

Chapter 4

Getting data to the Computer

We now have voltage data coming from our Arduino into the computer serial port, and we can see that data with the Arduino serial port monitors. But suppose we want to analyze our data in another program. How would we get the data into Excel or LoggerPro, or SPSS or even our beloved Python that we learned to use in PH150?

There are lots of ways to do this, but an easy way is to use our Python skills and write a simple code using the Python serial port library. That is what we are going to do in this lab.

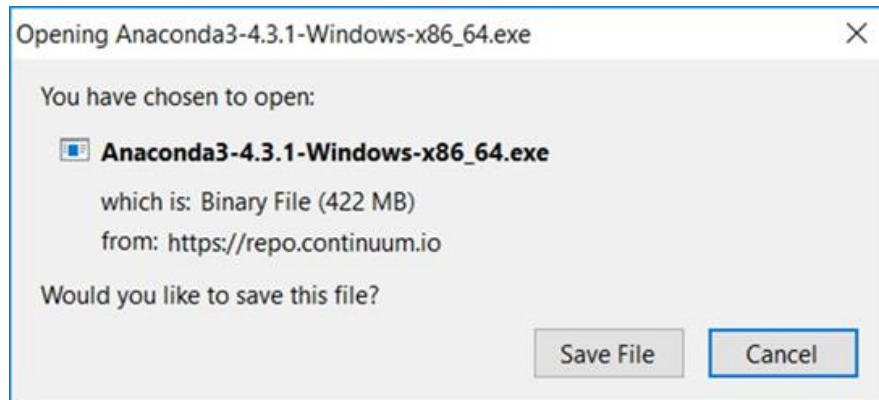
4.1 How to get Python

But wait, you might say, I didn't use snakes in PH150. What are you even talking about? Or maybe you got rid of Python because you never thought you would use it again. Whatever the case, Python is another way to make computer code like our Arduino app. Only, this code is designed to work on our computer, itself. This is just what we want for getting the data ready for analysis on our computer. If you already have Python, that is fine, you could skip ahead. If you don't, the next steps show how to get Python on your computer and how to get the Python serial library so you have the commands to read the serial port. Our physics department uses two different distributions of Python. Canopy, and Anaconda. Canopy is a little bit more "civilized" with more dialogue boxes and little command line work. Anaconda is a little bit more "linux-like" with more command line work. But both work fine. Instructions for getting both are in the sections below.

4.1.1 Getting Anaconda Python

The Anaconda distribution of Python is designed for scientific work (so it has most of the science libraries of functions already installed) It can be found at <https://www.continuum.io/downloads>. There is an install link for Windows, Mac, and Linux. Choose the one for your operating system¹.

When you click on a link, it is likely a dialog box will come up telling you that you are downloading something. In windows it looks like this



Choose “Save File” and when the file is downloaded open it to start the installation. You should see something like this:

Choose “Next” and follow the installation instructions. If all goes well, you should have a new set of apps. Here is what mine looked like on my Windows 10 computer.

The list of the apps has an app called Spyder. Let’s launch it to see what it does.

What we get is something like the Arduino program that we have been using to write Arduino sketches. Spyder is a place to write and run Python commands. Only this time there is no checking and uploading the code, because the Python code will run on our computer, not on an Arduino.

In the figure above, the command just says to print “hello new python users!” and that is all. When it runs, it prints our message on the small window to the right. But of course Python can do much more than print silly messages in little windows. We will have our Python system read a serial port and save our data to a file. But we need an additional piece of Python to do this. We need functions that can handle serial ports. These functions are already written by someone, and put together in a package called a “library.” But this library isn’t included in what we have downloaded. So we need to fix this next.

¹If you have a Linux computer, it is likely that you will want to actually see if Anaconda is in your distribution’s repository. If you are not a Linux user, you have no idea what that means and can ignore it.

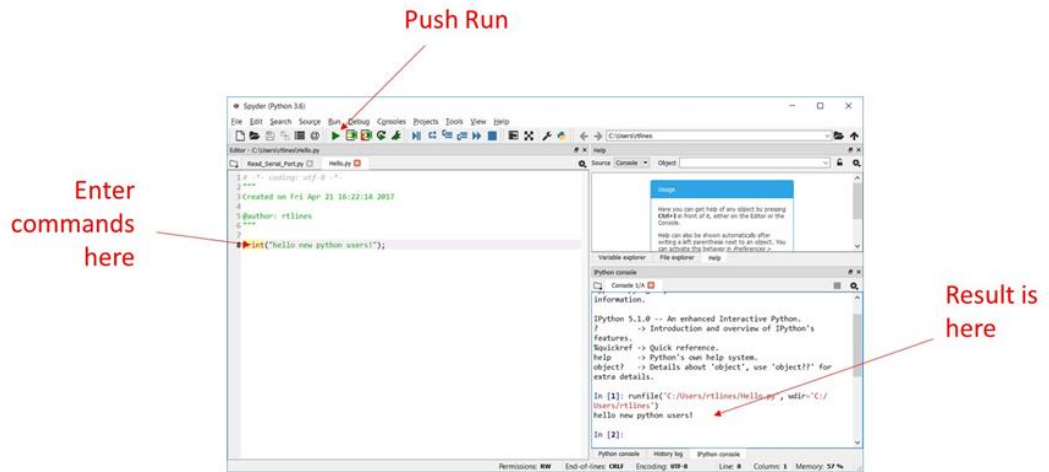


New Anaconda Apps



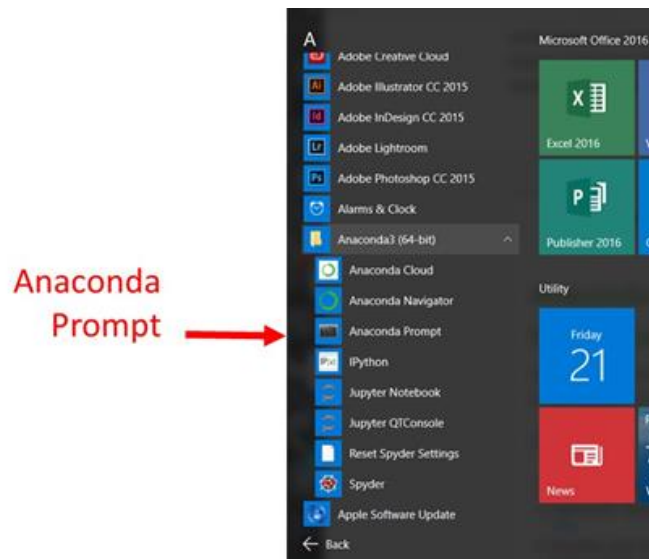
4.1.2 Getting the PySerial library for Anaconda.

