

Project Based Engineering Instrumentation With CircuitPython

A Brief Textbook Presented to the
Student Body of the University of South Alabama

Last Update: June 2, 2022
Copyright © Carlos José Montalvo

Manuscript Changes

1. Original tutorials in Google Docs created
2. Tutorials moved to LaTeX on this Github
3. December 21st, 2021 - Updated links for manuscript and hardware
4. Tutorials purchased by Tangibles that Teach and moved to url https://tangibles-that-teach.gitbook.io/instrumentation-lab-manual/-MbMx70LQzRmEG_hS7Ld/
5. May 30th, 2022 - Tangibles that teach went out of business and chapter began the move to Github

Changes Needed

1. All chapters from TTT need to be moved here
2. More theory is needed in this book or direction to further reading for the students
3. The new kit has a CPB - Might need to add a chapter explaining the difference and maybe even having the students getting the bluetooth to work.
4. The photo of the button lab could be better.
5. The circuit photo for the LSM6DS33 is not correct
6. Equations on thermistor need to be expanded
7. Equations relating voltage from protocol to Lux needs to be included
8. Example plots for light, sound, acceleration, etc needs to be expanded
9. Pendulum lab must be done in one of two ways. Either the pendulum is attached to a potentiometer or the CPX is mounted to the end of a string and data logged on board the CPX itself. The potentiometer is nice because you can record data with your laptop but the string idea is cool because you use the accelerometer. In either case you can make some really long pendulums
10. 3D printing a disc with holes on the outside to eventually mount to a shaft would be a really cool angular velocity sensor lab. Tangibles that teach could easily include a 3D printed disc that can mount to a pencil for ease of rotation. Could also include the CAD drawings so students can print more or even edit the design for better or worse performance.
11. Buying some load cells with the HMC converter and including them in the kit would add a whole lot different labs
12. Buying some magnetometers to measure magnetic field and do some sensor fusion would be neat. Could do roll, pitch and yaw calibration if we included a magnetometer.
13. Right now a lab just on roll and pitch estimation would be possible. Pendulum lab pretty much introduces them to this but could easily do an rc aircraft lab where they build an aircraft out of foam with an elevator and aileron so that the servos responds to roll and pitch change. There is a lab right now with just pitch but perhaps we could add roll to it.
14. Another cool project idea would be for the students to take temperature and light data on a cloudy day. Then have them infer if the amount of sunlight affected the temperature of the thermistor. They could plot the data with light on the x-axis and temperature on the y-axis and draw conclusions based on the plot they generate.
15. Could also have them take temperature and light data over the course of a whole day to plot sunrise and sunset and watch the ambient temperature rise and fall
16. For the aliasing lab, have the students sample as fast as possible and obtain the natural frequency of the system. Then have them sample at 1.0, 2.0 and 3.0 times the natural frequency they obtained. I originally picked 1,10 and 100 as arbitrary sampling frequencies and it would have been better to do 2,4 and 6.
17. On cool lab would be to take light data during sunset and watch light and temperature plummet.
18. Add this lab on frequency for notes

Acknowledgements

The author, Dr. Carlos Montalvo would like to acknowledge a few key members who made this textbook possible. First and foremost I would like to thank Adafruit for their entire ecosystem of electronics, tutorials, blogs and forums. Much of what I have learned here to teach Instrumentation was from Adafruit and the Adafruit Learn system and specifically people like Lady Ada and John Park who have helped shape CircuitPython and the Circuit-Playground Express to what it is today. I would also like to thank Dr. Saami Yazdani for creating the blueprint for Instrumentation at my university by creating a laboratory environment for an otherwise totally theoretical course. His course was the foundation for this textbook and for that I thank him for showing the way. Id like to also thank and acknowledge Tangibles that Teach for giving me the opportunity to morph this loose set of projects into a textbook that can be used for multiple universities and classrooms and of course help students learn and acquire knowledge through creating.

About this textbook

This textbook has been designed with the student and faculty member in mind. First, this textbook goes hand in hand with Engineering Instrumentation taught at the undergraduate level at many universities. The course begins with simple plotting and moves into data analysis, calibration and more complex instrumentation techniques such as active filtering and aliasing. This course is designed to get students away from their pen and paper and build something that blinks and moves as well as learn to process real data that they themselves acquire. There is no theory in these projects. It is all applied using the project based learning method. Students will be tasked with downloading code, building circuitry, taking data all from the ground up. By the end of this course students will be well versed in the desktop version of Python while also the variant CircuitPython designed specifically for microelectronics from Adafruit. After this course students will be able to understand Instrumentation at the fundamental level as well as generate code that can be used in future projects and research to take and analyze data. Python is such a broad and useful language that it will be very beneficial for any undergraduate student to learn this language. To the professors using this textbook, 1 credit hour labs are often hard to work into a curriculum and live demonstrations in the classroom cost time and money that take away from other faculty duties. Ive created this kit and textbook to be completely stand-alone. Students simply need to purchase the required materials and follow along with the lessons. These lessons can be picked apart and taught sequentially or individually on a schedule suited to the learning speed of the course. I hope whomever reads and learns from this textbook will walk away with an excitement to tinker, code and build future projects using microelectronics and programming.

