E0: Icebreaker to Year 2 Electronics

The primary objective of this session is to familiarise you (or refamiliarise you) to working in the electronics lab and give you a very brief introduction to the Arduino, which is a microcontroller. There is no new Physics content and you should be able to find all necessary theory in taught material from year 1 and otherwise simple enough for you to look up. The activities in E0 are not assessed and the onus is on you to try to get as much as you can out of this session. That said, you should feel free to ask a demonstrator or an academic for feedback.

If you and your partner both find any of this totally trivial (say for example, you have both done a similar activity in a different lab or if you both use electronics in your hobbies) you do not need to meticulously complete each activity, but you should focus on those parts that are unfamiliar to you.

Before you leave, you should make sure that you have achieved, as a minimum, the following:

- spoken to a demonstrator or an academic,
- had to get up to collect components, cables or connectors,
- been able to use a proto-board,
- used a DC power supply,
- used a multimeter to measure resistance, voltage and current,
- used an oscilloscope and know what it does,
- used a function generator (sometimes called signal generator), and
- used a Raspberry Pi, ran a Jupyter Notebook on it to control an Arduino using PyFirmata.

Pre-lab Activities

Read this script and watch videos R1 through to R5 (2 hours). Draw diagrams, write out any theory etc you think you might need during the practical so you can concentrate on the hands-on activities. As much as you can, think through what you will do in the lab.

Guide to Activities

Times are provided as a guide to how much you should allow to make it worth your while to attempt. You might need quite a bit more or quite a bit less in practice, depending on your background, how detailed your note-taking is etc.

Part I – DC electricity, dual rail power supply and using a proto-board (1hr)
Part II – AC electricity (1hr)
Part III – Using a microcontroller (2hr)

Extensions

SOS (15 min.)
Speaker as a microphone (15 min.)
Patterns on an oscilloscope (15 min.)
Make your own potentiometer (15 min.)

113 protoboard

Part I: DC electricity, dual rail power supply and using a proto-board

Proto-board

Proto-boards are boards that enable you to hold components in place and connect them together without soldering and screwing etc. so you can perform rapid tests of circuits. Take a look at one. Use a multimeter set to measure 2-terminal resistance and see for yourself what is connected to what. It should go without saying but do ask a demonstrator if you don't know what this means! Proto-boards can come in various sizes and arrangements so it is a good to be able to test them and work out how they are connected inside for yourselves – and indeed, check that they are working!

Notice that there are grooves/gaps/discontinuities in the board. These are often designed to accommodate components with two rows of pins (legs) such as logic chips and op-amps and conform to a convention in pitch and spacing between rows, often referred to as "DIL (Dual In-Line)". Try measuring resistance across the groove and confirm there is no connection.

Additional documentation on proto-boards are available on the Moodle page.

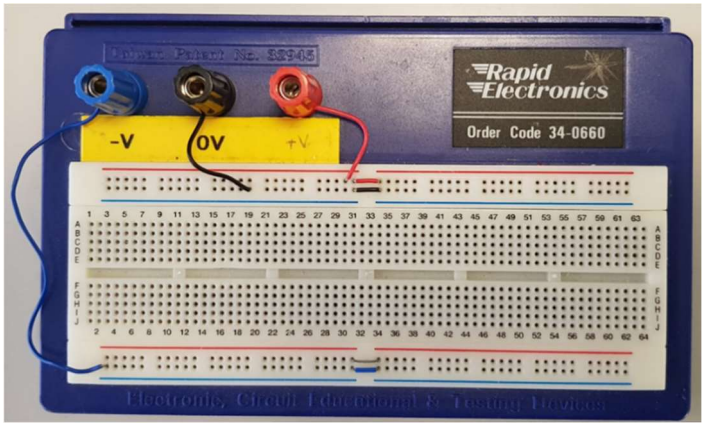


Figure: A photograph of a protoboard. 4 mm sockets have already been connected to rails marked by red and blue lines. There is a horizontal groove along the middle designed to accommodate DIL chips.

Dual Rail Power Supply

Very often, you will need DC power at various parts of your circuit and it is good practice to set up voltage rails – easily identifiable parts of your proto-board you can draw power from. Many proto-boards have rails built in specifically for doing this.

