# Lesson 6 –
# Simple web server using Python

- S.P. Chong

# Objectives

- In this lesson, you will learn to install & use a web development package called “Flask”, and create a simple web server to run in a Raspberry Pi, using Python.

- You will then learn how monitoring & control can be done via a LAN (Local Area Network).

- Finally, you will learn some networking concepts such as “DHCP reservation” & “port forwarding”, which will allow monitoring & control via internet / WAN (Wide Area Network).

# Installing & using Flask

- Flask is a web development package for Python, that is easy to set up and use.


You can refer to this website for more info.

http://flask.pocoo.org/

*overview* // *docs* // *community* // *extensions* // *donate*

*Flask is a microframework for Python based on Werkzeug, Jinja 2 and good intentions. And before you ask: It's BSD licensed!*

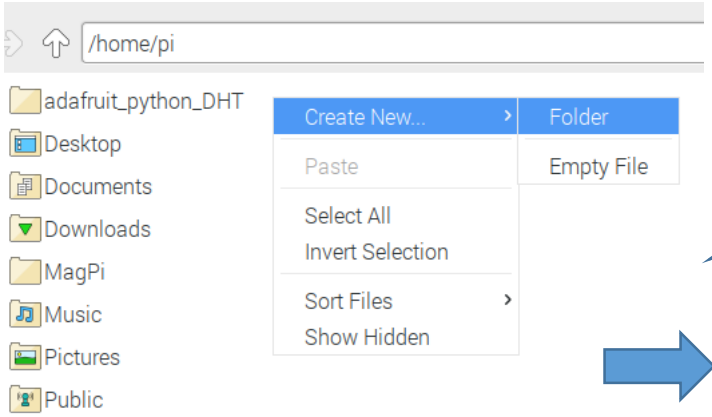
If it is not already installed in your RPi, use this command to install flask, .

sudo apt-get install python3-flask

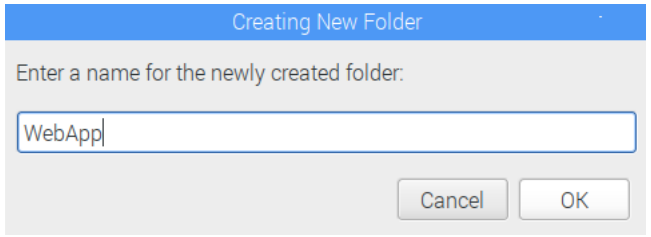
# Installing & using Flask (cont.)

/home/pi/WebApp (folder)
- app.py (file)
- static (folder)
  - style.css (file)
- templates (folder)
  - index.html (file)
  - xxxx.html (file)
  - yyyy.html (file)

- Next, create some folders & files in the SD card, as shown here.
- For now, create an empty folder called WebApp in the pi folder:




Right click, & select Create New -> Folder.




Type in the folder name & click OK.

Similarly to create a new Empty File.

# Installing & using Flask (cont.)

- After that, create the “app.py” python file, using the Python IDLE:

app.py - /home/pi/WebApp/app.py (3.5.3)

File Edit Format Run Options Window Help

```
from flask import Flask
app=Flask(__name__)

@app.route('/')
def index():
    return "Hello World!"

if __name__=="__main__":
    app.run(debug=True,host='0.0.0.0') #0.0.0.0 => accessible to any device on the network
```

Note that both “ “ & ‘ ‘ can be used to enclose a string.

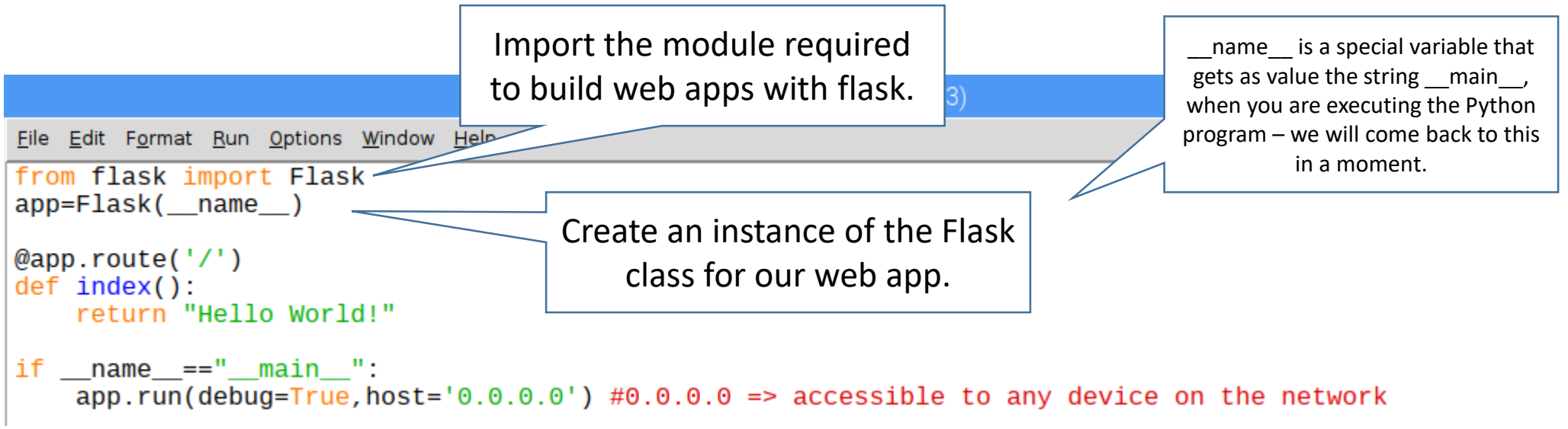
The codes will be explained next...

You can also read up this link to learn more:

http://flask.pocoo.org/docs/0.12/

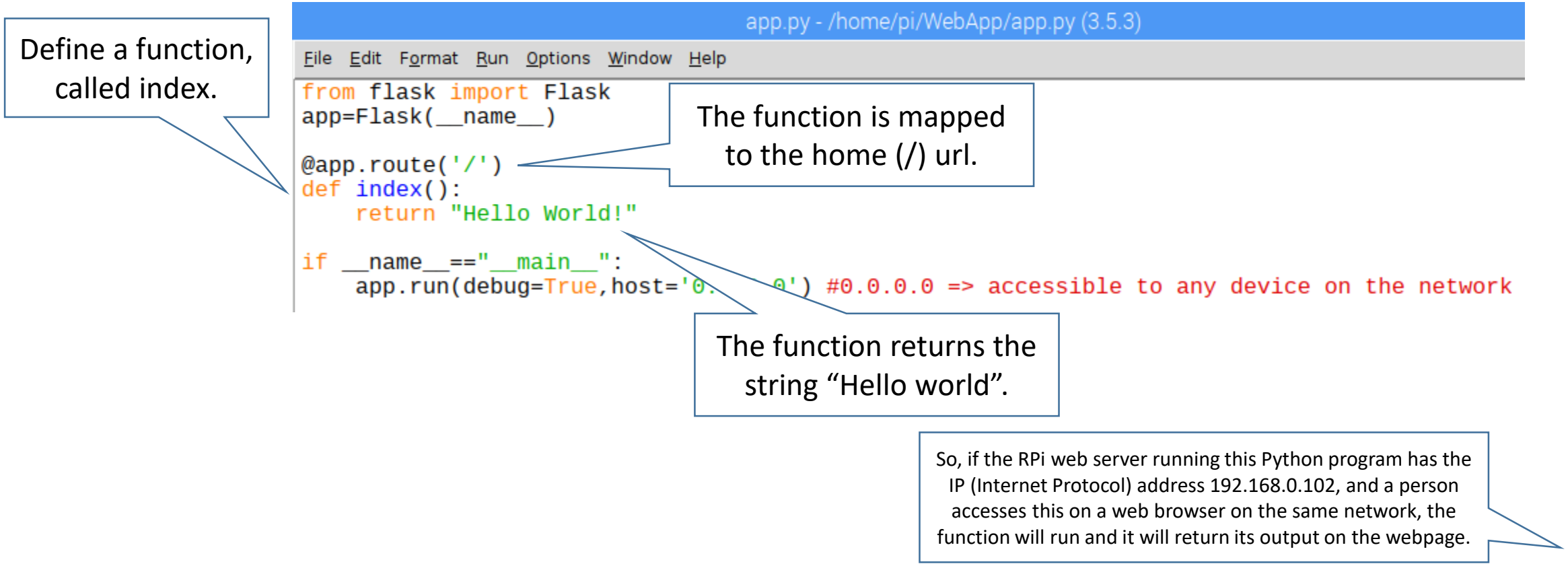
# Installing & using Flask (cont.)

- Some explanation below will be “touch n go”, because it is not a simple topic:



# Installing & using Flask (cont.)

- Here, we show how a function is mapped to the home url. We will see how to map to other url/route in a while.



# Installing & using Flask (cont.)

app.py - /home/pi/WebApp/app.py (3.5.3)

File Edit Format Run Options Window Help

```
from flask import Flask
app=Flask(__name__)

@app.route('/')
def index():
    return "Hello World!"

if __name__=="__main__":
    app.run(debug=True,host='0.0.0.0') #0.0.0.0 => accessible to any device on the network
```

If you are executing the file (by typing "python app.py" in a terminal, or by pressing F5 in Python IDLE), __name__ will be equal to __main__

Note that by setting host to 0.0.0.0, this web server is accessible to any device in the same network.

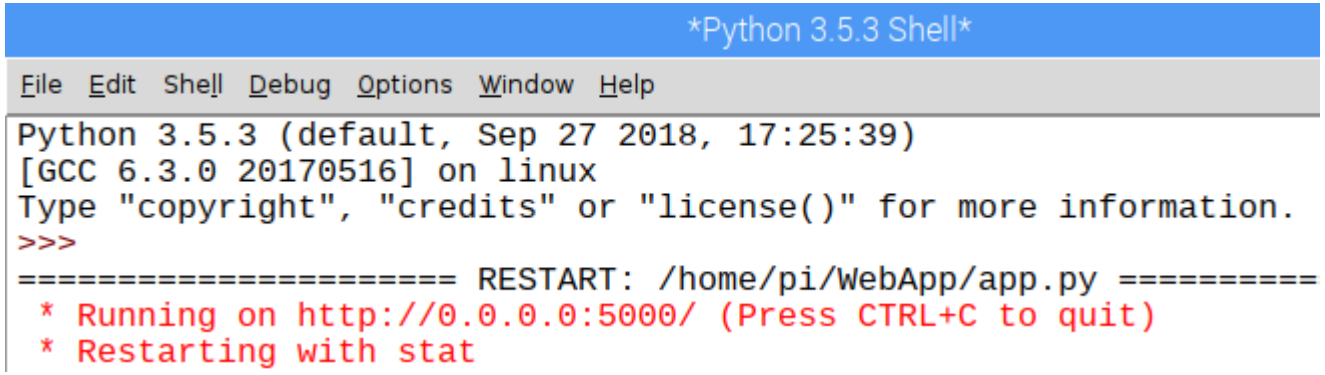
So, app.run will execute, with debug equal to true - this will print out possible errors on the web page, helping us trace the errors.

