Implementation/Assumptions Notes

Virtual Network Connect (VNC) was used to connect from a Windows 10 laptop to a Raspberry PI 4. No problems were encountered using the Raspberry PI in this mode. First, Tornado 6.1 was installed using the “pip install tornado” command in a terminal window. The client side was written in Python while the client side was written in HTML, Javascript and CSS. The client was written using a plain text editor. The server side was written using the Python 3 IDE. Both the client and the server were run on the Raspberry Pi. The HelloWorld client and server example programs on the Tornado website were used as templates to develop the code.

The client is used to send and receive messages to and from the server. The client has three input fields (an IP address field, a port number field and a uri field) to define the websocket. Below these fields is an “open” button which attempts to open a web socket with the server. If no server is present, an error message is displayed on the console. A series of command buttons are created: the Read1 command which reads one time stamp, humidity and temperature, the Read 10 command which reads 10 time stamps, humidities and temperatures, the Avg/Min/Max command which takes the average, minimum and maximum of ten readings and a Close command which closes the websocket. Replies to the commands are displayed on the client console. I attempted to left justify the messages on the client console but I could not figure out how to do that. If the connection is lost with the server, a “Connection Closed” message is displayed on the client console. Two other fields are displayed where the user can set the Humidity and Temperature at which alarms are to be set. Another button commands the server to check for over range Humidity and Temperature or both and set an alarm.

On the server side, once a connection with the client is established, the server listens for incoming messages which are commands from the client. A lot of the code used in the server is reused from Project 1. When the Single Read button is pressed, a time stamp is read, the humidity and temperature is read from the psuedosensor. The alarm command tells the server to toggle the alarm flag. A humidity set command tells the server to set the humidity alarm level to the level received in the message. The temperature set command tells the servers to set the temperature to the level received in the message. The levels are tested to see if they are within range. The alarm is tested to see if it is on. If so, the humidity and temperature that have been read are tested with the humidity and temperature alarm settings. If the humidity and temperature that where read from the sensor exceed the alarm settings, alarm text is added to the timestamp, humidity and temperature text. This text is sent to the client. The Read 10 button operates in a very similar fashion to the Single Read button except that it is in a loop that iterates 10 times. The AVG/Min/Max function reads the humidity and temperature ten times, computes an average humidity and temperature, finds the minimum and maximum of the ten values read and sends them to the client. The server closes the websocket if a close command is received.

The server program is Project2_server.py, the client program is Project2_client.html

