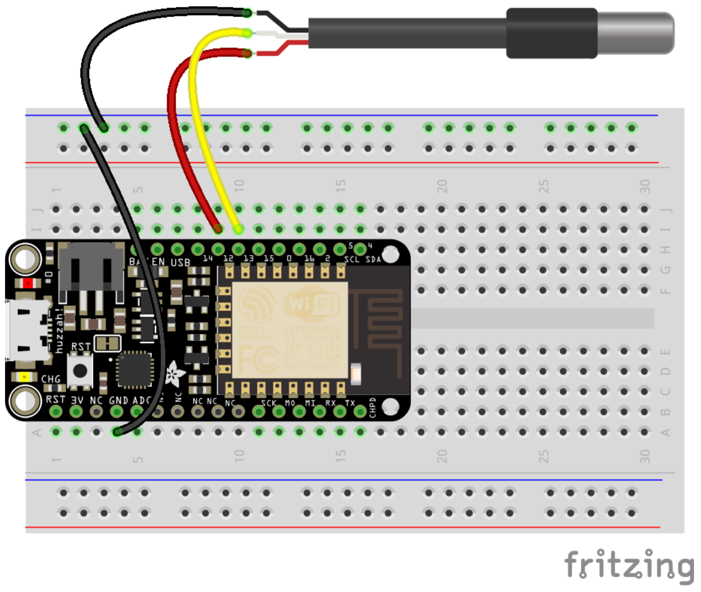
Lab 5 Guide: Timekeeping and digital sensors

In today's lab, you will start using digital sensors. Up until now, we have only used analog sensors. Instead of outputting an analog signal and then using your microcontroller to convert it to a digital signal using the Analog to Digital Converter, digital sensors do all of these steps together and send digital signals directly to your microcontroller. In today's lab, you will use a digital temperature sensor called a DS18B20 in which internal components do the same tasks: convert environmental conditions to analog electrical signals, convert analog signals to digital readings, and convert digital readings to temperature estimates. We will also explore response times of sensors, work with a digital clock to help timestamp sensor measurements and cross compare temperature measurements across different instruments.

1) Connecting the digital temperature sensor

In this section you will connect to and get readings from a digital temperature sensor. This sensor uses a one wire communication protocol with your microcontroller, meaning that it requires one connection to communicate data to the microcontroller. The sensor also requires power and ground connections.

1. Attach the digital temperature sensor (waterproof DS18B20 probe) to the microcontroller as shown in the diagram below
   a. Connect the black wire (ground) on the probe to the negative power rail. Make sure to connect the GND pin on the microcontroller to the negative power rail to complete the ground connection.
   b. Connect the red wire (power) on the probe to GPIO 12 on the microcontroller
   c. Connect the yellow wire (data) on the probe to GPIO 13 on the microcontroller



2. Connect the microcontroller to a computer and open Thonny. To operate this sensor, you'll need to transfer a few MicroPython libraries onto your microcontroller. Add the files: **ds18x20.py** and **onewire.py** from Canvas. Once you download the files, you can navigate to the directory they are in in the top Files (on your computer) panel in Thonny. Right click the file and select "Upload to /" to transfer it to your microcontroller. It should then appear in the bottom Files panel (on your microcontroller).

3. Also add the **print_temp.py** file from Canvas onto your microcontroller using Thonny. To run the file once it has been transferred to your microcontroller, you can simply type "import print_temp" in the Thonny Shell. Or you can open the file and run it using the green play button in Thonny. To stop the script hit CTRL + C.

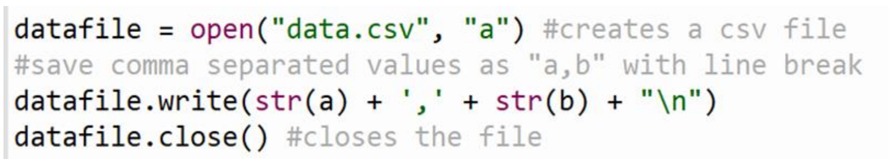
4. Take some preliminary temperature measurements by running the **print_temp.py** file. What is room temperature in the OTC? How consistent are your temperature readings?

