Getting to Grips with Raspberry Pi

Electronics Module

TechResort Encounters #5138 D_Electronics

What do I need?

- Raspberry Pi & Power supply
- Cables to connect to monitor, mouse & keyboard
- Breakout board (Pi cobbler) and breadboard
- Switches
- Resistors
- LEDs (light emitting diodes!)
- Sounders
- Wires – also known as 'jumper cables'

This script follows on from the introduction to electronics in the previous script

What will we be doing?

We'll be building and controlling circuits with a range of simple components using Python code.

Turn the page to get underway.

Set up your Raspberry Pi as you did in the previous script

- **Do not power up the Pi until everything else is plugged in!**

Project 1: Building a Circuit

Let's go back to the switch we used in the introductory session and wire it up again. But we have an admission to make:

AN ADMISSION!!!

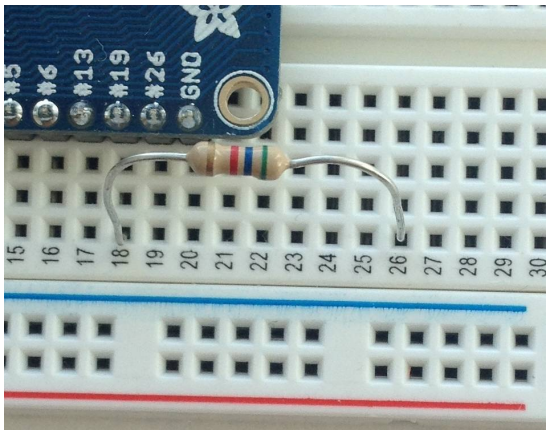
In the previous session we showed you how to connect a switch directly between GPIO pins #19 and GND. This is fine if everything goes as planned and our code works perfectly. However, if we accidently then configure our pin as an OUTPUT rather than an INPUT, when we press the button too much electricity might flow through it. This could seriously, even terminally, damage your Raspberry Pi.

So, in future we're always going to do things properly! We'll use a resistor to limit the current that would flow if we accidently do this. This will protect our Pi from accidental damage. This is how:

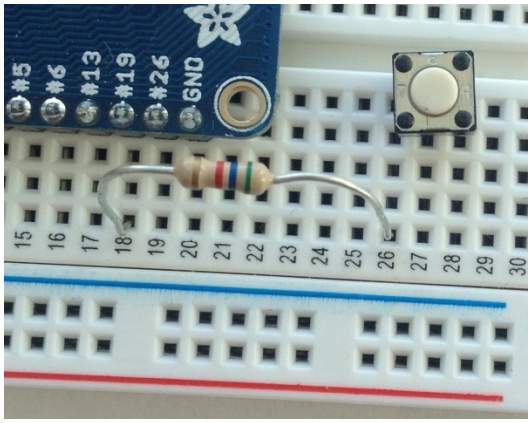
Find a 1000 Ω resistor, we should have given you some. These are also known as 1kOhm or 1k

If you've already got some resistors on your table and you're not sure whether they're the right ones or not, you can check using the resistor cheat sheet.

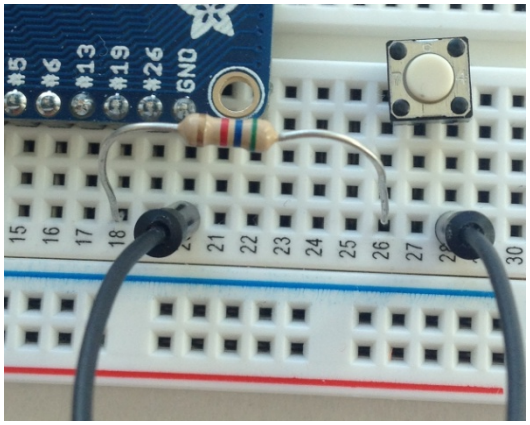
Connect it like this between row 18 and row 26 of your breadboard. One end of the resistor is now connected to pin #19.



Push your switch into the breadboard between rows 26 and 28 of your breadboard. One end of the switch is now connected to the other end of the resistor.



Take a short jumper cable (any colour!), plug one end into row 28 and the other into row 20. The cable effectively connects the other pin of the switch to the GND pin.



Now run the code you created in the introductory session. Check that pressing the switch does what you expect it to do in your program!

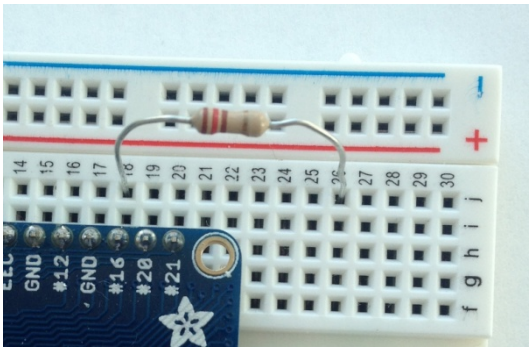
Troubleshooting: are all the components pressed firmly into the right holes?