Project2_server.py

```
import tornado.httpserver
import tornado.websocket
import tornado.ioloop
import tornado.web
import socket

# Code copied from Project 1
import time
from psuedoSensor import PseudoSensor
ps = PseudoSensor()
storeStruct = []
alarmOn = False
humid = 80.0
temp = 80.0

'''
The web socket handler is derived from the HelloWorld example from the Tornado
Websockets website
'''
# Create a web socket handler
class WSHandler(tornado.websocket.WebSocketHandler):
    def open(self):
        print ('new connection')

    def on_message(self, message):
        global alarmOn
        global humid
        global temp
        print ('message received:  %s' % message)
        # Test for which message is recieved
        if (message == "alarmOn"):
            if alarmOn == False:
                alarmOn = True
                self.write_message("Alarms On")
            else:
                alarmOn = False
                self.write_message("Alarms Off")
        elif (message == "Read1"):
            # read 1 time stamp, humidity and temperature
            ts = time.gmtime()
            h,t = ps.generate_values()
            print(h, humid, t, temp)
            # if the alarm is set, test to see if humidity and temperature
            # are exceeded. If so, add text to indicate alarm
            if alarmOn:
                if ((h >= humid) and (t < temp)):
                    displayStr = time.strftime("%Y-%m-%d %H:%M:%S   ",ts) + str(round(h,1)) + "%  " + str(round(t,1)) \
                      + " deg  Single Read   Humidity Alarm"
                elif ((h < humid) and (t >= temp)):
                    displayStr = time.strftime("%Y-%m-%d %H:%M:%S   ",ts) + str(round(h,1)) + "%  " + str(round(t,1)) \
                      + " deg  Single Read Temperature Alarm"
                elif ((h >= humid) and (t >= temp)):
                    displayStr = time.strftime("%Y-%m-%d %H:%M:%S   ",ts) + str(round(h,1)) + "%  " + str(round(t,1)) \
                      + " deg  Single Read Humid/Temp Alarm"
                else:
                    displayStr = time.strftime("%Y-%m-%d %H:%M:%S   ",ts) + str(round(h,1)) + "%  " + str(round(t,1)) \
                      + " deg  Single Read"
            else:
                displayStr = time.strftime("%Y-%m-%d %H:%M:%S   ",ts) + str(round(h,1)) + "%  " + str(round(t,1)) \
```

```
            + " deg  Single Read"
        self.write_message(displayStr)
    elif (message == "Read10"):
        # read 10 time stamps, humidities and temperatures
        for i in range(1,11):
            ts = time.gmtime()
            h,t = ps.generate_values()
            # if the alarm is set, test to see if humidity and temperature
            # are exceeded. If so, add text to indicate alarm
            if alarmOn:
                if ((h >= humid) and (t < temp)):
                    displayStr = time.strftime("%Y-%m-%d %H:%M:%S   ",ts) + str(round(h,1)) + "%  " + str(round(t,1)) \
                      + " deg " + str(i) + " Humidity Alarm"
                elif ((h < humid) and (t >= temp)):
                    displayStr = time.strftime("%Y-%m-%d %H:%M:%S   ",ts) + str(round(h,1)) + "%  " + str(round(t,1)) \
                      + " deg " + str(i) + " Temperature Alarm"
                elif ((h >= humid) and (t >= temp)):
                    displayStr = time.strftime("%Y-%m-%d %H:%M:%S   ",ts) + str(round(h,1)) + "%  " + str(round(t,1)) \
                      + " deg " + str(i) + " Humid/Temp Alarm"
                else:
                    displayStr = time.strftime("%Y-%m-%d %H:%M:%S   ",ts) + str(round(h,1)) + "%  " + str(round(t,1)) \
                      + " deg " + str(i)
            else:
                displayStr = time.strftime("%Y-%m-%d %H:%M:%S   ",ts) + str(round(h,1)) + "%  " + str(round(t,1)) \
                  + " deg " + str(i)
            self.write_message(displayStr)
            # sleep for 1 second
            time.sleep(1)
    elif (message == "AvgMinMax"):
        #Calculate the average, the minimum and maximum of 10 readings