If the "app.py" file is imported by another Python file, __name__ will be equal to "app.py" – we are not doing this.

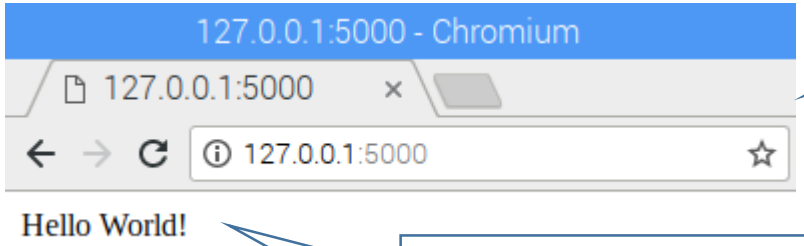
When everything has been tested, you may want to set debug to false.

# Installing & using Flask (cont.)

- Let's try out this simple web server:




Press F5 to run the program.




Use the Chromium browser, type 127.0.0.1:5000 to access the web server

The web server returns the string "Hello world" to the web browser.

Note: if you know the IP address of this RPi (e.g. 192.168.0.102), you can use another end device in the same LAN, type the IP address & port number in a browser (192.168.0.102:5000), and get the same result.

# Installing & using Flask (cont.)

- Let's see how more pages can be added to the "web site":

app.py - /home/pi/WebApp/app.py (3.5.3)

File Edit Format Run Options Window Help

```
from flask import Flask
app=Flask(__name__)

@app.route('/')
def index():
    return "Hello World!"

@app.route('/page2')
def page2():
    return "Second page"

if __name__=="__main__":
    app.run(debug=True,host='0.0.0.0') #0.0.0.0 => accessible to any device
```

Add the route '/page2'

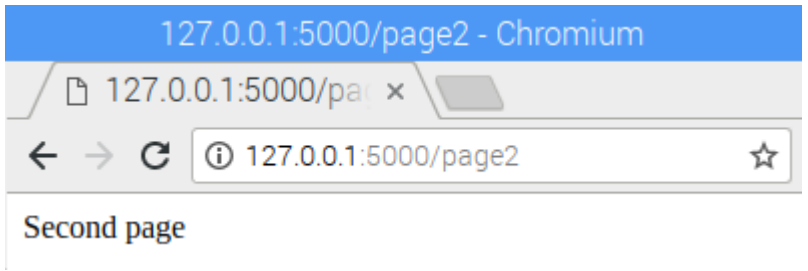
Define the associated function 'page2'.

You may need to issue the command fuser –k 5000/tcp before running the program again.

pi@raspberrypi: ~

File Edit Tabs Help

```
pi@raspberrypi:~ $ fuser -k 5000/tcp
```




Run this program again, browse to …/page2 and see what you get.

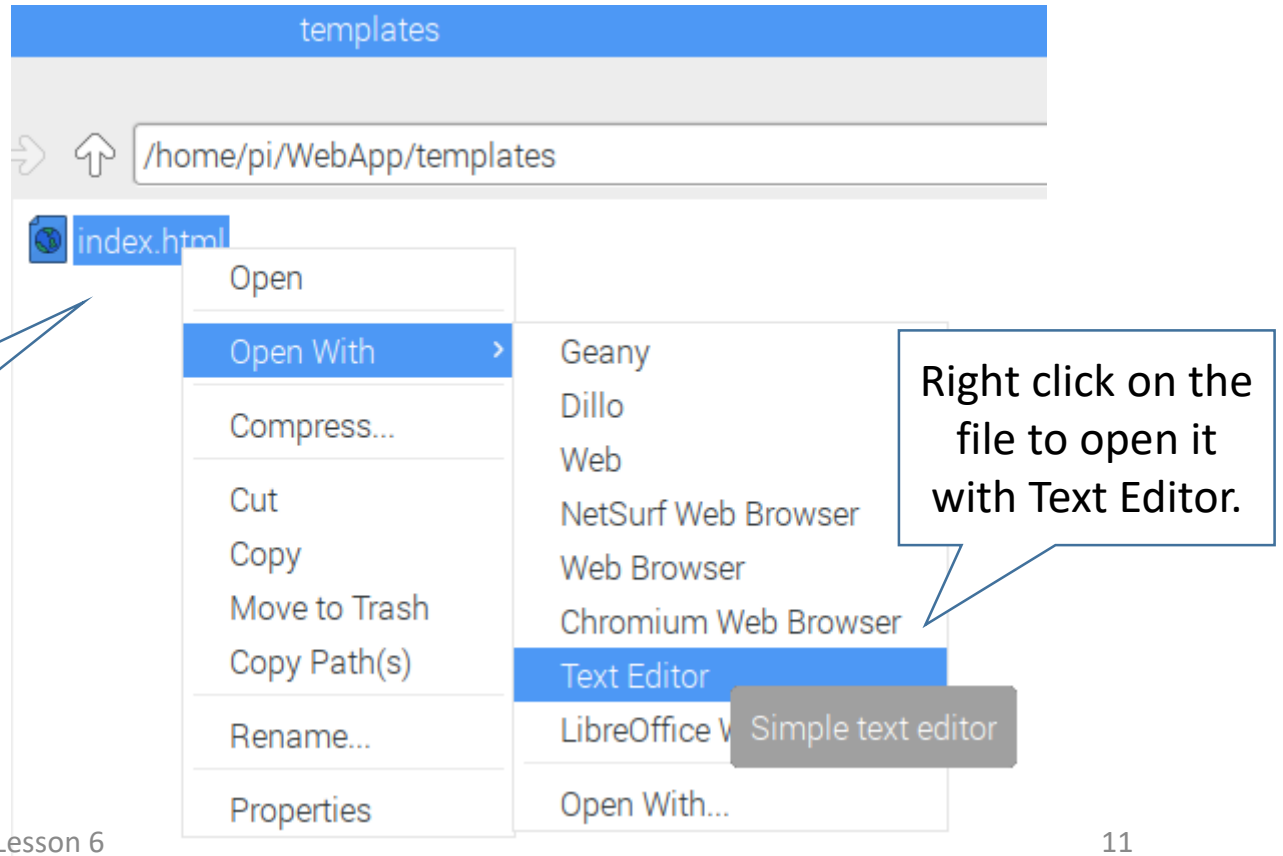
# Rendering template

- Instead of using Python code to produce the contents for the web pages, let's see how a HTML (Hypertext Markup Language) file can be used instead.

/home/pi/WebApp (folder)
app.py (file)
static (folder)
style.css (file)
templates (folder)
index.html (file)
xxxx.html (file)
yyyy.html (file)

Create a folder called "templates" & an empty file called "index.html" in the folder.

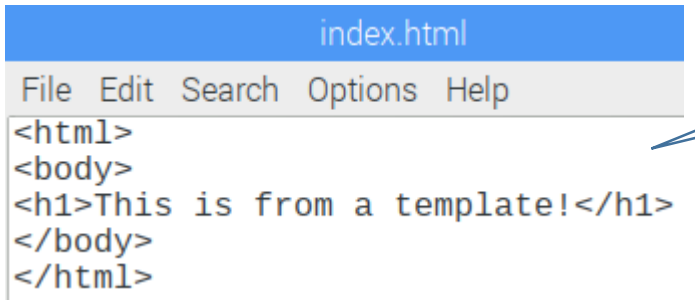
You should have something like this.



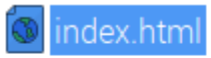

Right click on the file to open it with Text Editor.

# Rendering template (cont.)

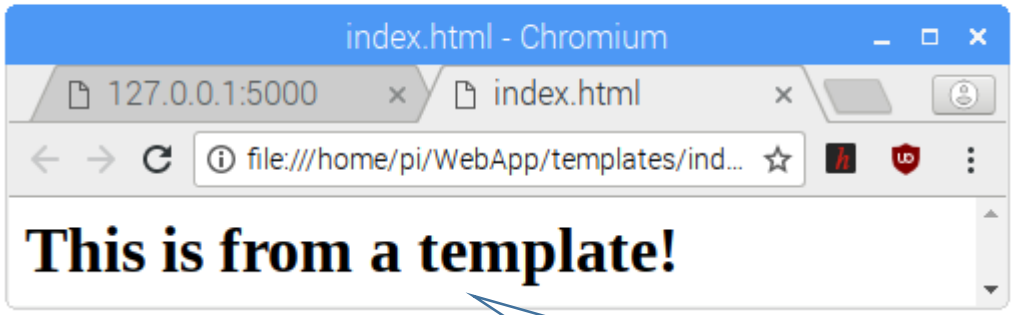
- The approach is to put the contents to be displayed in a HTML file and then using Python / Flask to “render” that file.




Type in these lines.




This HTML file, if opened with a web browser, will display a line of text.




You can learn more about HTML from these websites.

https://www.codecademy.com/en/tracks/web

https://www.w3schools.com/html/default.asp

The h1 tags will make the text bold, in large font size.

# Rendering template (cont.)

- This is how Python/Flask can “render” a HTML file:

app.py - /home/pi/WebApp/app.py (3.5.3)

File Edit Format Run Options Window Help

```
from flask import Flask
from flask import render_template
app=Flask(__name__)

@app.route('/')
def index():
    return render_template("index.html")

@app.route('/page2')
def page2():
    return "Second page"

if __name__=="__main__":
    app.run(debug=True,host='0.0.0.0') #0.0.0.0 => accessible to any device on the network
```

So, if you have many web pages for your web site, you can organise them as folders & HTML files, and add the Python codes to render them.


If you run this program, and use a browser to browse to 127.0.0.1:5000 (IP address:port number) as before, you should see the line of text, bold & in large font size.

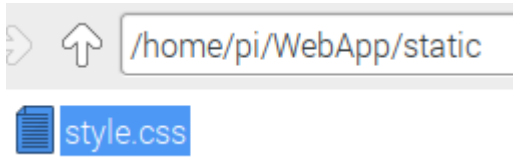
# Cascading Style Sheet

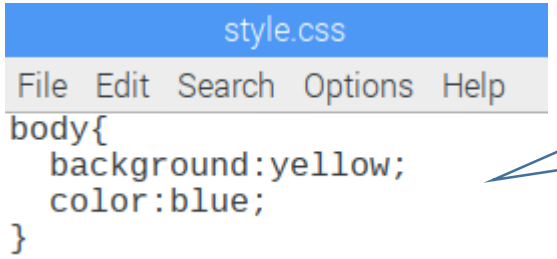
- CSS (Cascading Style Sheet) can be used to change the way a website looks – text colour, background colour etc.

Create a folder called "static" & an empty file called "style.css" in the folder.

/home/pi/WebApp (folder)
app.py (file)
static (folder)
style.css (file)
templates (folder)
index.html (file)
xxxx.html (file)
yyyy.html (file)

You should have something like this.




Right click on the file to open it with Text Editor, and change it to this.

Again, you can learn more about CSS from W3school.com

# Cascading Style Sheet (cont.)

index.html

File Edit Search Options Help

```
<html>
<head>
  <link rel="stylesheet" href='/static/style.css'/>
</head>
<body>
  <h1>This is from a template!</h1>
</body>
</html>
```

Back to the HTML file, add in a link to the static folder & the CSS file.


