

## Lab session week 7: IoT Programming with Arduino and Websites

### Aim

The aim of this tutorial is for students to be able to confidently control edge devices from a website.

### Important Information

- Please use your own laptop.
- If your laptop does not have a USB type A socket, you will need to bring an adaptor.
- Be gentle with the hardware.
- You can work with a physical Raspberry Pi board or a virtual desktop to complete this task.
- Do not connect your laptop or a physical Raspberry Pi via Ethernet to the Swinburne network.
- If you are using a physical Raspberry Pi, you will need to find a way to connect it to your laptop without using the Swinburne network.

### Task 1: Getting ready the edge device

Write a program in Arduino that turns ON or OFF two LEDs independently depending on the data received via serial communication. For example, the first LED will turn on if the Arduino board receives a 1, and it will turn OFF if a 2 is received. You can wire the two LEDs as shown in Figure 1.

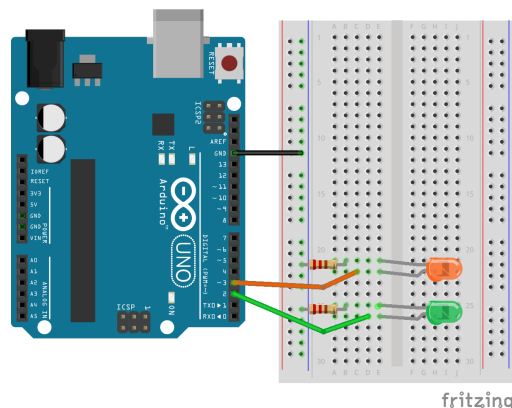


Figure 1: Wiring two LEDs

Test your code using the serial monitor of the Arduino IDE.

## Task 2: Getting ready Flask in the edge server

It has been a while since we installed Raspberry Pi OS, let's update the package list and upgrade the system.

```
sudo apt-get update
sudo apt-get upgrade
```

Most likely flask will be already installed; if not, you can install flask with the following command:

```
sudo apt-get install python-flask
```

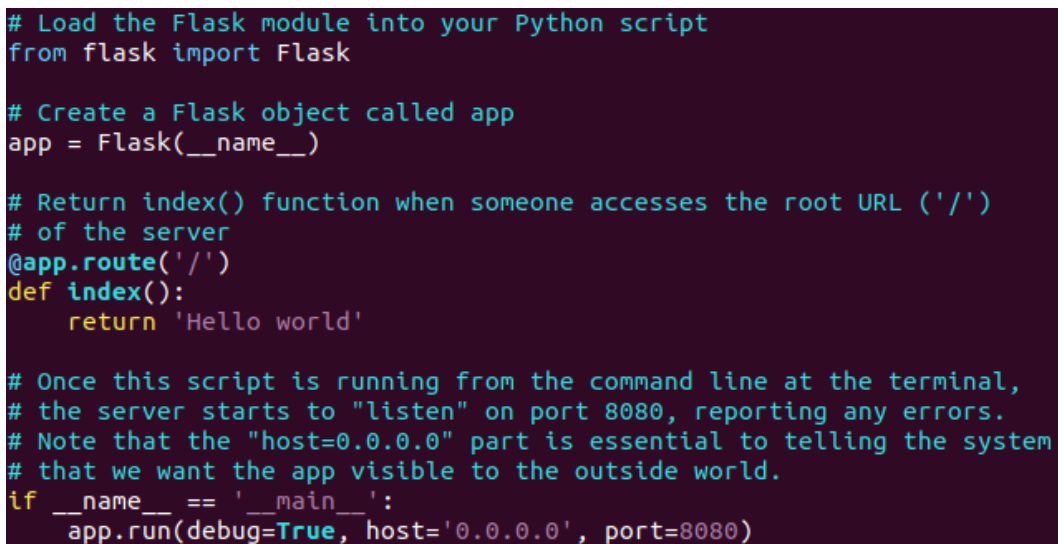
Now we need to create a folder to store the content of the website. Create a folder, and enter in that folder with the following commands:

```
mkdir web-server
cd web-server
```

We can test flask with a hello-world website. Create a file named flask-helloworld.py, and type the code in Figure 2:

```
nano flask-helloworld.py
```

nano is a command line text editor, use `Ctrl + o` to save, and `Ctrl + x` to exit.

