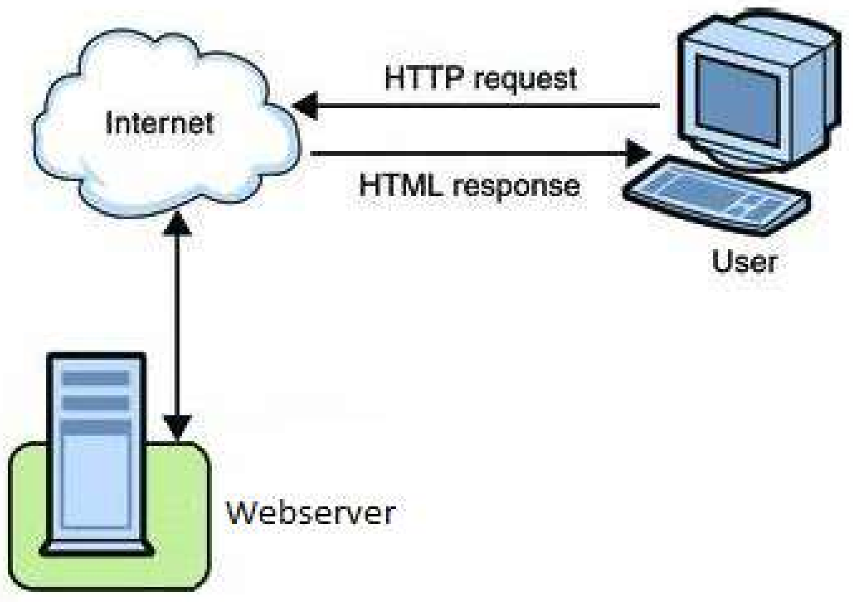
Lab Sheet: 4

Objective: Configuring Raspberry Pi webserver and controlling output devices through Wifi network

Hardware Requirements: Raspberry Pi; SD card at least 4GB, Monitor with HDMI (High definition multimedia interface) connector, Micro USB power supply, USB keyboard, USB mouse, Ethernet cable or WiFi dongle, breadboard, two LEDs (read and green), two 370Ω Resistors and Jumper wires.

Internet of Things: This lab sheet demonstrates a basic idea of Internet of Things; IOT is a concept of collecting things or articles and connecting them to the internet; we all enjoying the benefits of IOT with our phones, laptops or desktops; when something is connected to the internet that may send and receive information; i.e. If any object is connected to the Internet; so as that the object is connected to you through your device; hence you may send information (requesting for something) and receive information (status of the request). IOT makes things smart; now a days everything around us connected to the internet and turned into smart such as smart cars, smart homes, smart class rooms, etc.

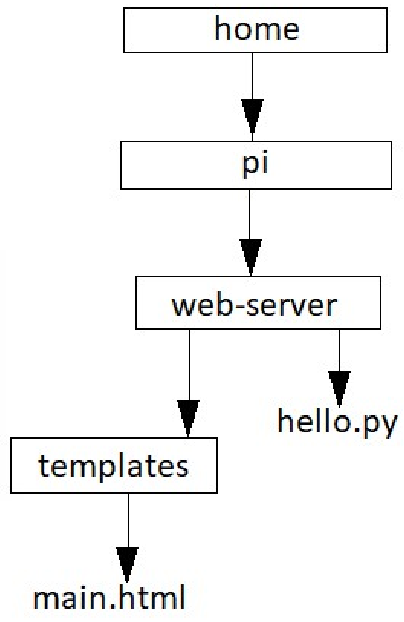
Webserver: Webserver, is a dedicated hardware or software (or both working together) that serves information word wide when client requests i.e. it delivers information to end users over the internet using Hyper Text Transfer Protocol (HTTP). i.e. The communication between client and server takes place using the Hypertext Transfer Protocol. In general, webservers contain one or more websites, to publish any website you need a webserver; The primary function of a web server is to store, process and deliver web pages to clients; whenever a browser needs a file which is hosted on a webserver; the browser request the server via HTTP; and the server services the request by sending the requested document to the browser.