When you run the python program, and use a browser to browse the web server, you will see the styling.

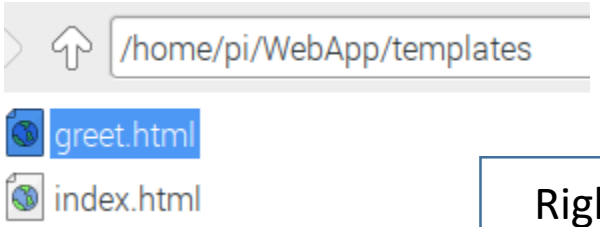
# Dynamic contents (passing parameters)

- It is possible to pass “parameters” from a web browser to a web server, so that “dynamic contents” can be created.

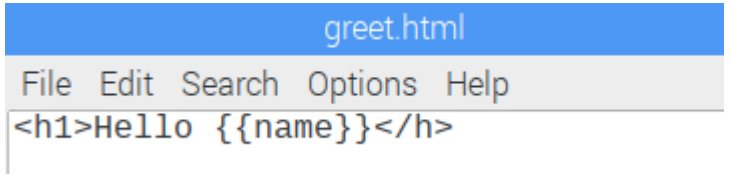
Create an empty file called “greet.html” in the “templates” folder.

/home/pi/WebApp (folder)
- app.py (file)
- static (folder)
  - style.css (file)
- templates (folder)
  - index.html (file)
  - greet.html (file)
  - yyyy.html (file)

You should have something like this.




Right click on the file to open it with Text Editor, and change it to this.




Note that in this html file, a parameter ‘name’ is passed in – more on this later.

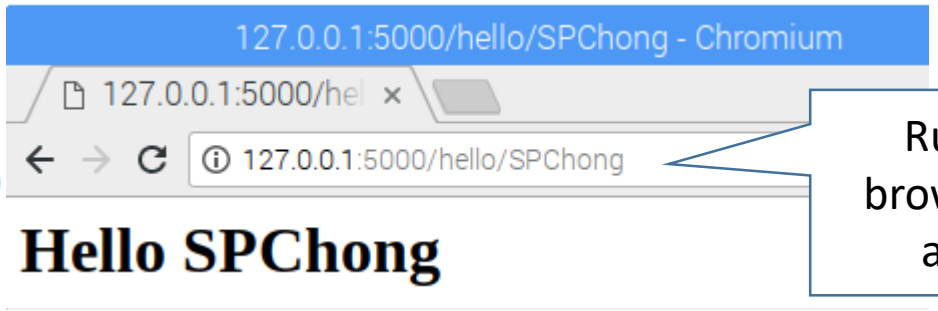
# Dynamic contents (cont.)

app.py - /home/pi/WebApp/app.py (3.5.3)

File Edit Format Run Options Window Help

```
from flask import Flask
from flask import render_template
app=Flask(__name__)

@app.route('/')
def index():
    return render_template("index.html")

@app.route('/hello/<who>')
def hello(who):
    return render_template("greet.html",name=who)

@app.route('/page2')
def page2():
    return "Second page"

if __name__=="__main__":
    app.run(debug=True,host='0.0.0.0')
```

Back to the app.py file, add the route '/hello/<who>' and define the associated function 'hello', which takes in the parameter 'who'.




Run this program again, browse to …/hello/SPChong and see what you get.

What exactly happens?

# Dynamic contents (cont.)

app.py - /home/pi/WebApp/app.py

File Edit Format Run Options Window Help

```
from flask import Flask
from flask import render_template
app=Flask(__name__)

@app.route('/')
def index():
    return render_template("index.html")

@app.route('/hello/<who>')
def hello(who):
    return render_template("greet.html",name=who)

@app.route('/page2')
def page2():
    return "Second page"

if __name__=="__main__":
    app.run(debug=True,host='0.0.0.0') #0.0.0.0 => acces
```


1. When you browse to …/hello/SPChong…

2. This route is taken, and 'who' takes the value 'SPChong'…

3. This function is executed, and 'name' takes the value of 'who', which is 'SPChong'…

4. The function renders the template file 'greet.html', and with 'SPChong' as the name...

5. So, the web browser will show 'Hello SPChong'.

greet.html

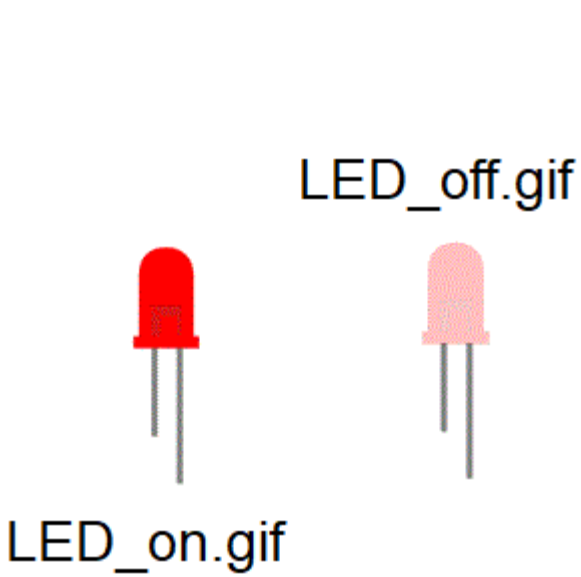
File Edit Search Options Help

```
<h1>Hello {{name}}</h>
```

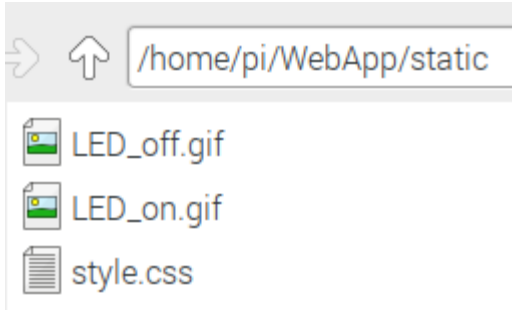
The "parameter" (SPChong) is passed from the web browser's URL (1), to the web server's webapp.py (2, 3), to the web server's greet.html (4), & finally to the web server's web page display (5).

# Adding images & links

- Let's first see how images can be used in web pages, and how web pages can be linked together.



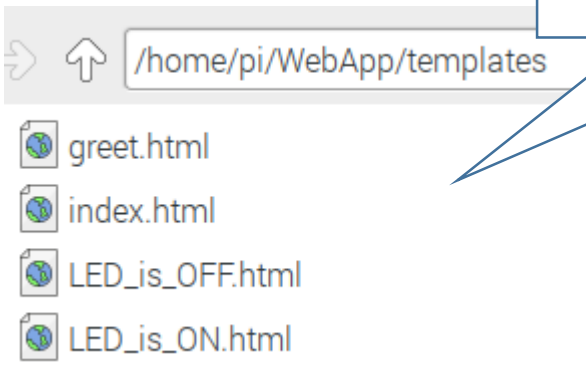

Create 2 images as shown, each around 64 x 128 pixels.




Save them as LED_on.gif & LED_off.gif, in the WebApp/static folder.

# Adding images & links (cont.)

Create 2 empty files called "LED_is_OFF.html" & "LED_is_ON.html", in the "templates" folder.




Use a text editor to change the files to the following:

LED_is_OFF.html

File Edit Search Options Help

```
<img src="/static/LED_off.gif" alt="LED off gif file">
<p>
<form action="/LED_is_ON">
  <button type="submit">Turn ON LED</button>
</form>
```

This line will insert the image from the static folder into the web page.

These lines will create a form, with a submit button ("Turn ON LED"), which when clicked, will lead to an action (i.e. going to the "LED_is_ON" page).

LED_is_ON.html

File Edit Search Options Help

```
<img src="/static/LED_on.gif" alt="LED on gif file">
<p>
<form action="/LED_is_OFF">
  <button type="submit">Turn OFF LED</button>
</form>
```

Likewise for the "LED_is_ON" page.

W3school.com is a good place to learn html.

# Adding images & links (cont.)

app.py - /home/pi/WebApp/app.py (3.5.3)

File Edit Format Run Options Window Help

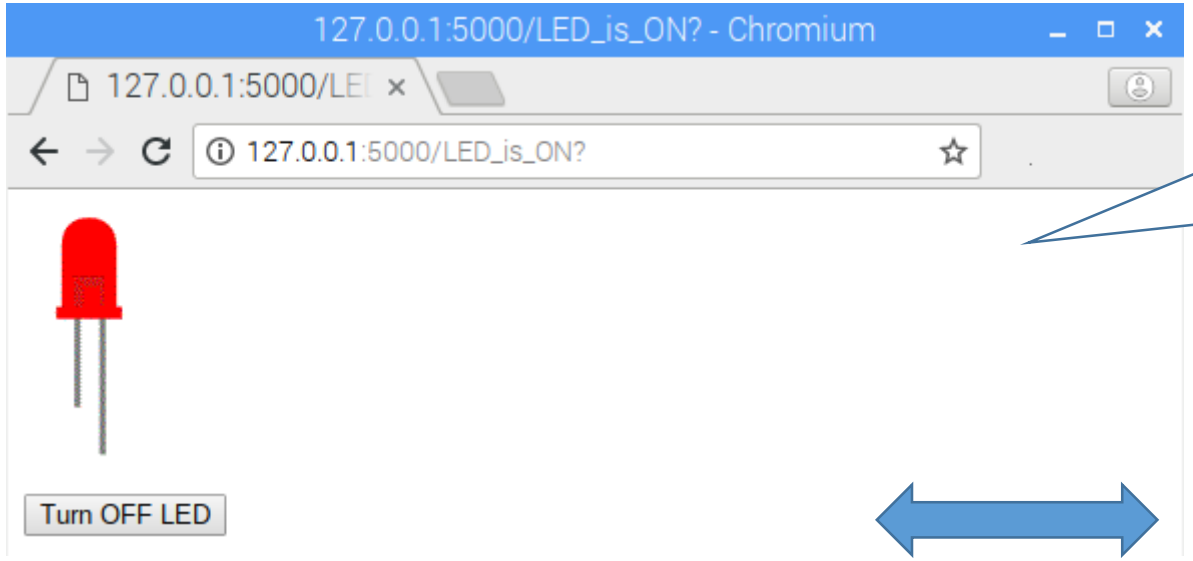
```
from flask import Flask
from flask import render_template
app=Flask(__name__)

@app.route('/LED_is_ON')
def show_LED_is_ON ():
    return render_template("LED_is_ON.html")

@app.route('/LED_is_OFF')
def show_LED_is_OFF ():
    return render_template("LED_is_OFF.html")

#other "web pages" not shown

if __name__=="__main__":
    app.run(debug=True,host='0.0.0.0') #0.0.0.0 =      essible to any device on the network
```

The routes & associated functions must be added to the app.py, which acts as the web server code.

When a user browses to …:5000/LED_is_ON, the "LED_is_ON.html" page will be served.

