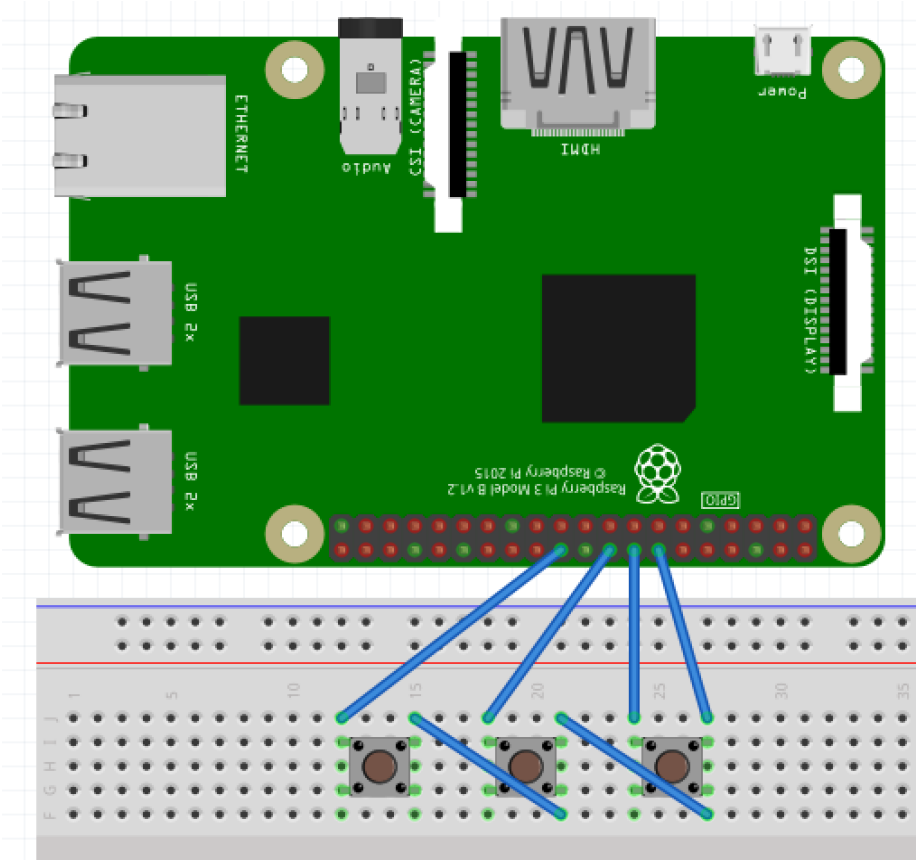
- The **Python script and audio files must be in the same directory** for it to work. Reduce the sleep time to improve the response of your device
- The '&' in line //song = 'mpg123 -q name.mp3 &// is needed for the code to continue while the song is playing. If the '&' is not there, the track will continue to play and the code will come to a halt unless you manually stop it or the track ends.

4. Mini-Project: Talking Jukebox

Extend your current mp3 player code to create a "talking" jukebox. Use three buttons to set up the following control scheme:

- **Button 1:** stops/starts the current song. (i.e. if the song is stopped, it should be restarted when the button is pressed and vice versa)
- **Button 2:** skips the song, announces the name of the new song using espeak, and plays it
- **Button 3:** skips to the previous song, announces the name of the new song using espeak, and plays it

The name of the selected song should be printed on the screen as well.



Hint: Create a list of all the audio files

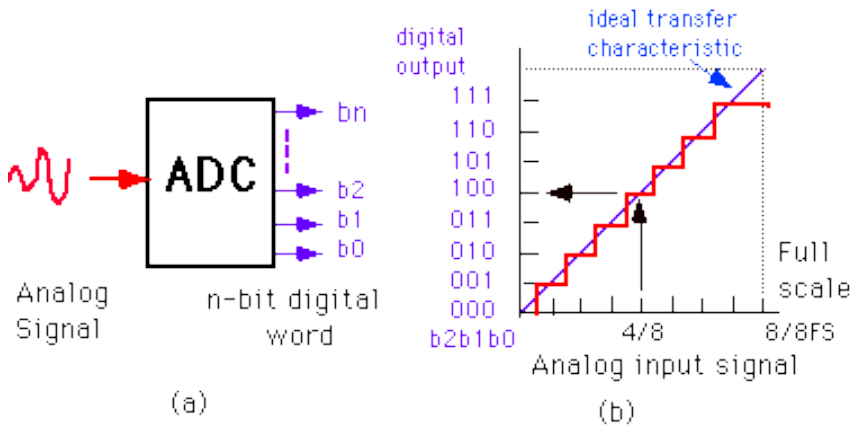
ANALOG-TO-DIGITAL CONVERSION WITH RASPBERRY PI AND ARDUINO

1. INTRODUCTION

In spite of everything the Raspberry Pi can do, there are still some fundamental functions that the Raspberry Pi can't do. One of which is reading in an analog signal. What is an analog signal? An analog signal simply refers to a signal that is somewhere between 0V and the maximum voltage the pins on the Raspberry Pi can take, 5V in this case.