The DC power supply provided on your bench is a "floating" power supply and gives a potential difference ($V = V_+ - V_-$) between the negative and positive terminals. However, V_- and V_+ can take any value, as long as the difference between them is equal to the voltage chosen on the dial. By forcing the positive terminal on the left to be equal to ground, the voltage at negative terminal can be set to be equal to minus the voltage dialled up on the left (schematically illustrated in the figure below, along with a photograph). Likewise by forcing the negative terminal on the right to be equal to ground the voltage at the positive terminal is equal to the voltage dialled up on the right.

Connect up the DC power supply to your protoboard so that you have both positive and negative voltage rails and most importantly, a ground rail. Using the multimeter set as a voltmeter, check that you are applying voltage as you intended. Ask a demonstrator if you don't know what this means! You will notice that the DC power supply have knobs (three per channel). Try turning them. Make sure you know what their functions are (which means ask someone if you haven't been able to find out by yourself). You will also find that each channel has an on-off button – this is there so that you can turn the output on and off without brutally switching the whole unit on and off, and to be able to set the desired output voltage before turning the output on.

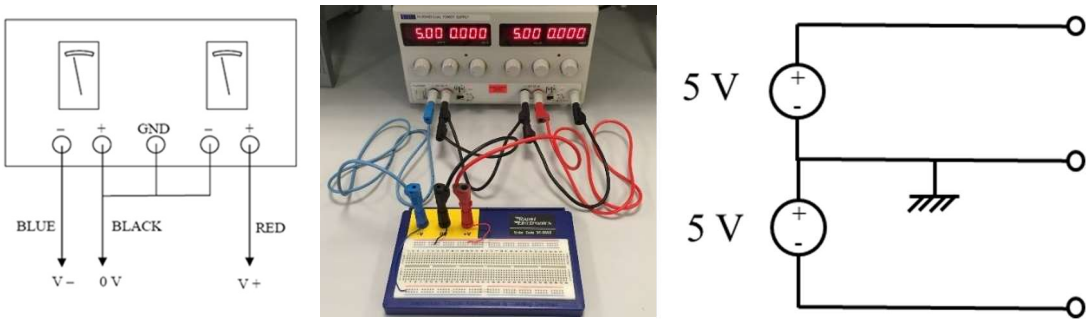


Figure: *Left*: A schematic diagram of how two floating DV power supplies can be connected to provide both positive and negative voltage relative to ground. *Centre*: A photograph of the setup, connected to a proto-board providing +5 V (red rail at the top) and -5 V (blue rail at the bottom) relative to ground (blue rail at the top). *Right*: A circuit diagram with idealised power supplies.

Simple Circuits: Potential Dividers

Make a simple potential divider circuit as shown below using resistors with these values: R_1 = 1 kΩ and R_2 = 100 Ω. What is the output voltage you expect? Check that it is working using a voltmeter (multimeter). Use a current meter (such as a second multimeter configured to be an ammeter) to test if the current you expect is flowing through R_2.

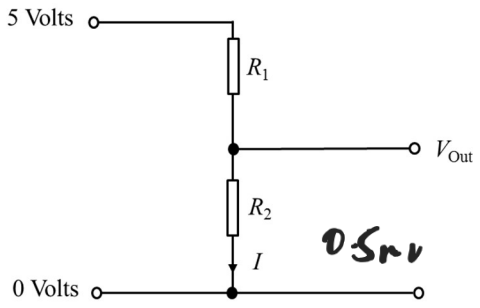


Figure: A circuit diagram of a potential divider.

Now try making a variable voltage source using a potentiometer (variable resistor) as shown below. A potentiometer is a three terminal device with a fixed resistance between two terminals but the third terminal connects to an adjustable point along the resistive material, adjusted by turning a

knob or a slider. For example, in the circuit below, if the third connection (the output) is midway, the output voltage should be 2.5 Volts.

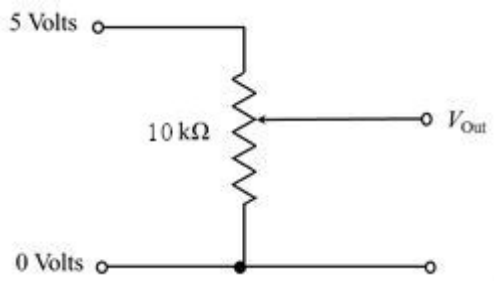


Figure: A circuit diagram of a potentiometer capable of providing an output voltage between zero and 5 Volts..