        hSum = 0
        hMin = 101.0
        hMax = -.1
        tSum = 0
        tMin = 101.0
        tMax = -21.0
        for ii in range(1,10):
            tss = time.gmtime()
            hh,tt = ps.generate_values()

            # Calulate the sum of the 10 readings
            hSum = hh + hSum
            tSum = tt + tSum

            # Test to see if the current reading is greater than the current maximum. If so, set the
            # maximum to the current maximum. Do likewise for the minimum.
            if tt > tMax:
                tMax = tt
            if hh > hMax:
                hMax = hh
            if hh < hMin:
                hMin = hh
            if tt < tMin:
                tMin = tt

            # sleep for 1 second
            time.sleep(1)

        #calculate the average of the ten readings
        hAvg = hSum/10
        tAvg = tSum/10

        #Display the average humidity and temperature
```

```
            displayStr = time.strftime("%Y-%m-%d %H:%M:%S   ",tss) + str(round(hAvg,1)) + \
                  "%  Avg  " + str(round(tAvg,1)) + " deg  Avg"
            self.write_message(displayStr)

            #Display the minimum humidity and temperature
            displayStr = time.strftime("%Y-%m-%d %H:%M:%S   ",tss) + str(round(hMin,1)) + \
                  "%  Min  " + str(round(tMin,1)) + " deg  Min"
            self.write_message(displayStr)

            #Display the minimum humidity and temperature
            displayStr = time.strftime("%Y-%m-%d %H:%M:%S   ",tss) + str(round(hMax,1)) + \
                  "%  Max  " + str(round(tMax,1)) + " deg  Max"
            self.write_message(displayStr)

        elif (message == "Close"):
            #Close the connection
            print ('connection closed')
            self.close()

        elif(message[:6] == "Humd ="):
            #Set the humidty alarm.
            #First test to see if a valid number was entered
            try:
                #round the number to one decimal place
                t_humid = round(float(message[6:]),1)
                #test to see if the humidity entered is within valid range
                if ((t_humid >= 0.0) and (t_humid <= 100.0)):
                    humid = t_humid
                    print (f"Humidity Alarm Set to {humid}")
                    self.write_message(f"Humidity alarm set to {humid}")
                else:
                    print ("Humidity entered is out of range")
                    self.write_message("Humidity entered is out of range")
            except:
                print ("Humidity entry is not a valid number")
                self.write_message("Humidity entry is not a valid number")
        elif(message[:6] == "Temp ="):
            #test to see if the temperature entered is a valid number
            try:
                #rond off to one decimal place
                t_temp = round(float(message[6:]),1)
                #test to see if temperature entered is within range
                if ((t_temp >= -20.0) and (t_temp <= 100.0)):
                    temp = t_temp
                    print (f"Temperature Alarm Set to {temp}")
                    self.write_message(f"Temperature alarm set to {temp}")
                else:
                    print ("Temperature entered is out of range")
                    self.write_message("Temperature entered is out of range")
            except:
                print ("Temperature entry is not a valid number")
                self.write_message("Temperature entry is not a valid number")
        else:
            print('unknown command recieved')

    def check_origin(self, origin):
        return True


application = tornado.web.Application([
    (r'/ws', WSHandler),
])
```

```
if __name__ == "__main__":
    http_server = tornado.httpserver.HTTPServer(application)
    http_server.listen(8888)
    myIP = socket.gethostbyname(socket.gethostname())
    print ('*** Websocket Server Started at %s***' % myIP)
    tornado.ioloop.IOLoop.instance().start()
```