Now that we have Python, we need to update it with the PySerial library so that we can read our data from the computer serial port. If you have installed



the Anaconda package as your Python distribution, follow along here. If you have Canopy, see the instructions below (section 4.2.1). If you have a different distribution entirely, ask your instructor for help.

We will use the Anaconda prompt to get the PySerial library. The Anaconda prompt is an app that let's us modify the Python libraries that we have installed. The Anaconda prompt is found in the list of apps that were installed in Anaconda. If you are using Windows, you might find it in your app list like this.

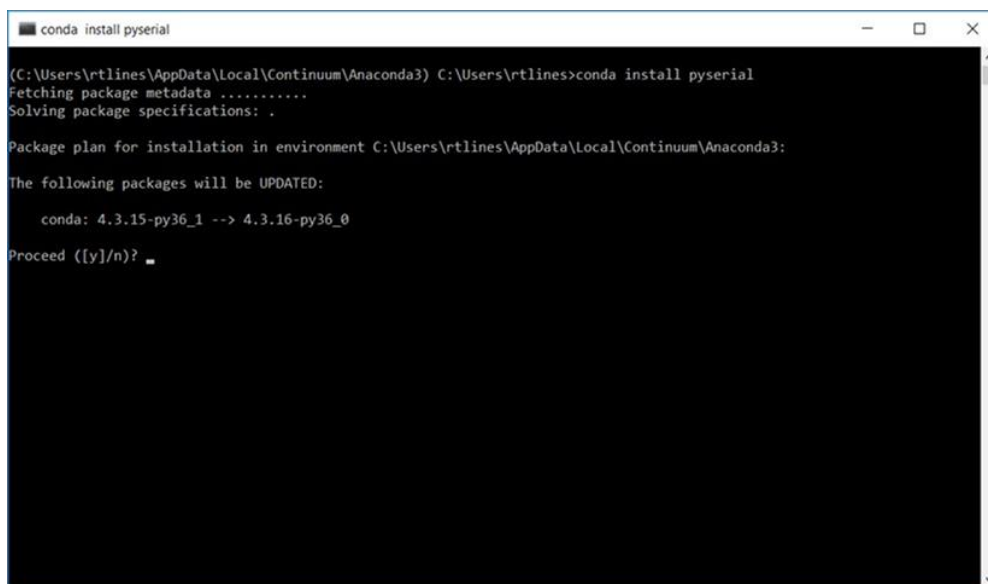


Once you launch the Anaconda prompt it just looks like a big black box. We can type commands in that box that will modify the Anaconda Python



```
Anaconda Prompt
(C:\Users\rtlines\AppData\Local\Continuum\Anaconda3) C:\Users\rtlines>conda install pyserial
```

programs that we installed. In our case we want to type in the command `conda install pyserial`.



```
conda install pyserial
(C:\Users\rtlines\AppData\Local\Continuum\Anaconda3) C:\Users\rtlines>conda install pyserial
Fetching package metadata .....
Solving package specifications: .

Package plan for installation in environment C:\Users\rtlines\AppData\Local\Continuum\Anaconda3:

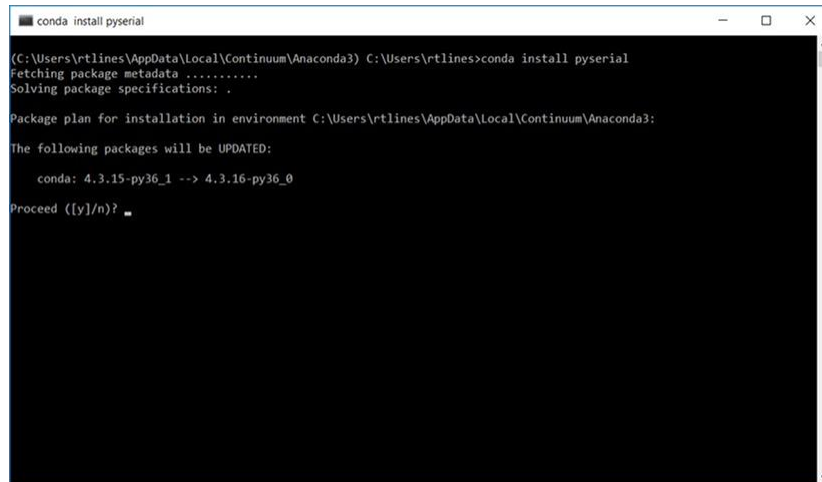
The following packages will be UPDATED:

    conda: 4.3.15-py36_1 --> 4.3.16-py36_0

Proceed ([y]/n)?
```

Notice that the Anaconda prompt app responds to our command. What it responds will depend on your computer and your Python distribution. You need a network connection to install new libraries, so if you get an error, you

