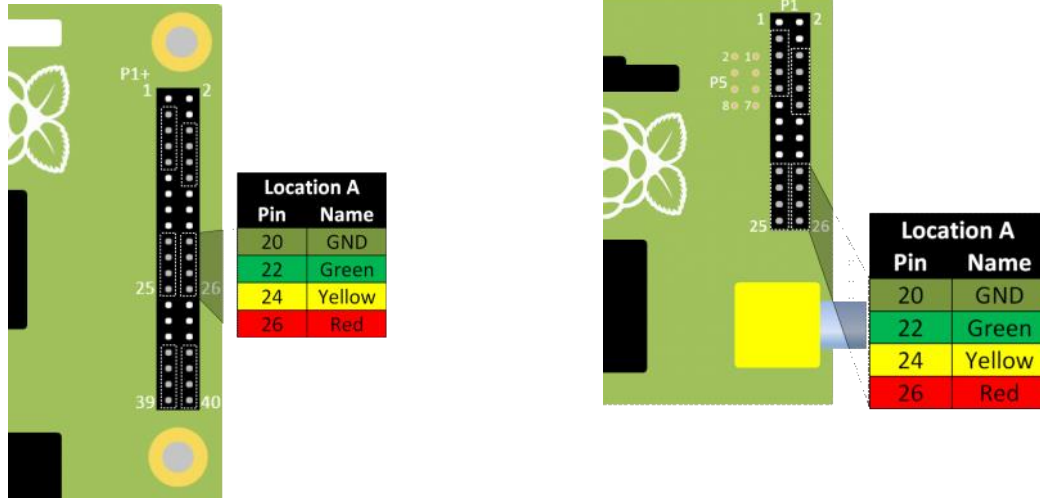


Worksheet 1 - Whereabout Clock Scanner

1 PiStop Setup

Attaching the hardware (position "A")

To begin with we shall use Location A



TIP: To find **Pi-Stop Location A** on the 40 pin GPIO header, ensure you count 7 pins directly up from the bottom (that will leave 7 unconnected pins below the Pi-Stop).

Your Pi-Stop should be facing towards the outside of the Raspberry Pi and inserted on the row of pins nearest the edge.

2 Starting the desktop and Python 3 (Idle)



NOTE:

If you are prompted to **login** the default is as follows:

Username is *pi*

Password is *raspberrypi*

To start the Raspberry Pi desktop (if you are currently in command-line mode) type the following command and press enter:

`startx`

For this workshop we will use the desktop and the Python 3 (Idle) editor. If you are more comfortable using another editor then feel free to use that.

Open the Terminal, locate the Terminal icon on the desktop, or through the Accessories menu.



Enter the following command:

`sudo idle3`



TIP:

For older installations on the Raspberry Pi you will have to start Idle3 (the editor) with a **sudo** command. This is because using the GPIO pins was **protected** and required **super user do** permissions, newer installations have relaxed this requirement (so you can often use the Idle3 link from the menu on your own setup).

3 Testing the PiStop

Open the terminal command line (as shown above).

Open folder using the following command:

```
cd /home/pi/Desktop/WhereaboutClock
```

Run the display.py code (which will perform a basic demo):

```
python3 -m pistop.display
```

Once you have confirmed the LEDs are flashing, you can continue with the worksheet.

4 Creating our scanner

On Idle3, use the File->Open... menu to locate and open the scanner01.py (in /home/pi/Desktop/WhereaboutClock):

```
#!/usr/bin/python3
from time import sleep
import pistop.display as DISP

SCAN_PERIOD = 10 #seconds

run = True
with DISP.display(hw_id=["A"]) as myDisplay:
    while(run):
        try:
            print("SCANNING...")
            ###Your code here###
            sleep(SCAN_PERIOD)
        except KeyboardInterrupt:
            # On keyboard interrupt signal stop running
            run = False
print("End")
```

First we can run this template code to see what it does.

