Practical No. 7

Aim: Trigger a set of led Gpios on the pi via a Python Flask web server

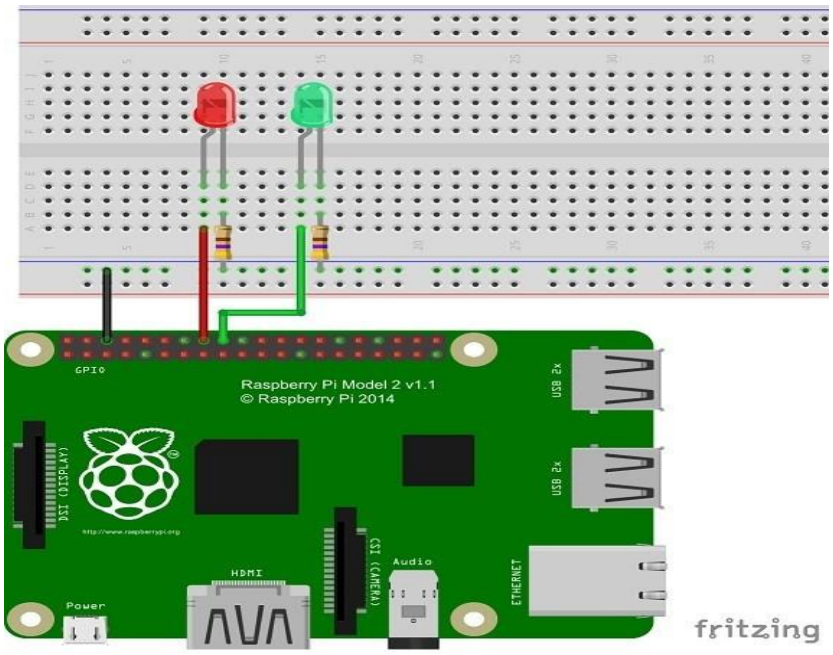
Step 1: install Flask, you'll need to have pip installed. Run the following commands to update your Pi and install pip:

```
pi@raspberrypi ~ $ sudo apt-get update
pi@raspberrypi ~ $ sudo apt-get upgrade
pi@raspberrypi ~ $ sudo apt-get install python-pip python-flask
```

Then, you use pip to install Flask and its dependencies:

```
pi@raspberrypi ~ $ sudo pip install flask
```

Step 2: Simply connect two LEDs to pins GPIO 23 and GPIO 24, as the figure below illustrates.



Step 3: Creating the Python Script

This is the core script of our application. It sets up the web server and actually interacts with the Raspberry Pi GPIOs.

To keep everything organized, start by creating a new folder:

```
pi@raspberrypi ~ $ mkdir web-server
pi@raspberrypi ~ $ cd web-server
pi@raspberrypi:~/web-server $
```

Create a new file called *app.py*.

```
pi@raspberrypi:~/web-server $ nano app.py
```

Copy and paste the following script to your Raspberry Pi (this code is based on Matt Richardson great example).

```
'''

Adapted excerpt from Getting Started with Raspberry Pi by Matt Richardson

Modified by Rui Santos
Complete project details: https://randomnerdtutorials.com

'''

import RPi.GPIO as GPIO
from flask import Flask, render_template, request
app = Flask(__name__)

GPIO.setmode(GPIO.BCM)

# Create a dictionary called pins to store the pin number, name, and pin
state:
pins = {
   23 : {'name' : 'GPIO 23', 'state' : GPIO.LOW},
   24 : {'name' : 'GPIO 24', 'state' : GPIO.LOW}
```

```
   }

# Set each pin as an output and make it low:
for pin in pins:
   GPIO.setup(pin, GPIO.OUT)
   GPIO.output(pin, GPIO.LOW)

@app.route("/")
def main():
   # For each pin, read the pin state and store it in the pins dictionary:
   for pin in pins:
      pins[pin]['state'] = GPIO.input(pin)
   # Put the pin dictionary into the template data dictionary:
   templateData = {
      'pins' : pins
      }
   # Pass the template data into the template main.html and return it to
the user
   return render_template('main.html', **templateData)

# The function below is executed when someone requests a URL with the pin
number and action in it:
@app.route("/<changePin>/<action>")
def action(changePin, action):
   # Convert the pin from the URL into an integer:
   changePin = int(changePin)
   # Get the device name for the pin being changed:
   deviceName = pins[changePin]['name']
   # If the action part of the URL is "on," execute the code indented
below:
   if action == "on":
      # Set the pin high:
      GPIO.output(changePin, GPIO.HIGH)
      # Save the status message to be passed into the template:
      message = "Turned " + deviceName + " on."
   if action == "off":
      GPIO.output(changePin, GPIO.LOW)
      message = "Turned " + deviceName + " off."

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

   # Along with the pin dictionary, put the message into the template data
dictionary:
   templateData = {
      'pins' : pins
   }
```

```
    return render_template('main.html', **templateData)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

Step 4: Creating the HTML File

Create a new folder called templates:

```
pi@raspberrypi:~/web-server $ mkdir templates
pi@raspberrypi:~/web-server $ cd templates
pi@raspberrypi:~/web-server/templates $
```

Create a new file called *main.html*.

```
pi@raspberrypi:~/web-server/templates $ nano main.html
```

Copy and paste the following template to your Pi:

```
<!DOCTYPE html>
<head>
    <title>RPi Web Server</title>
    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs
s" integrity="sha384-
1q8mTJOASx8j1Au+a5WDVnPi2lkFfwwEAa8hDDdjZlpLegxhjVME1fgjWPGmkzs7"
crossorigin="anonymous">
    <!-- Optional theme -->
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap-
theme.min.css" integrity="sha384-
fLW2N01lMqjakBkx3l/M9EahuwpSfeNvV63J5ezn3uZzapT0u7EYsXMjQV+0En5r"
crossorigin="anonymous">
    <!-- Latest compiled and minified JavaScript -->
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"
integrity="sha384-
```

```
0mSbJDEHialfmuBBQP6A4Qrprq5OVfW37PRR3j5ELqxss1yVqOtnepnHVP9aJ7xS"
crossorigin="anonymous"></script>
</head>

<body>
   <h1>RPi Web Server</h1>
   {% for pin in pins %}
   <h2>{{ pins[pin].name }}
   {% if pins[pin].state == true %}
      is currently <strong>on</strong></h2><div class="row"><div
class="col-md-2">
      <a href="/{{pin}}/off" class="btn btn-block btn-lg btn-default"
role="button">Turn off</a></div></div>
   {% else %}
      is currently <strong>off</strong></h2><div class="row"><div
class="col-md-2">
      <a href="/{{pin}}/on" class="btn btn-block btn-lg btn-primary"
role="button">Turn on</a></div></div>
   {% endif %}
   {% endfor %}
</body>
</html>
```

Step 5: Launching the Web Server

To launch your Raspberry Pi web server move to the folder that contains the file *app.py*:

```
pi@raspberrypi:~/web-server/templates $ cd ..
```

Then run the following command:

```
pi@raspberrypi:~/web-server $ sudo python app.py
```

Your web server should start immediately!

Step 6: Open your Raspberry Pi address in your browser by entering its IP address, in my case: [http://192.168.1.98/.](http://192.168.1.98/)