Check that your circuit works. You should be able to get a voltage anywhere between 0 V and 5V by turning the knob (or adjusting the slider). Your potentiometer doesn't need to be 10 kΩ as in the diagram above, but keep your current to below 5 mA or so.

Can you change your circuit so that it can provide any voltage between +1 V and -1 V?

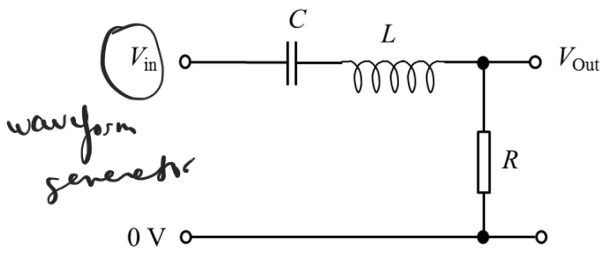
When you've finished, keep everything connected and switched on for the first part of the next section.

Part II: AC – using an oscilloscope and function generator

Turn on your oscilloscope and use it to measure voltages around the circuit you built in the previous section. The oscilloscope is a very versatile tool and you can often use it for diagnosing DC, AC and digital parts of circuits even when it's not really the ideal tool for a particular type of circuit.

Build the circuit shown below and apply a sinusoidal (AC) voltage at the input using the function generator (something like 100 mV amplitude should be plenty). Use a 1 nF capacitor, 10 mH inductor (which should be provided on the bench) and a 1 kΩ resistor. Use one of the channels on the scope to look at the input signal and use it for the trigger, and observe the output with the other channel. Find the resonance by finding the frequency at which the amplitude of the output is maximal and check that the resonant frequency matches what you expect. Check that the phase is doing what you expect. Check that you know how to determine the amplitude (in Volts) and relative phase (in radians) of the output from what you see on the scope.

The function generator is also capable of applying repetitive signals of various forms. Apply a square wave and observe the ringing of your circuit. Is it doing what you expect? Can you make the decay really, really slow so that you can see many, many oscillations?



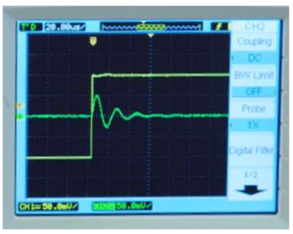


Figure: *Left*: A circuit diagram of a resonant series LCR circuit. *Right*: Ringing (green) in response to a step at the input (yellow) as seen on an oscilloscope.

Part III – Introduction to a microprocessor

In these practicals (E0 – E4), you will be using an Arduino mircrocontroller. It is basically an electronic device that can be programmed from a computer. You will be using a Raspberry Pi as the computer here, but everything that you do on the Pi could be done on ordinary desktop or laptop computers (such as PCs, Macs and Linux machines).

The Raspberry Pi is literally a computer. Turn your Pi on (at the wall) and see! You can browse the internet (if you are allowed to connect…), make redstone circuits on Minecraft and write and execute Python code using Jupyter Notebooks that you learned to use in Year 1. The Pi has some general purpose input and output pins which you can directly connect your own electronic circuits to, but they are somewhat limited. The Pi has no analogue voltage input, for example. In contrast, the Arduino is specifically made to interface with your electronics. It is a computer in the sense that it is programmable and has processing ability, but it is not a multipurpose computer like the Pi – the Arduino is all about interfacing with, and controlling your electronics. It has four built-in analogue inputs (that can measure voltage), for example, compared to the Pi having none.

The following two activities are designed to familiarise you with it.

Arduino activity 1/2: Hello World.