Either run from the command line:

```
python3 scanner01.py
```

Or press F5 (if you have changed the file you will be prompted to save the file - select yes).

The code should run and we will see the following displayed on the screen.

```
SCANNING...
SCANNING...
SCANNING...
```

This should repeat every 10 seconds (as specified by the value of the "SCAN_PERIOD"), this will continue forever or until "run" is no longer "True".

You can stop the program by pressing ctrl+c, this will stop most python programs automatically. In this case we use a try...except block which allows us to detect when ctrl+c is pressed (which generates a KeyboardInterrupt) and set run = False. This will stop our "while" loop and let our program finish cleanly.



NOTE:

Any line starting with # is a comment and is ignored by the computer (you don't need to type them for the workshop, but it is often useful to include comments in your code to make it clear what you intend the code to do).

Spaces are very important in Python, each indent in the code represents a separate block of code. For example everything that is below and to the right of the while statement will run until you press ctrl+c. Then it will print "End" to the screen, since this is a separate block, this won't run until the "while" block has completed.

More about the code

The code also sets up the PiStop as a display called myDisplay. This is achieved by importing the module with the line:
`import pistop.display as DISP`

This allows us to use all the code contained within this module and refer to it with DISP in our code.

You can look at the code this contains by opening the file:
`/home/pi/Desktop/WhereaboutClock/pistop/display.py`

Before we start our loop we create "myDisplay" which sets our PiStop on hardware position "A" on the Raspberry Pi. The "myDisplay" object allows us to control one or more PiStop lights and use them as our Whereabout Clock display. By using the special "with" block, it allows us to use the "myDisplay" object within the code block and then clean up automatically once we are outside of this block (switching off LEDs etc.).

5 Display scanning activity

Since we want our Whereabout Clock to work without a screen, we will want it to show some activity while it performs the scans of the network. To this we can use the demoLine() function that is built into the PiStop display module.

Where it says `###Your code here###`, add the following on the next lines:



```
###Your code here###  
myDisplay.demoLine() <-- Add new the code to your file here.
```

1a



TIP: Be sure to check that you have the same number of spaces before each line so that they are in-line with the other statements.

Run you code (again by pressing F5 and save, or via the command line):
`python3 scanner01.py`

Press ctrl+c to stop it again.

6 Performing an ARP scan

As discussed in the first worksheet, ARP allows us to get a list of all the devices and computers on our network. For the purposes of this workshop we will use a module that performs this scan and returns the results to us in a structured format.

Add a new import (near the top of the file) to import the arp_scan module:



```
from time import sleep  
import pistop.display as DISP  
from arp_scan.arp_scan import ArpScanLinux as SCAN
```

Update your code to include the new lines:



```
###Your code here###  
response = SCAN.scanMOCK()  
print(response)  
myDisplay.demoLine()
```

1b

Run you code (again by pressing F5 and save, or via the command line):
`python3 scanner01.py`

Press ctrl+c to stop it again.

We will see the something similar displayed on the screen.

```
SCANNING...  
SIMULATION  
Interface: eth0, datalink type: EN10MB (Ethernet)
```

```
Starting arp-scan 1.7 with 256 hosts (http:\\webpage.com)
192.168.0.13    40:b4:cd:16:61:16    Device Manufacturer Info
192.168.0.254  ac:bd:ce:df:e1:f2    Device Manufacturer Info
192.168.0.6    a0:8e:78:2e:2e:e2    Device Manufacturer Info
192.168.0.8    ec:9b:f3:7a:7a:7a    Device Manufacturer Info
192.168.0.11   84:b5:41:00:88:66    Device Manufacturer Info
192.168.0.7    00:08:22:bf:ba:b3    Device Manufacturer Info
```

8 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.7: 256 hosts scanned in 3.11 seconds (123 hosts/sec). 31 responded

Next we can send this output to be processed into something we can use more easily in python.

