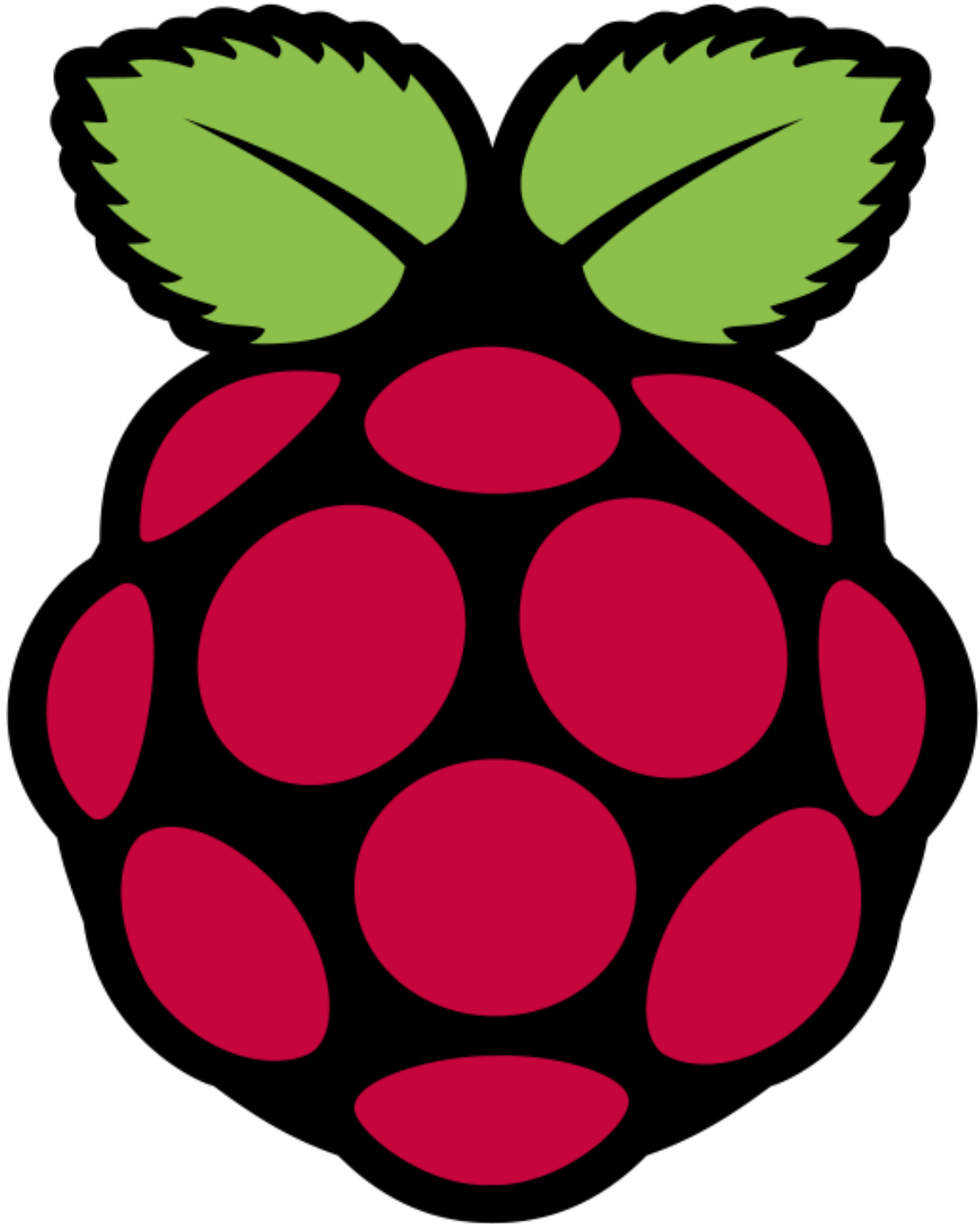


Raspberry Pi



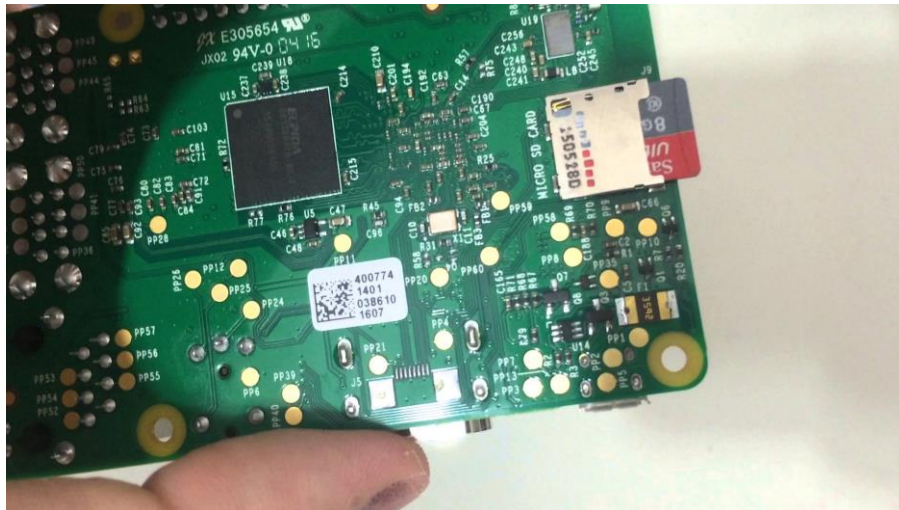
Antonio Manuel Núñez Domínguez

Introducción

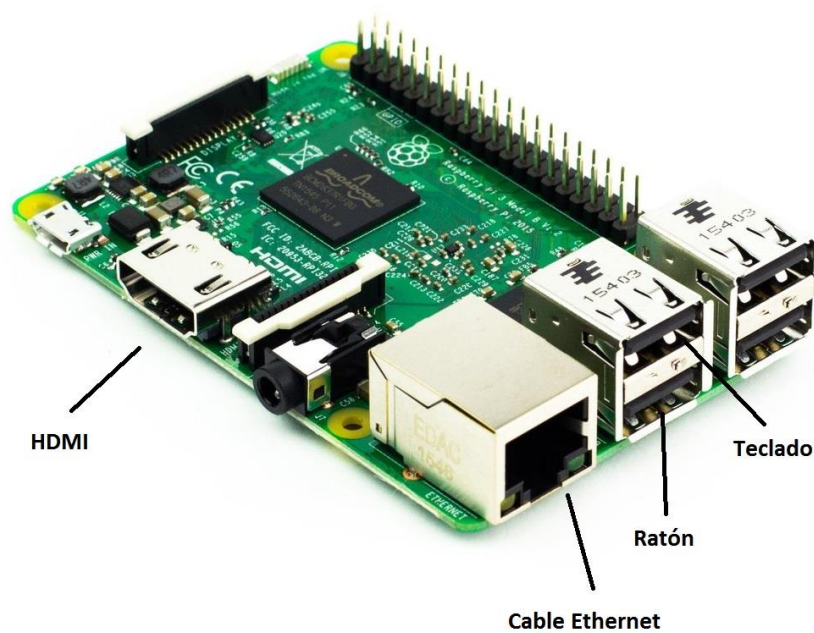
En estas prácticas, se va a trabajar con una Raspberry Pi 3. Raspberry es una placa de reducido tamaño, basada en un computador. Sus características permiten ejecutar un sistema operativo entero. Hacer que ejecute un SO va a ser la primera parte de estas prácticas.