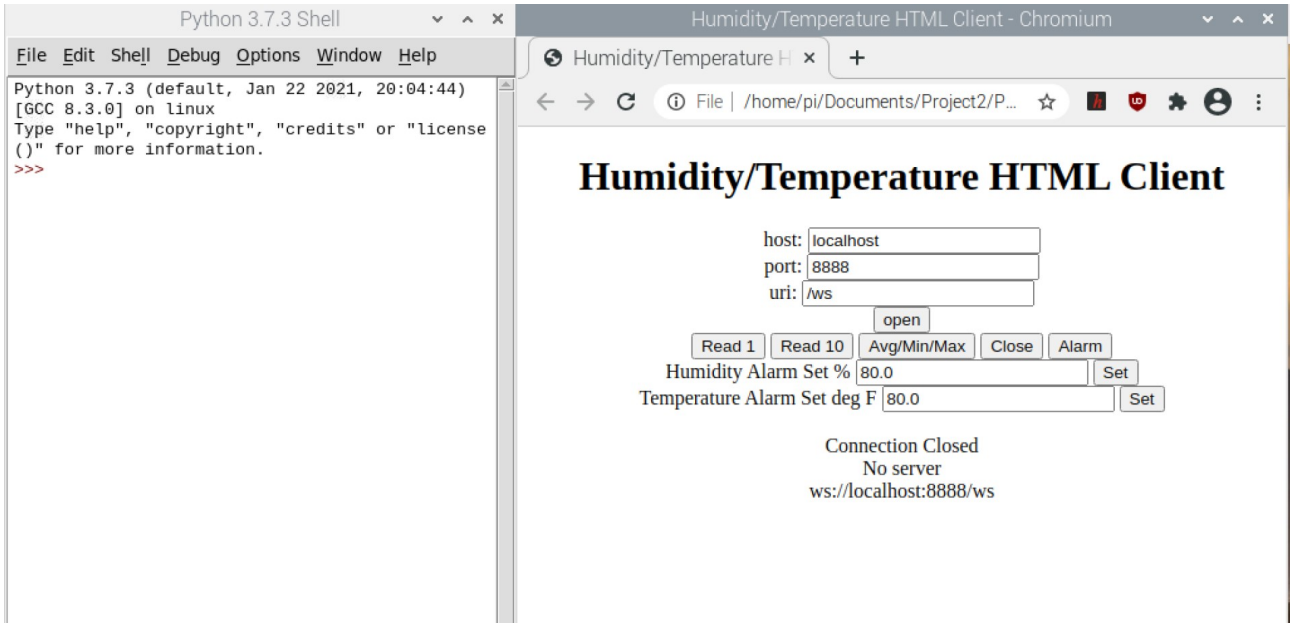
Project2_client.html

```
<!doctype html>
<html>
  <head>
    <title>Humidity/Temperature HTML Client</title>
    <meta charset="utf-8" />
    <style type="text/css">
      body {
        text-align: center;
        min-width: 500px;
      }
    </style>
    <script src="http://code.jquery.com/jquery.min.js"></script>
    <script>

    // From the HelloWorld tornado client/erver websocket example,
    // create a websocket instance


      // log function
      log = function(data){
        $("div#terminal").prepend("</br>" + data)
        console.log(data);
      };

      $(document).ready(function () {
        $("div#message_details").hide()

            var ws;

        $("#open").click(function(evt) {
          evt.preventDefault();

          var host = $("#host").val();
          var port = $("#port").val();
          var uri = $("#uri").val();

          // create websocket instance
              log("ws://" + host + ":" + port + uri);
          ws = new WebSocket("ws://" + host + ":" + port + uri);

          // Handle incoming websocket message callback
          ws.onmessage = function(evt) {
            log("Message Received: " + evt.data)
            };

          // Close Websocket callback
          ws.onclose = function(evt) {
            log("Connection Closed");
            $("#host").css;
            $("#port").css;
            $("#uri").css;
            $("div#message_details").empty();
            };

          // Open Websocket callback
          ws.onopen = function(evt) {
            $("#host").css;
            $("#port").css;
            $("#uri").css;
            $("div#message_details").show();
```

