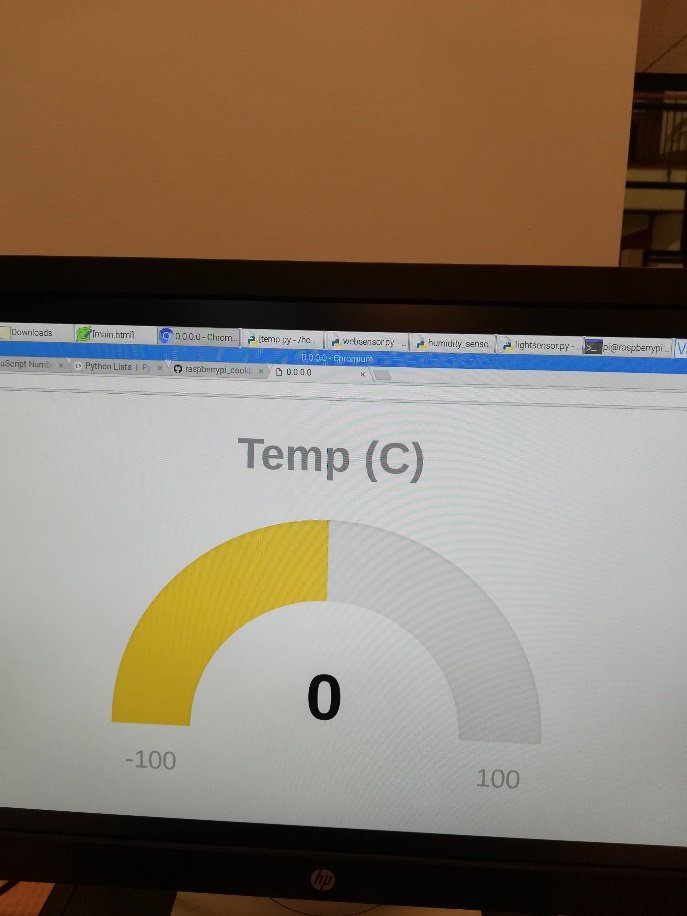
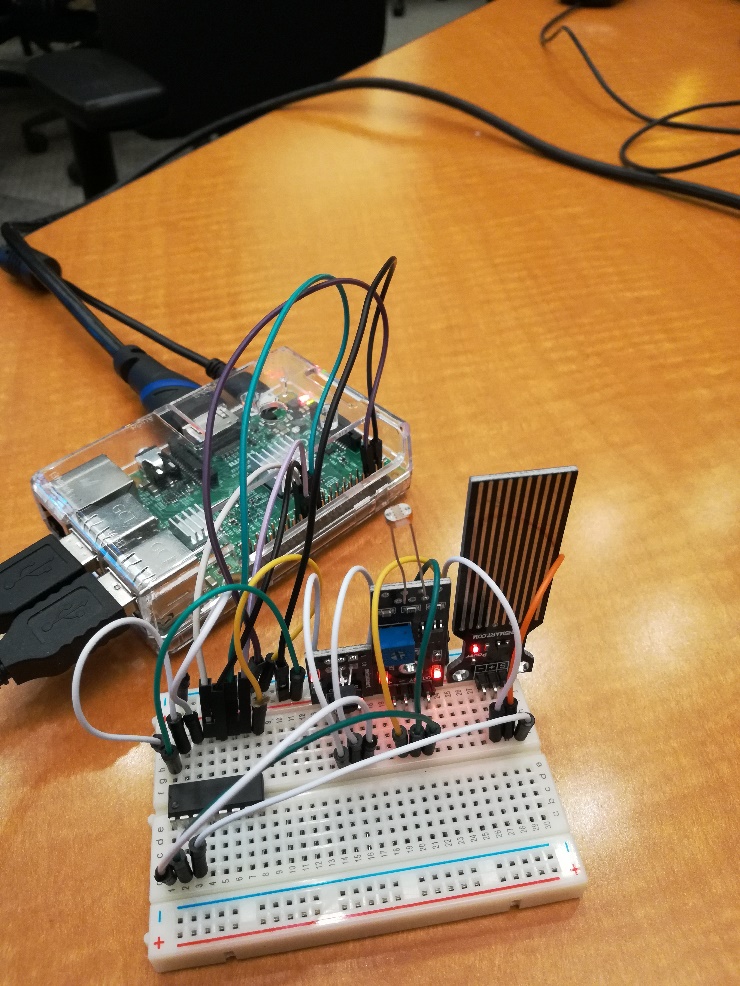
**Raspberry Pi Weather Station**