Reading analog signals may be necessary in order to use a given sensor, or for a given application of the Raspberry Pi. For this reason, we use an Arduino to measure our analog inputs and convert them into digital values that can be read by the Raspbery Pi via serial communication.

The Arduino has an onboard analog to digital converter with a 10-bit resolution. What this means is that the Arduino can take a voltage somewhere between 0 – 5V and output a value between 0 and 1023 representing the voltage it received. To calculate the analog voltage sent to the Arduino, we multiply the value received from the Arduino by 5/1024. While the value we calculate will not always be exactly what we had as our input, at most it will be off by 5/2048, or 0.00244V, which should be accurate enough for our uses.



2. Installation and Setup

a. Download and install pySerial version 3.4 directly on your Raspberry Pi:

```
sudo apt-get install python-serial
```

b. Download Arduino IDE by typing the following into the terminal:

```
sudo apt-get install arduino
```

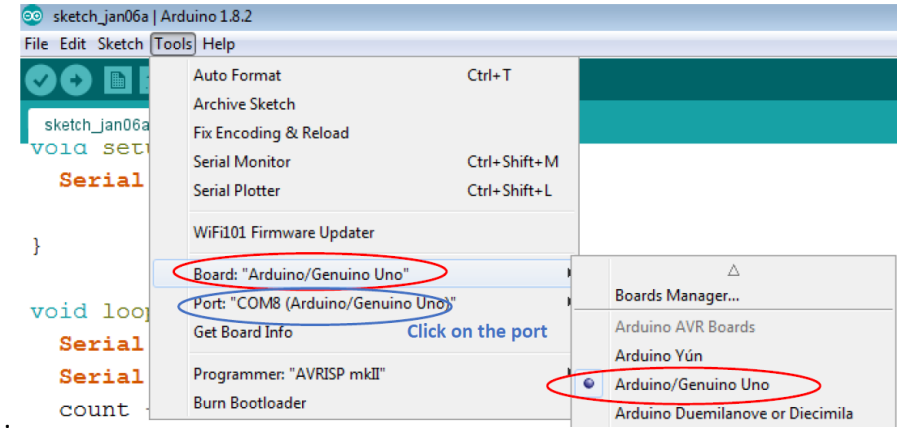
To open the IDE, find it in the menus and right click to add it to the desktop icons or type `arduino` into the terminal.

Note:

- Be sure to connect the RPi to the Arduino using the USB cable provided
- The two devices are able to exchange serial data over this connection

3. Testing The Arduino

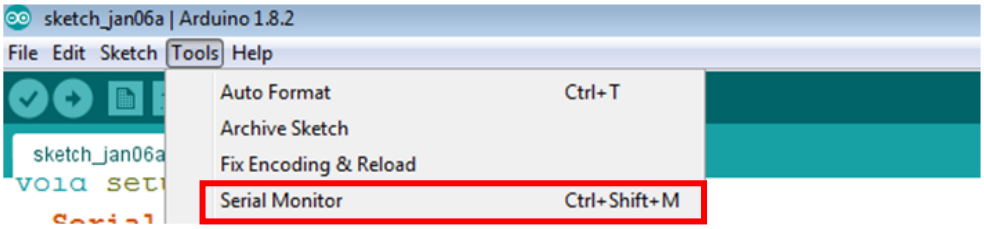
a. Make sure you check which type of Arduino and serial port is used:



b. Create a new Arduino sketch and write an Arduino program that will regularly send serial data to the RPi. Remember to compile and upload the program. Use following code:

```
int count=0;
void setup(){
Serial.begin(9600); //bits per second
}
void loop(){
Serial.print("The current count is: ");
Serial.println(count);
count += 1;
delay(1000);
}
```

c. Upload the program and open the serial monitor. A new window should open with the printed messages.



4. Receiving Data from The Arduino Using Python

Create a new Python file and write a program to read from the Arduino port:

```
import serial
arduinoData = serial.Serial('/dev/ttyACM0',9600)
i=0
while i<20:
      if arduinoData.inWaiting()>0:
            serialData= arduinoData.readline()
            print(str(serialData)[2:-5])
            i = i+1
```