Installing FLASK and Setting up your RPi Webserver: To create a simple web server, we will be using a Python micro-frame work called Flask; using Flask you may setup your Raspberry Pi as a webserver; Flask uses a template engine called Jinja2 that may send dynamic data from your Python script to your HTML file. To install Flask, you will need to have pip installed. Run the following commands to update your Pi and install pip:

- sudo apt-get update
- sudo apt-get upgrade
- sudo apt-get install python-pip python-flask
- sudo pip install flask

To keep everything organized, we need to create two folders named web-server and template for storing files; templates should be created inside web-server; templates is a container for your html files; the following figure shows a general tree structure of a simple web-server.

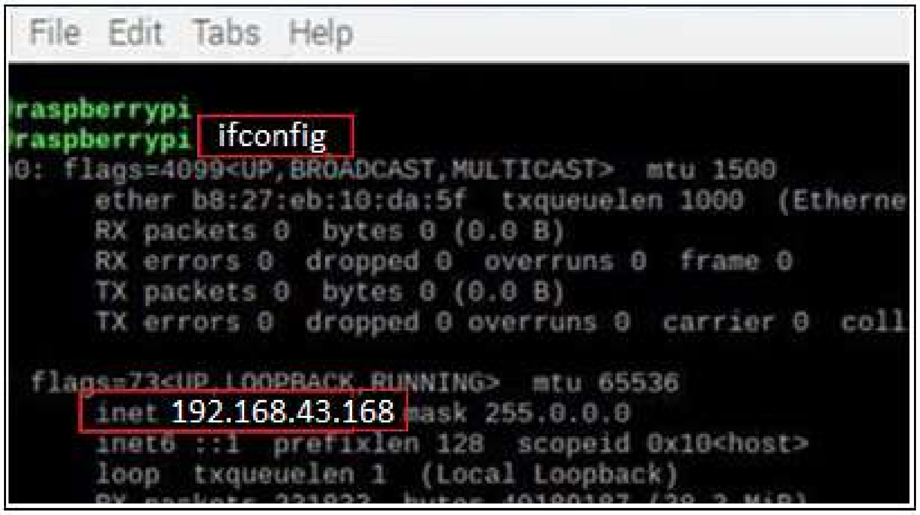


1. **My first web Raspberry Pi webserver:** Now, let us create our first python script with Flask for displaying “Hello Word !” in a browser window; save the following script as hello.py; run and verify the output.

```
# Raspberry Pi webserver Done at Amrita
# Amritapuri Campus - for CSE

from flask import Flask         # load Flask module into Python script
app = Flask(__name__)           # create Flask object called app:

@app.route('/')
#run the hello() function when someone accesses the root URL (‘/’) of the server
```

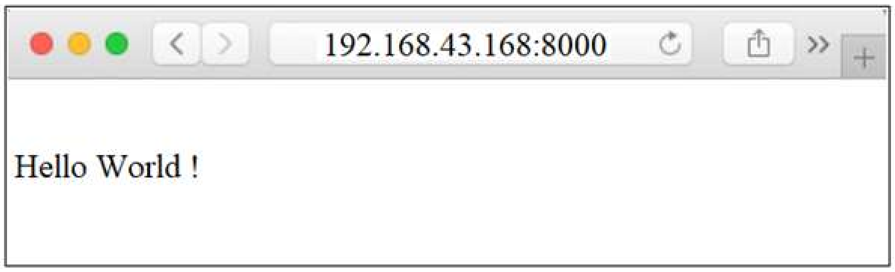
```
def hello():                    # defining function hello
    return 'Hello World !'      # it returns 'Hello World !'
# the server starts to listen port 8000, and report any errors
# 192.168.43.168 is the IP address of the pi which is connected to internet
if __name__ == '__main__':
    app.run(debug=True, port=8000, host='192.168.43.168')
```