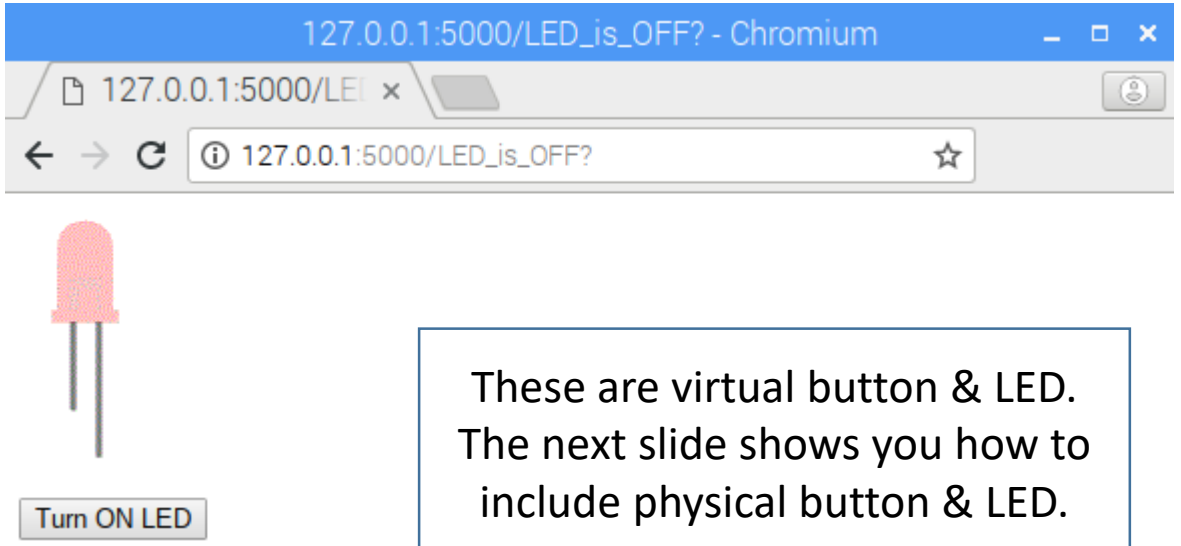
Likewise for the "LED_is_OFF".

# Adding images & links (cont.)




As expected, in the “LED_is_ON” page, the LED is lit, and a button is made available to turn off the LED.

Clicking the buttons allows a transition between the 2 pages.



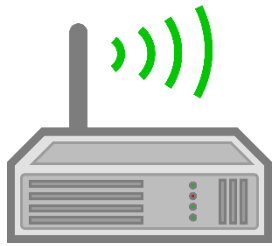

These are virtual button & LED. The next slide shows you how to include physical button & LED.

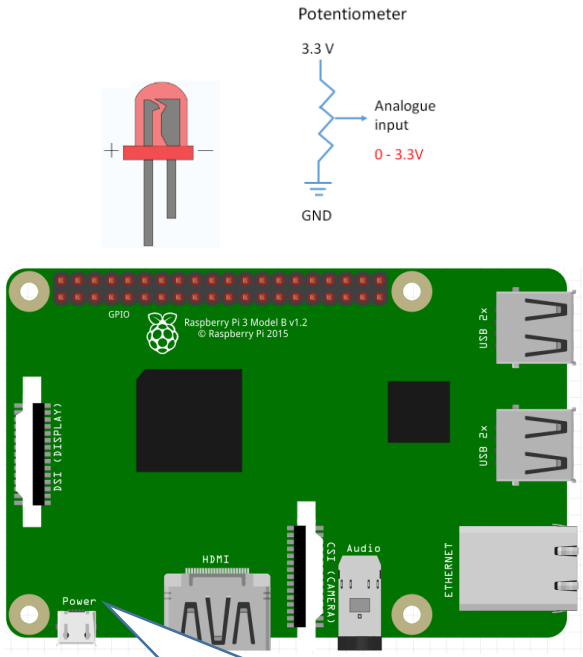
# Monitoring & control thru. network

- Next, you will learn to use python (with Flask) together with HTML codes to remotely monitor a potentiometer, and to control an LED.

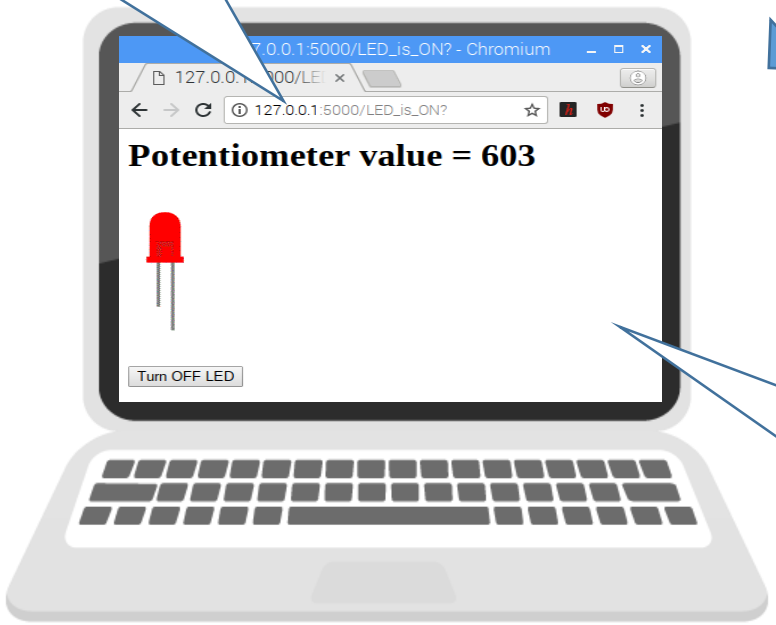
The laptop & the RPi are in the same network.

Router






You will need to enter the IP address of the RPi, instead of 127….




The web browser is in the laptop, and a person can use this to do remote monitoring & control.

The RPi is the web server, and is connected directly to the potentiometer & the LED.

# Monitoring & control thru. a network (cont.)

- The app.py file must be modified as follows:

app.py - /home/pi/WebApp/app.py (3.5.3)

File Edit Format Run Options Window Help

```
from flask import Flask
from flask import render_template

#set up for monitoring potentiometer
import spidev
my_spi=spidev.SpiDev()
my_spi.open(0,0)

#set up for controlling LED
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(27,GPIO.OUT)

app=Flask(__name__)

def read_ADC(): #function to read potentiometer
    my_spi.max_speed_hz=1350000
    r=my_spi.xfer2([1,0b10000000,0])
    result=((r[1]&3)<<8)+r[2]
    return result

@app.route('/LED_is_ON')
def show_LED_is_ON():
    pot_val=str(read_ADC()) # read potentiometer value
    GPIO.output(27,1) #turn on the physical LED
    return render_template("LED_is_ON.html",value=pot_val) #pass pot_val to webpage

@app.route('/LED_is_OFF')
def show_LED_is_OFF():
    pot_val=str(read_ADC()) # read potentiometer value
    GPIO.output(27,0) #turn off the physical LED
    return render_template("LED_is_OFF.html",value=pot_val) #pass pot_val to webpage

#other "web pages" not shown

if __name__=="__main__":
    app.run(debug=True,host='0.0.0.0') #0.0.0.0 => accessible to any device on the network
```

Add in the set-up code for monitoring the potentiometer (an analogue input).

Add in the set-up code for controlling the LED (a digital output).

Write a function to read the potentiometer, from channel 0 of the MCP3008 ADC.

In the “LED_is_ON” route / function, read the potentiometer & turn on the red LED on the shield.

Pass the potentiometer value to the web page to display.

Similarly for the “LED_is_OFF”

# Monitoring & control thru. a network (cont.)

- The html files must be modified as follows:

LED_is_ON.html

File Edit Search Options Help

```
<h1>Potentiometer value = {{value}}</h>

<p>
<img src="/static/LED_on.gif" alt="LED on gif file">

<p>
<form action="/LED_is_OFF">
  <button type="submit">Turn OFF LED</button>
</form>
```

A line / paragraph is added to display the potentiometer value passed in.

Likewise, here.