may just not be connected to a network.



```
conda install pyserial

(C:\Users\rtlines\AppData\Local\Continuum\Anaconda3) C:\Users\rtlines>conda install pyserial
Fetching package metadata .....
Solving package specifications: .

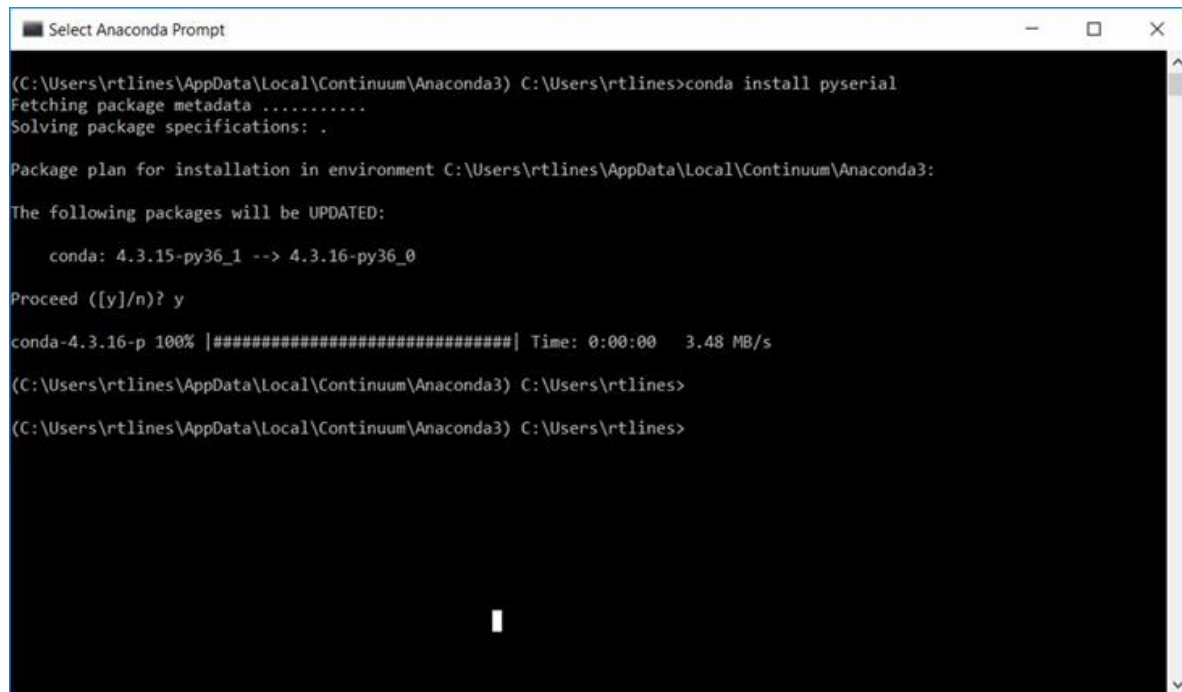
Package plan for installation in environment C:\Users\rtlines\AppData\Local\Continuum\Anaconda3:

The following packages will be UPDATED:

  conda: 4.3.15-py36_1 --> 4.3.16-py36_0

Proceed ([y]/n)?
```

If you already have PySerial installed, you will be told so, or asked if you wish to update it if an update is available. If you don't have PySerial, it will ask you if you want to install it. Answer “yes” and PySerial will be installed. If any error messages are generated, ask for help from your instructor. Hopefully you will see a happy end result like this and you will be all ready to start writing



```
Select Anaconda Prompt

(C:\Users\rtlines\AppData\Local\Continuum\Anaconda3) C:\Users\rtlines>conda install pyserial
Fetching package metadata .....
Solving package specifications: .

Package plan for installation in environment C:\Users\rtlines\AppData\Local\Continuum\Anaconda3:

The following packages will be UPDATED:

  conda: 4.3.15-py36_1 --> 4.3.16-py36_0

Proceed ([y]/n)? y

conda-4.3.16-p 100% |#####| Time: 0:00:00 3.48 MB/s

(C:\Users\rtlines\AppData\Local\Continuum\Anaconda3) C:\Users\rtlines>

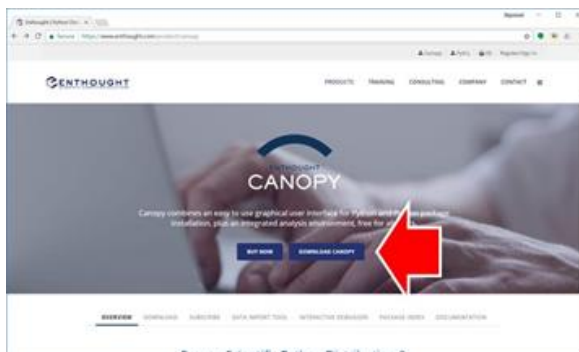
(C:\Users\rtlines\AppData\Local\Continuum\Anaconda3) C:\Users\rtlines>
```

code to get data from the serial port.

4.2 Getting Canopy Python

If you choose the Canopy system, follow these instructions. You may already have the Canopy distribution of Python. If so skip down to adding PySerial in the next section.

Canopy is a little more polished in it's setup. And it works well. So let's see how to install this version of Python and enhance it with the Pyserial library. We will start at the Canopy home site <https://www.enthought.com/products/canopy/>. Partway down the page there is a blue “Download Canopy” button.

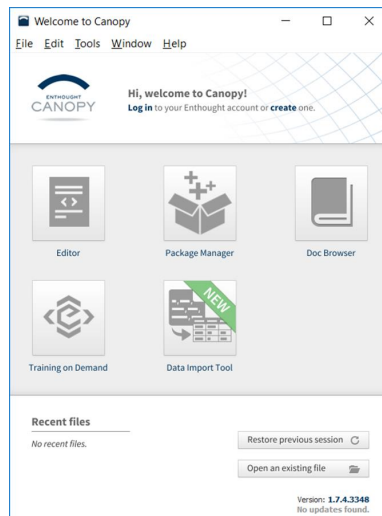
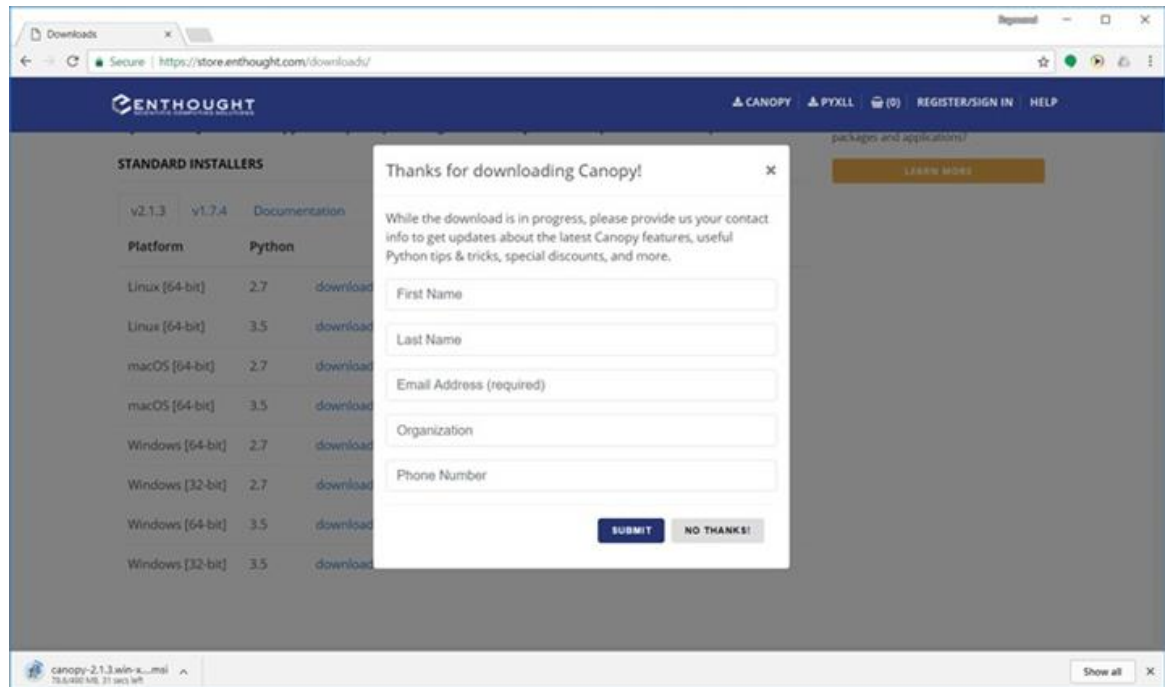