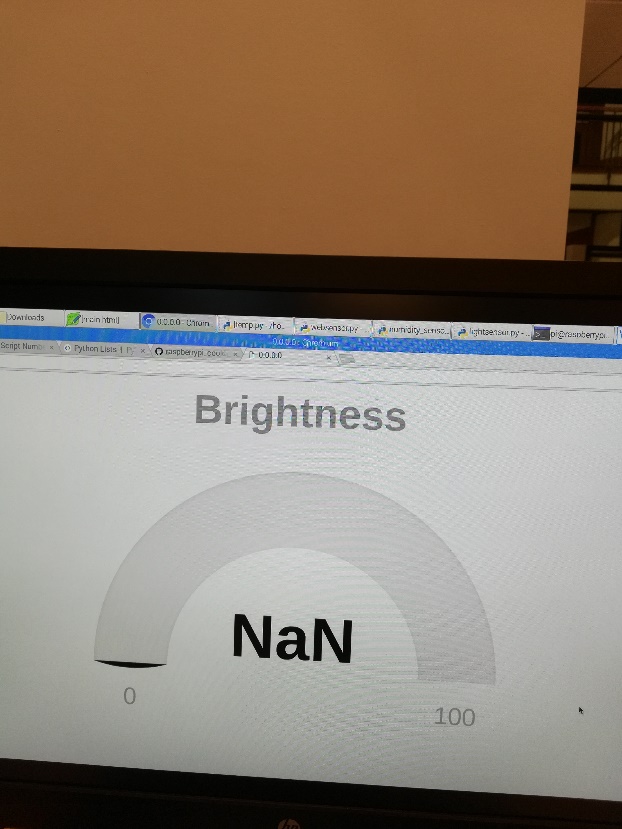
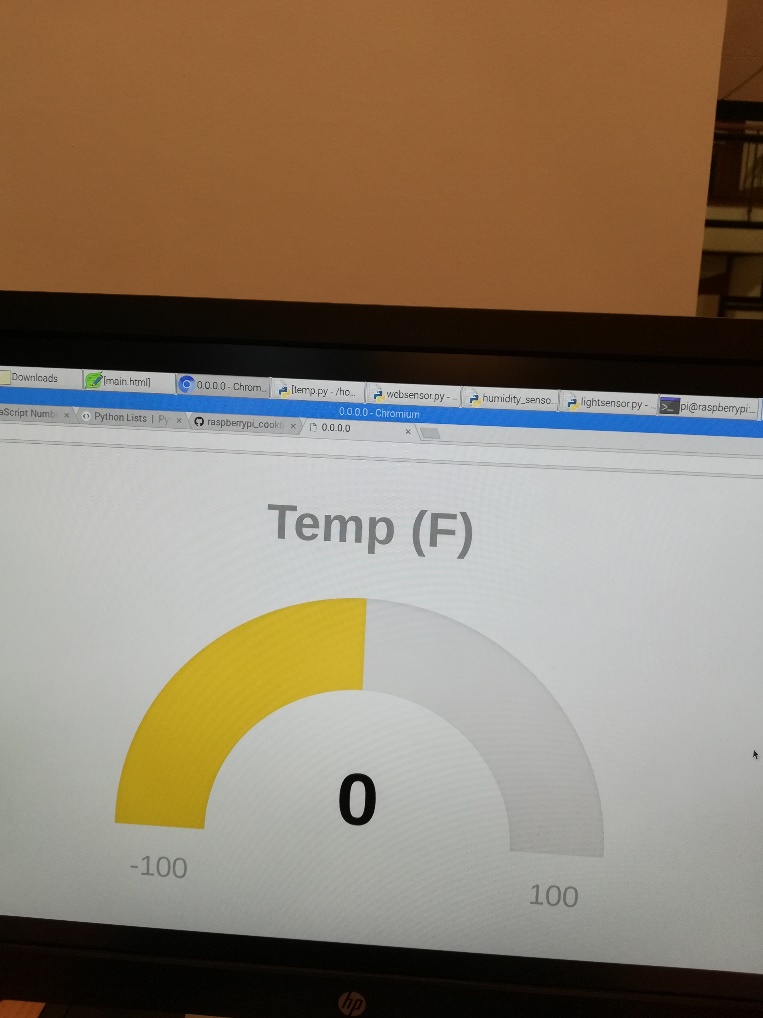
Kevin Sass