LED_is_OFF.html

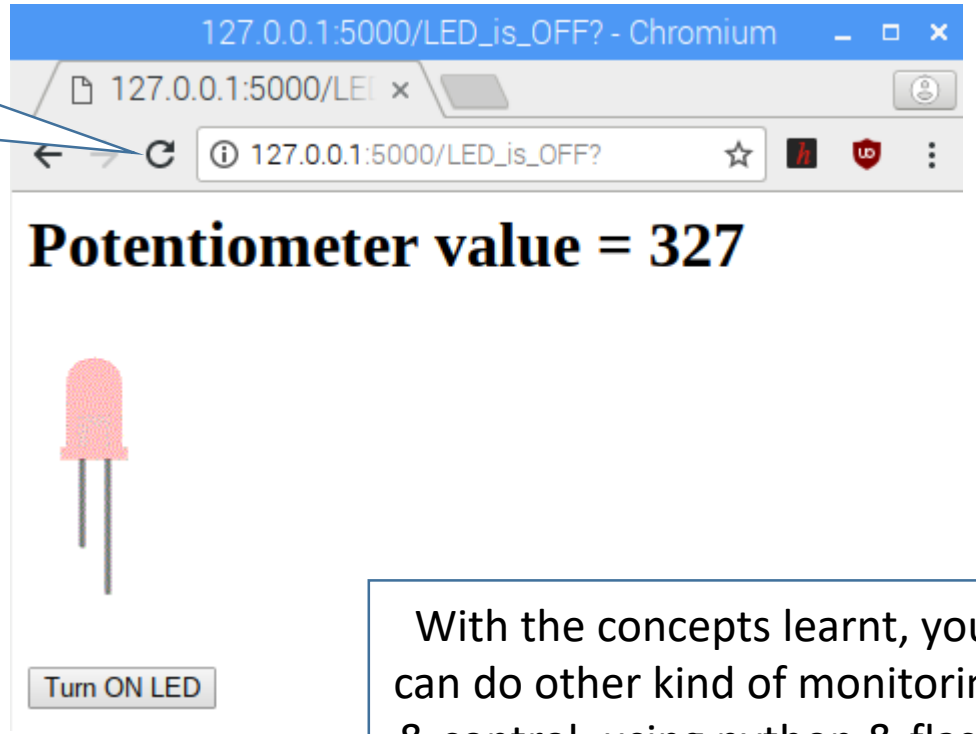
File Edit Search Options Help

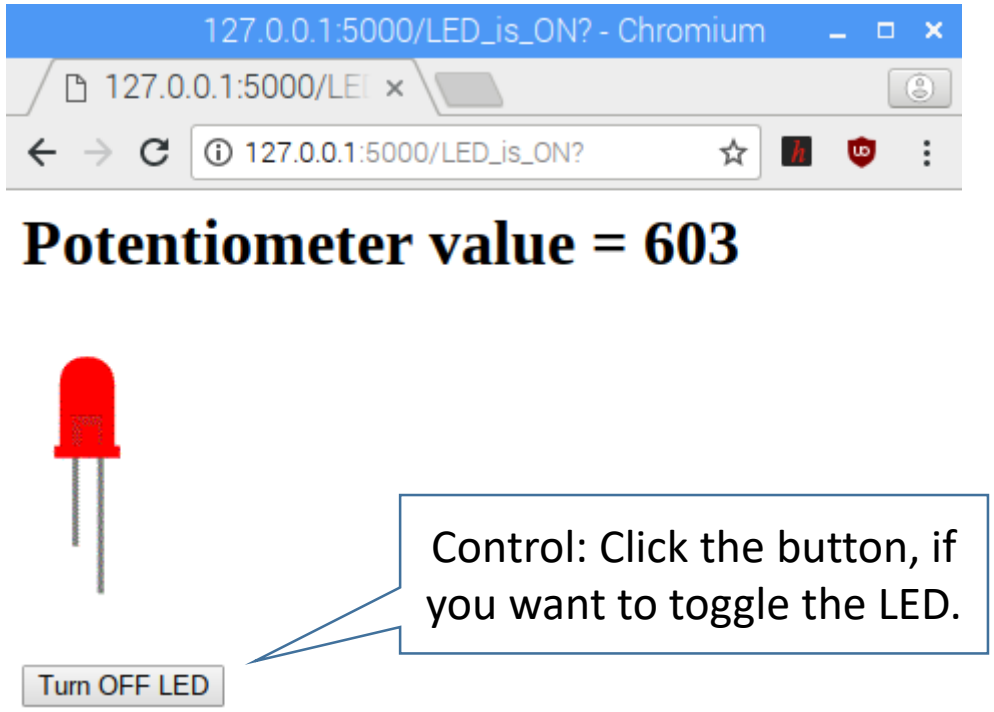
```
<h1>Potentiometer value = {{value}}</h>

<p>
<img src="/static/LED_off.gif" alt="LED off gif file">

<p>
<form action="/LED_is_ON">
  <button type="submit">Turn ON LED</button>
```

# Monitoring & control thru. a network (cont.)

- Browse to either web page, and see the monitoring & control in action:

Monitoring: Refresh the page, if you want to see the new potentiometer value.



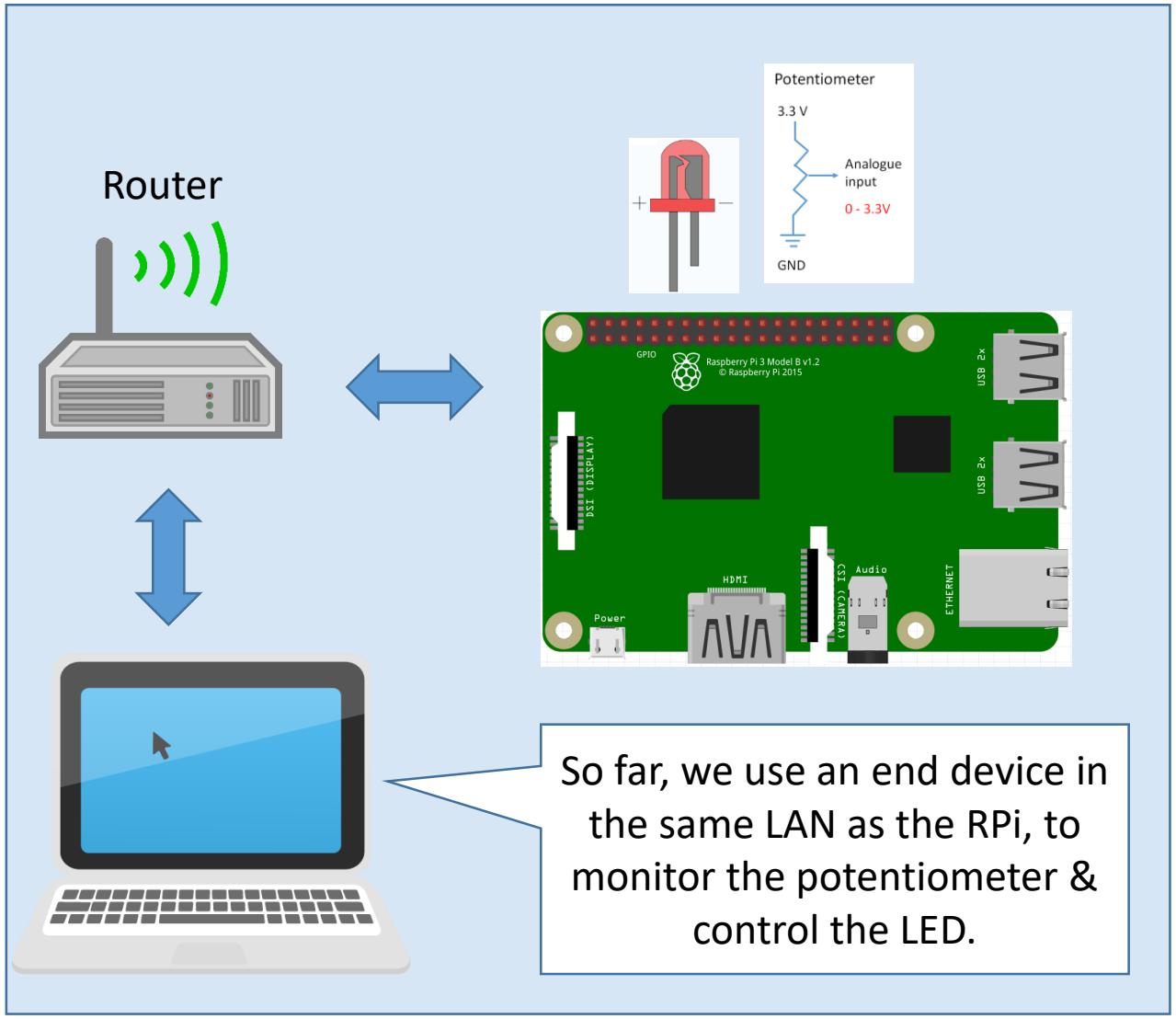




Control: Click the button, if you want to toggle the LED.

With the concepts learnt, you can do other kind of monitoring & control, using python & flask, in your project.

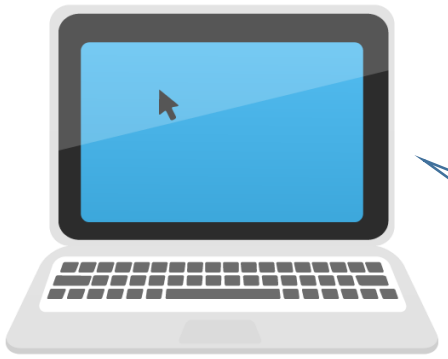
# DHCP reservation & port forwarding

- DHCP reservation & port forwarding are 2 important concepts to learn, if we want to do monitoring & control through the internet.

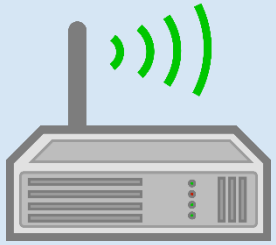
# DHCP reservation & port forwarding (cont.)

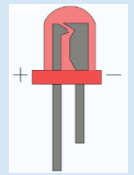
- What should we do, when the control & monitoring is to be done through the internet i.e. from an end device NOT in the same LAN as the RPi?

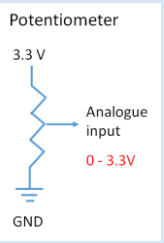



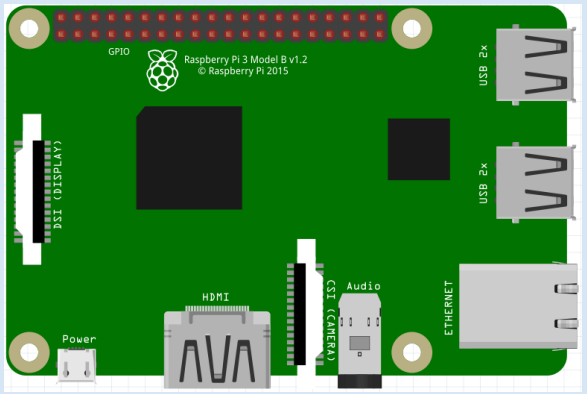
An end device NOT in the same LAN as the RPi.

Router








**Step 1**: Access the router's admin page using an end device in the LAN.

192.168.1.254

Or: 192.168.1.1

# DHCP reservation & port forwarding (cont.)

Note: Different routers have different UI's (user interfaces). These screen captures are from Cisco's Packet Tracer.

Wireless Router0

Physical | Config | GUI | Attributes

Wireless-N Broadband Router WRT300N

Setup | Setup | Wireless | Security | Access Restrictions | Applications & Gaming | Administration | Status

Basic Setup | DDNS | MAC Address Clone | Advanced Routing

**Internet Setup**

Internet Connection type: Automatic Configuration - DHCP

Optional Settings (required by some internet service providers)
Host Name:
Domain Name:
MTU: Size: 1500

**Network Setup**

Router IP
IP Address: 192 . 168 . 1 . 254
Subnet Mask: 255.255.255.192

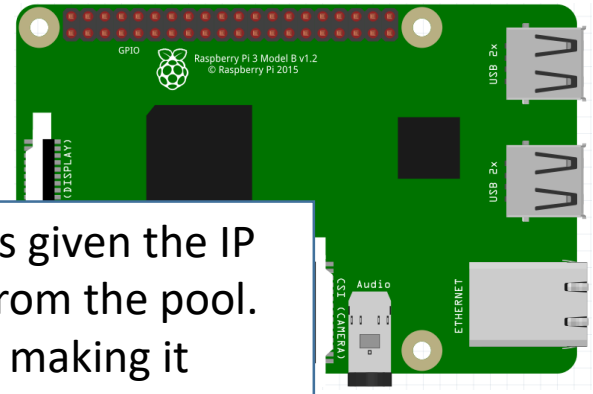
DHCP Server Settings
DHCP Server: Enabled / Disabled — DHCP Reservation
Start IP Address: 192.168.1. 100
Maximum number of Users: 50
IP Address Range: 192.168.1. 100 - 149
Client Lease Time: 0
Static DNS 1: 0 . 0

Help...

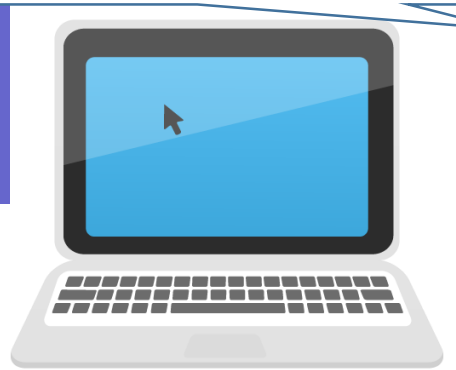
Indeed, the router's IP address is 192.168.1.254

Note that here, the RPi is given the IP address 192.168.1.102 from the pool. But this is NOT fixed, making it inconvenient to locate the web server...

The end devices in the LAN will be assigned IP addresses from this pool.

