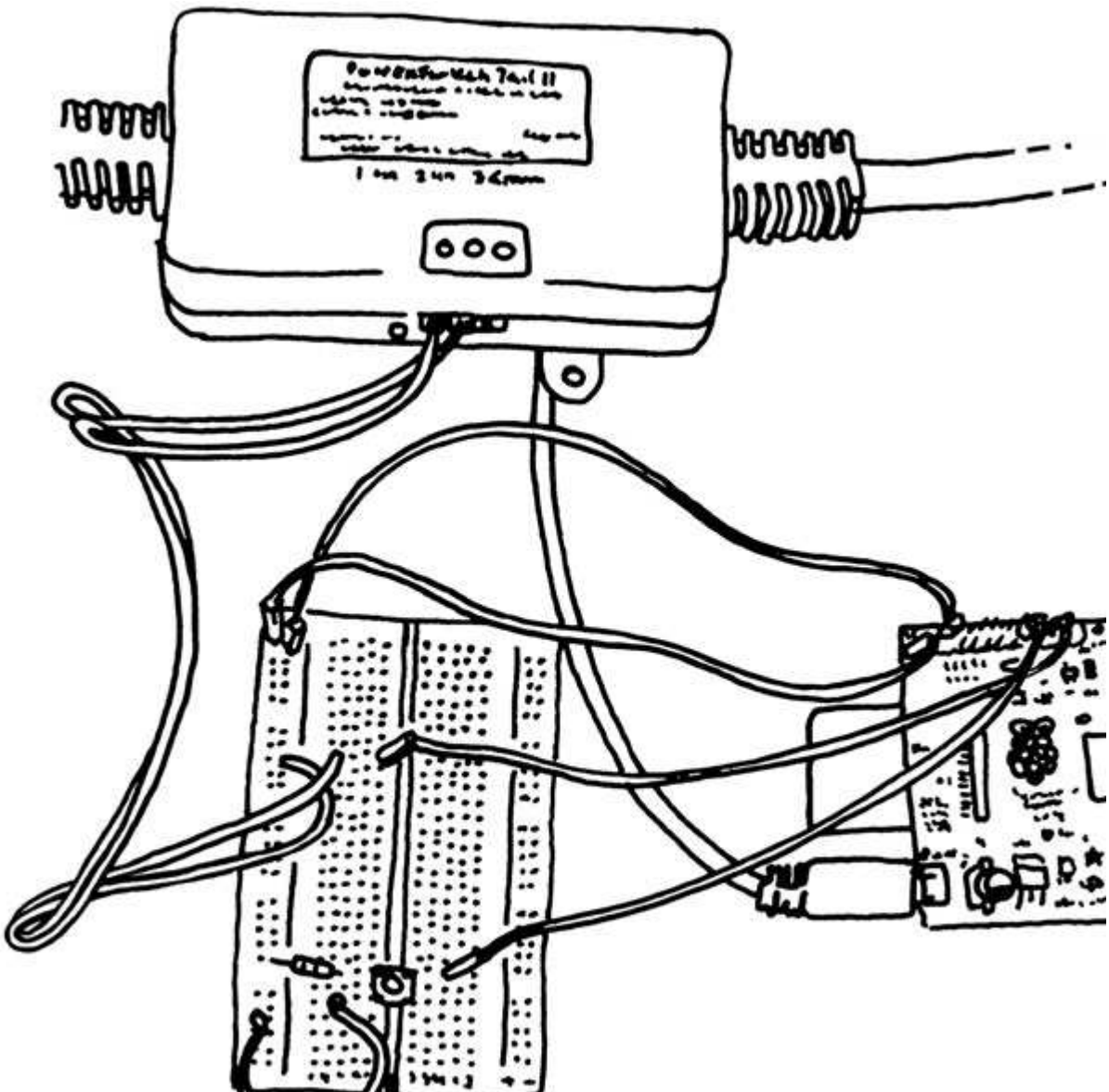
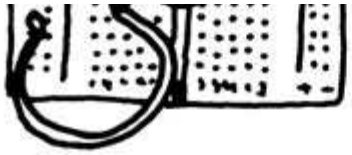
Project: WebLamp

In chapter 7 of Getting Started with Raspberry Pi, Shawn and I show you how to use Raspberry Pi as a simple AC outlet timer using the Linux job scheduler, `cron`. Now that you know how to use Python and Flask, you can now control the state of a lamp over the web. This basic project is simply a starting point for creating Internet-connected devices with the Raspberry Pi.