Update your code to include the new line. We can remove the print line now we have seen the output of the scan() command:



```
###Your code here###
response = SCAN.scanMOCK()
seen = SCAN.parseResponse(response)
print(seen)
myDisplay.demoLine()
```

1c

Run you code (again by pressing F5 and save, or via the command line):
python3 scanner01.py

Press ctrl+c to stop it again.

The program will produce a result similar to the following:

```
SCANNING...
SIMULATION
[['192.168.0.7', '00:08:22:bf:ba:b3'], ['192.168.0.10', 'fc:a6:67:44:22:11'], ['192.168.0.2',
'6c:72:20:05:54:45'], ['192.168.0.5', 'f8:3d:ff:43:ee:34'], ['192.168.0.1', '00:03:78:ad:ba:ca'],
['192.168.0.11', '84:b5:41:00:88:66']]
```

This will perform an ARP scan of the network and return a list of the IP address and the associated MAC address seen on the network in an array format.



NOTE: Since your Raspberry Pi may not have a network connection the arp-scan module allows us to simulate an ARP scan and will return "MOCK" results (which will match the devices we are looking for).

To use the real ARP scanner at home, install ARP using:

```
sudo apt-get install arp-scan
```

Then replace the function:

```
response = SCAN.scanMOCK()
```

With:

```
response = SCAN.scan()
```

This module also supports performing (and simulating) ARP scans on windows systems too (just replace ArpScanLinux with ArpScanWin in the import line).



IMPORTANT: You should only perform ARP scans on networks that you own or have permission to do so. Recording details of devices on a network without permission could be considered a breach of privacy and may otherwise be unlawful.

You can see the code for this module in this file:

```
/home/pi/Desktop/WhereaboutClock/arp_scan/arp_scan.py
```

7 Checking for devices

We now have a list of devices we have found on our network, we can now check if we recognise any of them as the devices of our family.

i.e. Have they connected to our network and therefore currently at home!

In order to do this we have to prepare a list of known devices and how we want to classify them, this has been done in the file: /home/pi/Desktop/WhereaboutClock/devices.py

```
# Edit the file for the people/devices you want to track
class ident():
    #-1=No Display
    IGNORE    = -1
    Home      = -1
    Me        = 0
    John      = 1
    Jane      = 2
    WARN      = 5

home = [
    {"name":"TV",          "mac":"00:03:78:ad:ba:ca",      "ident":ident.Home},
    ... ..
    {"name":"Bob Tablet",  "mac":"84:b5:41:00:88:66",      "ident":ident.IGNORE},
    {"name":"Jane Tablet", "mac":"34:d2:70:36:63:36",      "ident":ident.Jane},
    {"name":"Home Gateway", "mac":"ac:bd:ce:df:e1:f2",      "ident":ident.Home}
]
```

For each device we are interested in we can add a line, which details the device name, MAC address and the identifier (this format is known as a python dictionary which allows us to refer to elements by name, ident, mac etc.).

This means we can allocate several devices to a particular person and therefore to a particular position on our Whereabout Clock display. The file has been setup to match the simulated data from our previous ARP scans, however feel free to change the identifier names (me, John, Jane etc.) to suit your family. Ensure you leave the MAC address the same as these will correspond to the simulated ARP scan output.



NOTE: You can find your own MAC addresses for your own devices and update this file at home.

Add a new import (near the top of the file) to import the devices file. Again we can remove print line from before now we have seen the output:



```
from time import sleep
import piston.display as DISP
from arp_scan.arp_scan import ArpScanLinux as SCAN
import devices
```

Add the following to your code:



```
###Your code here###
response = SCAN.scanMOCK()
seen = SCAN.parseResponse(response)
myDisplay.demoLine()
# Check if seen items are expected
for itemseen in seen:
    displayDevices(itemseen)
```

Run you code (again by pressing F5 and save, or via the command line):
python3 scanner01.py

Press ctrl+c to stop it again.