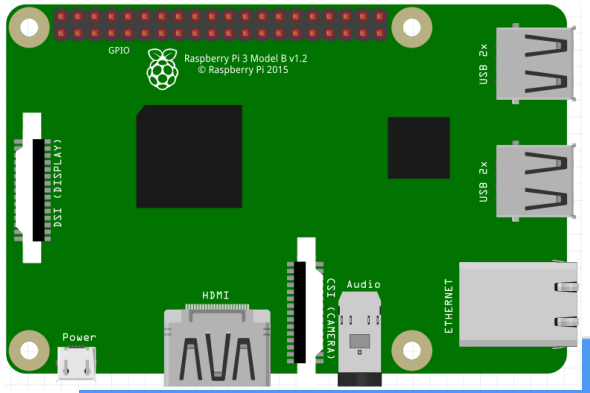



192.168.1.102



192.168.1.100

# DHCP reservation & port forwarding (cont.)




**Step 2**: At a RPi terminal, enter the command as shown, to get the MAC address, a 48-bit number that is fixed, and unique to the RPi.

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
pi@raspberrypi:~ $ cat /sys/class/net/wlan0/address
b8:27:eb:35:75:4e
pi@raspberrypi:~ $ cat /sys/class/net/eth0/address
b8:27:eb:60:20:1b
pi@raspberrypi:~ $
```

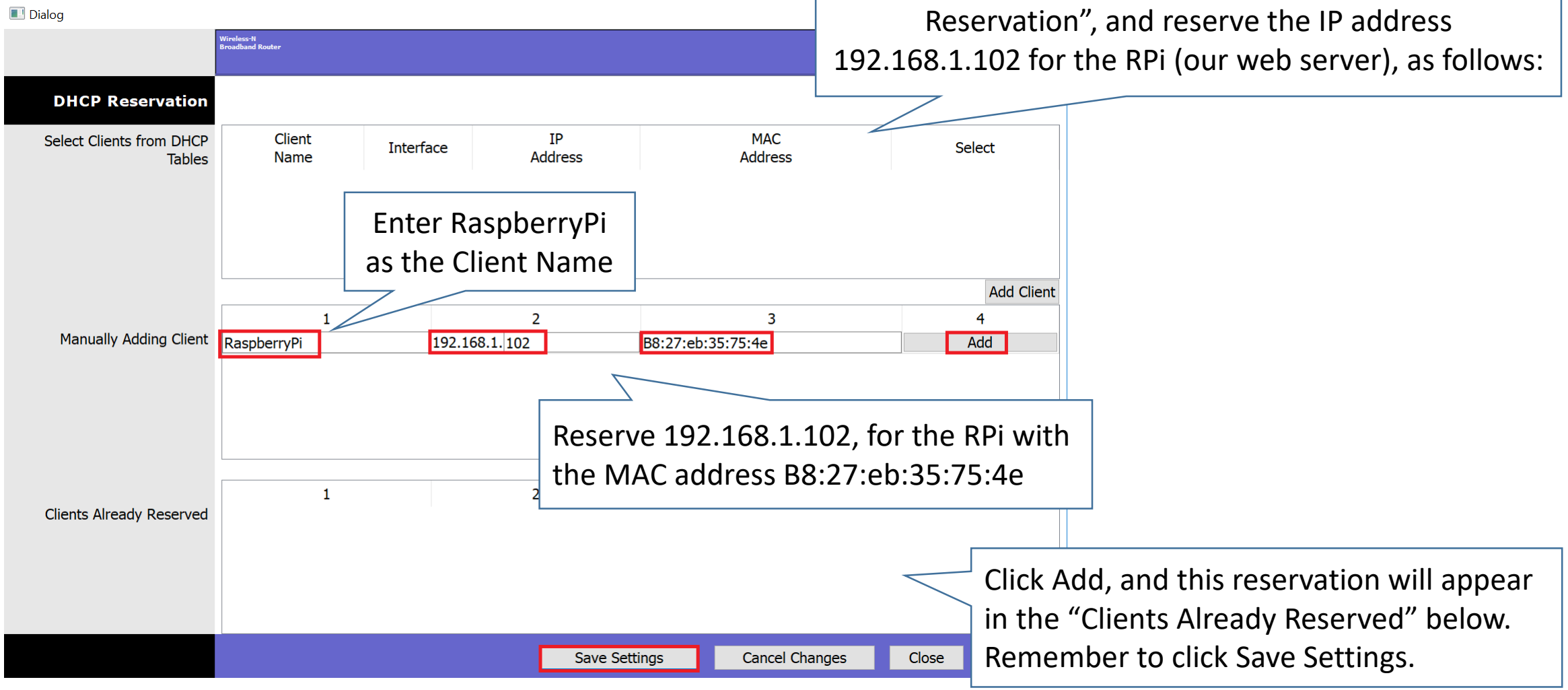
The 1st MAC address is for wireless access, while the 2nd MAC address is for wired access. Let's assume we use wireless access.

The (wireless) router can be configured, so that it will always assign the IP address 192.168.1.102 to the RPi (our web server), which has the MAC address B8:27:eb:35:75:4e

B8:27:eb:35:75:4e → 192.168.1.102

This is called "DHCP Reservation based on MAC address".

# DHCP reservation & port forwarding (cont.)




**Step 3**: Back to the router's admin page, click "DHCP Reservation", and reserve the IP address 192.168.1.102 for the RPi (our web server), as follows:

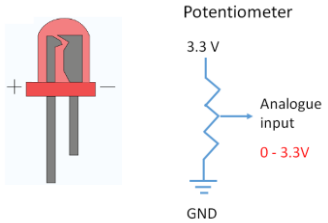
Enter RaspberryPi as the Client Name

Reserve 192.168.1.102, for the RPi with the MAC address B8:27:eb:35:75:4e

Click Add, and this reservation will appear in the "Clients Already Reserved" below. Remember to click Save Settings.

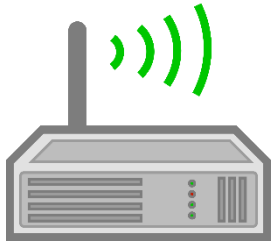
# DHCP reservation & port forwarding (cont.)

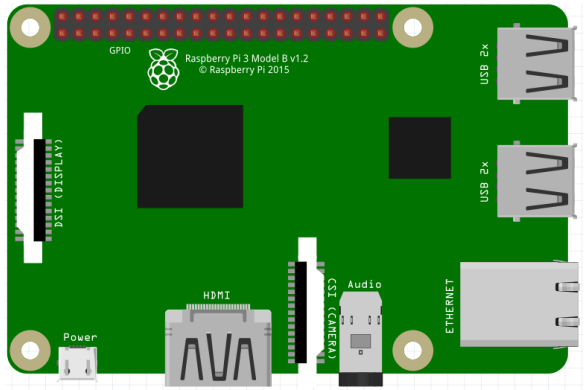
After this is done, the RPi (our web server) will always get back the same IP address.




I am the end device with the MAC address B8:27:eb:35:75:4e

Router

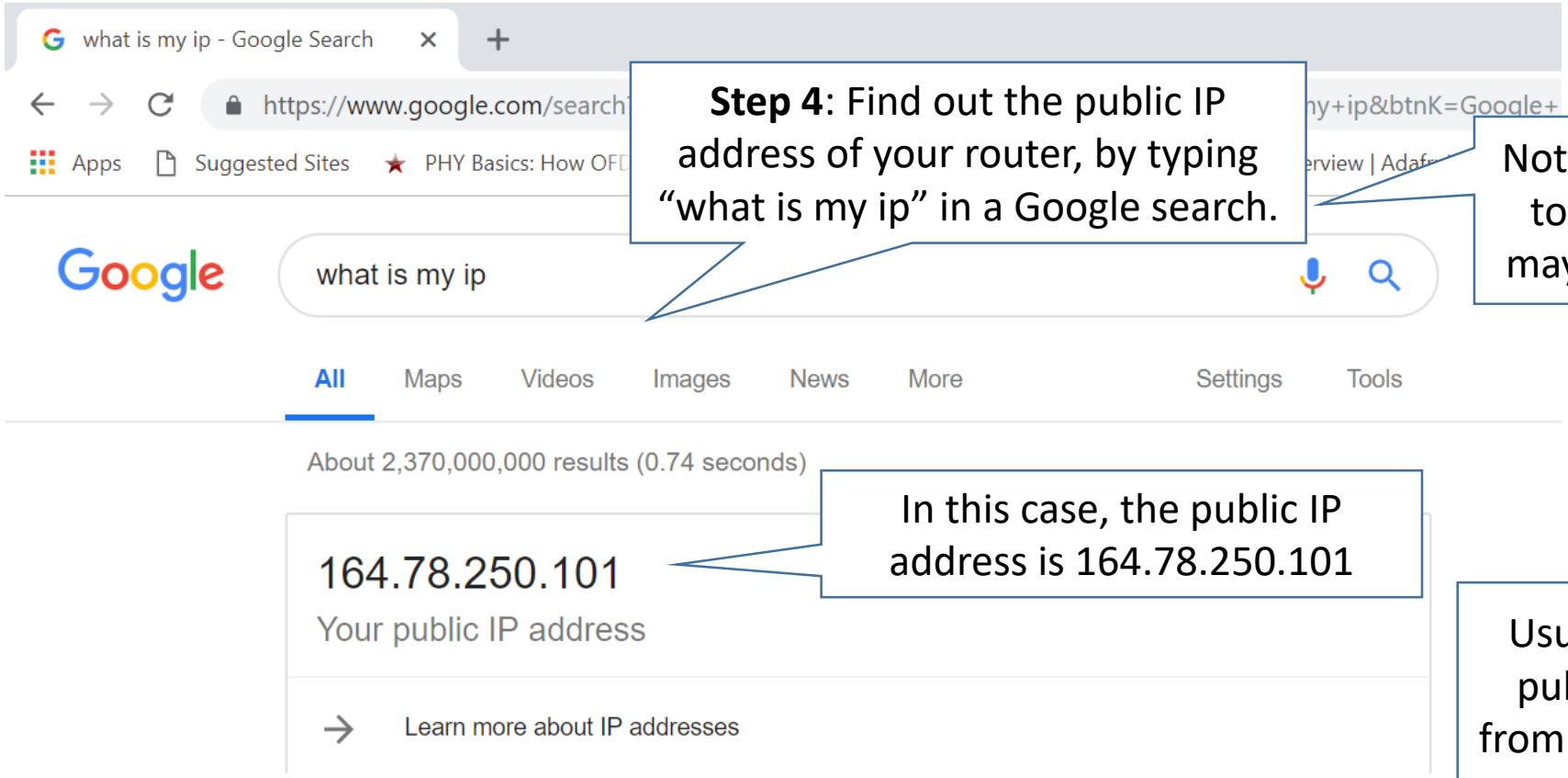





I have reserved this IP address for you: 192.168.1.102

And the web server can be accessed more easily.

# DHCP reservation & port forwarding (cont.)




**Step 4**: Find out the public IP address of your router, by typing "what is my ip" in a Google search.

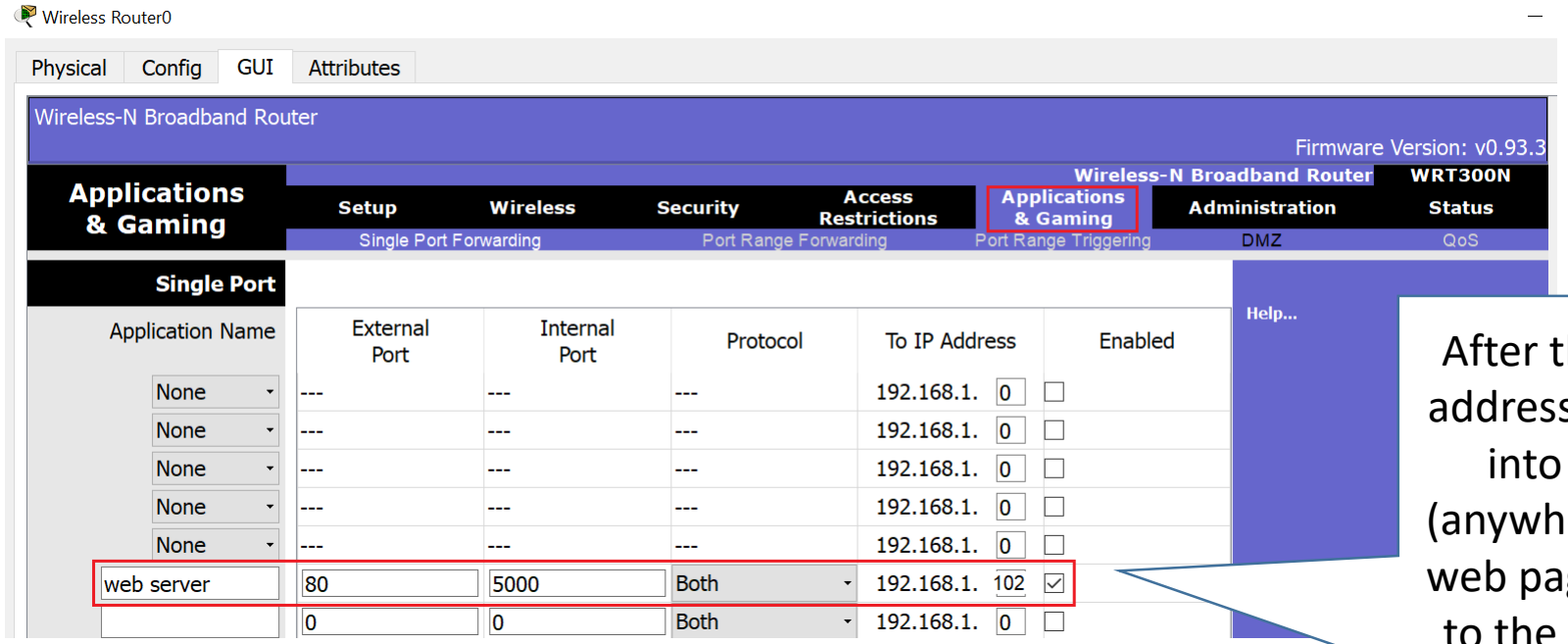