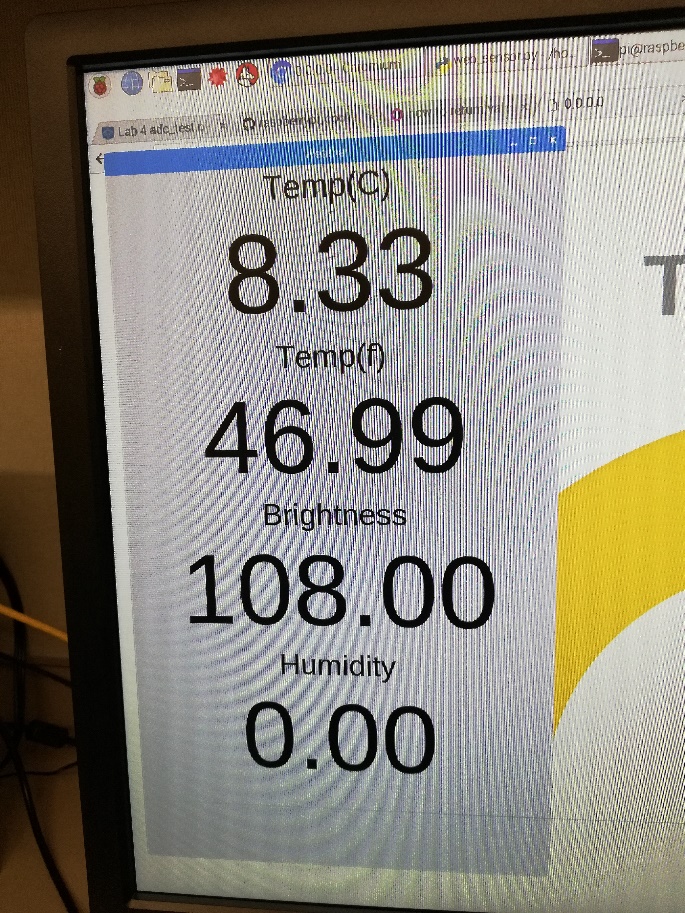
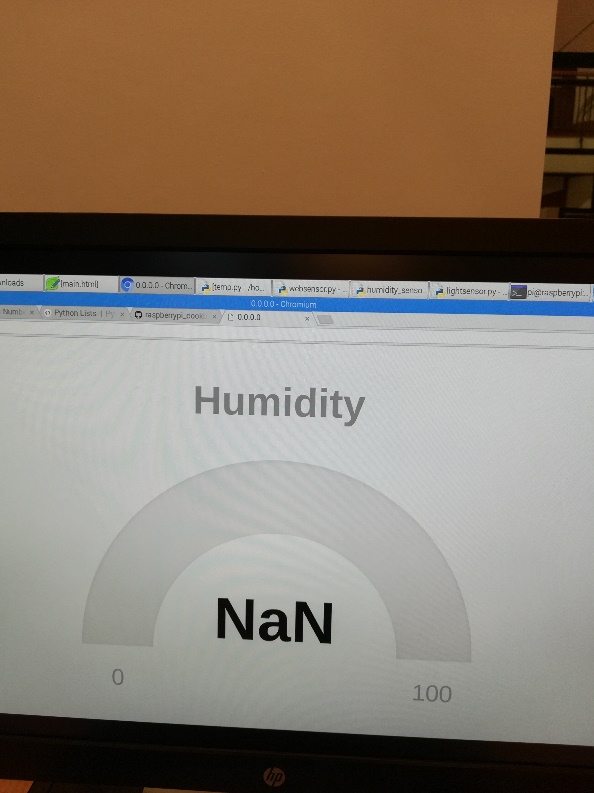
Corey Zheng

We wanted to create a weather station out of our Raspberry Pi that could detect temperature, humidity, and light/brightness in real time using sensors and then display the values onto the web to be viewed from a computer or phone, allowing for convenient viewing. We were able to create a GUI interface and display the sensor values on our screen. However, we could not figure out why our web page would not display those same sensor values. We edited the HTML code correctly, but it still would not work. Only the default HTML code from the recipe worked which displayed the CPU temperature. Even the professor could not figure out why it wouldn’t work and stated that it did not make sense.