A screenshot of a terminal window with a dark background and light-colored text. The text shows the code for a Flask application. It starts with a comment and an import statement for Flask. Then it creates a Flask object named 'app'. Next, it defines a route for the root URL '/' with a function 'index()' that returns 'Hello world'. There are more comments explaining the server's operation and the importance of the host parameter. Finally, it has a main block that runs the app on port 8080 with debug mode enabled.

```
# Load the Flask module into your Python script
from flask import Flask

# Create a Flask object called app
app = Flask(__name__)

# Return index() function when someone accesses the root URL ('/')
# of the server
@app.route('/')
def index():
    return 'Hello world'

# Once this script is running from the command line at the terminal,
# the server starts to "listen" on port 8080, reporting any errors.
# Note that the "host=0.0.0.0" part is essential to telling the system
# that we want the app visible to the outside world.
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=8080)
```

Figure 2: Flask code for Hello-World

Once you have coded the hello world website, you can test it executing the following command (`Ctrl + c` to close it):

```
python flask-helloworld.py
```

If everything is working fine, you should see something similar to Figure 3 when typing localhost URL in the Raspberry Pi OS browser: `localhost:8080`.

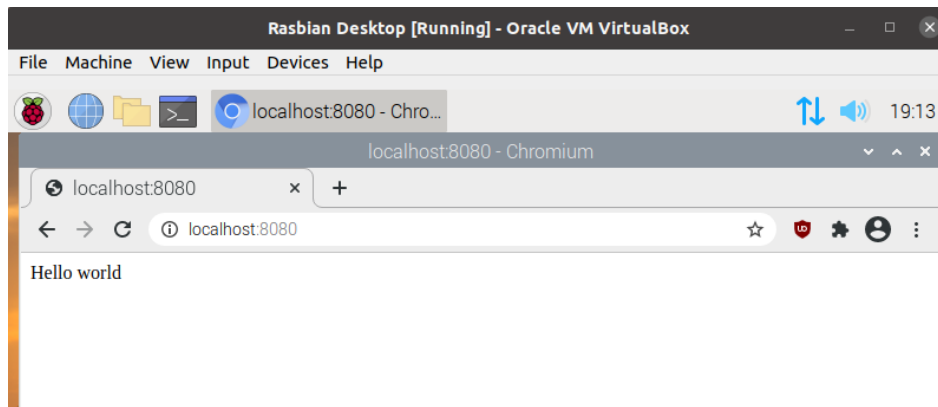


Figure 3: Hello-World website from Raspbian

Alternatively, you can type the ip address of your Raspberry Pi in the browser of your computer and see something like Figure 4: `192.168.1.181:8080`.

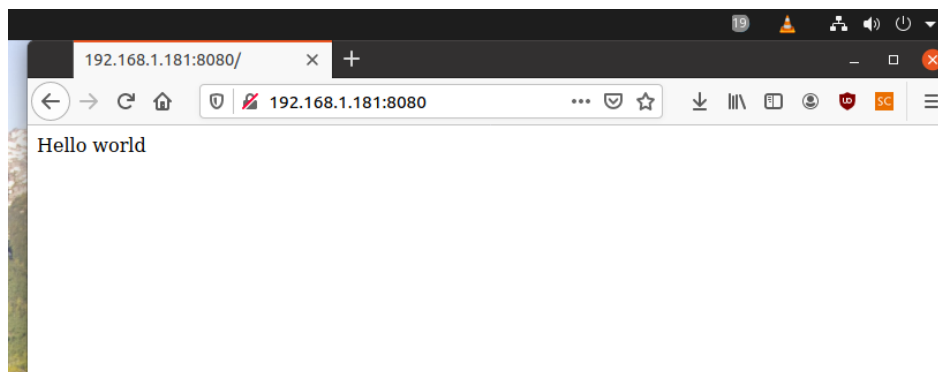


Figure 4: Hello-World website from my computer

### Task 3: Programming Flask website to control Arduino

Alright, it is time to develop the website that will control Arduino. We will edit a file named `led_control.py` or something similar that you like.

```
nano led_control.py
```