This will take you to the download page where you can choose the version for your operating system. When you click the proper download link, it will ask you for some information.

Once the download completes, install Canopy. When you run the program, you will see something like this

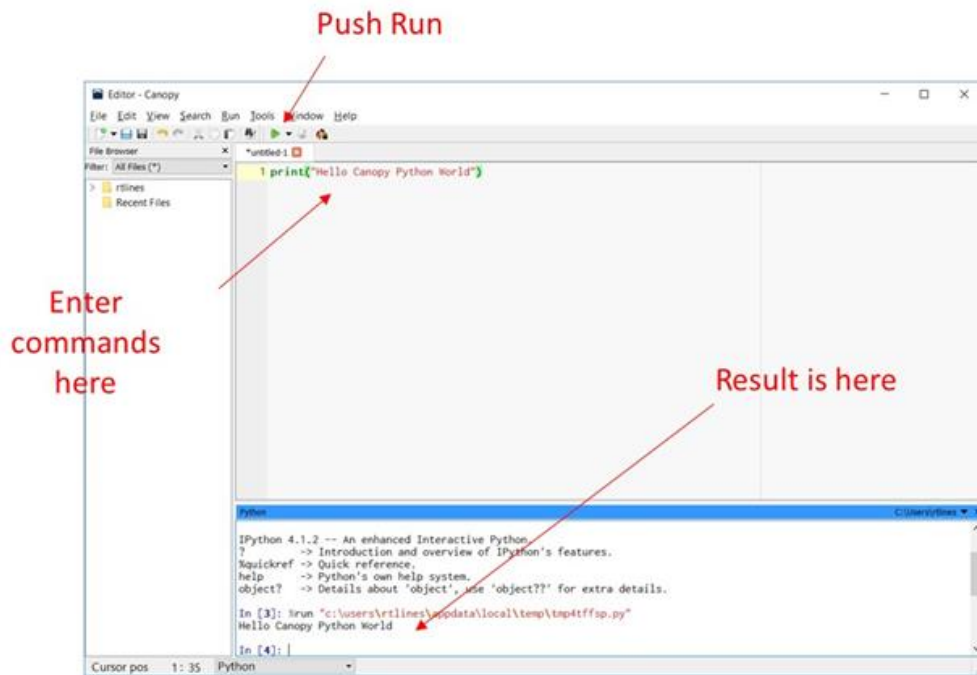
The editor lets us write Python programs. If you choose this you get a window like our Arduino program. The editor is a place to write and run Python commands very like our Arduino software. Only, this time there is no checking and uploading the code, because the Python code will run on our computer, not on an Arduino.

In the figure above, the command just says to print “Hello Canopy Python World” and that is all. When it runs, it prints our message on the small window at the bottom of the Editor window. But of course Python can do much more than print silly messages in little windows. We will have our Python system read a serial port and save our data to a file. But we need an additional piece of Python to do this. We need functions that can handle serial ports. These functions are already written by someone, and put together in a package called a “library.” But this library isn’t included in what we have downloaded, so we need to fix this next.

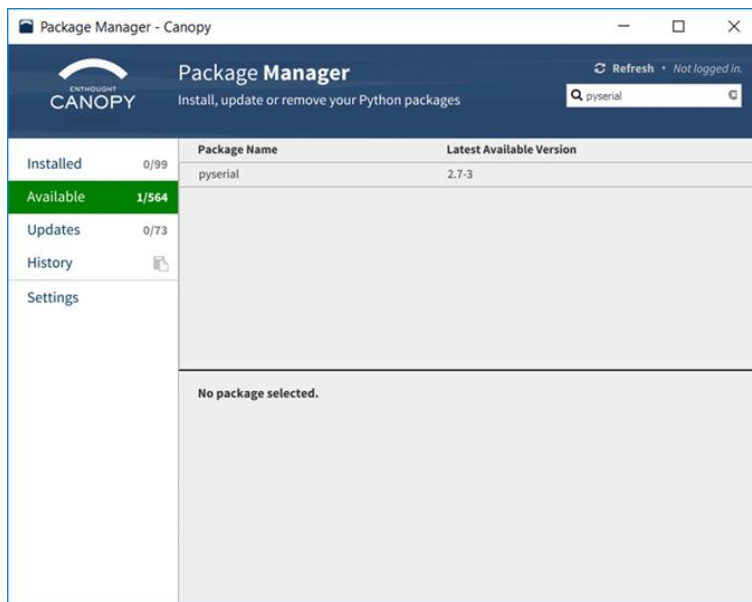


4.2.1 Getting the PySerial library for Canopy

Now that we have Python, we need to update it with the PySerial library so that we can read our data from the computer serial port. If we go back to the Canopy main window we will see a "Package Manager." The Package manager lets us add new parts of Python, and that is just what we want to do. Choose

