

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

In this practical you'll create a standalone web server with a Raspberry Pi that can switch on and off one LED.

You can replace the LED with any output (like a relay or a transistor).

In order to create the web server you will be using a Python microframework called Flask

Parts Required

Here's the hardware that you need to complete this practical:

1. Raspberry Pi (any Pi should work, I recommend using Raspberry Pi 3) – read Best Raspberry Pi Starter Kits
2. SD Card (minimum size 8Gb and class 10)
3. Micro USB Power Supply
4. Ethernet cable or WiFi dongle
5. Breadboard
6. 2x LEDs
7. 2x 470Ω Resistors
8. Jumper wires

Installing Flask

We're going to use a Python microframework called Flask to turn the Raspberry Pi into web server.

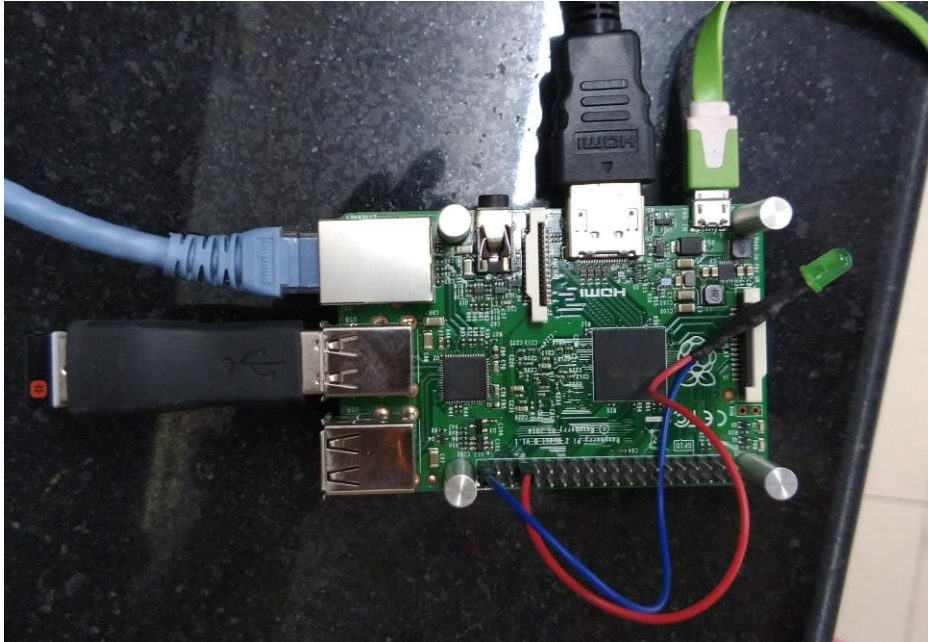
To install Flask, you'll need to have pip installed. Run the following commands to update your Pi and install pip:

```
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
```

Connection Setup



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 *Actuator.py*.

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

```
import RPi.GPIO as GPIO
from flask import Flask, render_template, request

app = Flask(__name__)

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

#define actuators GPIOs
ledNormal = 13

#initialize GPIO status variables
```

```
ledNormalSts = 0

# Define led pins as output
GPIO.setup(ledNormal, GPIO.OUT)

# turn leds OFF
GPIO.output(ledNormal, GPIO.LOW)

@app.route("/")
def index():
    # Read Sensors Status
    ledNormalSts = GPIO.input(ledNormal)

    templateData = {
        'title' : 'GPIO output Status!',
        'ledNormal' : ledNormalSts,
    }
    return render_template('index-actuator.html', **templateData)

@app.route("/<deviceName>/<action>")
def action(deviceName, action):
    if deviceName == 'ledNormal':
        actuator = ledNormal
    #if action == 'on':
    #    GPIO.output(actuator, GPIO.HIGH)
    if action == 'on':
        GPIO.output(actuator, GPIO.HIGH)
    if action == 'off':
        GPIO.output(actuator, GPIO.LOW)

    ledNormalSts = GPIO.input(ledNormal)

    templateData = {
        'ledNormal' : ledNormalSts,
    }
    return render_template('index-actuator.html', **templateData)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=10, debug=True)
```

Creating the HTML File

Keeping HTML tags separated from your Python script is how you keep your practical organized.

Flask uses a template engine called Jinja2 that you can use to send dynamic data from your Python script to your 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 *index-actuator.html*.

```
pi@raspberrypi:~/web-server/templates $ index-actuator.html
```

```
<!DOCTYPE html>

<head>

    <title>GPIO Control</title>

    <link rel="stylesheet" href='../static/style.css'/>

</head>

<body>

    <h1>Actuators</h1>

    <h2> Status </h2>

    <h3> LED ==> {{ ledNormal }}</h3>

    <br>

    <h2> Commands </h2>

    <h3>
        Normal LED Ctrl ==>

        <a href="/ledNormal/on" class="button">LED ON</a>

        <a href="/ledNormal/off" class="button">LED OFF</a>

    </h3>

</body>

</html>
```