Note that you may not be able to do this in school. So, you may want to try this at home?

In this case, the public IP address is 164.78.250.101

Usually, each home only has ONE public IP address. The addresses from the router (182.168.1.100-149) are private IP addresses, which cannot be used for internet access.

# DHCP reservation & port forwarding (cont.)

**Step 5**: Back to the router's admin page, set up the "port forwarding" as follows, to forward all requests for web pages to the web server:




80 is well-known port number for web server.

After this, if 164.78.250.101:80 (public IP address : external port number) is entered into a web browser in an end device (anywhere in the internet), the request for web page will be forwarded, by the router, to the web server at 192.168.1.102:5000 (private IP address : internal port number).

# DHCP reservation & port forwarding

After this is done, an end device anywhere in the internet can access the RPi web server.

1. I want to access the RPi web server.
I know the Public IP Address (router's) (164.78.250.101) and External Port (80).

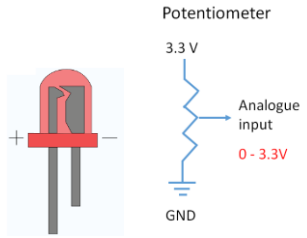
Or Public Port.

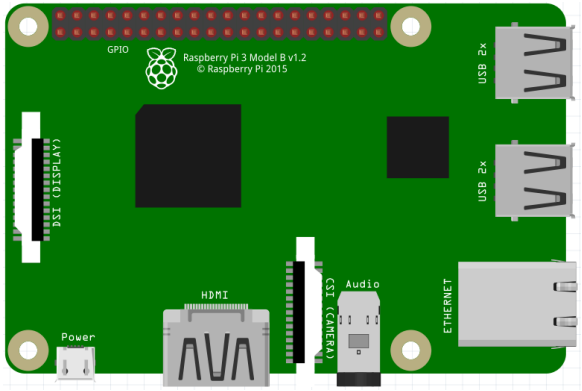
End device not in the same LAN


Router

Or Private Port.

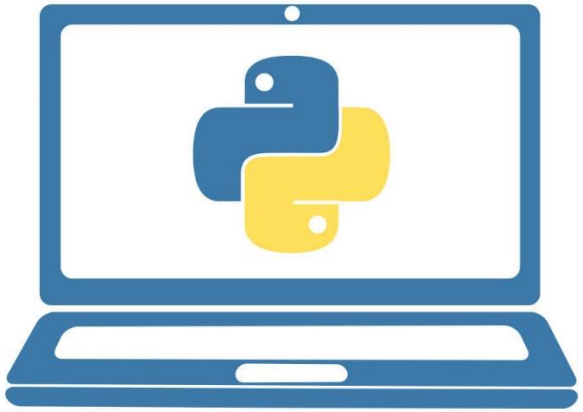
2. Such request will be forwarded to the Private IP Address (192.168.1.102) and Internal Port (5000).




3. Here you are, the web page you requested.

# Lab Exercises



- Exercise 6.1 – Hello World!
- Exercise 6.2 – Adding a photo to a web page
- Exercise 6.3 – Adding a camera to update the photo
- Exercise 6.4 – Adding a PIR sensor to trigger the camera

# Exercise 6.1 – Hello World!

Step 1 - Create the following folders & files in your pi folder:

/home/pi/WebApp (folder)
- app.py (file)
- templates (folder)
  - index.html (file)

Step 2 – Edit the app.py file to the following:

app.py - /home/pi/WebApp/app.py (3.5.3)

File Edit Format Run Options Window Help