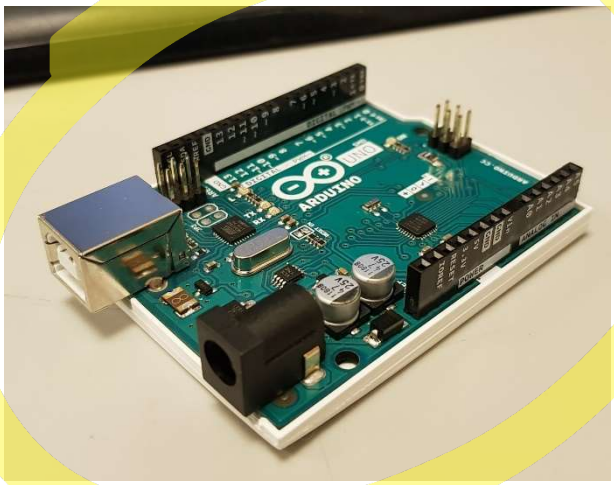
This activity will guide you through connecting an Arduino to your Raspberry Pi; uploading some code; and checking that the board is working. You should start by familiarising yourself with the Pi (power it on, etc). If you don't find using the Raspberry Pi's user interface intuitive, there is a help-sheet available on the Moodle page, which in turn contains links for further support. Also, feel free to ask a demonstrator!

What you will need:

- Arduino
- Base board
- USB cable
- Raspberry Pi

Mounting the Arduino on the base board (you will only need to do this if it isn't mounted already).

The Arduino needs to be clipped into its base board, which helps to protect the Arduino. To do this, take the Arduino and the base, sit the Arduino onto the base (observing the correct alignment), and push it down. There should be a little bit of resistance, but then the two parts should be firmly mated together.



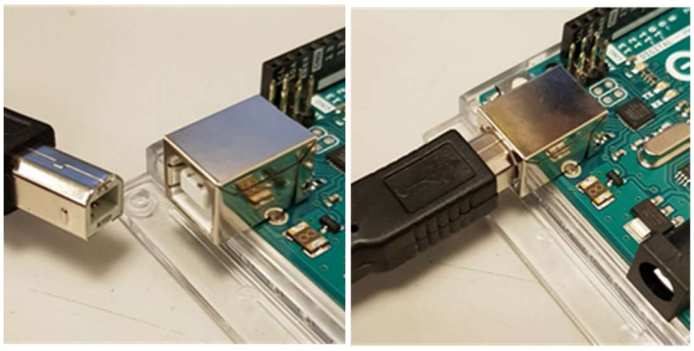
Downloading the IDE.

The Arduino is programmed using its own Integrated Development Environment (IDE). Lab computers (and Raspberry Pis) already have this installed (the app is simply called 'Arduino', or 'Arduino IDE'), but if you want to install it on your own computer, it's free, and can be downloaded from: https://www.arduino.cc/download

Connecting the Arduino.

The Arduino connects to a computer via a USB cable. This is the same type of cable used with desktop printers. The Arduino and Cable should be both provided on the bench.

1. Start by plugging the 'square' end of the cable into the Arduino.



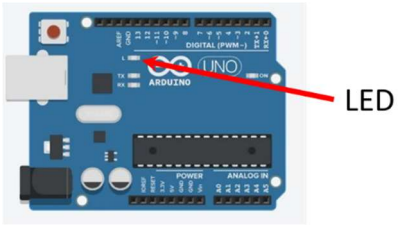
2. Then plug the other end of the USB cable into your computer. If the Arduino is correctly connected (and if your Pi is on), the red 'ON' LED will illuminate.

The IDE.

1. Run the "Arduino" application, from the "*Raspberry -> Programming"* menu. When opened for the first time, it should look something like this:

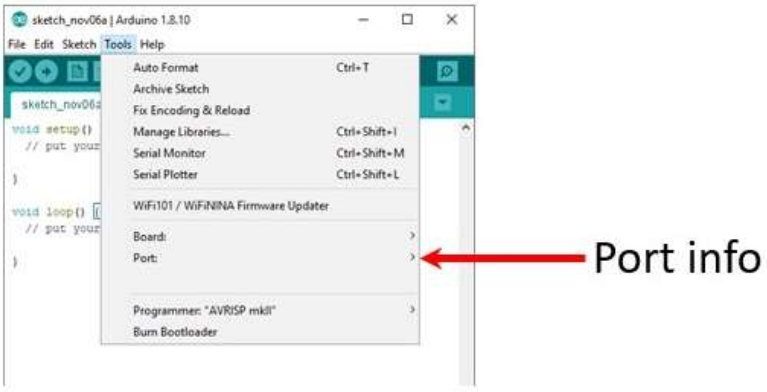
2. We're now going to check the Arduino is working. Traditionally, when learning a new language; using new hardware; or when debugging a system, programmers will start with some 'hello world' code. Often, this involves a simple print statement, to print something onto the screen (e.g. "Hello World"). This checks that the system is basically working.