**Finished Project**

****

****

****

**Project Objectives**

|  |  |
| --- | --- |
| **Proposal Objectives** | **Were These Objectives Met?** |
| Detect & display temperature | At first yes, but now no |
| Detect & display brightness | At first yes, but now no |
| Detect & display humidity | At first yes, but now no |
| Use a web page to display values | Yes |
| Create a live updating chart on web page | No, we replaced the live chart display with a live gauge display learned from class but edited it a bit |

**Project Approach**

We implemented the project by first redoing the temperature and brightness exercises from our previous labs and applying the same methods to implement the humidity sensor. We used a MCP3008 Analog-to-Digital converter, used channels 0 – 2 on it for the three sensors, and combined all 3 sensors and circuits on one breadboard. We then created a web page (learned from our previous lab) and edited the code to display the temperature, brightness, and humidity values on it with a visual aid of live updating gauges. When we noticed that the web page would not upload the values, we created a GUI interface instead, which correctly displayed the sensor values.

The recipes we used were 13.2 (Measuring Light), 9.5 (Setting Up SPI), 13.5 (Measuring a Voltage), 13.8 (Measuring Temperature with an ADC), 13.19 (Displaying Sensor Values), 7.17 (Writing a Simple Web Server in Python), and finally 15.2 (Displaying Sensor Readings on a Web Page).

We thought this project idea wouldn’t be too difficult but we ended up facing many challenges. Our first challenge was we couldn’t get the values displayed on a web page. We overcame this by instead deciding to just make a GUI interface to display the values, which we succeeded in doing. Our second challenge was coming to our final class to present, only to find out that my Pi wouldn’t start up. I found a crack on my memory card and deduced that it broke and was the main culprit. We resolved this by notifying the professor and planning for a further date to present our final project, and also reluctantly starting over on the code work we had created. Our final challenge was that our sensors weren’t working properly. They weren’t changing when they were supposed to, just staying one number. We resolved this by adding in the code ‘spi.max\_speed\_hz=50000’, which fixed the problem and the sensor values now correctly displayed on the GUI, changing when we touched them.

The only feature we had to abandon was creating live updating charts to graph/plot the temperature, humidity, and brightness values in real time. The reason we decided this was because all of the many tutorials we found seemed too long, complicated, and daunting in regards to the extra amount of installing and coding necessary. We instead decided to use what we already learned (how to make a gauge display) and customize it a little to better suit our needs.

**Conclusion**

Overall, due to the series of unfortunate events we faced, we found this final project to be a bit frustrating, especially since we could not figure out why our sensors weren’t working properly and wasted a lot of time doing it. However, it did feel pretty rewarding when we were successful in bringing up the sensor values on the screen, and being able to create our own web page so easily seemed pretty amazing to me. The project had its highs and lows. This lab class was very enjoyable. The labs assignments were detailed, clear, and not too difficult. They showed us so many interesting and practical ways we can use our mini-computers, such as a methane detector, using a camera, sending an email remotely, detecting motion, controlling LED’s, and even programming a rover robot. The professor was very helpful and kind, always eager to drop whatever he was doing and help right away whenever I had a question or issue. I made friends with my partner and the classmates around me. This class was a very good and enjoyable experience.

**Appendix**

**Websensor.py:**

from bottle import route, run, template

import temp, lightsensor, humidity\_sensor

@route('/tempC')

def tempC():

return temp.read\_temp()[0]

@route('/tempF')

def tempF():

return temp.read\_temp()[1]

@route('/brightness')

def brightness():

return lightsensor.read\_brightness()

@route('/humidity')

def humidity():

return humidity\_sensor.read\_humidity()

@route('/')

def index():

return template('main.html')

@route('/raphael')

def index():

return template('raphael.2.1.0.min.js')

@route('/justgage')

def index():

return template('justgage.1.0.1.min.js')

run(host='0.0.0.0', port=80)

**temp.py:**

import spidev, time

spi = spidev.SpiDev()

spi.open(0,0)

def analog\_read(channel):

spi.max\_speed\_hz=50000

r = spi.xfer2([1, (8 + channel) << 4, 0])

adc\_out = ((r[1]&3) << 8) + r[2]

return adc\_out

def read\_temp():

reading = analog\_read(0)

voltage = reading \* 3.3 / 1024

temp\_c = voltage \* 100 - 50

temp\_f = temp\_c \* 9.0 / 5.0 + 32

return temp\_c, temp\_f

time.sleep(1)

def read\_brightness():

reading = analog\_read(1)

return reading

time.sleep(1)

def read\_humidity():

reading = analog\_read(2)

return reading

time.sleep(1)