Sesión 1 de laboratorio

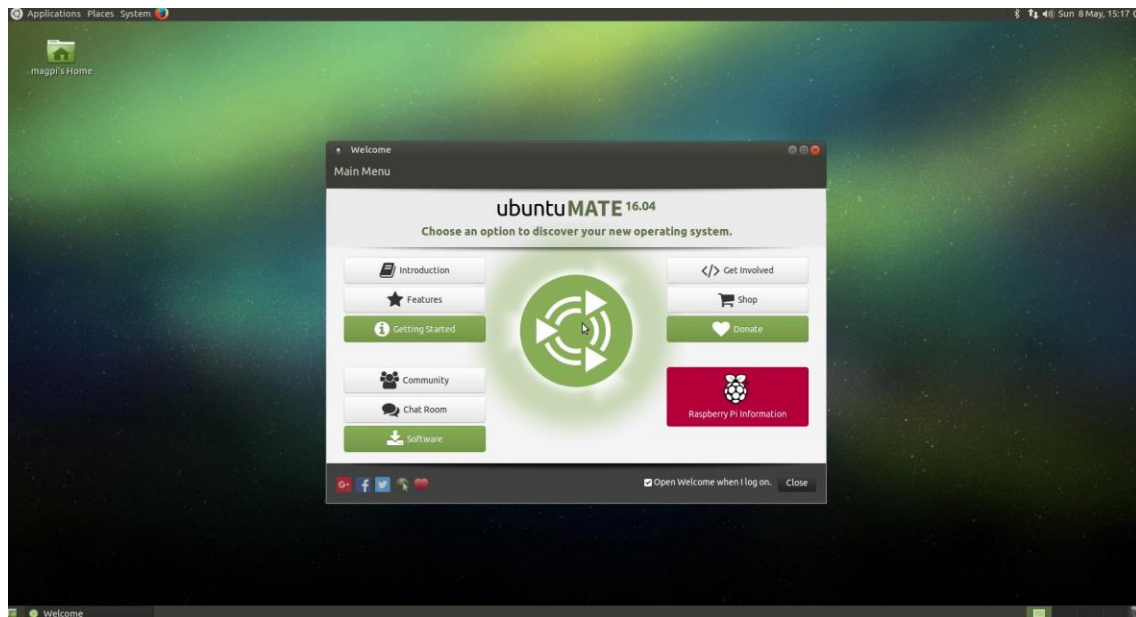
El primer paso es descargar la imagen de UBUNTU MATE, el sistema operativo basado en Linux. La imagen del SO se introdujo en una tarjeta SD. Esta tarjeta SD se introduce en ella ranura para tarjetas de la placa de Raspberry.



Una vez que se introduce la tarjeta SD, se deben conectar los cables de Ethernet, ratón, teclado, y HDMI.



Cuando este todo listo y se alimente la placa vía USB, arrancará el sistema y se podrá configurar.



Por supuesto, hay que configurar la conexión a internet. La primera vez se hizo como viene en el pdf de las prácticas de laboratorio. En las siguientes sesiones se tuvo que hacer de otra manera, ya que no siempre cogíamos la misma placa, por lo que la MAC cambiaba y teníamos que configurarlo de nuevo.

Tarea 1 – Manejo de GPIOs

Esta tarea consiste en manejar el encendido y apagado de LEDs a través de la línea de comandos:

Primero tenemos que crear un pin mediante el siguiente comando:

```
echo 17 > /sys/class/gpio/export
```

A continuación se informa a la Raspberry PI si el pin va a ser de salida:

```
echo out > /sys/class/gpio/gpio17/direction
```

Para encender el LED utilizaremos:

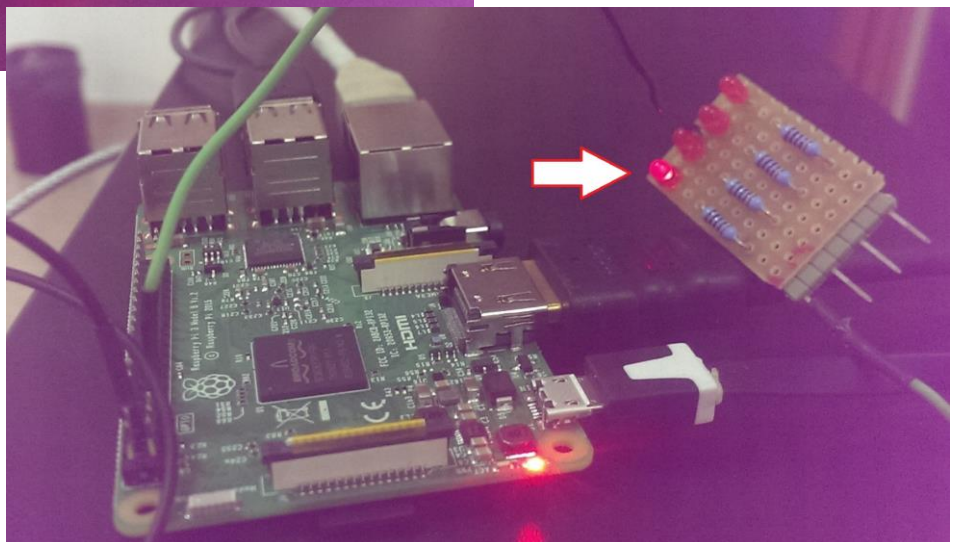
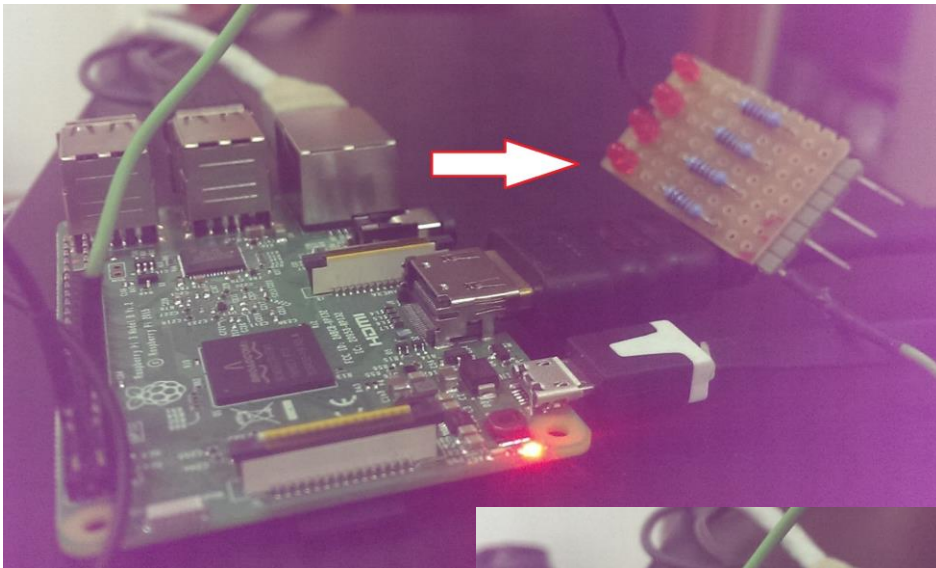
```
echo 1 > /sys/class/gpio/gpio17/value
```

Para apagar el LED utilizaremos:

```
echo 0 > /sys/class/gpio/gpio17/value
```

Para eliminar la entrada GPIO se utiliza:

```
echo 17 > /sys/class/gpio/unexport
```



Tarea 2 – Control de LEDs con Python

En esta tarea el objetivo es controlar 2 LEDs mediante código Python. Para ello, el primer paso es instalar la librería Rpi.GPIO para poder controlar los GPIO con Python. En nuestro caso, se está utilizando Ubuntu Mate, el cual ya lo trae instalado.

El siguiente paso consiste en crear un fichero blink.py, en el que se va a introducir el código que permitirá controlar los LEDs. Este código encenderá y apagará alternativamente los LEDs:

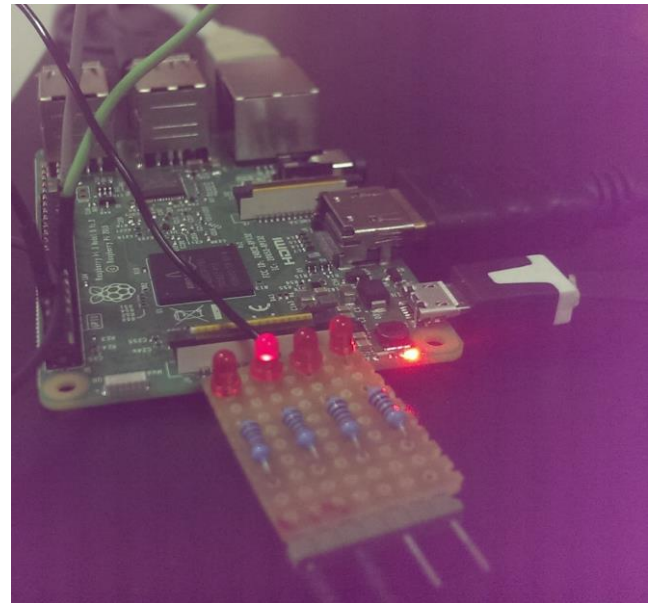
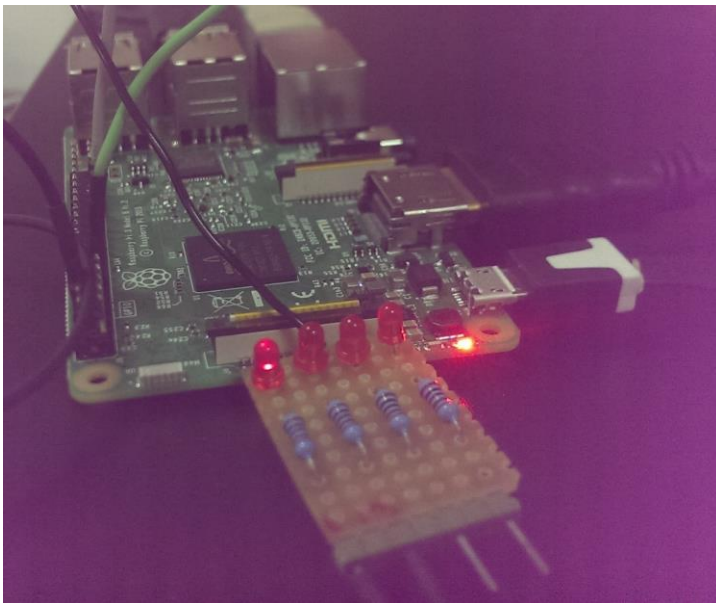
```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT) ## GPIO 17 como salida
GPIO.setup(27, GPIO.OUT) ## GPIO 27 como salida

def blink():
    print "Ejecucion iniciada..."
    iteracion = 0
    while iteracion < 30: ## Segundos que durara la funcion
        GPIO.output(17, True) ## Enciendo el 17
        GPIO.output(27, False) ## Apago el 27
        time.sleep(1) ## Esperamos 1 segundo
        GPIO.output(17, False) ## Apago el 17
        GPIO.output(27, True) ## Enciendo el 27
        time.sleep(1) ## Esperamos 1 segundo
        iteracion = iteracion + 2 ## Sumo 2 porque he hecho dos parpadeos
    print "Ejecucion finalizada"
    GPIO.cleanup() ## Hago una limpieza de los GPIO

blink() ## Hago la llamada a la funcion blink
```

Después se ejecuta desde la consola el siguiente comando:

```
sudo python blink.py
```



También se puede ver el funcionamiento desde el siguiente vídeo haciendo Ctrl + Click [aquí](#)

Tarea 3 – Montaje de servidor web

Para realizar esta tarea, comenzaremos creando una carpeta “raíz” en la cual se añadirán los códigos en Python. Además en esta carpeta crearemos otra, llamada “templates” en la cual se creará un archivo llamado main.html.

Primero se comprueba que funciona el crear el servidor web con este sencillo código en Python:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

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

Una vez que se haya comprobado que funciona correctamente, se puede seguir con algo un poco más complicado, como por ejemplo mostrar la hora del sistema. Para ello, se necesita el siguiente código en Python:

```
from flask import Flask, render_template
import datetime
app = Flask(__name__)

@app.route("/")
def hello():
    now = datetime.datetime.now()
    timeString = now.strftime("%Y-%m-%d %H:%M")
    templateData = {
        'title': 'HELLO!',
        'time': timeString
    }
    return render_template('main.html', **templateData)

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

En la carpeta “templates” se añadirá este código HTML a main.html

```
<!DOCTYPE html>
<head>
    <title>{{ title }}</title>
</head>

<body>
    <h1>Hello, World!</h1>
    <h2>The date and time on the server is: {{ time }}</h2>
</body>
</html>
```

Tarea 4 – WebLamp

En esta parte se controlará desde el servidor web el encendido y apagado de dos LEDs.

Primero se hace el código en Python en el fichero 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' : 'LED 1', 'state' : GPIO.LOW},
    25 : {'name' : 'LED 2', '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
@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."
    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)
```


El código HTML que se escribe en main.html es el siguiente:

```
<!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 %}
  </p>
  {% endfor %}

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

</body>
</html>
```

Se iniciará el servidor desde la línea de comandos, y si todo funciona correctamente, se podrá controlar el encendido y apagado de los LEDs desde el navegador, como se muestra en el vídeo siguiente, haciendo Ctrl + Click [aquí](#).