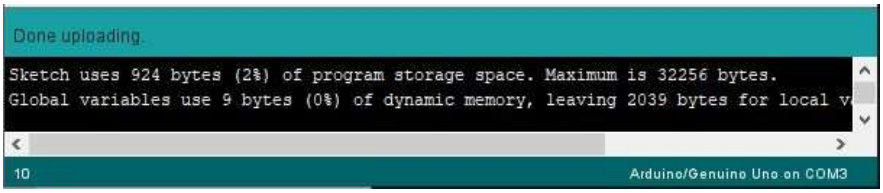
   With the Arduino, we don't have the option to print anything to the screen (at this point). However, the Arduino does have an LED built onto its board. We can blink this on and off as a simple "hello world".



3. There are many types of Arduino board, so we need to select the one we're using (Arduino Uno). In the IDE, navigate to the "*Tools*" menu and find the "*Board*" sub-menu. Mouse-over this submenu and select "*Arduino Uno*". A check mark should appear next to the selection. In future, you can use this to check that the correct board type is selected.
4. Also in the "*Tools*" menu, and find the *"Serial Port"* sub-menu. Mouse-over the submenu and find an entry which looks similar to: "*/dev/ttyACM0".* Select this entry. This is your Serial Port name, which you will need to refer to later. You may also see this referred to as a COM port.
   - The Arduino connects to a computer via USB, which is a serial connection. To handle multiple USB devices being used on a single computer, each device is given a serial port name/number, to identify it.

- Note: After certain events, the port number might automatically increment (the computer thinks it's a new USB device). For instance, you might see "/dev/ttyACM1", or higher. This is fine, but you need to be aware of it.

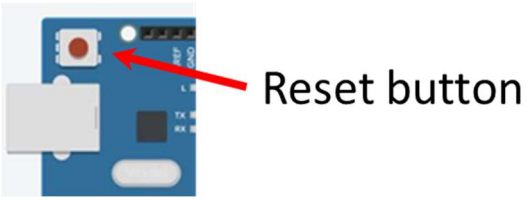


5. Next, we'll open some example code. We're not going to write any code here, but it's worth noting that the Arduino is programmed in the "C" language. You'll be using Python for most of the subsequent tutorials, so we won't be covering "C" in any detail.
6. Navigate to the "*File->Examples->01.Basics*" menu and select "Blink". Typically, a new IDE window will open, containing the example code you've just opened.
    - The top portion of code is commented and explains what this code does. Don't worry about the content of the code at this point.
7. Next, click the "Verify" button  to compile and check the code. In the bottom portion of the IDE window, you should see a progress bar.
    - When complete, the progress bar should be replaced with some text saying "Done compiling".
    - In addition, some white text should be visible in the black section underneath. This reports the amount of memory the program will use.
    - 

    - If there is an error, it will be reported here and will use a different text colour.
8. To upload the code, simply press the "Upload" button  to upload to the Arduino. Again, a progress bar should appear and a successful upload will result in the message "Done uploading". Again, white text will appear in the black section underneath.
9. If the upload was successful, you should see the LED now blinking on and off. By default, the code switches the LED on for 1 second and off for 1 second (a 50% duty cycle, with a period of 2 seconds).

Resetting the Arduino.

Sometimes, things don't work as expected and you might have to restart your hardware. The Arduino has a convenient reset button, located next to the USB socket. Press this, and you will notice the program stops running momentarily and the LED will flash more quickly.



When the Arduino has restarted, the code should start running again.