Infact the code needs address of your Raspberry Pi and port address also; you may get it by running ***ifconfig*** command on your terminal as shown in the above picture.



```
raspberrypi
raspberrypi ifconfig
0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
     ether b8:27:eb:10:da:5f  txqueuelen 1000  (Etherne
     RX packets 0  bytes 0 (0.0 B)
     RX errors 0  dropped 0  overruns 0  frame 0
     TX packets 0  bytes 0 (0.0 B)
     TX errors 0  dropped 0 overruns 0  carrier 0  coll

  flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
     inet 192.168.43.168 mask 255.0.0.0
     inet6 ::1  prefixlen 128  scopeid 0x10<host>
     loop  txqueuelen 1  (Local Loopback)
```

Launching Pi Webserver: To launch your Raspberry Pi web server, run the script hello.py; your web server starts immediately; now you must open a web browser that is connected to same wifi network as your Raspberry and type the IP address with port number (192.168.43.168:8000) in the right format; “Hello World !” should appear in your browser window, as shown in the output figure.



Hello World !

2. **Webserver with Template:** Modify the above script, hello.py to read date and time of your Pi Webserver; and pass the date and time to a HTML template as an argument

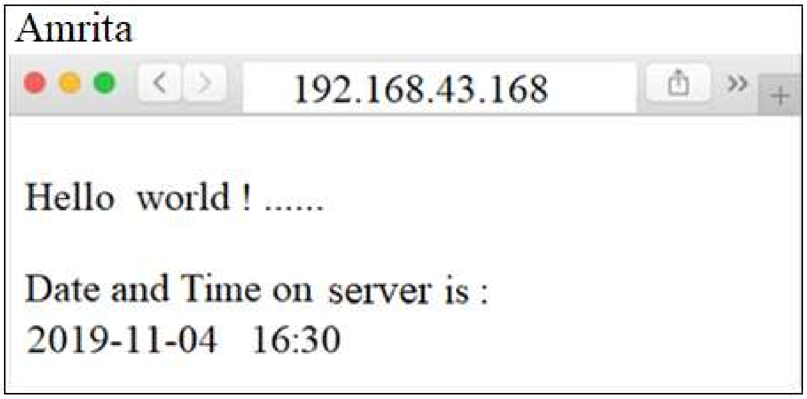
Templates: Templates are nothing but HTML files; **o**ne may have a separate HTML template with placeholders or spots for any dynamic data to be inserted. Here we have a template named main.html, and saved it in the subfolder /templates;

```
< html>  <head>
    <title>{{ title }}</title>
  </head>
  <body>
    <h3>Hello world !.....</h3>
    <h3>Date and Time on server is: {{ time }}</h3>
  </body>
</html>
```

In the above HTML template, anything given in double curly braces is interpreted as a variable, whose value would be given by the Python script via the render_template function. Now, we write a Python script - hellotime.py

```
from flask import Flask, render_template
import datetime
app = Flask(__name__)
@app.route("/")

def hello():
  now = datetime.datetime.now()
  servertime = now.strftime("%d-%m-%y  %h:%m")

  templateData = {
    'title' : 'Amrita',
    'time': servertime     }
    return render_template('main.html', **templateData)

if __name__ == "__main__":
    app.run(host='192.168.43.168', port=8000, debug=True)
```