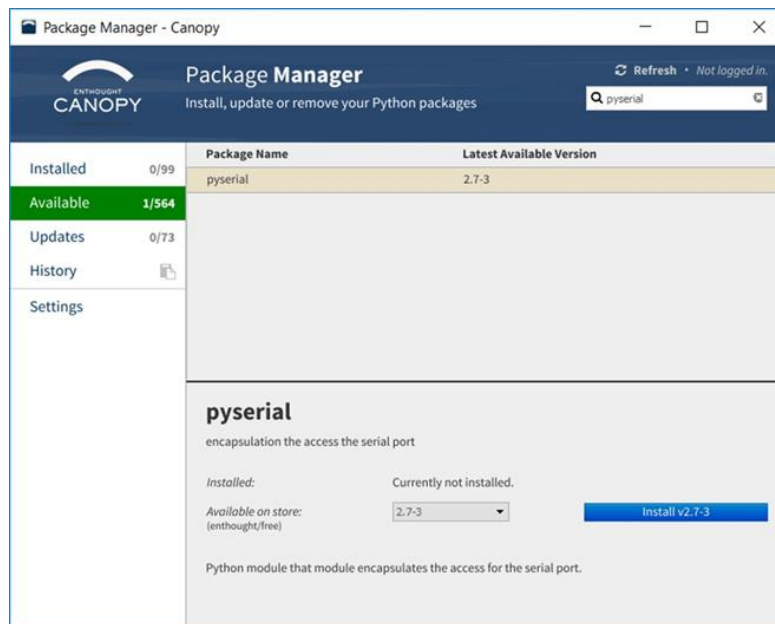


the Package Manager and in its search box type in “pyserial.”



It won't initially find pyserial because it is not yet installed and we are, by

default, looking at what is installed. But choose the “Available” tab. Now we see pyserial in the package list. Select Pyserial and choose the install button



that appears. If all goes well, you will see a red “uninstall” button appear.

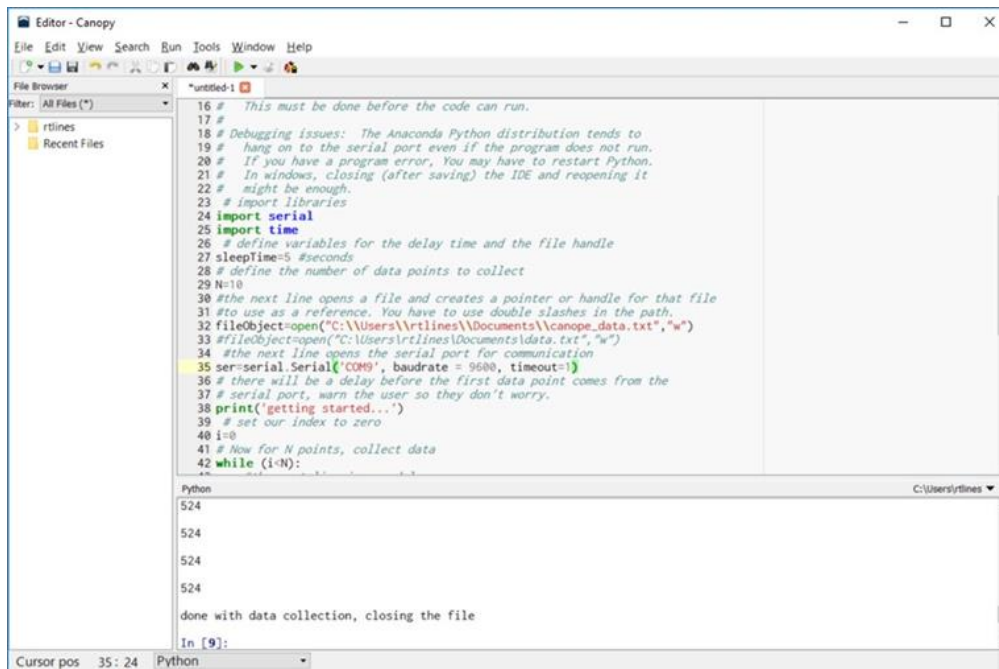
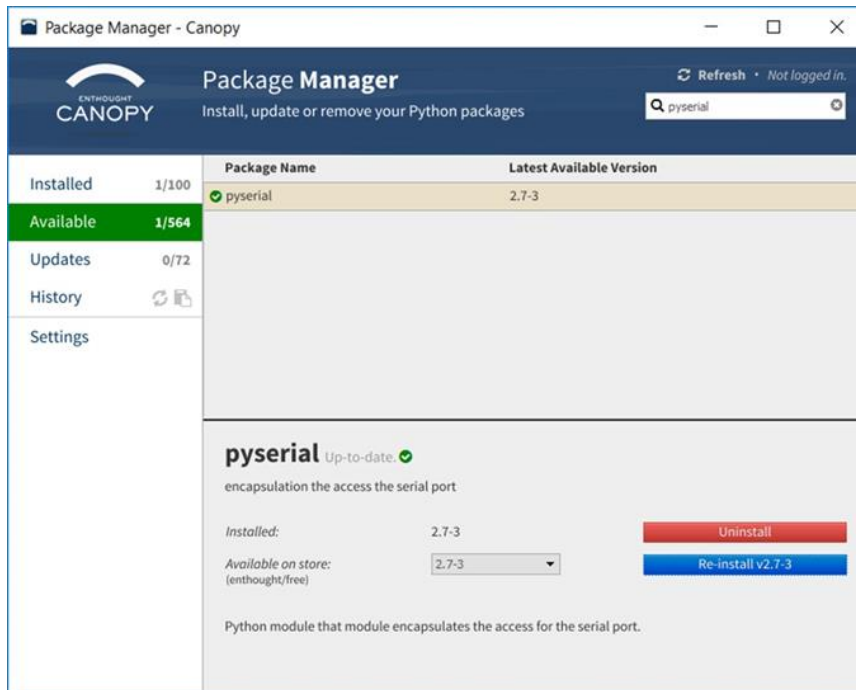
Now we can go back to the editor and write our code to read the serial port.