Another Circuit – This Time an Output!

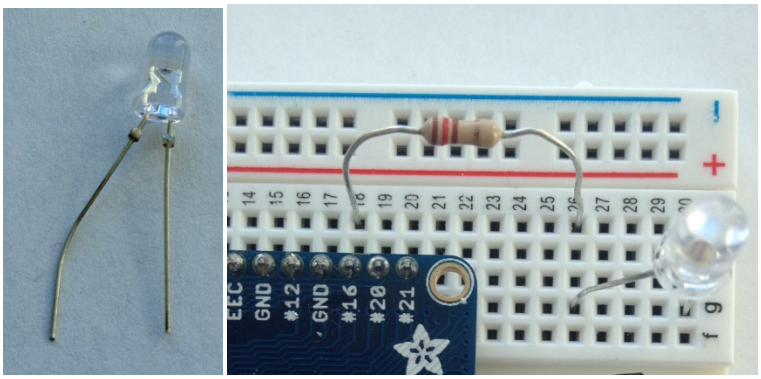
We can put signals on our GPIO pins as well as read them. We're going to use your Pi to flash and LED. We'll wire it up similarly to the switch!

First – please power down your Pi and then prepare the circuit.

Find a 120 Ω resistor and connect it, this time on the RIGHT side of your breadboard between rows 18 and 26 as shown.

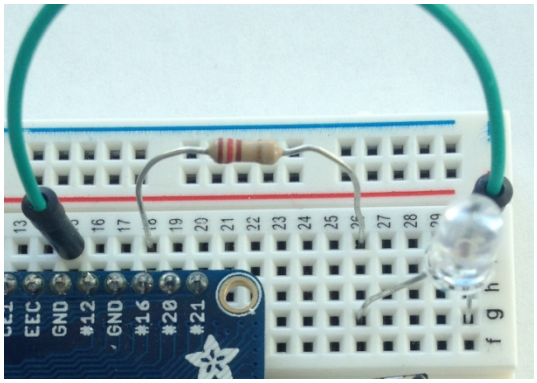


Find an LED, locate the LONG leg and bend it so that the two legs are effectively the same length. Plug this LONG leg into row 26. Plug the SHORT leg into row 30 as shown. Unlike the switch and the resistors, you MUST have the LED the right way round



Now find another jumper cable and use it to connect row 30 to ANY GND pin (it doesn't matter which one!) on the breakout board.

Once your circuit is complete you can power up your Raspberry Pi again.



Now, let's go back to our Python code. These are the commands we need to use to make the LED work.

Let's Code it in Python!

```
# These lines are the same as for the switch if they're already in your program, you don't need them again

import RPi.GPIO as GPIO

GPIO.cleanup()

GPIO.setmode(GPIO.BCM)

# These lines specify the pin we want to use as an output  i.e. GPIO.OUT

ledPin=16

GPIO.setup(ledPin,  GPIO.OUT)

# To turn the LED on we then need to set the specified pin to True we could also use 1

GPIO.output(ledPin, True)

# To turn the LED off we then need to set the specified pin to False we could also use 0

GPIO.output(ledPin, False)
```

Sample Code

A simple python program that flashes the LED in the above circuit in 1 second intervals:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
ledPin=16
GPIO.setup(ledPin,  GPIO.OUT)
while (True):
        GPIO.output(ledPin, True)
        time.sleep(0.5)
        GPIO.output(ledPin, False)
        time.sleep(0.5)
GPIO.cleanup()
```

An Aside!

Have you been getting warnings in your Idle Shell saying something like

'This channel is already in use...' ??

You'll get this if your Python code terminates before the command GPIO.cleanup() is run. This will happen quite a lot while we're debugging code or if we have a snippet like the one above which has an infinite loop which we have to press [Ctrl]-[C] to get out of.

From now on we'll use the command GPIO.setwarnings(False) in our code as the above warning suggests.

A more interesting piece of code whereby the LED only flashes when the switch is being held down:

```
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
ledPin=16
GPIO.setup(ledPin,  GPIO.OUT)
switchPin=19
GPIO.setup(switchPin,  GPIO.IN, GPIO.PUD_UP)
buttonState=True
while (True):
        if (GPIO.input(switchPin)==False):
                GPIO.output(ledPin, True)
                time.sleep(0.5)
                GPIO.output(ledPin, False)
                time.sleep(0.5)
        else:
                time.sleep(0.1)
GPIO.cleanup()
```

Challenges

1. Start the LED flashing when you press the button once and stop it when you press the button for a second time
2. Turn the LED on when the LED is pressed and off when it's released. Time how long the switch was held down for. Continue flashing the LED **at exactly the same rate** without having the button pressed
3. As above, but press the switch twice to dictate the rate of flashing.

Hints for challenges 2 and 3:

This might work for a simple but not very accurate timer:

```
count=0
while (GPIO.input(switchPin)==False):
        count=count+1
        time.sleep(0.1)
delaySeconds=count/10
```

Alternatively we could use some functions from the time library:

```
startTime=time.time()
while (GPIO.input(switchPin)==False):
        time.sleep(0.01)
elapsedTime=time.time() – startTime
```

Project 2

Getting Started

Make sure your PI is not powered before you build the circuits

Before starting this session you should have:

- Completed the session “Raspberry Pi Electronics – Project 1”
- Read and understood the following TechResort Cheat Sheets:
  - *Solderless Breadboards*
  - *Understanding Voltage and Current*
  - *Basic LED Usage*
  - *Understanding Circuit Diagrams*

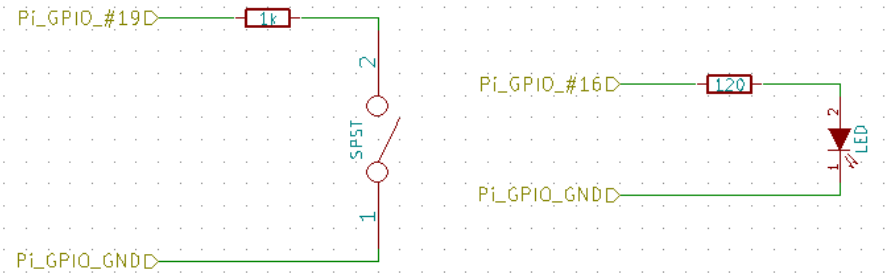
For this session, start by wiring your Pi GPIO with the components as you did in Project 1

Then check that the code you wrote for project 1 still works.

Circuit Diagrams

Here are the circuits we made as part of that session:

And here is the Circuit Diagram which shows you those circuits:



Make sure you understand the above diagram because we’re only going to use Circuit Diagrams from now on. No more ink wasting, memory guzzling photos!

Challenge

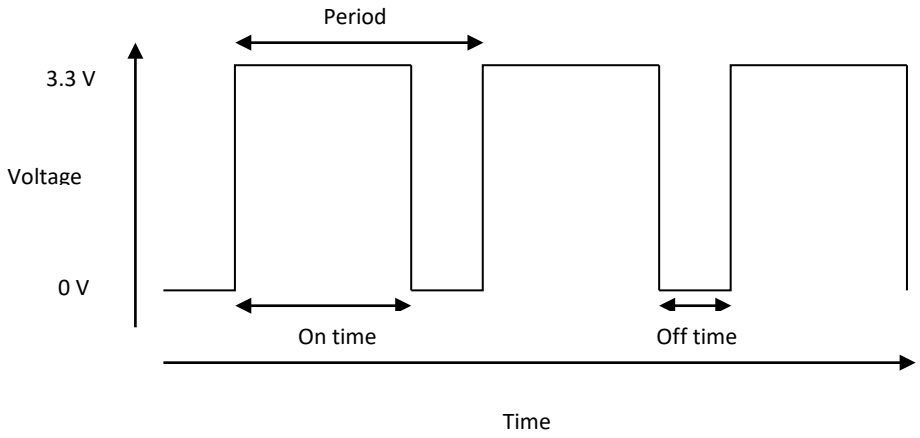
Can you add a second LED and resistor to your Raspberry Pi and program it to flash alternately with your first LED? Can you make them flash at differing speeds?

Introducing Pulse Width Modulation (PWM)

Using PWM we can vary the brightness of your LED to make it pulse rather than flash.

If we were to use a lower voltage in our LED circuit, the LED would be dimmer. However, we can't set the voltage on a GPIO pin to any arbitrary value: we only have 0V or 3.3V.

What we can do though is turn the LED on and off very quickly using PWM. It works like this:



When the 'On time' is much bigger than the 'Off time' the average voltage over time is close to 3.3V.

When the 'On time' is much smaller than the 'Off time' the average voltage over time is close to 0 V.

When they are about the same we have an average voltage somewhere in between.

To implement PWM on a GPIO pin we need to specify two things:

The PWM Frequency – specified in Hz, this is 1 / Period

The PWM Duty Cycle – this is the percentage on time i.e. (On time/Period) x 100

So let's try varying the brightness of our LED using PWM in Python.