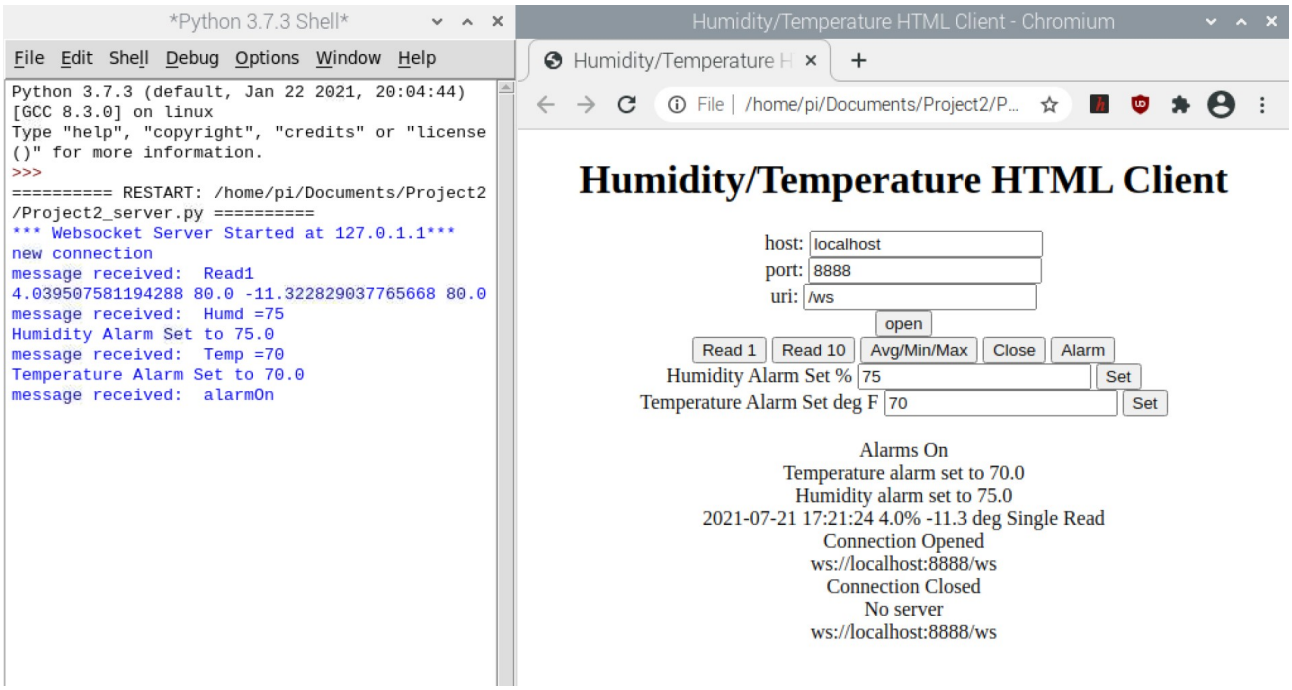
```
    log("Connection Opened");
   };

 // Listen for websocket errors
   ws.addEventListener('error', function () {
     log("No server");
   });

});

 //Send a read 1 command to the server
 Read1.addEventListener('click', function(){
   ws.send("Read1");
   // Handle incoming websocket message callback
   ws.onmessage = function(evt) {
     log(evt.data)
     };
});

 //Send a read 10 command to the server
 Read10.addEventListener('click', function(){
   ws.send("Read10");
   // Handle incoming websocket message callback
   ws.onmessage = function(evt) {
     log(evt.data)
     };
});

 // Send a calculate Average/Minimim and Maximum command to the server
 AvgMinMax.addEventListener('click', function(){
   ws.send("AvgMinMax");
   // Handle incoming websocket message callback
   ws.onmessage = function(evt) {
     log(evt.data)
     };
});

 // Tell the server to Toggle the Alarm On/Off flag
 Alarm.addEventListener('click', function(){
   ws.send("alarmOn");
   // Handle incoming websocket message callback
   ws.onmessage = function(evt) {
     log(evt.data)
     };
});

 // Send a close connection command to the server
 Close.addEventListener('click', function(){
   ws.send("Close");
   // Handle incoming websocket message callback
   ws.onmessage = function(evt) {
     log(evt.data)
     };
});

 // Send Humidity set alarm websocket message function
 $("#halarmset").click(function(evt) {
     ws.send("Humd =" + $("#humidity_alarm").val());
 });

 // Send Temperature set alarm websocket message function
 $("#talarmset").click(function(evt) {
     ws.send("Temp =" + $("#temp_alarm").val());
```

```
        });

      });
    </script>
  </head>

  <body>
    <!--
      From the HelloWorld tornado client/erver websocket example,
      create input text fields for the host IP, socket and uri
    -->
    <h1>Humidity/Temperature HTML Client</h1>
    <div id="connection_details">
      <label for="host">host:</label>
      <input type="text" id="host" value="localhost" /><br />
      <label for="port">port:</label>
      <input type="text" id="port" value="8888" /><br />
      <label for="uri">uri:</label>
      <input type="text" id="uri" value="/ws" /><br />
      <input type="submit" id="open" value="open" />
    </div>

    <!-- Create buttons for the commands -->
    <div>
       <input type="button" id = "Read1" value = "Read 1" >
       <input type="button" id = "Read10" value = "Read 10" >
       <input type="button" id = "AvgMinMax" value ="Avg/Min/Max" >
       <input type="button" id = "Close" value = "Close" >
            <input type="button" id = "Alarm" value = "Alarm" >
    </div>

    <!-- Create an input field for the humidity alarm -->
    <div>
       <label for="humidity_alarm">Humidity Alarm Set %</label>
       <input id="humidity_alarm" type="text" value="80.0">
       <input type="submit" id="halarmset" value="Set">
    </div>

    <!-- Create an input field for the temperature alarm -->
    <div>
       <label for="temp_alarm">Temperature Alarm Set deg F</label>
       <input id="temp_alarm" type="text" value="80.0">
            <input type="submit" id="talarmset" value="Set">
    </div>

    <div id="terminal">

    </div>

  </body>
</html>
```