And just like how the previous Flask example showed how you can have the same code work on multiple pins, you'll set up this project so that if you want to control more devices in the future, it's easy to add.

Connect a Power Switch Tail II to pin 25 of the Raspberry Pi, as shown in the illustration below. If you don't have a Power Switch Tail, you can connect an LED to the pin to try this out for now.





If you have another PowerSwitch Tail II relay, connect it to pin 24 to control a second AC device. Otherwise, just connect an LED to pin 24. We're simply using it to demonstrate how multiple devices can be controlled with the same code.

Create a new directory in your home directory called `webLamp`.

In `webLamp`, create a file called `weblamp.py` and put in the code from below:

weblamp.py

```
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 = {
   24 : {'name' : 'coffee maker', 'state' : GPIO.LOW},
   25 : {'name' : 'lamp', '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
@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":
```

```
    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."
    if action == "toggle":
       # Read the pin and set it to whatever it isn't (that is, toggle it):
       GPIO.output(changePin, not GPIO.input(changePin))
       message = "Toggled " + deviceName + "."

    # 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 = {
       'message' : message,
       'pins' : pins
    }

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

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

Since there's a lot going on in this code, I've placed on the explanations inline, as comments.

Create a new directory within `WebLamp` called `templates`.

Inside `templates`, create the file `main.html`:

`main.html`

```
<!DOCTYPE html>
<head>
   <title>Current Status</title>
</head>

<body>
   <h1>Device Listing and Status</h1>

   {% for pin in pins %}
   <p>The {{ pins[pin].name }}
   {% if pins[pin].state == true %}
      is currently on (<a href="/{{pin}}/off">turn off</a>)
   {% else %}
      is currently off (<a href="/{{pin}}/on">turn on</a>)
   {% endif %}
```

```
    {% else %}
       is currently off (<a href="/{{pin}}/on">turn on</a>)
    {% endif %}
    </p>
    {% endfor %}

    {% if message %}
    <h2>{{ message }}</h2>
    {% endif %}

</body>
</html>
```

There are two templating concepts being illustrated in this file. First:

```
{% for pin in pins %}
<p>The {{ pins[pin].name }}
{% if pins[pin].state == true %}
   is currently on (<a href="/{{pin}}/off">turn off</a>)
{% else %}
   is currently off (<a href="/{{pin}}/on">turn on</a>)
{% endif %}
</p>
{% endfor %}
```

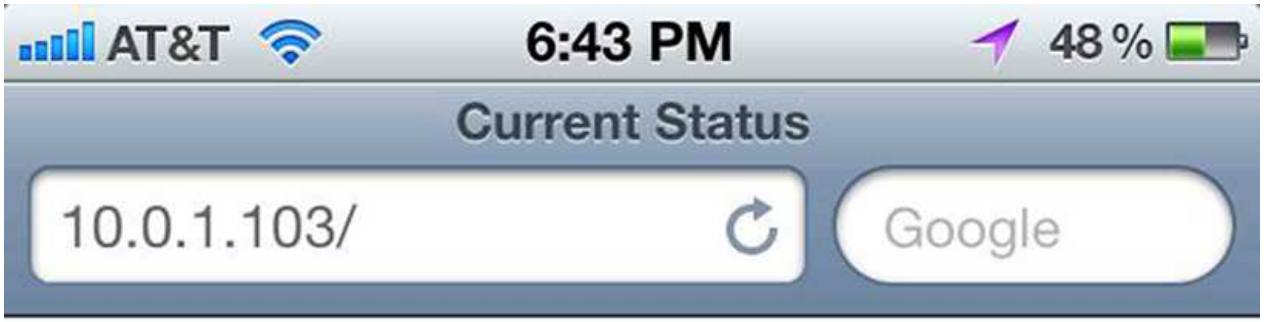
This sets up a for loop to cycle through each pin, print its name and its state. It then gives the option to change its state, based on its current state. Second:

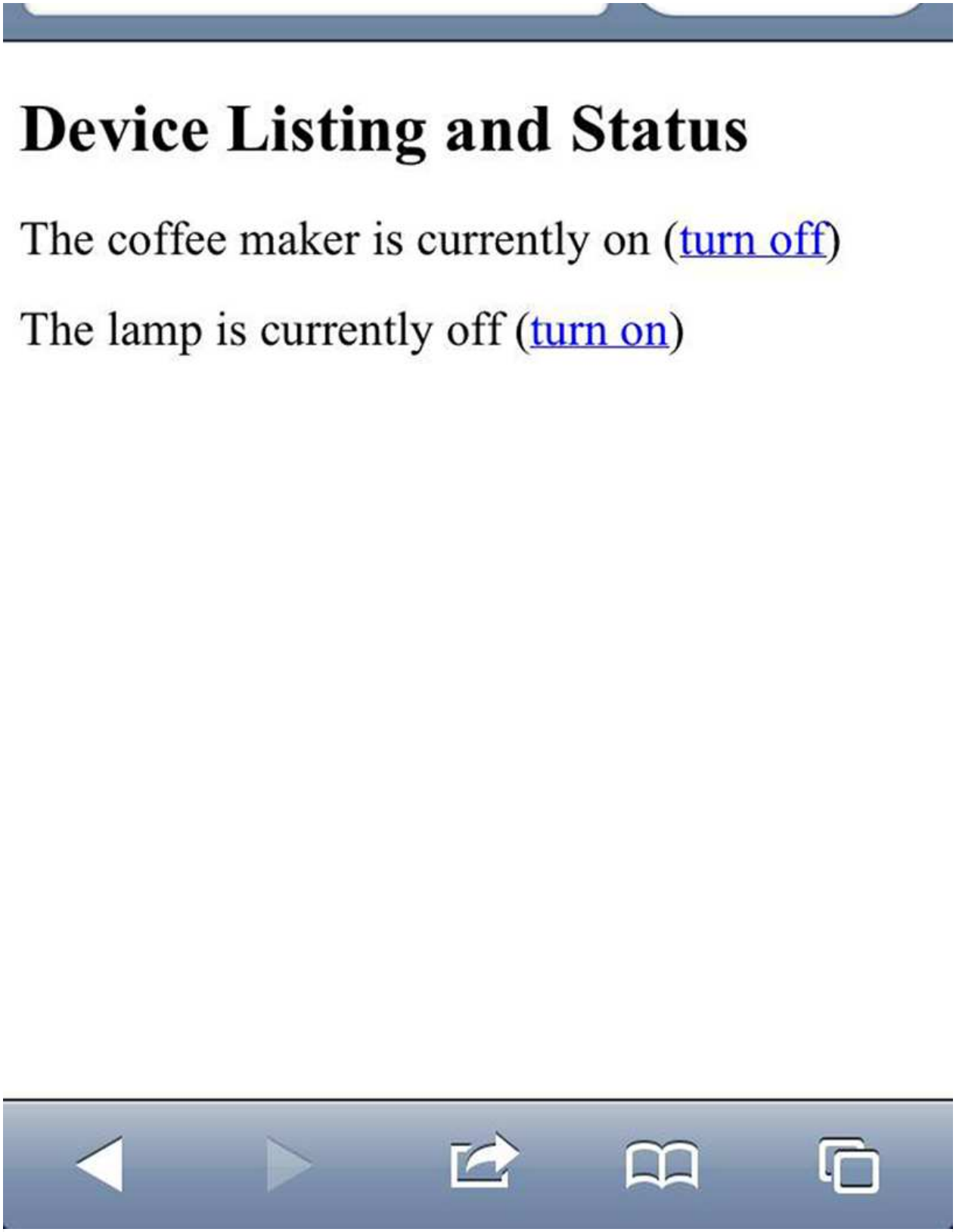
```
    {% if message %}
    <h2>{{ message }}</h2>
    {% endif %}
```

This if statement will print a message passed from your Python script if there's a message set (that is, a change happened).

In terminal, navigate to the `WebLamp` directory and start the server (be sure to use Control-C to kill any other Flask server you have running first):

```
pi@raspberrypi ~/WebLamp $ sudo python weblamp.py
```





The best part about writing the code in this way is that you can very easily add as many devices that the hardware will support. Simply add the information about the device to the pins dictionary. When you restart the server, the new device will appear in the status list and its control URLs will work automatically.

There's another great feature built in: if you want to be able to flip the switch on a device with a single tap on your phone, you can create a bookmark to the address `http://ipaddress`

There's another great feature built in: if you want to be able to flip the switch on a device with a single tap on your phone, you can create a bookmark to the address `http://ipaddress/pin/toggle`. That URL will check the pin's current state and switch it.