**weather\_gui.py:**

from Tkinter import \*

import RPi.GPIO as GPIO

import temp

class App:

def \_\_init\_\_(self, master):

self.master = master

frame = Frame(master)

frame.pack()

label = Label(frame, text='Temp(C)', font=("Helvetica", 32))

label.grid(row=0)

self.reading\_label = Label(frame, text='12.34', font=("Helvetica", 110))

self.reading\_label.grid(row=1)

label1 = Label(frame, text='Temp(f)', font=("Helvetica", 32))

label1.grid(row=2)

self.reading\_label1 = Label(frame, text='12.34', font=("Helvetica", 110))

self.reading\_label1.grid(row=3)

label2 = Label(frame, text='Brightness', font=("Helvetica", 32))

label2.grid(row=4)

self.reading\_label2 = Label(frame, text='12.34', font=("Helvetica", 110))

self.reading\_label2.grid(row=5)

label3 = Label(frame, text='Humidity', font=("Helvetica", 32))

label3.grid(row=6)

self.reading\_label3 = Label(frame, text='12.34', font=("Helvetica", 110))

self.reading\_label3.grid(row=7)

self.update\_reading()

def update\_reading(self):

tempc, tempf = temp.read\_temp()

brightness = temp.read\_brightness()

humidity = temp.read\_humidity()

reading\_tempc = "{:.2f}".format(tempc)

reading\_tempf = "{:.2f}".format(tempf)

reading\_brightness = "{:.2f}".format(brightness)

reading\_humidity = "{:.2f}".format(humidity)

self.reading\_label.configure(text=reading\_tempc)

self.reading\_label1.configure(text=reading\_tempf)

self.reading\_label2.configure(text=reading\_brightness)

self.reading\_label3.configure(text=reading\_humidity)

self.master.after(500, self.update\_reading)

root = Tk()

root.wm\_title('Weather')

app = App(root)

root.geometry("600x1000+0+0")

root.mainloop()

**main.html:**

<html>

<head>

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js" type="text/javascript" charset="utf-8"></script>

<script src="raphael"></script>

<script src="justgage"></script>

<script>

function callback(tempStrC, status){

if (status == "success") {

tempC = parseFloat(tempStrC).toFixed(2);

c.refresh(tempC);

setTimeout(getTempC, 1000);

}

else {

alert("There was a problem");

}

}

function callback1(tempStrF, status){

if (status == "success") {

tempF = parseFloat(tempStrF).toFixed(2);

f.refresh(tempF);

setTimeout(getTempF, 1000);

}

else {

alert("There was a problem");

}

}

function callback2(brightnessStr, status){

if (status == "success") {

brightnessV = parseFloat(brightnessStr).toFixed(2);

b.refresh(brightnessV);

setTimeout(getBrightness, 1000);

}

else {

alert("There was a problem");

}

}

function callback3(humidityStr, status){

if (status == "success") {

humidityV = parseFloat(humidityStr).toFixed(2);

h.refresh(humidityV);

setTimeout(getHumidity, 1000);

}

else {

alert("There was a problem");

}

}

function getTempC(){

$.get('/tempC', callback);

}

function getTempF(){

$.get('/tempF', callback1);

}

function getBrightness(){

$.get('/brightness', callback2);

}

function getHumidity(){

$.get('/humidity', callback3);

}

</script>

</head>

<body>

<div id="tempC" class="100x80px"></div>

<div id="tempF" class="100x80px"></div>

<div id="bright" class="100x80px"></div>

<div id="humid" class="100x80px"></div>

<script>

var c = new JustGage({

id: "tempC",

value: 0,

min: -100,

max: 100,

title: "Temp (C)"

});

getTempC();

</script>

<script>

var f = new JustGage({

id: "tempF",

value: 0,

min: -100,

max: 100,

title: "Temp (F)"

});

getTempF();

</script>

<script>

var b = new JustGage({

id: "bright",

value: 0,

min: 0,

max: 100,

title: "Brightness"

});

getBrightness();

</script>

<script>

var h = new JustGage({

id: "humid",

value: 0,

min: 0,

max: 100,

title: "Humidity"

});

getHumidity();

</script>

</body>

</html>