Arduino introduction pt2/2: Using Python

This activity will guide you through using Python to communicate with an Arduino

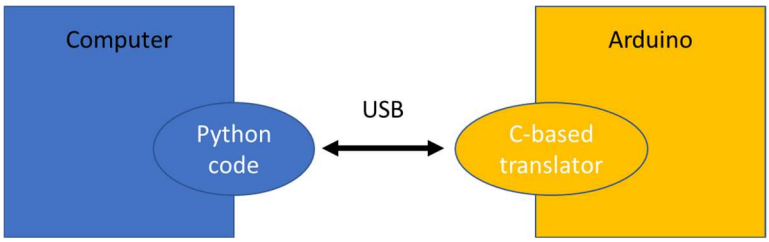
What you will need:

- Arduino
- USB cable
- Raspberry Pi.
- pushbutton, breadboard, wires, 1kΩ resistor

Introduction.

As mentioned in the previous introduction activity, the Arduino hardware is programmed using a variant of the "C" language, but we can also use Python to control it. We won't be running Python code directly on the Arduino hardware, but will instead run it on our computers. On the Arduino, some "C" code ("Firmata') will be running constantly, acting as a translator between our Python code, and the Arduino.

A simplified block diagram illustration of this is shown below:



… actually, though, as you may have noticed, this is a bit of an oversimplification. Please discuss if you want to know more!

The Arduino code.

The code which will run on the Arduino is called 'Firmata'. We need to open this in the IDE and upload it to the Arduino.

Usefully, Firmata is already provided in the IDE. It can be found under:

"File->Examples->Firmata->StandardFirmata".

Open this file. You'll notice it's a very long piece of code, but don't worry about it. Following the same connection and upload procedure from the 'Hello World' activity, upload the StandardFirmata code to your Arduino. Also: make note of the Serial Port name (e.g. '/dev/ttyACM0') for your Arduino (this can be found under *"Tools->Port").*

Using Python with Arduino.

To start, we're going to use Python to blink the on-board LED, similar to the 'Hello World' activity. We'll blink it at a period of 2 seconds (1 second ON, 1 second OFF).

1. Open your Python programming environment. This document will refer specifically to Jupyter Notebooks.
2. Open the first Jupyter Notebook for E0, ('Blink LED'). The following steps take you through how the code in that notebook is constructed (you do not need to type them):
3. First, we import the *"pyfirmata"* and *"time"* libraries, so we can make use of them in the code:

```
import pyfirmata
import time
```

4. Next, we define the Arduino board (note, the Serial Port name here needs to be the same as you previously observed in the Arduino IDE):

```
board = pyfirmata.Arduino('/dev/ttyACM0')
```

5. Then, we simply create a while loop to turn the LED on and off. There are some specific naming conventions within Firmata, for addressing the Arduino inputs and outputs.

```
# Start the While loop
while True:
# Switch the LED on.
# NOTE: board.digital[13] is the specific digital output for the on-board pin.
# NOTE: write(1) tells the Arduino to send a digital 1 to the pin.
# NOTE: 1 is high, which will switch the LED on.
```

board.digital[13].write(1)

 # Wait 1 second.

 # NOTE: time.sleep(s), where s is the time you want the code to sleep.

time.sleep(1)

#Switch the LED off.

board.digital[13].write(0)

 # Wait 1 second.

time.sleep(1)

6. The finished code (without the extra comments above) should look something like:

```
# -*- coding: utf-8 -*-
"""
PyFirmata: Blink the LED

"""

import pyfirmata
import time

# Define the board with the corrcect port
board = pyfirmata.Arduino('/dev/ttyACM0')

# While loop to switch the LED on and off.
while True:
    board.digital[13].write(1)
    time.sleep(1)

    board.digital[13].write(0)
    time.sleep(1)
```

Nb. You will find Jupyter notebooks on the Moodle page, there to help you with some of the activities. Please feel free download them, copy, paste and edit to suit your needs.

Running the code.

To run the file from within the Jupyter Notebook, simply click the *"Run"* button.

After running the code, you should see the LED repeatedly blink on and off.

You might also observe that the "RX" LED is also blinking. This LED is signifying that the Arduino hardware is receiving communication signals from the Python code on your computer.

Serial port issue.

When interrupting the code by using the 'stop' button (or other interrupts), PyFirmata fails to properly close the Serial port connection. This will cause any subsequent attempts to run code to fail, with an error.

A crude method for working around this issue is to restart the Kernel after the code is interrupted.

A more graceful and programmatical method is to include some additional lines in your code, to 'catch' the interrupt signal. By catching the interrupt, we can handle it in a more suitable manner. PyFirmata *does* include a method for closing the board connection, so we can use that.