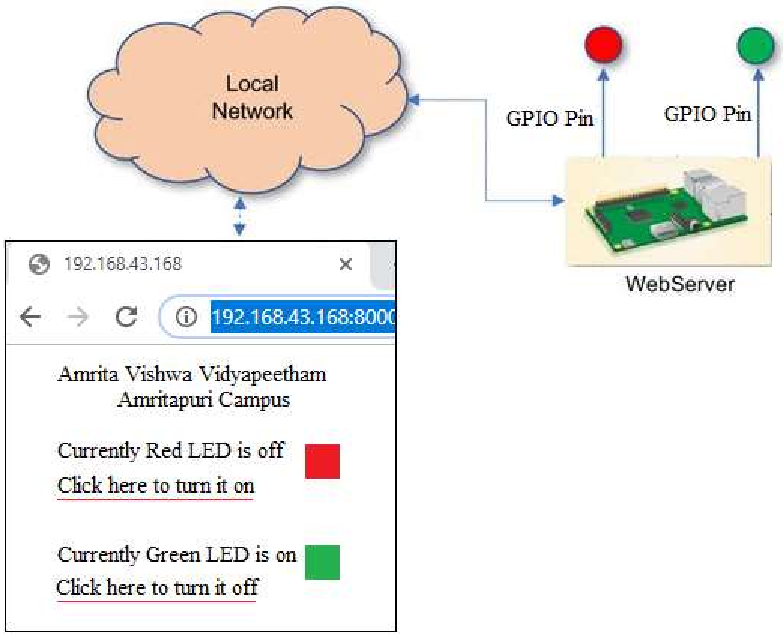
Here in this code we created a formatted string "servertime" using the "now" object. Next important thing in the code is, that we have created a dictionary of variables or a set of keys (such as title and time) along with whose values to be passed to the template main.html. Run hellotime.py; open a web browser with your RPi IP address and port number (192.168.43.168:8000); you will be provided with the following screen shot that shows the real time clock values of your Pi server; content of the template changes dynamically every time when you refresh it; accordingly the values of the variables (title and time) will be passed by Python script. In our case, title has a fixed value, but time will be changing in every second with Pi real time clock.



Hello world ! ......

Date and Time on server is :
2019-11-04 16:30

3. **Reading and Controlling GPIOs:** Here, we switch on and off two LEDs (Red and Green) which are connected to two GPIO pins of our Raspberry Pi -Webserver via a website which can be accessed worldwide; the following figure illustrates the operation clearly.

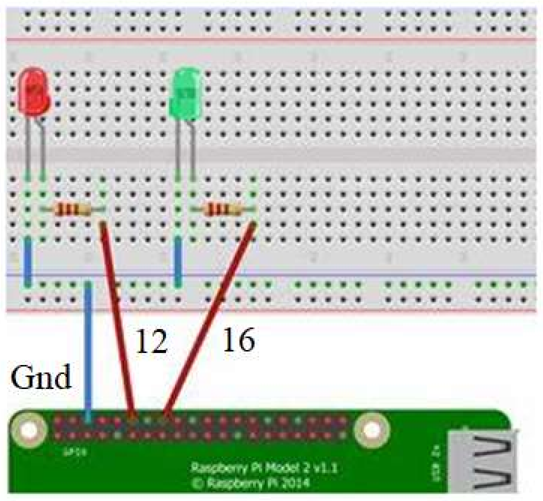


Amrita Vishwa Vidyapeetham
Amritapuri Campus

Currently Red LED is off 🟥
Click here to turn it on

Currently Green LED is on 🟩
Click here to turn it off

Schematic: Connect the positive lead or anode of a red LED to the GPIO pin 12 (follow physical pin numbering scheme) through a 270 ohm resistor, and the cathode to GND pin; in this way connect a green LED to the GPIO pin 16 as shown in the following schematic to be used in this lab session.



We will create a template (main.html) that request a controller (amrita.py) to change the state of devices (LEDs); and receives the current state of devices. i.e. whenever a request is received from an end user, in response the Phython script checks the current state of the GPIO and drives the LED on or off and sends the new state back to the web application.