The program output will be similar to below:

```
SCANNING...
SIMULATION
['192.168.0.7', '00:08:22:bf:ba:b3']
['192.168.0.11', '84:b5:41:00:88:66']
['192.168.0.3', 'b8:27:eb:44:0d:55']
['192.168.0.11', '84:b5:41:00:88:66']
['192.168.0.8', 'ec:9b:f3:7a:7a:7a']
['192.168.0.11', '84:b5:41:00:88:66']
#0 00:08:22:bf:ba:b3 192.168.0.7 (My Phone)
#-1 84:b5:41:00:88:66 192.168.0.11 (Bob Tablet)
#-1 b8:27:eb:44:0d:55 192.168.0.3 (DVD Player)
#-1 ec:9b:f3:7a:7a:7a 192.168.0.8 (Home Printer)
```

About the code

The new code allows us to step through our list of "seen" devices one by one. Each device in the list is set to "itemseen", which is an array that contains the MAC address and the IP address of the seen device. We process each "itemseen" in the list until they have all been checked.

The added code will call the following function to display information about the device found:

```
def displayDevices(itemseen):
    '''This function allows the devices seen to be displayed'''
    for item in devices.home:
        if item["mac"] in itemseen[SCAN.MAC]:
            mac = itemseen[SCAN.MAC]
            ip = itemseen[SCAN.IP]
            name = item["name"]
            ident = item["ident"]
            # Display active devices
            print("#{:<2} {:<17}\t{:15}\t({})".format(ident, mac, ip, name))
    return
```

We check each "item" in our "devices" list (the ones defined in our devices.py file of our family devices). If we find the item's "mac" address matches the mac address of the "itemseen", then we have found one of our family devices on the network.

We set the "name" and "ident" to equal the found device, so this information can be displayed.

Finally, we can display the devices we've found using the format command. The format command will place each of the values listed into the positions marked with {}. The {:<17} allows us to specify this value should allow 17 spaces justified to the left, and {:15} will allow 15 spaces for the value justified to the right. The \t will create a tab character which will further aid with displaying the information in a neat format.


i.e.

```
["192.168.0.7", "00:08:22:bf:ba:b3"]
Becomes:
#0 00:08:22:bf:ba:b3 192.168.0.7 (My Phone)
```

8 Display on the PiStop

The final step for this worksheet is to display the result of our scan on our PiStop display. You will notice that each of our devices in the `devices.py` file have a particular "ident" value assigned to them. This value can be used to specify which light on the PiStop should light up for a particular person. For instance "Jane" is given `ident.John = 1`, which will light up the second LED on the PiStop.

```
2  Me    ---
1  John  ---
0  Jane  ---
```

A small icon of the PiStop display, which is a vertical rectangular device with three circular LEDs arranged vertically. The top LED is currently lit.

We also have a special ident value of -1, which won't light up an LED, allowing us to set certain devices to be ignored in our display. For example, we probably don't want to know if our TV or printer are at home or not...hopefully they are always in!

To use our display we can add the following line to the `displayDevices()` function:



```
# Display active devices
print("#{<2} {<17}\t{<15}\t({})".format(ident, mac, ip, name))
myDisplay.pos(ident, DISP.PS.ON)
```

1e

Run your code (again by pressing F5 and save, or via the command line):
`python3 scanner01.py`

Press `ctrl+c` to stop it again.

We should now see each scan occurring on the display and then displaying the detected devices via the LEDs showing who is at home!

About the code

The `myDisplay` code allows us to reference each of the PiStop LEDs by numbers 0 to 2 and specify which LEDs we want to change (i.e. the `ident` value) and the new state (`ON = DISP.PS.ON`). If we send a negative number it will be ignored, so if `ident=-1`, we won't display any devices marked as `ident.IGNORE`.

Normally we would want to switch off the LED next time if the devices were no longer present, however our `demoLine()` function from before will clear any previously lit LEDs once it has re-scanned.