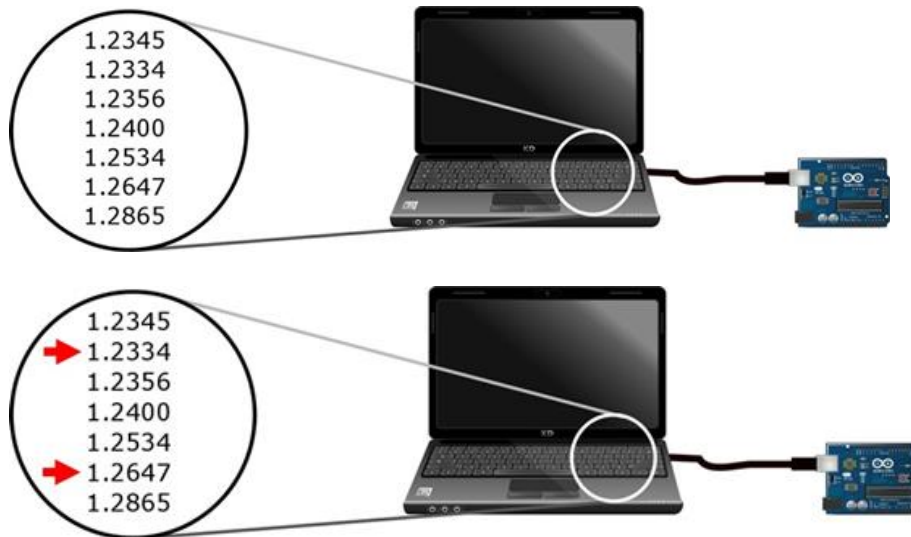
4.3 Getting data from the Arduino

It might help to know how a serial port works. The serial port in the computer takes in data from the serial cable. It stores the data in a temporary place in memory called a *buffer*. Whatever data comes into the port goes into that memory location.

Suppose that we set up our Arduino to send data to the port. The Arduino sends data every few milliseconds. But, suppose we only want data every few seconds. We only want some of the data that the Arduino has sent. The computer has placed every data point sent by the Arduino into its buffer, and the buffer must be read in sequence. You can’t skip data values. But this is just what we want to do, skip some data values and take a value every few seconds. A way to do this is to continuously read in data, but only store it every few seconds. We will use this technique in the code that follows.

Just for fun, let’s think of an actual data collection. Suppose we want to measure a voltage from our Arduino, but we want to measure it many times, each time five seconds apart. This way we are looking at how the voltage changes over time. We might want 10 total measurements.





You might be an expert Python programmer, and if so you can probably see how to write this code. If so, go ahead and do so. But if not, let me introduce some of the Python code elements we will need, and then give an example code.

The first new code piece is making files for our data. We create a file with a line like this (see the actual code below):

```
fileObject=open("C:\\Users\\rtlines\\Documents\\data.txt","w")
```

The “fileObject” is a variable that contains all the file information like the path and file name. It is way easier to type than to include all that information each time we use a file. So we will use fileObject variables. In the code below I named the fileObject variable “dataFile.” Of course, you will have to choose your own path where you will place the data file (you can’t use mine, because you don’t have my computer!) and you will need to choose your own file name. Notice the weird double slashes “\\.” These are a Python thing and you need to write the path this way.

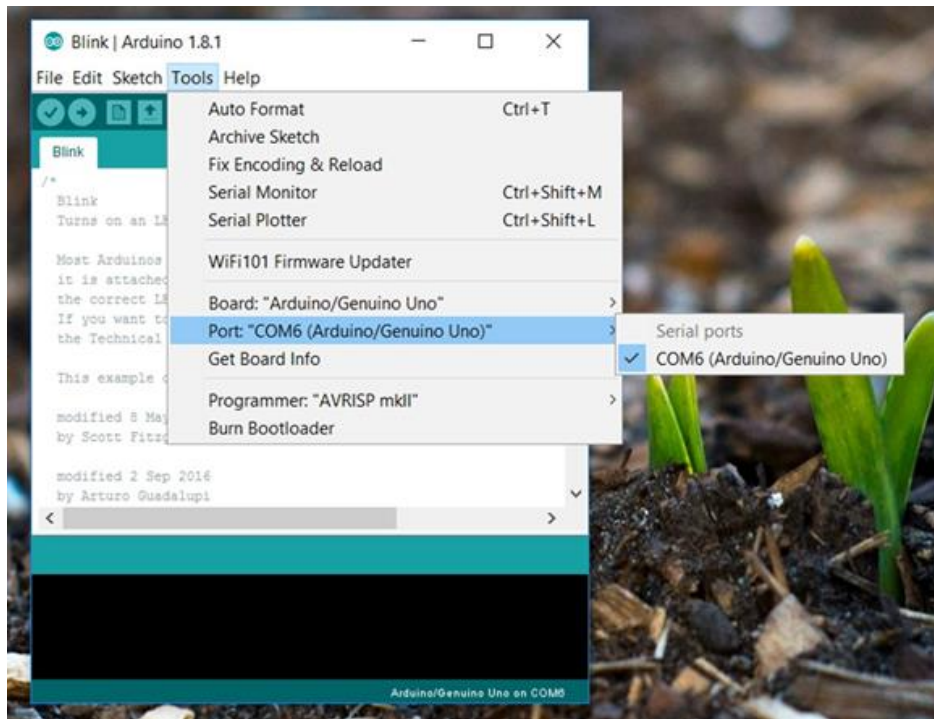
The other new code piece is dealing with serial ports. Like with our Arduino code, we have to set up the serial port. We do that with the line like the this (also see the actual code below).

```
ser=serial.Serial('COM6', baudrate = 9600, timeout=1)
```

Note that when I was using this code my Arduino was in COM6. But that might not be true for you. Use the Arduino app to find out where your Arduino is connected.

The COM port number must match. This looks a little uglier on a Mac, but is much the same. Once the serial port is set up, a line like

```
arduinoData=ser.readline().decode('ascii')
```



will read data from the serial port and “decode it” so that it is text that we can use in another program. The rest of the code writes the data point to a file. I have it calculating the time since the beginning of our data collection (we might just need that in a future lab) and outputting that into the file as well.

We are going to want a name for the amount of time in between data points. In the example code, the amount of time to delay in between data points is named *sleeptime* and the number of data points is named *N*.

At the end of the program we close the file

```
fileObject.close()
```

```
ser.close()
```

so that it will be ready to use next time. Your complete code might look something like this.