2) Measuring response time

One key feature of sensors is the response time, the time it takes for them to equilibrate with the environmental conditions. It is important to understand the response time of any sensor to be able to accurately characterize the environment. In this section you will explore the response time of your digital temperature sensor.

1. Conduct an initial exploration: Run the **print_temp.py** file while warming the sensor in your hand. This code prints a temperature measurement every second. Watch the plotter in the Thonny window. How fast does the temperature change? How might this affect how you would use the sensor to take environmental measurements?

2. Edit the **print_temp.py** file to save the data being printed to the screen to a .csv file like you did in the salinity lab. In the file, save the seconds elapsed in one column and the temperature in an adjacent column. <u>*Note:*</u> *See the example below to help you write your own code. You'll need to substitute 'a' and 'b' with seconds elapsed and temperature in your code. You will also need to decide where in the code to implement each line.*



```
datafile = open("data.csv", "a") #creates a csv file
#save comma separated values as "a,b" with line break
datafile.write(str(a) + ',' + str(b) + "\n")
datafile.close() #closes the file
```

3. Set up a water bath in a cup with some ice in it.

4. Insert the room temperature DS18B20 probe into the ice bath and start running your new **print_temp.py** file to save the data as your probe responds. Once you have run the code for long enough that the temperature has equilibrate you can stop the script using CTRL + C. To close the data file you'll need to write that line of code in the Shell. Then you will be able to open the file to work with the data.

5. Use the data file you saved to create a plot of temperature vs. seconds elapsed. Save a screenshot of your plot.

*****Save a screenshot of your plot for the lab assignment*****

6. How long does it take for the sensor to reach a stable temperature indicative of the water bath temperature (e.g. what's the response time)?

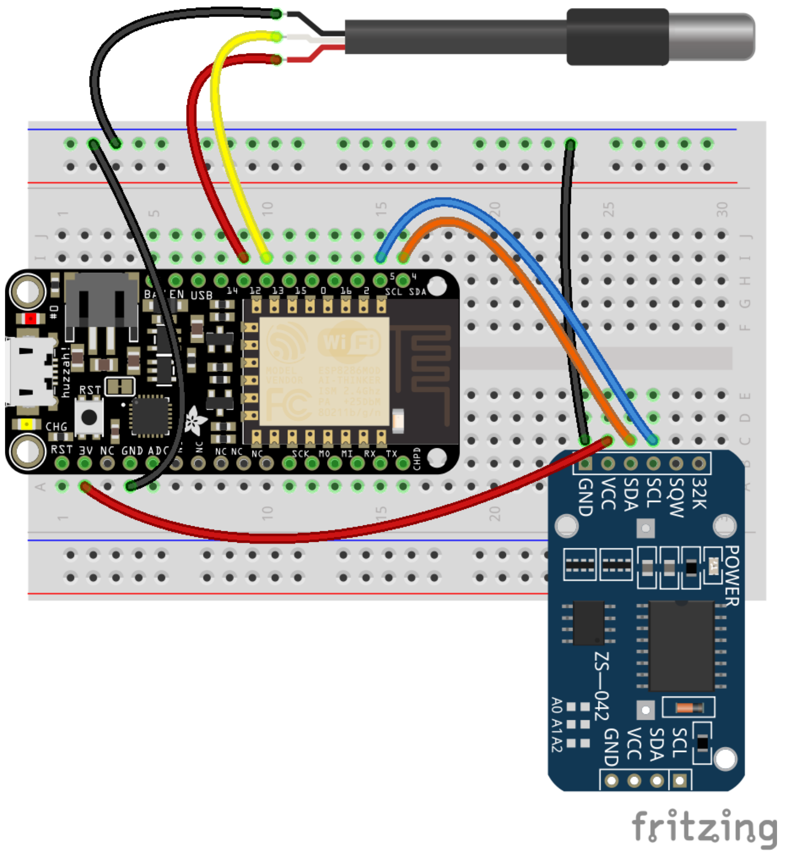
7. Why would it be important to measure the response time of a sensor before using it in the environment?