Tarea 4 – WebLamp y sensor de temperatura

En esta parte se mostrará la temperatura recogida por el sensor **tmp36GZ**, mediante la placa Arduino, que pasará los datos vía serie.

El código en Arduino es el siguiente:

```
int LDRPin = 0;
int boton = 8;

void setup () {
    Serial.begin(9600);
}

void loop () {
    if(Serial.available()) {

        char c = Serial.read();
        int value = analogRead(LDRPin);
        float millivolts = (value / 1023.0) * 5000;
        float celsius = millivolts / 10;
        Serial.print(celsius);
        Serial.println(" C");
    }
}
```

Hay que añadir la siguiente función al código Python de webLamp:

```
@app.route("/<temper>")
def leerTemp(temper):

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

    if temper == "L":

        arduino.write("N")
        temperatura[1]['temp'] = ""
        time.sleep(0.8)
        while arduino.inWaiting() != 0:
            temperatura[1]['temp'] += arduino.read(1)

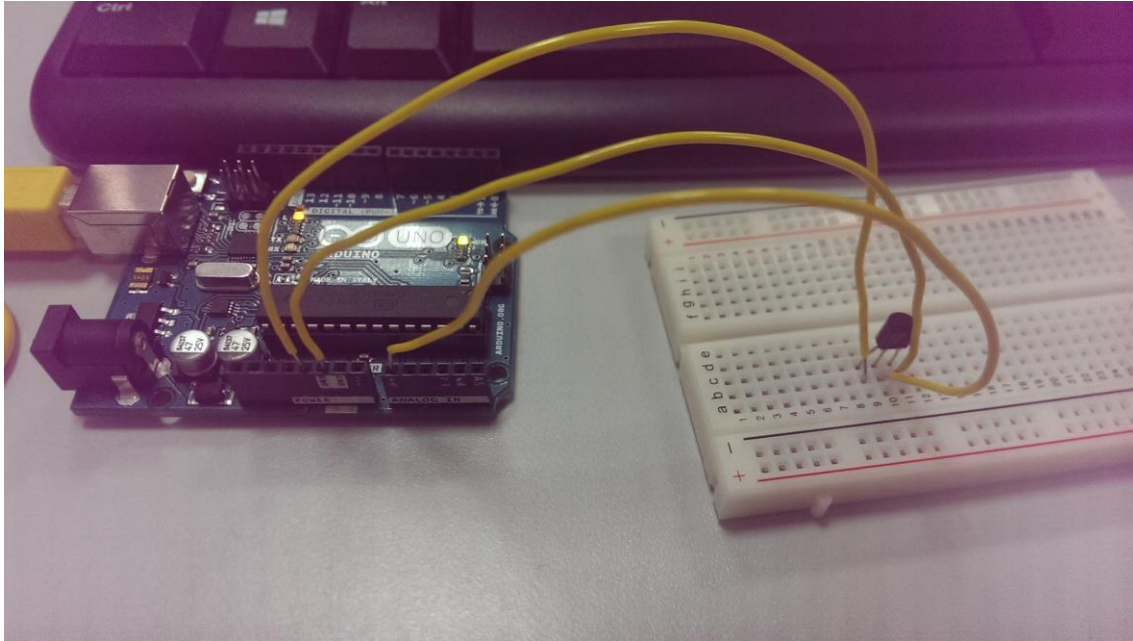
    templateData = {
        'temperatura' : temperatura,
        'pins' : pins
    }

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

Por último añadiríamos una simple línea al código main.html

```
<h2>Actualiza temperatura: (<a href="/L">Leer</a>)</h2>  
El valor es : {{temperatura}}
```

Aquí se puede ver la conexión del sensor de temperatura:



Y se puede ver el funcionamiento de esta tarea haciendo Ctrl + Click [aquí](#).