```
from flask import Flask
from flask import render_template
app=Flask(__name__)

@app.route('/')
def index():
    return render_template("index.html")

if __name__=="__main__":
    app.run(debug=True,host='0.0.0.0') #0.0.0.0 => accessible to any device on the network
```

Step 3 – Edit the index.html file to the following:

index.html

File Edit Search Options Help

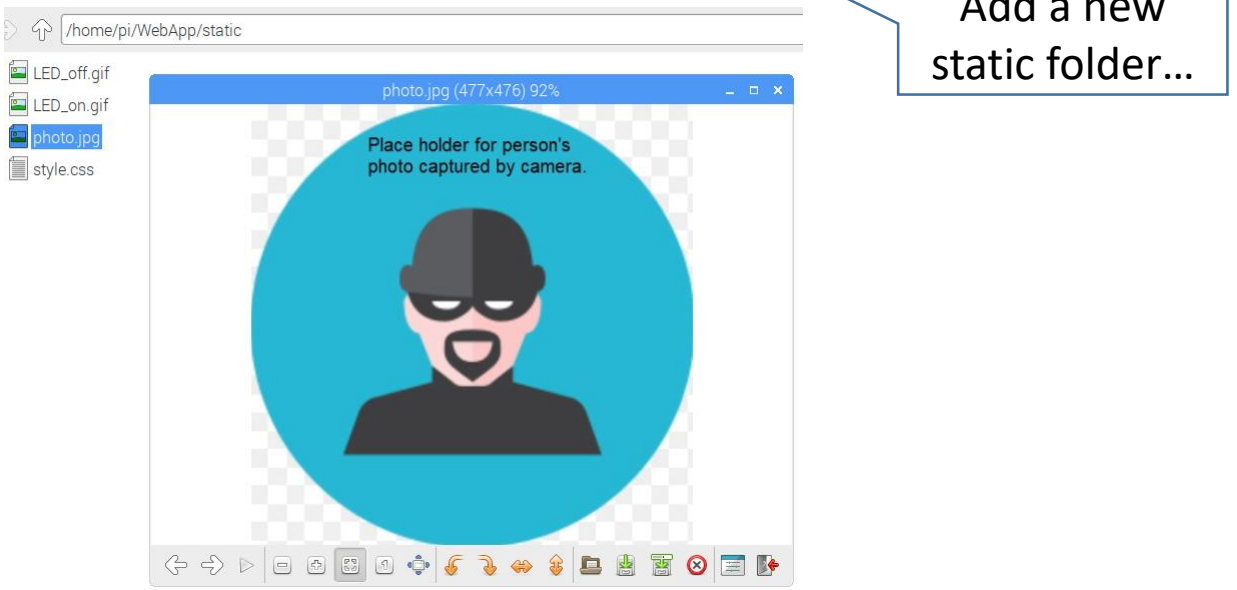
```
<html>
<body>
<h1> Hello World! </h1>
</body>
</html>
```

Step 4 – Run the app.py web server, and use the Chromium browser to visit 127.0.0.1:5000

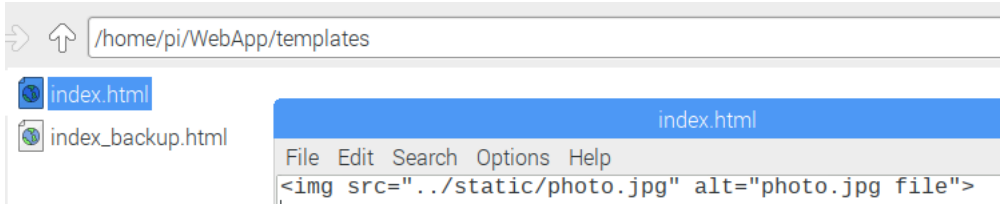
What do you see?

# Exercise 6.2 – Adding a photo to a webpage

Step 1 - Google for “intruder icon jpg”, and save the image (similar to the following) as “photo.jpg” in the WebApp/static folder:




Step 2 - Using a text editor, modify the index.html file in the WebApp/templates folder to the following:

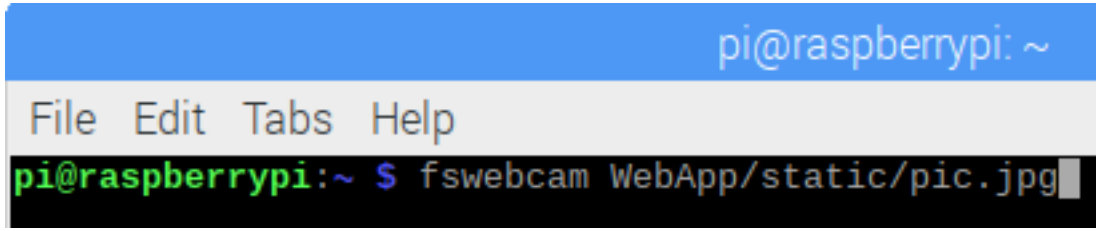



Step 3 - Double click on the index.html file (to open it using the Chromium browser). What do you see?

# Exercise 6.3 – Adding a camera to update the photo

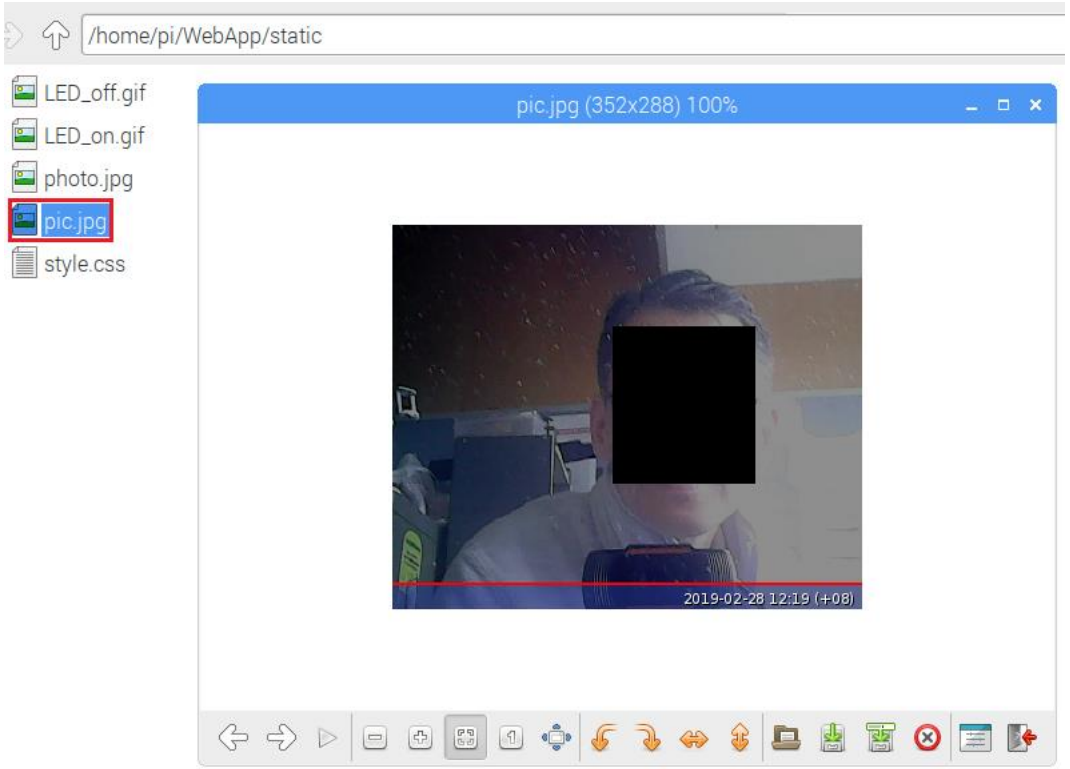
Make sure you have a webcam in a USB port!

Step 1 - At a RPi terminal, type:
fswebcam WebApp/static/pic.jpg




If necessary, install fswebcam by typing
sudo apt-get install fswebcam
at a RPi terminal.

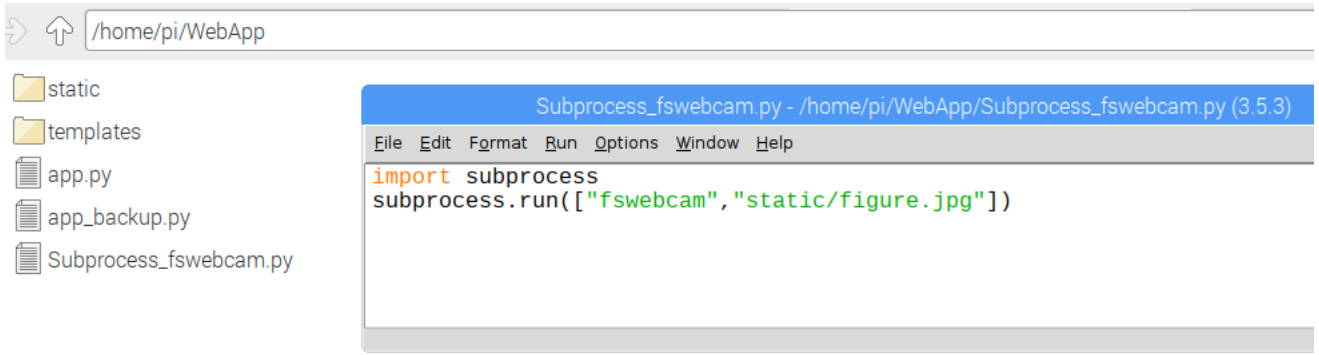
Step 1b - A photo will be captured by the webcam, and saved as pic.jpg in the WebApp/static folder:



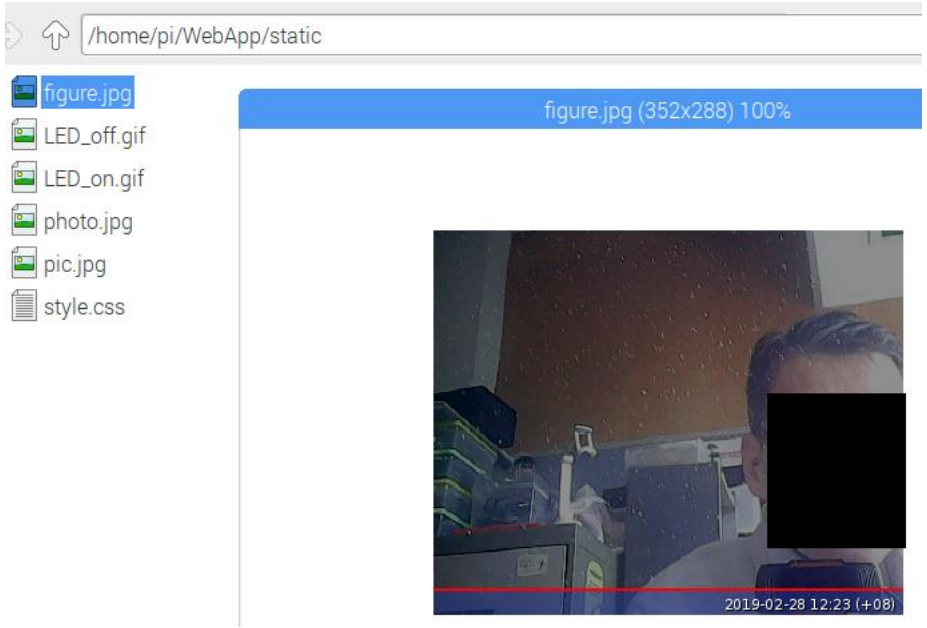

fswebcam is a small and simple webcam app for Lunix. It can capture images from a number of different sources and perform simple manipulation on the captured image. The image can be saved as one or more PNG or JPG files. You can learn more about fswebcam from this:
https://www.raspberrypi.org/documentation/usage/webcams/

# Exercise 6.3 – Adding a camera to update the photo (cont.)

Step 2 - The same can be achieved by using a Python program. Type these 2 lines, save it (any name will do, with extension .py) in the WebApp folder, and run it:

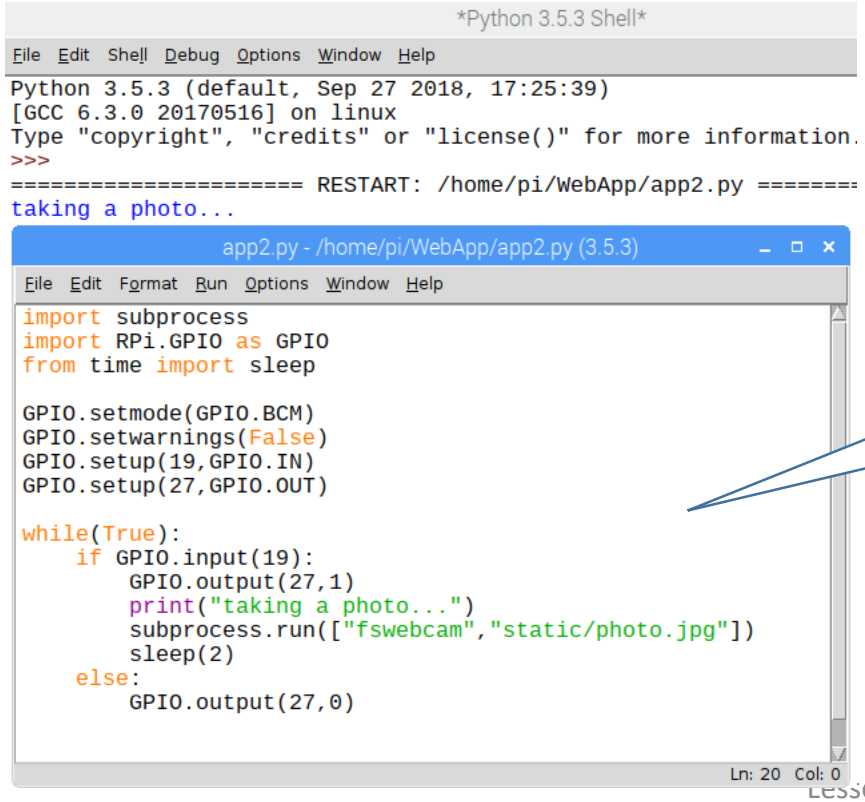



Step 2b - You will find that a photo (named as figure.jpg) has been saved in the WebApp/static folder:



# Exercise 6.4 – Adding a PIR sensor to trigger the camera

Step 1 - Connect a PIR sensor to the RPi (or use the button on the IoT shield, at GPIO pin 19), write the following Python program, and save it as app2.py in the WebApp folder




When the button is pressed, the LED (at GPIO pin 27) will light up, the message "taking a photo…" will be printed onto the screen, and fswebcam will be used to take a photo & save it as photo.jpg (replacing the old photo) in the static folder.

Step 2 - Run this program by double clicking on it. It will most likely be run using Thonny, another Python IDE.

# Exercise 6.4 – Adding a PIR sensor to trigger the camera (cont.)

Step 3 - Run the app.py (web server, unmodified) using the Python IDLE. Note that RPi can run more than one Python program concurrently.

app.py - /home/pi/WebA

File Edit Format Run Options Window Help

```
from flask import Flask
from flask import render_template

app=Flask(__name__)

@app.route('/')
def index():
    return render_template("index.html")

if __name__=="__main__":
    app.run(debug=True,host='0.0.0.0') #0.0.0.0 => accessible to any device on the network
```

You may need to issue the command fuser –k 5000/tcp before running the web server program again.

With the methods illustrated in these 4 exercises, can you see how you can implement an “Intruder Alarm” project?

Step 4 - Use the Chromium browser to visit 127.0.0.1:5000 again. Clearing the cache/history if necessary, see if you get an updated photo at the webpage each time you press the button:


Thank you