3) Connecting and setting the real time clock (RTC)

Sections 3 and 4 of this lab can be done in either order.

In this section, you'll connect, set, and use a digital real time clock (RTC). The model we will use is a DS3231 Real Time Clock. As you explored in the previous section, time can provide important context for environmental measurements. Across longer time scales, timestamped measurements are critical to developing time series data to be able to characterize oceanographic patterns and processes and be able to understand how conditions change over different time scales. Even measurements from the most accurate of sensors can be meaningless without the context of when the measurement was taken. Many microcontrollers have on-board clocks, but we will use an eternally connected one here because they do not drift as much. This clock uses another common digital communication protocol called I^2C (Inter-Integrated Circuit). These digital sensors rely on two connections (SDA which stands for Serial Data, and SCL which stands for Serial Clock) in addition to ground and power. Because different I^{2}C sensors have unique addresses, you can connect multiple devices to the same SCL and SDA pins and communicate with them individually.

1. Connect the external clock to the microcontroller as shown below.
   a. Confirm the clock has the coin cell battery inserted correctly with the + side out
   b. Connect VCC on the clock to 3V on the microcontroller
   c. Connect GND on the clock to the negative power rail.
   d. Connect SCL on the clock to SCL (GPIO 5) on the microcontroller
   e. Connect SDA on the clock to SDA (GPIO 4) on the microcontroller



2. Download the **urtc.py** file from Canvas and transfer it on to your microcontroller. This is a necessary library to be able to communicate with your external clock.

3. In the Thonny shell, set up the clock with the following commands.

```
import urtc
from machine import I2C, Pin

i2c = I2C(scl = Pin(5), sda = Pin(4)) # assigns the I2C connection
rtc = urtc.DS3231(i2c) # assigns the RTC
```

4. Read the current clock time with the following command:

```
rtc.datetime() # prints the current date/time
```

5. Your clock is probably not displaying the correct time. Set the clock with the following commands. In this example, the year is 2020, month is September (the 9th month), day is 15, weekday is 2, hour is 13 (1pm), minute is 30, second is 0, and microsecond is 0. Weekday is 1-7 for Mon-Sun. You'll want to input the actual current time.

```
rtc.datetime((2020, 9, 15, 1, 13, 30, 0, 0)) # set the date/time
rtc.datetime()                               # confirm date/time you set
```

6. **You should now have the correct time set on your clock!** This should stay set even when the clock becomes unplugged. If you take out the coin cell battery, that can remove the time you set and you will need to reset it to the time you want.

7. You can isolate different components of the time with the commands below. Try this in the shell to see how it works and confirm the times are correct.

```
d = rtc.datetime()   # saves current date and time as a variable named d
print(d.year)        # prints just the year of the timestamp saved in d
print(d.month)       # prints just the month of the timestamp saved in d
print(d.day)         # prints just the day of the timestamp saved in d
print(d.hour)        # prints just the hour of the timestamp saved in d
print(d.minute)      # prints just the minute of the timestamp saved in d
print(d.second)      # prints just the second of the timestamp saved in d
```

8. Edit your **print_temps.py** file again to add the time to the printed and saved temperature readings. You'll need to add the code from Step 3 to your script to be able to work with the clock data. Save the date in the format YYYY/MM/DD HH:MM:SS. To save your time to a file, you'll need to put them in string format. Substitute your new timestamp in place of seconds elapsed from code edits from Part 2. Use the example below to help you out.

```
# saves the date as a string called date (YYYY/MM/DD)
date = str(d.year) + '/' + str(d.month) + '/' str(d.day)
```

9. Take a screenshot of your printed timestamped temperature measurements from the Thonny Shell for your lab assignment.

*****Save a screenshot of your function output in the Shell for the lab assignment*****

4) Cross comparison of temperature sensors

When using sensors, it is helpful to be able to cross compare readings from different instruments to help calibrate and confirm the accuracy of your measurements. If more than one instrument gives you similar results, you have increased confidence in your measurements. In this section you will compare your digital temperature measurements to the classroom CTD.