[Download here](#)

```
#-----
# Python Code to read a stream of data from the serial port
# and save it to a file
#-----
# The idea is to read a series of voltages from an Arduino
# connected to the serial port where the Arduino is being
# used as the Analog to Digital converter. Both the voltage
```

```

# and the time the voltage was taken are sent to the serial port.
#
# We will use two libraries, serial and time
# The serial library is used to read the serial port
# The time library is used to space out our data collection by
# adding a delay in between data points. The amount of time
# to wait in between data points is called "timeBetween."
#
# We may have to install the serial library. If you have the
# Anaconda Python for Windows, you can open an Anaconda
# window and use the command 'conda install pyserial'
# This must be done before the code can run.
#
# Debugging issues: The Anaconda Python distribution tends to
# hang on to the serial port even if the program does not run.
# If this happens, try sending the python command ser.close()
# at the command prompt. If this doesn't work, You may have to
# restart Python.
# In windows, closing (after saving) the IDE and reopening it
# might be enough.
#-----
# import libraries
import serial
import time

# define variables for the delay time we wait between data points
timeBetween=5 #seconds

# define the number of data points to collect
N=20

#the next line opens a file and creates a pointer or handle for that
# file
# to use as a reference. You have to use double slashes in the path.
# The pointer, "dataFile" takes the place of all the path and file
# name so it is easier to use in the code below
# This line worked for Brother Lines, but won't work for you as it is.
# You need to replace "rtlines" with your username at a minimum.
dataFile=open('C:\\Users\\rtlines\\Documents\\data2.txt','w')

#the next line opens the serial port for communication
ser=serial.Serial('COM3', baudrate = 9600, timeout=1)

#there will be a delay before the first data point comes from the
# serial port, warn the user so they don't worry.
print('getting started...')

# set our index to zero
i=0

# Now for N points, collect data and write it to a file
while (i<N): #Begin data collection loop
    #We will take data every "timeBetween" seconds. We need to know
    # when we start waiting so we can tell if it is time to collect
    # data yet. Use the time.time() to get the current time in
    # seconds

```

```

# since Jan 1,1970. Yes that is a weird way to measure time, but
# computers do it this way.
waitStart=time.time()

#Data comes to the serial port fast. We will continually read
# the data as fast as it comes, but only save it every timeBetween
# seconds. The next while loop keeps us reading in data, but only
# when the current time - waitStart >= timeBetween will we use
# the data.
while (time.time()-waitStart<timeBetween): #Begin Data read loop
    # Get data from the serial port
    # it should have a time and a voltage
    arduinoData=ser.readline().decode('ascii')
    # end of the Data read loop

    # the next line just prints the voltage point on the console so the
    user
    # feels like something is happening.
    print(arduinoData)
    # This next line writes combines the time since we started and the
    Arduino
    # value from the serial port into one string
    writeString=str(arduinoData) #+ " \n"
    # The next line writes our time plus Arduino value to the file.
    dataFile.write(writeString)
    # and finally we increment the loop counter
    i=i+1      # end Data collection loop

# Print out a message saying we are done
print("done with data collection, closing the file and the serial port"
      )

# Close the file
dataFile.close()
# Close the serial port
ser.close()
#-----
#-----

```

Make sure you understand what each line does. Discuss each line with a group member or with the instructor. Python code runs one line at a time. That is different than our Arduino code that must be checked and translated before it goes to the Arduino. Errors in Python code show up as the code runs. If you use the Spyder IDE, the errors show up in the little output box to the right. If you use Canopy they show up in the lower box of the editor.

I modified my simple voltmeter to give the time that the data was taken and to send both the time and the voltage to the serial port. Here is my sketch (remember since it is a simple voltmeter it can only handle 0V to +5V.)

[Download here](#)

```

////////////////////////////////////
// very simple voltmeter that also returns the time since
// the data collection started with the voltage.
// will measure 0 to 5V only!
// Voltages outside 0 to 5V will destroy your Arduino!!!

```

```

////////////////////////////////////
int AI0 = 0;
float delta_v_min=0.0049;    // volts per A2D unit
int value = 0;
float voltage = 0.0;

////////////////////////////////////

void setup() {
    // put your setup code here, to run once:
    //Initiate Serial Communication
    Serial.begin(9600);      //9600 baud rate
}

////////////////////////////////////

void loop() {
    // read in the voltage
    // in A2D units form the serial port
    value = analogRead(AI0);
    Serial.print("_time_in_milliseconds_");
    // the millis() function gives the time
    // in milliseconds since the sketch started
    Serial.print(millis());
    // convert to voltage units using delta_v_min
    voltage = value * delta_v_min;
    Serial.print("_voltage_");
    Serial.println(voltage, 4);
}

////////////////////////////////////
////////////////////////////////////

```

You can't use the Arduino serial monitor and get data from the serial port using Python at the same time. So turn off the serial monitor if it is running.

Now that we have our data in a file, we can analyze it. You might try opening the file in Excel or another spreadsheet program and plotting the data. If you know Python, you could add more code to plot the data right in the code that takes it from the serial port. But, if you are new to Python, you could plot the data in something like a Excel or LoggerPro. Ask for help if you don't know how to do this. We will plot data in future labs.

4.4 Getting Pyserial if you have a Mac

Of course, we have a diversity of computers on campus. The instructions I gave above are for a PC type computer.

4.4.1 Anaconda Mac Users 4.1.2

Go to the Anaconda Prompt and type in: `conda install -c anaconda pyserial`. This will install pyserial -v3.4.

If this does not work for some reason, try going to the Anaconda Navigator, on the left hand side, click Environments. Next, on the right hand side, change the drop-down from Installed to Not Installed. Then enter serial in the search box. Check the box for pyserial and click the Apply button. If you have Spyder running, then relaunch Spyder.

4.4.2 Canopy Mac Users 4.2.1

The steps are the same as the Microsoft Version.

4.4.3 Manually install pyserial through the terminal on a Mac. Coding in emacs/VI

This is kind of a “last resort” approach, so only try this if the other methods above fail.

1. First, go to <https://pypi.python.org/pypi/pyserial> and download *pyserial-3.4.tar.gz* or the version that correlates with the version of python you are using. If you are using python 2 then instead of 3.4, you should look for a file that starts with 2.x; however, if you are using python 3 then 3.4 is the correct version of pyserial you are looking for (but please consider using python 3.x!).
2. Be sure that you downloaded to your Downloads folder.
3. Go to your search bar and type in *terminal* and open that application.
4. Type into the command line `cd Downloads` then press enter.
5. Next, type in `tar -xzf pyserial-3.4.tar.gz` then press enter. Type in `cd pyserial-3.4`, press enter
6. Then type in `sudo python3 setup.py install`.
7. If you are using python 2 then only type in *python* where it says *python3*.