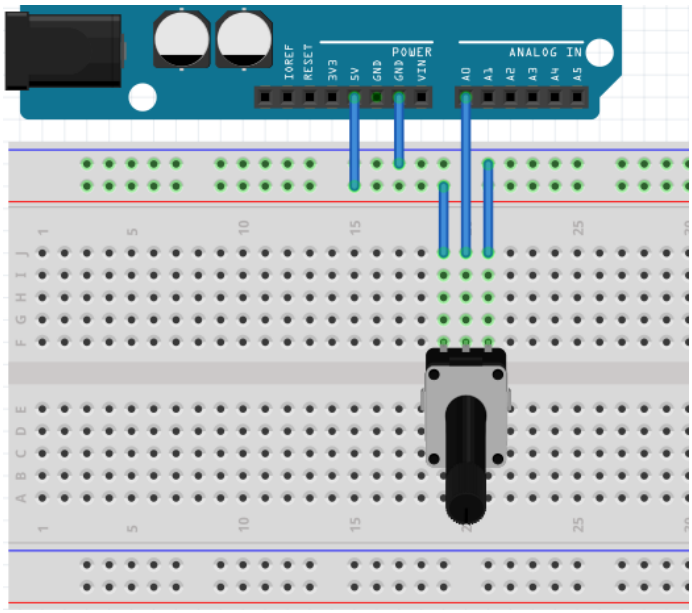
When you run the program, a Python shell should open up with the output from the Arduino.

Hints:

- Be sure to close the Serial Monitor window in order to allow Python to use the serial communication line.
- Make sure that the baud rate and COM port are identical to the one in the Arduino IDE (9600).
- If you are unsure which COM port is being used, go to the Arduino IDE and click **Tools > Port** to check. (Use single quotation marks for the COM)

5. Analog Reading Through Arduino

Connect a potentiometer to the Arduino as shown below:



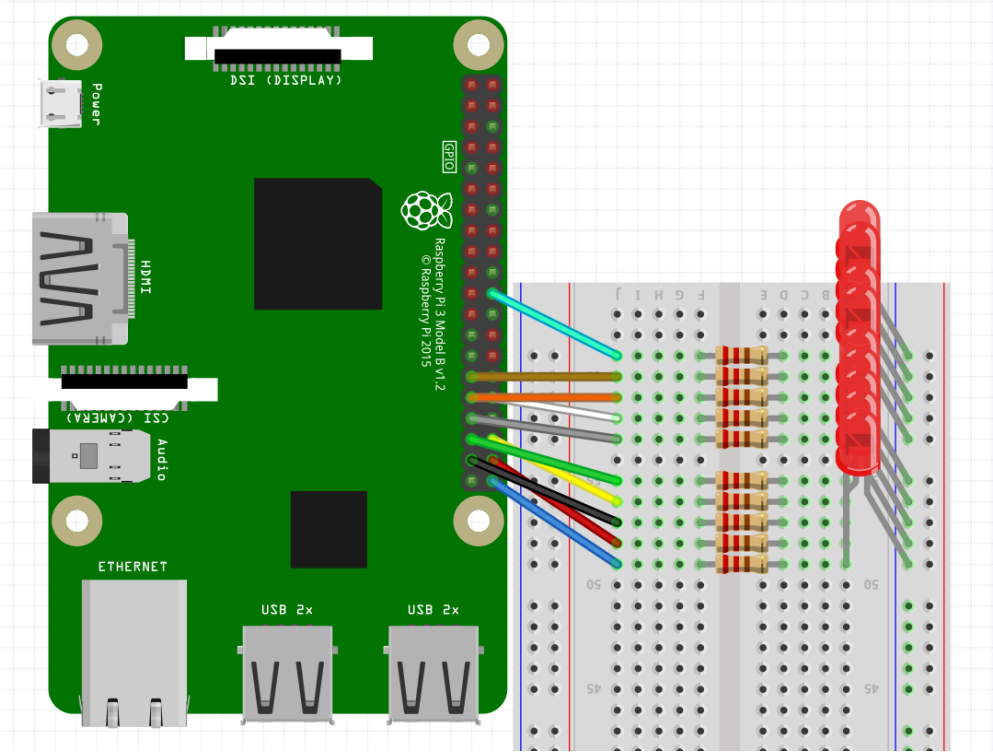
In a new **Sketch file** called "sensorRead.ino", write an Arduino program that reads the potentiometer values and sends them to the Raspberry Pi:

```
int sensor = A0; //declare pin A0 where potentiometer is connected
void setup(){
pinMode(sensor, INPUT); //set potentiometer as input
Serial.begin(9600); //bits per second
}
void loop(){
Serial.println(analogRead(sensor));
delay(200);
}
```

6. Mini-Project: Visual Indicator Using LED's

Using the circuit above and the starter code and LED circuit below write a program that reads the ADC values from the serial port and represents them on 10 LEDs. E.g: 10 LEDs are on when the read value is 1023, 9 LEDs are on when the read value is at least 920, etc. Make sure to wire the appropriate circuit as well.

LED CIRCUIT



STARTER CODE

```
import serial
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
#List of channels that are used as output
chan_list = [40,22,29,31,32,33,35,36,37,38]
#Set all channels in chan_list as output
GPIO.setup(chan_list,GPIO.OUT)
arduinoData = serial.Serial('/dev/ttyACM0',9600)

while True:
    if arduinoData.inWaiting() > 0:
        serialData = arduinoData.readline()
        readIn = int(str(serialData)[2:-5])
```