1. Visit the classroom CTD station and learn how to retrieve real time readings of the tank's water from it. The classroom CTD is a SBE37 from Sea-Bird Electronics, a common supplier of oceanographic instruments.

2. Set up your constructed sensor so you can take simultaneous readings from your sensor and the CTD. How does your sensor compare to the CTD readings?

3. Take 5 replicate temperature measurements of the tank water from the CTD and report the average below:

4. Simultaneously put your DS18B20 temperature sensor into the tank, run your **print_temps.py** script, and watch the sensor equilibrate to the tank temperature. Be sure you use the information you gathered on response time earlier to make sure you collect data from your temperature sensor at the appropriate time. Record 5 replicate temperature readings and find the difference between each reading and the average CTD reading you calculated above.

CTD Avg. temperature (°C)	**DS18B20 temperature (°C)**	**T_{diff} (CTD - DS18B20)**

5. Use the CTD measurements to assess the accuracy (μ) of your temperature sensor. Use the equation below to calculate accuracy as you have in previous labs.

$$\mu = \frac{1}{n}\sum_{i=0}^{n}(T_{diff})$$

μ = _______________

6. Use the CTD measurements to assess the precision (σ) of your temperature sensor. Use the equation below to calculate precision as you have in previous labs.

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=0}^{n} \left(T_{diff} - \mu\right)^2}$$

σ = _______________

7. Look at the DS18B20 datasheet on Canvas. How does the accuracy you calculated compare to the accuracy reported on the datasheet?

8. Look at the SBE37 datasheet on Canvas. How does the accuracy of the SBE37 compare to that of the DS18B20?

9. Imagine you are conducting a research project. In which situations would it be appropriate to use the SBE37? In which situations would it be appropriate to use the DS18B20?

To submit for credit: Lab Assignment 5 on Canvas: Download the template to type in your answers to the questions and insert the necessary photos to demonstrate you've reached the relevant milestones for this lab.

Extra time? Try setting the clock from the internet using a Network Time Protocol (NTP)

As you may have noticed in the lab, it is difficult to set the RTC manually with very high accuracy. A more accurate way to set the clock is to use an NTP server, which is an online portal that enables a local machine to calculate the current time. MicroPython has a built-in utility called ntptime to automatically obtain the current time from an NTP server (the default is pool.ntp.org) and use it to reset the RTC. Here, you will use a modified version of this utility, which can also be used to reset the external RTC. You will need to load the **ntptimeDS3231.py** file (available on Canvas) onto your microcontroller

This exercise requires that your microcontroller is connected to WiFi (e.g. through the OTCnet router). By default, the microcontroller is set to Access Point (AP) mode, meaning it serves as a router – you can log directly onto your microcontroller's AP via your laptop's WiFi, for example, when you communicate with it via WebREPL. However, to set the RTC over the internet requires your microcontroller to be in Station Mode (STA). The commands below set up your microcontroller in Station Mode, meaning you can use it to log on to another Access Point, like OTCnet. In Station mode, another machine cannot log on to your microcontroller (e.g you will not be able to connect to it using WebREPL in this mode). Use the commands below to put your microcontroller in Station mode:

```
from network import WLAN, STA_IF
wlan = WLAN(STA_IF)
wlan.active(True)                        # activates the interface
wlan.connect('OTCnet', 'sEnse3five1') # connects to OTC wifi
```

Get the current time using the commands below

```
import ntptimeDS3231
ntptimeDS3231.time()
```

Set the time on the DS3231 RTC with the commands below. Note that this will set your clock in UTC (Coordinated Universal Time).

```
ntptimeDS3231.settimeDS3231()
datetime = rtc.datetime()
print(datetime)
```

You should now see the clock has been set to the correct time. To get out of Station mode use the command below:

```
wlan.active(False)
```