We will insert the code in Figures 5, 6 and 7. We are configuring serial data communication in the main function (Figure 7) at 9600 baud in port `/dev/ttyS0`, make sure those are the correct values for your configuration. This time the website will be hosted at default port 80.

```

import serial
import time
from flask import Flask, render_template

app = Flask(__name__)

# Dictionary of pins with name of pin and state ON/OFF
pins = {
    2 : {'name' : 'PIN 2', 'state' : 0},
    3 : {'name' : 'PIN 3', 'state' : 0}
}

# Main function when accessing the website
@app.route("/")
def index():
    # TODO: Read the status of the pins ON/OFF and update dictionary

    # This data will be sent to index.html (pins dictionary)
    templateData = {
        'pins' : pins
    }
    # Pass the template data into the template index.html and return it
    return render_template('index.html', **templateData)

```

Figure 5: Flask first snippet - imports, dictionary, main website function

```

# Function with buttons that toggle depending on the status
@app.route("/<changePin>/<toggle>")
def toggle_function(changePin, toggle):
    # 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 toggle == "on":
        # Set the pin high:
        if changePin == 2:
            ser.write(b"1")
            pins[changePin]['state'] = 1
        if changePin == 3:
            ser.write(b"3")
            pins[changePin]['state'] = 1
        # Save the status message to be passed into the template:
        message = "Turned " + deviceName + " on."
    if toggle == "off":
        if changePin == 2:
            ser.write(b"2")
            pins[changePin]['state'] = 0
        if changePin == 3:
            ser.write(b"4")
            pins[changePin]['state'] = 0
        # Set the pin low:
        message = "Turned " + deviceName + " off."

    # This data will be sent to index.html (pins dictionary)
    templateData = {
        'pins' : pins
    }
    # Pass the template data into the template index.html and return it
    return render_template('index.html', **templateData)

```

Figure 6: Flask second snippet - cool function that toggles LEDs

```

# Function to send simple commands
@app.route("/<action>")
def action(action):
    if action == 'action1':
        ser.write(b"1")
        pins[2]['state'] = 1
    if action == 'action2':
        ser.write(b"2")
        pins[2]['state'] = 0
    if action == 'action3':
        ser.write(b"3")
        pins[3]['state'] = 1
    if action == 'action4':
        ser.write(b"4")
        pins[3]['state'] = 0
    # This data will be sent to index.html (pins dicctionary)
    templateData= {
        'pins' : pins
    }
    # Pass the template data into the template index.html and return it
    return render_template('index.html', **templateData)

# Main function, set up serial bus, indicate port for the webserver,
# and start the service.
if __name__ == "__main__":
    ser = serial.Serial('/dev/ttyS0',9600, timeout=1)
    ser.flush()
    app.run(host='0.0.0.0', port = 80, debug = True)

```

Figure 7: Flask third snippet - simple function to turn ON/OFF LEDs and main function

As you can see, our code is using Flask templates in order to update the dynamic data of the website (e.g., if the LED is ON or OFF). Thus, we need to add a template called index.html in templates folder with the content of Figure 8.

```

mkdir templates
nano templates/index.html

```

Before starting the service, make sure Arduino is connected correctly to your Raspberry Pi (virtual or physical). Once everything is ready, we can start again the webserver. This time though, we will start it with admin permissions (sudo command) due to hosting the website at default port 80:

```

sudo python led_control.py

```

You should see something like Figure 9. If you type `Ctrl + c` you will stop the service (the website will stop).

Figure 10 shows the website developed to control Arduino.