Here is an example, with comments to help you understand what's going on (you should be able to find this Notebook on the Moodle page too.):

```
# -*- coding: utf-8 -*-
"""
PyFirmata: Arduino graceful close.

Gracefully close serial port.

"""

import pyfirmata
import time


""" *** Board configuration section *** """

board = pyfirmata.Arduino('/dev/ttyACM0')


""" *** Main code section *** """

# This is the main function, where the bulk of the code goes.

def main():
    print('running main')

    # blink the LED
    while True:
        board.digital[13].write(1)
        time.sleep(1)

        board.digital[13].write(0)
        time.sleep(1)


""" *** Error catching section *** """

# Firstly, check the code is running as __main__ (this is not related to the main() function, above)
if __name__ == "__main__":

    # Call the main() function
    try:
        main()
        #main() function will run...

    # EXCEPT if there's a KeyboardInterrupt exception
    # NOTE: KeyboardInterrupt is a broad term, and doesn't exclusively refer to the user's keyboard.
    # NOTE: Other exceptions can be caught here.  KeyboardInterrupt is only one example.
    except KeyboardInterrupt:

        print('code interrupted')

        # Use the PyFirmata method for closing the board connection.
        board.exit()
        print('board connection closed gracefully')
```

When you run this code, you should note that it will print a few statements. When the main() function is run, it will print:

```
running main
```

When you press the stop button, or send another interrupt, it will print:

```
code interrupted
```

And finally, when it has properly closed the board connection:

```
board connection closed gracefully
```

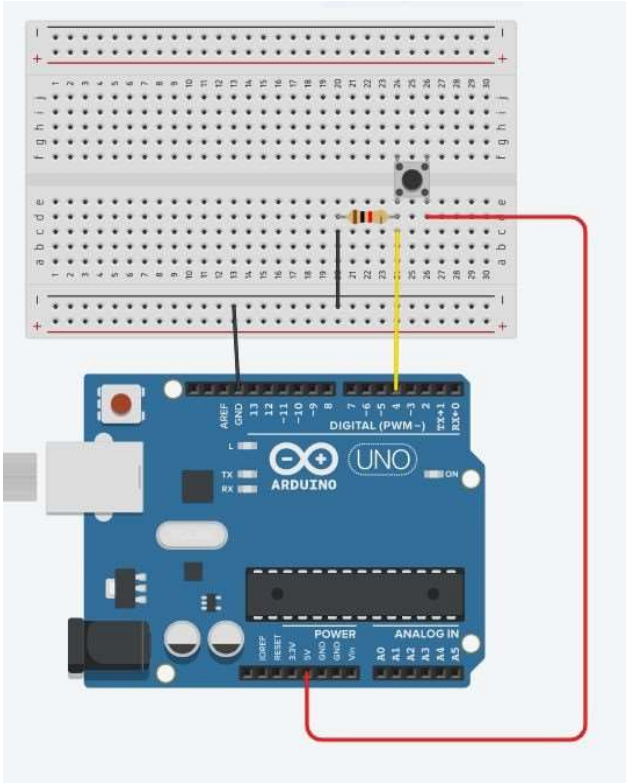
These print statements are included purely so you can observe what is happening. You can remove the print statements, if you like.

It's worth noting that you can use this method of handling interrupts to manage other issues that may occur in your Python code. For instance, you can handle specific errors in the same way.

Reading from an input.

Finally, we can read one of the Arduino's digital inputs and respond to it in some manner. A simple example of this is to wire a pushbutton switch to the Arduino, read the digital state of the switch (1 or 0), and flash the LED if the switch is pressed.