After that you are ready to use the serial library in python.

4.4.4 Mac Pathway and Port Notation

Mac computers list paths to files differently than PC computers.

Mac Pathway

In fact, Mac computers don't make file locations obvious to users at all! But our python code needs to know where to put the files we build, so we will need to understand how to do this.

On the line of code that starts with *dataFile* you will replace it with *dataFile = open("/Users/rtlines/Documents/data.csv", "w")*. This is in the form of /User-s/username/folder name/(optional if you have a folder within the previous folder) folder name/file name + extension (e.g. .csv or .txt).

You can find your username by going to your download folder then right click on any file in there and select **Get Info**. Next make sure that the arrow to the left of *General* is pointing down. Once it is pointing down look at the line that reads, **Where: Macintosh HD → Users → your username will be here → Downloads**.

Mac Port

Mac computers also deal with serial ports differently. So we need to change that next line of code after the *dataFile* line that begins with *ser = serial...* The line of code will need look something like *ser = serial.Serial('/dev/cu.usbmodem1411', baudrate = 9600, timeout = 1)*. However, yours may vary slightly by a different usbmodem number. To find out what to place in between the apostrophes is by going to your arduino code, click on **Tools**, scroll down to **Port** and write down what is written there minus what is in the parenthesis.

4.4.5 Mac version of the python code

Download here

```
#-----
# Python Code to read a stream of data from the serial port
#   and save it to a file
#-----
#   The idea is to read a series of voltages from an Arduino
#   connected to the serial port where the Arduino is being
#   used as the Analog to Digital converter. Both the voltage
#   and the time the voltage was taken are sent to the serial port.
#
# We will use two libraries, serial and time
#   The serial library is used to read the serial port
#   The time library is used to space out our data collection by
#   adding a delay in between data points. The amount of time
#   to wait in between data points is called "timeBetween."
#
# We may have to install the serial library. If you have the
#   Anaconda Python for Windows, you can open an Anaconda
#   window and use the command 'conda install pyserial'
#   This must be done before the code can run.
#
# Debugging issues: The Anaconda Python distribution tends to
#   hang on to the serial port even if the program does not run.
#   If this happens, try sending the python command ser.close()
```

```

# at the command prompt. If this doesn't work, You may have to
# restart Python.
# In windows, closing (after saving) the IDE and reopening it
# might be enough.
#-----
# import libraries
import serial
import time

# define variables for the delay time we wait between data points
timeBetween=5 #seconds

# define the number of data points to collect
N=20

#the next line opens a file and creates a pointer or handle for that
# file
# to use as a reference. You have to use double slashes in the path.
# The pointer, "dataFile" takes the place of all the path and file
# name so it is easier to use in the code below
# This line worked for Brother Lines, but won't work for you as it is.
# You need to replace "rtlines" with your username at a minimum.
# PC version next:
# dataFile=open("C:\\Users\\rtlines\\Documents\\data.txt", "w")
# MAC version next
dataFile = open('/Users/rtlines/Documents/data.csv', 'w')

#the next line opens the serial port for communication
# PC version next
#ser=serial.Serial('COM3', baudrate = 9600, timeout=1)
# MAC version next
ser = serial.Serial('/dev/cu.usbmodem1411', baudrate = 9600, timeout =
1)

#there will be a delay before the first data point comes from the
# serial port, warn the user so they don't worry.
print('getting started...')

# set our index to zero
i=0

# Now for N points, collect data and write it to a file
while (i<N): #Begin data collection loop
    #We will take data every "timeBetween" seconds. We need to know
    # when we start waiting so we can tell if it is time to collect
    # data yet. Use the time.time() to get the current time in
    # seconds
    # since Jan 1,1970. Yes that is a weird way to measure time, but
    # computers do it this way.
    waitStart=time.time()

    #Data comes to the serial port fast. We will continually read
    # the data as fast as it comes, but only save it every timeBetween
    # seconds. The next while loop keeps us reading in data, but only

```

```

# when the current time - waitStart >= timeBetween will we use
# the data.
while (time.time()-waitStart<timeBetween): #Begin Data read loop
    # Get data from the serial port
    # it should have a time and a voltage
    arduinoData=ser.readline().decode('ascii')
    # end of the Data read loop

    # the next line just prints the voltage point on the console so the
    # user
    # feels like something is happening.
    print(arduinoData)
    # This next line writes combines the time since we started and the
    # Arduino
    # value from the serial port into one string
    writeString=str(arduinoData) #+ " \n"
    # The next line writes our time plus Arduino value to the file.
    dataFile.write(writeString)
    # and finally we increment the loop counter
    i=i+1      # end Data collection loop

# Print out a message saying we are done
print("done with data collection, closing the file and the serial port"
      )

# Close the file
dataFile.close()
# Close the serial port
ser.close()
#-----
#-----

```

Make sure you understand each line of this code. It is a little fancy in that it tries to check to make sure the SD card file is working properly and warns you if something is wrong. But most of the code is comments. So don't be discouraged by the length.

4.5 Lab Assignment

1. Finish any part of the last lab that you haven't done.
2. Using Python, Save Arduino data to a file on your computer
 - (a) Wire up a simple voltmeter and load its sketch. Build a simple circuit to test like we did back in section (2.1). Start your Arduino and check to make sure voltages are going to the serial port by looking at the serial monitor or plotter.
 - (b) Check with your group members to make sure their simple voltmeters are working. Help if they are not.
 - (c) Start your Python system (Spyder or Canopy if you are following the instructions given above) and write the program to read the serial port and save the data to a file.

- (d) Close the serial monitor and/or serial plotter. Then run your Python code. Check to make sure that the file of data is written properly. The voltages written in the file should match what your power supply and circuit provided.
 - (e) Make sure you save this program and record what you did in your lab notebook.
 - (f) Make sure your lab group members all have Python programs that run and save data correctly. Help if they do not.
3. Together with your group, fill out a “brainstorming sheet” in preparation for your design project later in the semester.