#### Task 4: Programming Flask website to control and receive data from Arduino

Modify the website in order to receive and display data from Arduino (e.g., potentiometer value, state of switch buttons, etc.).

```

<!DOCTYPE html>
<head>
  <title>Arduino Web Server</title>
</head>
<body>

  <h1> Arduino Web Server </h1>
  <h2> Toggle buttons </h2>
  {% for pin in pins %}
    <h3>{{ pins[pin].name }}
    {% if pins[pin].state == 1 %}
      is currently <strong>on</strong></h3><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></h3><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 %}

  <h2> Commands </h2>
  <h3> LED pin 2: <a href="/action1" >TURN_ON</a>
  <h3> LED pin 2: <a href="/action2" >TURN_OFF</a>
  <h3> LED pin 3: <a href="/action3" >TURN_ON</a>
  <h3> LED pin 3: <a href="/action4" >TURN_OFF</a>

</body>
</html>

```

Figure 8: HTML Template code

```

pi@raspberrypi:~/web-server $ sudo python led_control.py
* Serving Flask app "led_control" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 239-013-285
192.168.1.150 - - [27/Jan/2021 02:36:28] "GET / HTTP/1.1" 200 -
192.168.1.150 - - [27/Jan/2021 02:36:35] "GET /2/on HTTP/1.1" 200 -
192.168.1.150 - - [27/Jan/2021 02:36:39] "GET /3/on HTTP/1.1" 200 -
192.168.1.150 - - [27/Jan/2021 02:36:45] "GET /action2 HTTP/1.1" 200 -
192.168.1.150 - - [27/Jan/2021 02:36:49] "GET /action1 HTTP/1.1" 200 -
192.168.1.150 - - [27/Jan/2021 02:36:52] "GET /action4 HTTP/1.1" 200 -

```

Figure 9: Starting the Webserver

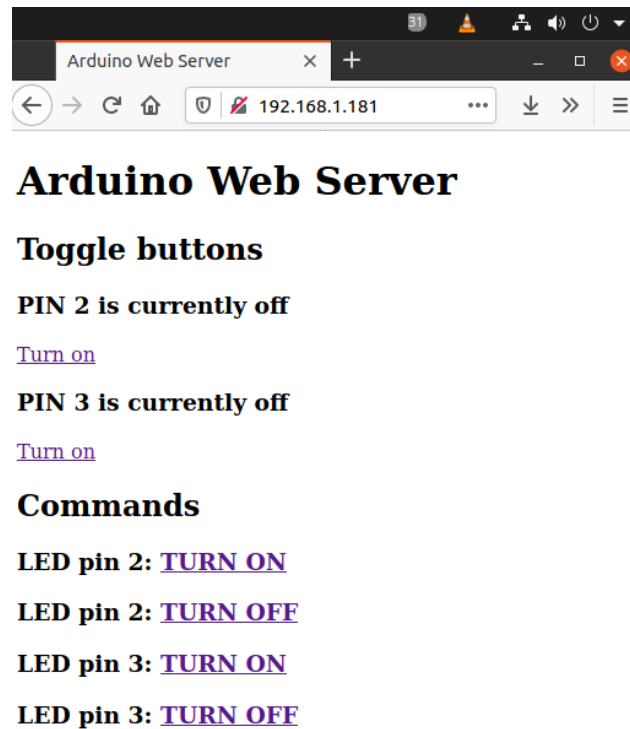


Figure 10: Website to control Arduino

## Task 5: Flask and Raspberry Pi GPIO

If you have a physical Raspberry Pi, use GPIOZero and Flask to control LEDs and sensors connected to the GPIO of the Raspberry Pi.

### Resources

#### Arduino

- Arduino Language Reference: <https://www.arduino.cc/reference/en/>
- Arduino Foundations: <https://www.arduino.cc/en/Tutorial/Foundations>
- Arduino Built-In Examples: <https://www.arduino.cc/en/Tutorial/BuiltInExamples>

#### Raspberry Pi

- <https://www.raspberrypi.org/>

#### GNU/Linux

- Linux Journey is a site dedicated to making learning Linux fun and easy. <https://linuxjourney.com/>

- Introduction to Linux: A Hands on Guide. <https://tldp.org/LDP/intro-linux/intro-linux.pdf>
- Introduction to Linux (LFS101), the Linux Foundation training course: <https://training.linuxfoundation.org/training/introduction-to-linux/>

## **Python**

- Python for Beginners (Programmers). <https://www.python.org/about/gettingstarted/>
- Flask. <https://flask.palletsprojects.com/en/1.1.x/>