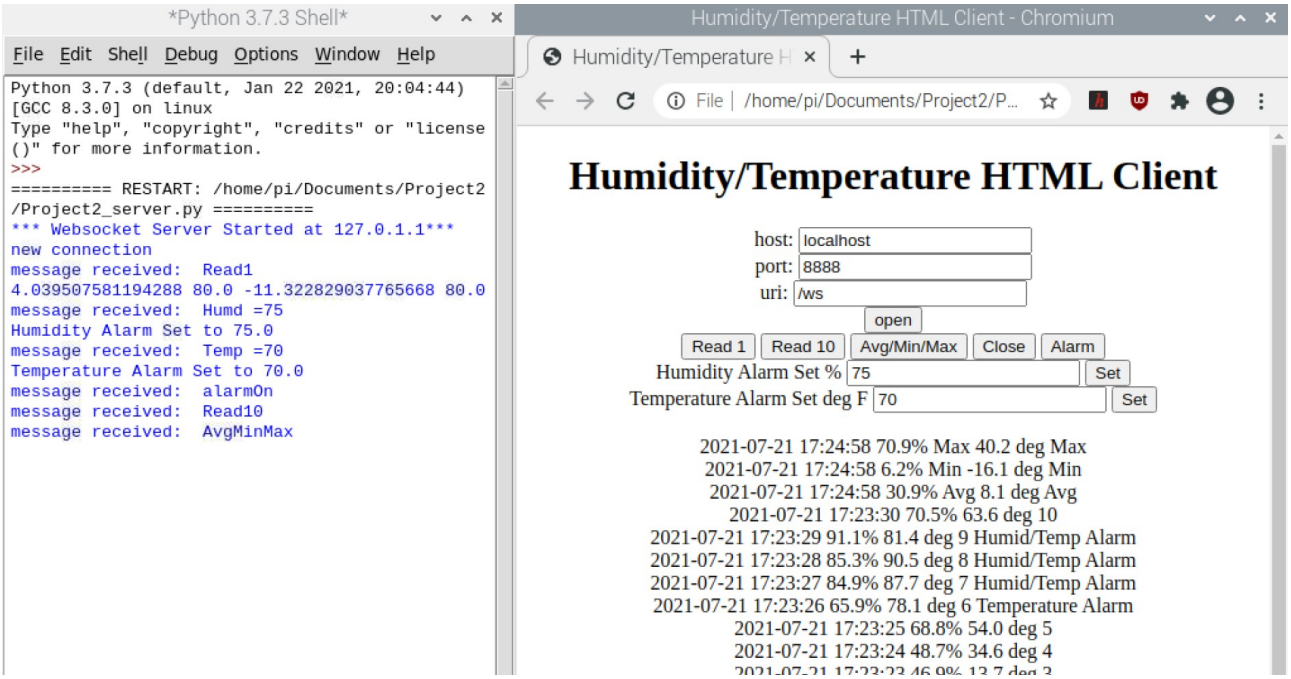
Screenshots



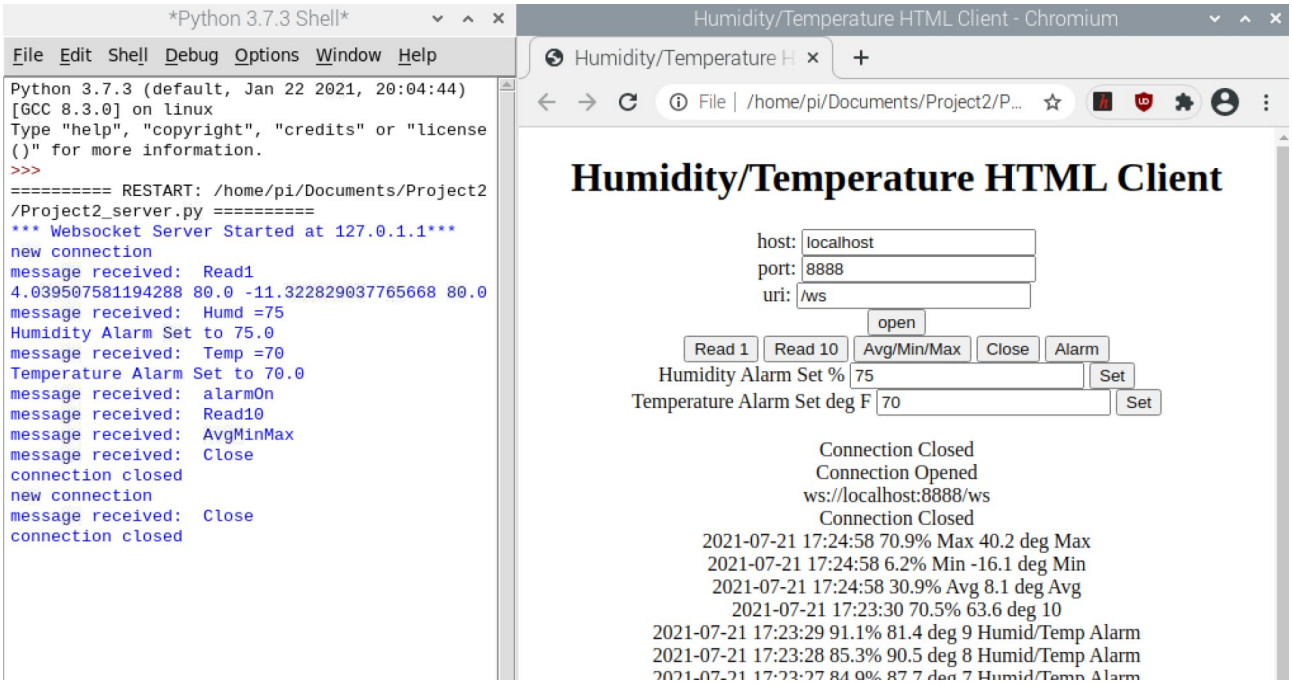
Attempt to connect with no server running



Start server, Read 1, set alarms, turn alarms on.



Average/Min/Max



Close Connection