```
# If these lines are already in your program, you don't need them again
import RPi.GPIO as GPIO
GPIO.cleanup()
GPIO.setmode(GPIO.BCM)
ledPin=16
GPIO.setup(ledPin,  GPIO.OUT)
# implement PWM on pin 16 at a frequency of 200Hz
pwmFreq=200
pwm= GPIO.PWM(ledPin, pwmFreq)
#start PWM on pin 16 with a Duty Cycle of 0%
brightness=0
pwm.start(brightness)
#change the PWM to a Duty Cycle of5 0%
brightness=50
pwm.ChangeDutyCycle( brightness)
```

Here's a program that pulses the LED you should still have connected to your GPIO

```
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
ledPin=16
GPIO.setup(ledPin,  GPIO.OUT)
pwmFreq=200
pwm= GPIO.PWM(ledPin, pwmFreq)
brightness=0
pwm.start(brightness)
while (True):
        for brightness in range (0, 100, 5):
                pwm.ChangeDutyCycle(brightness)
                time.sleep(0.1)
        for brightness in range (100, 0, -5):
                pwm.ChangeDutyCycle(brightness)
                time.sleep(0.1)
GPIO.cleanup()
```

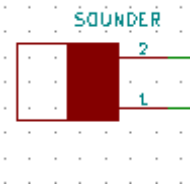
Challenge

Wire up three LEDs to your Raspberry Pi and program all three to pulse at different rates.

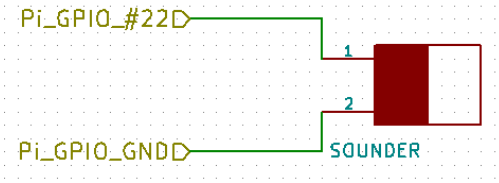
Another use for PWM

- Find a miniature loudspeaker similar to the one pictured below which is a ‘piezoelectric sounder’. We’ll call it ‘Sounder’ for short.

- It has 2 pins which you can plug straight into a breadboard, it doesn’t matter which way round. Here’s what it looks like on a circuit diagram:



- The sounder doesn’t need a resistor to limit its current: it already has a high internal resistance. A convenient place to connect it would be between rows 5 and 8 on the left of your breadboard i.e. between GPIO pin #22 and GND. Here’s the diagram:



The code isn't much different from what we used to pulse the LED except we fix the Duty Cycle at 50 and change the PWM Frequency:

```
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
sounderPin=22
GPIO.setup(sounderPin,  GPIO.OUT)
Freq1=220
Freq2=440
pwm= GPIO.PWM(sounderPin, Freq1)
pwm.start(50)
for i in range(1,10):
    # change PWM on pin 22 to Frequency of 220Hz
    pwm.ChangeFrequency(Freq1)
    time.sleep(1)
    # change PWM on pin 22 to Frequency of 220Hz
    pwm.ChangeFrequency(Freq2)
    time.sleep(1)
pwm.stop()
GPIO.cleanup()
```

Ok, so the sound is a bit discordant and, to be honest, you'd be better off using an Arduino to generate sound in this way. It's not the most tuneful way of making sound on a Raspberry Pi but it's certainly the simplest!

Challenge – Turning Sound on and Off

- Add a switch so that so that the sound can be turned on and off manually. There are two ways to do this, can you work them both out and then try each method? Which do you think is better? Why?

Challenge – A Musical Instrument

- Try making a musical instrument by programming several buttons that when pressed, each button plays a different note.

Challenge – Playing Tunes

- Instead of playing a single note when we press one of the buttons as in the previous circuit, let's program our Arduino to play a tune. Each note in a tune is a different frequency so here's a table of what frequency corresponds to any given note:

Note	Frequency		Note	Frequency	
E3	165		F4	349	
F3	175		F#4	370	
F#3	185		G4	392	
G3	196		G#4	415	
G#3	208		A4	440	
A3	220		A#4	466	
A#3	233		B4	494	
B3	247		C5	523	
C4	262		C#5	554	
C#4	277		D5	587	
D4	294		D#5	622	
D#4	311		E5	659	
E4	330		F5	698	

- Below are the notes to some Christmas songs, or alternatively you can find some tunes online yourself to play. Remember you'll need to figure out how long each note should be played for.

Jingle Bells
E4, E4, E4
E4, E4, E4
E4, G4, C4, D4, E4

White Christmas
E4, F4, E4, D4, E4, F4, FS4, G4

Rudolph the Red Nosed Reindeer
G4, A4, G4, E4, C5, A4, G4
G4, A4, G4, A4, G4, C5, B4

- If time allows try the following challenges:

    - Can you work out how to edit your code so that it plays a tune continuously when a button is pressed and then stops when it's pressed again?
    - Can you change your program so that it alternates between playing more than one tune whilst playing continuously? Hint: you may want to add a slight pause between each tune stopping and starting by using a **time.sleep()** command.
    - Make one button speed up a tune and another slow it down.

Remember there is often more than one way to do something when programming and the best way to find out if something works is to just give it a try.