One more file to be created as style.css in a folder static within web-server folder

```
body {
    background: lightgrey;
    color: black;
```

```
}  
  
.button {  
  font: bold 15px Arial;  
  text-decoration: none;  
  background-color: #AAEEAA;  
  color: #333333;  
  padding: 2px 6px 2px 6px;  
  border-top: 1px solid #CCCCCC;  
  border-right: 1px solid #333333;  
  border-bottom: 1px solid #333333;  
  border-left: 1px solid #CCCCCC;  
}
```

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 ..
```

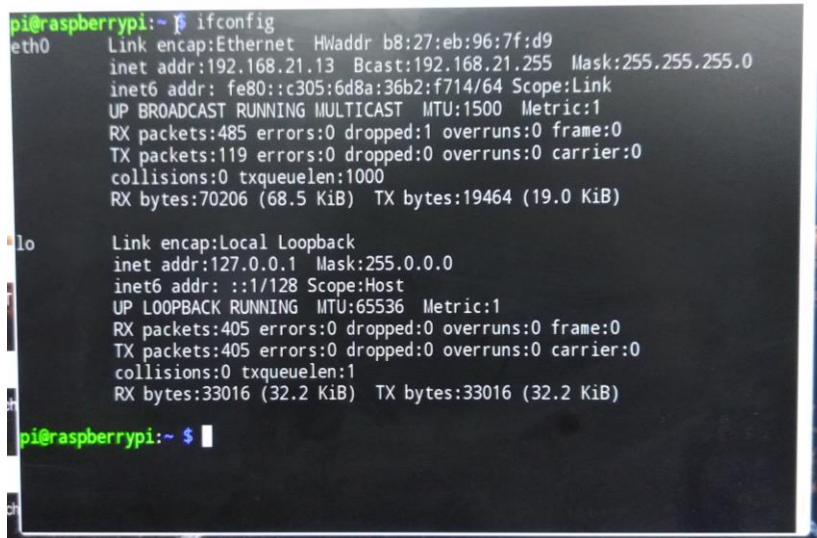
Then run the following command:

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

Your web server should start immediately on the terminal!

Now check the Inet address of your device by ifconfig.

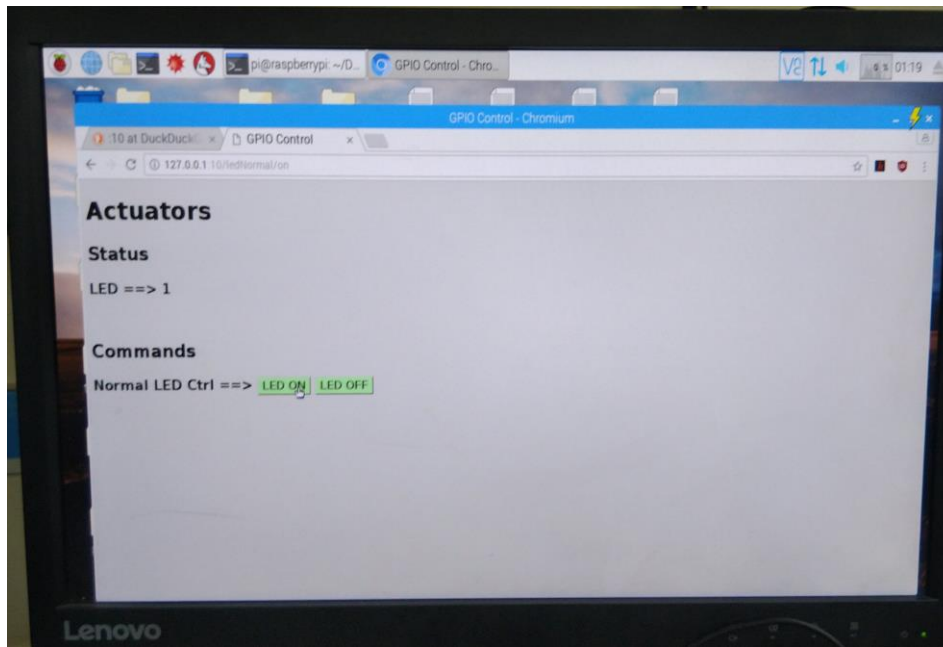
It may be- 127.0.0.1



```
pi@raspberrypi:~$ ifconfig  
eth0      Link encap:Ethernet  HWaddr b8:27:eb:96:7f:d9  
          inet addr:192.168.21.13  Bcast:192.168.21.255  Mask:255.255.255.0  
          inet6 addr: fe80::c305:6d8a:36b2:f714/64 Scope:Link  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:485 errors:0 dropped:1 overruns:0 frame:0  
          TX packets:119 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:70206 (68.5 KiB)  TX bytes:19464 (19.0 KiB)  
  
lo        Link encap:Local Loopback  
          inet addr:127.0.0.1  Mask:255.0.0.0  
          inet6 addr: ::1/128 Scope:Host  
          UP LOOPBACK RUNNING  MTU:65536  Metric:1  
          RX packets:405 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:405 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1  
          RX bytes:33016 (32.2 KiB)  TX bytes:33016 (32.2 KiB)  
  
pi@raspberrypi:~$
```

Demonstration

Type the same on chromium web browser task bar as 127.0.0.1:10 the webserver should display like following.



Once you click on LED on button. It will glow the LED.