Wire up the circuit shown in fig 3:



The concept here is that when the pushbutton is pushed on, the voltage at input[4] will be +5V. When not switched, the voltage will be 0V. The resistor ensures there is not a short circuit between +5V and 0V when the pushbutton is on.

```
1  import pyfirmata
2  import time
3
4  board = pyfirmata.Arduino('/dev/ttyACM0')
5
6  # Start the firmata iterator
7  it = pyfirmata.util.Iterator(board)
8  it.start()
9
10 # define the input pin
11 board.digital[4].mode = pyfirmata.INPUT
12
13 while True:
14     sw = board.digital[4].read()
15
16     if sw is True:
17         board.digital[13].write(1)
18     else:
19         board.digital[13].write(0)
20     time.sleep(0.1)
```

The code required is slightly different from previous:

- Line 7 and 8 define and start the Firmata 'iterator', which is a loop used to read inputs.
- Line 11 changes the mode of pin 4 to be an INPUT (otherwise, the default is OUTPUT).
- Line 14 defines 'sw' as the read value from the input.
- Line 20 puts a brief pause in the code. This can be useful to avoid overloading the Arduino CPU.

If you run this code, the LED will come on any time you push (and hold) the button. As soon as you release the button, the LED should turn off. The switch being used here is a 'momentary' switch.

Extensions

These activities are here to consolidate your learning and are not strictly necessary for you to be able to get what you need out of subsequent practicals, although, as you can see, you will not need much time to give them a go! These are not in any order – so if you are limited for time and energy, just try the ones that interest you.

Arduino and Python SOS (15 min.)

Can you make the LED display an SOS code (dot dot dot, dash dash dash, dot dot dot etc.)? Just write a code in Python and run it! [Not really intended for this lab, but you could also think about how you could get python to read from a string (say, gotten from a .txt file, or one that you type in) and then express it in Morse code on the LED!]

Make your own potentiometer (15 min.)

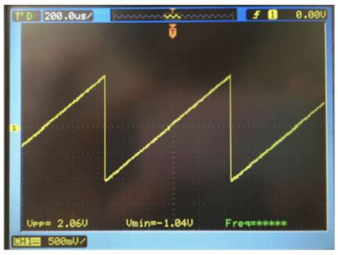
Get a pencil and lay down some carbon on paper. Measure the resistance between two points – try different distances. Try applying a voltage across the ends of your deposit and see whether or not you have written your own potentiometer, operating like the circuit at the end of Part I!

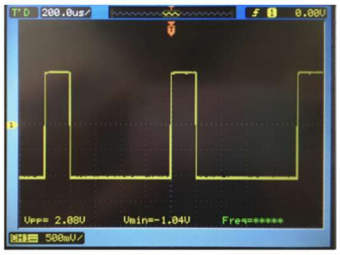
Speaker as a microphone (15 min.)

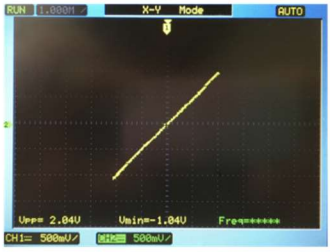
Grab a speaker (a diaphragm one) and connect it to an input of your scope and see if you can see a signal. Compare how your speaking voice compares with a whistle! You could also try making noise by connecting the speaker to a function generator, and then seeing if you can pick it up with a second speaker. Do the wave forms look the same? Do they do what you expect? You could, if you can find one with suitable connections, try a real microphone too.

Patterns on an oscilloscope (15 min., but open ended)

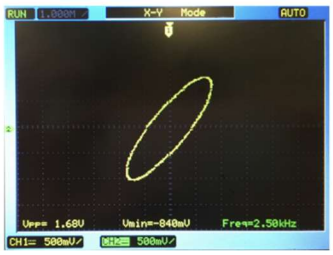
Here are some patters on an oscilloscope. Can you use equipment on your bench to generate these? Just get your scope to show qualitatively similar patterns!

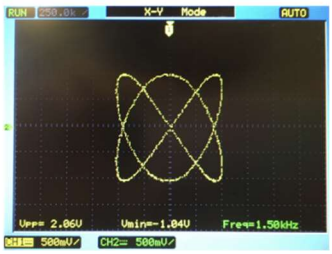
(a) 

(b) 

(c) 

(d) 

(e) 

(f) 

You might need to grab a capacitor for (e) and you might need to borrow something from another pair of students for (f). You might find some unexpected things happening when you attempt these – have a play around and try to work out what's going on! Can you find a really pretty pattern to impress your friends (or prospective students on an open day)?

Anything else you can think of (that we can quickly and reasonably accommodate) – there will be a box of standard components that are often used with Arduinos – feel free to try things out!

End of E0. KT/JM