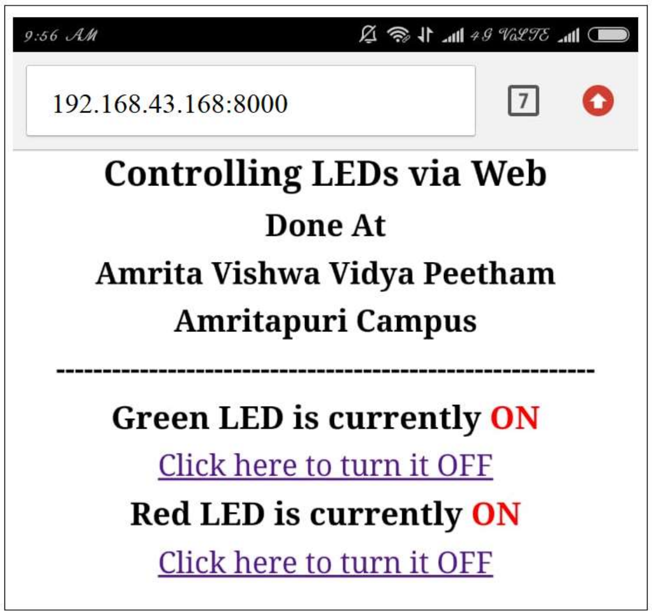
```
<html>  <head>
 <title>RPi Web Server</title>
 </head> <body> <center>
<h1>Controlling LEDs via Web</h1>
<h3>Done At</h3>
<h3>Amrita Vishwa Vidya Peetham</h3>
<h3>Amritapuri Campus</h3>
<h3>-------------------------------------------------------</h3>
{% for i in pins %}
<h2>{{pins[i].name}} is currently

{% if pins[i].state == true %}
<font color='red'> <strong> ON </strong> </font> </h2>
<a href="/{{i}}/off">Click here to turn it OFF</a>

{% else %}
<font color='blue'><strong> OFF </strong> </font> </h2>
<a href="/{{i}}/on">Click here to turn it ON</a>

{% endif %}
{% endfor %}

</center>
</body>
</html>
```

Following Python script (amrita.py) setup a web server and interacts with the Raspberry Pi GPIOs; the script will be running on our Raspberry Pi webserver continuously for checking the status of LEDs and controlling them; this a very straightforward code to control GPIO Pins.

```
import RPi.GPIO as GPIO
from flask import Flask, render_template, request
app = Flask(__name__)
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

# Create a dictionary (called pins)
# to store pin number, name and state
pins = {
  12 : {'name' : 'Red LED', 'state' : False},
  16 : {'name' : 'Green LED', 'state' : False}
  }
#Configure each pin as an output and make it False
for i in pins:
  GPIO.setup(i, GPIO.OUT)
  GPIO.output(i, False)

@app.route("/")
def main():
   # read pin state and store it in pins dictionary
   for i in pins:
     pins[i]['state'] = GPIO.input(i)

  # Put the pin dictionary into the data dictionary:
   amrita = {
     'pins' : pins
     }
  # Pass amrita to the template main.html
   return render_template('main.html', **amrita)

# The function below is executed when someone requests a URL
# with the pin number and action in it:
@app.route("/<mypin>/<mytask>")
def action(mypin, mytask):

# Convert the pin number read through URL into an integer:
  mypin = int(mypin)

  # If the action part of the URL is "on,"
  # then turn the LED on
  if mytask == "on":
    GPIO.output(mypin, True)
  # else turn the LED off
  if mytask == "off":
    GPIO.output(mypin, False)

  # For each pin, read the pin state and store it in the pins dictionary:
  for i in pins:
   pins[i]['state'] = GPIO.input(i)

  # Put pins dictionary into amrita dictionary:
  amrita = {
   'pins' : pins
  }
```

```
 # Pass amrita into the template main.html and return it to the user
  return render_template('main.html', **amrita)


if __name__ == "__main__":
  app.run(host='192.168.43.168', port=8000, debug=True)
```

Run the script and open a browser with IP address and port address; you will be provided with two links one for turning on the Red LED and the other for Green LED; through which you can control your LEDs. In this way you may control any device which is connected to your local network only; and cannot be controlled from any different network. In order to control your devices worldwide, Of course you need to obtain unique public IP address of your router from any service provider. Since IP address is not easy to remember, you may have a domain name.



Controlling LEDs via Web
Done At
Amrita Vishwa Vidya Peetham
Amritapuri Campus

Green LED is currently ON
Click here to turn it OFF
Red LED is currently ON
Click here to turn it OFF