Contents

1 Purchase Equipment	5
1.1 Parts List	5
1.2 List of Items in Kit	5
1.3 Assignment	6
2 Download Python for Desktop	6
2.1 Thonny	6
2.2 Spyder	7
2.3 Other Options	7
2.4 Setting up your IDE	8
2.5 Scripting	9
2.6 Built-In Help Function and dir()	9
2.7 Assignment	11
3 Getting Started with the CPX/CPB	12
3.1 Parts List	12
3.2 Setting up your Circuit Playground	12
3.3 TL;DR	21
3.4 Assignment	21
4 Troubleshooting Guide	22
5 External LEDs and Push Buttons	22
5.1 Parts List	22
5.2 Learning Objectives	22
5.3 LED with no Code	23
5.4 LED with a push button	24
5.5 LED with code	25
5.6 LED with CPX button	26
5.7 Assignment	29
6 Using the CPX/CPB as a Data Acquisition System (DAQ)	30
6.1 Parts List	30
6.2 Learning Objectives	30
6.3 Extra Help	30
6.4 Getting Started	30
6.5 Method 1 - Copying Serial Monitor Data	33
6.6 Method 2 - Automatically Populate a Spreadsheet	33
6.7 Method 3 - Logging Data Directly to on board memory	36
6.8 Plotting Logged Data	41
6.9 Assignment	42

1 Purchase Equipment

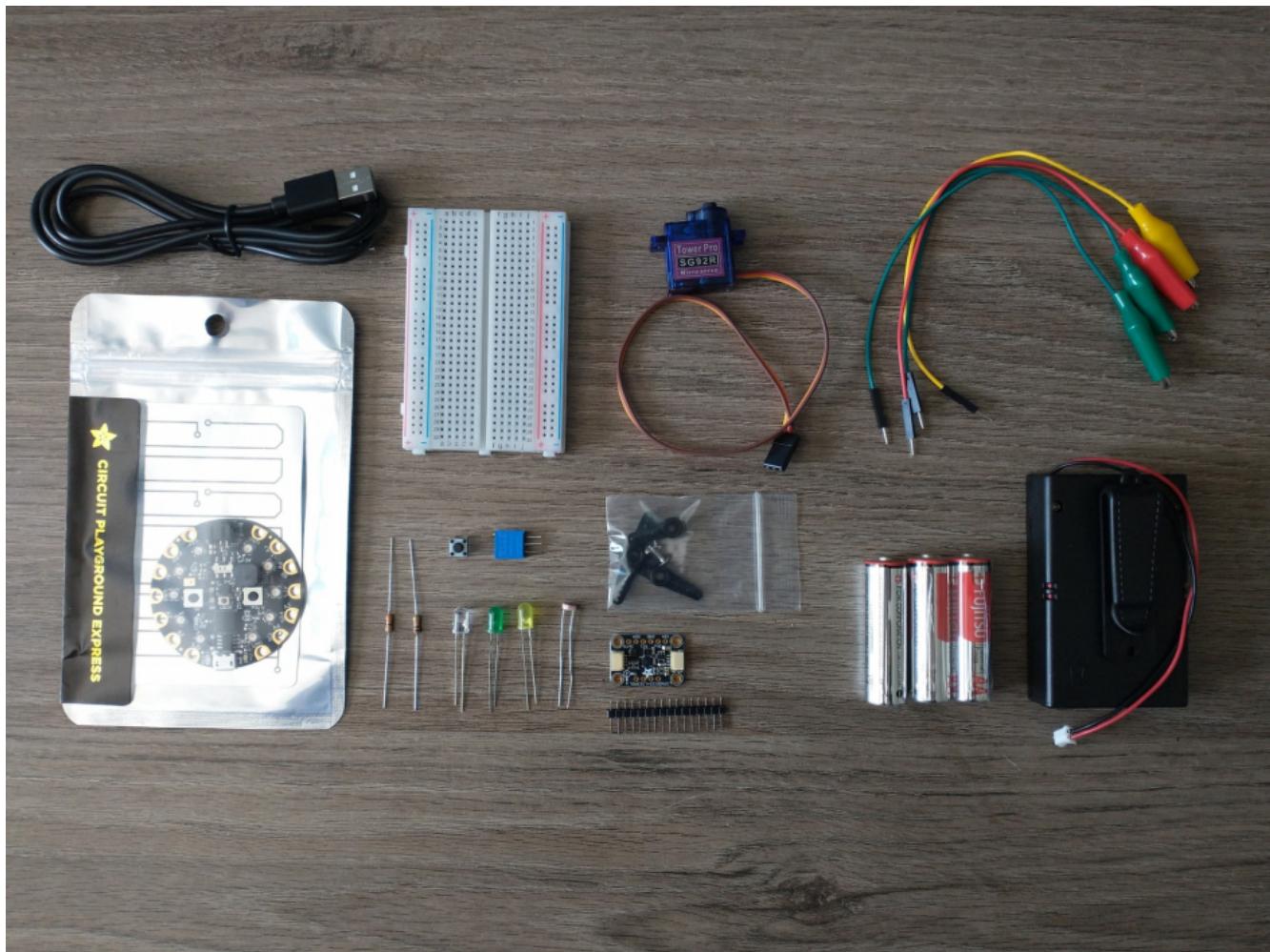
1.1 Parts List

Laptop + internet connectivity

In this class you're going to build some circuits that will enhance your learning experience. Rather than just solving problems by hand you're going to take and analyze data. Over the summer of 2020, I began to work with Tangibles that Teach and they have graciously bundled all components together. At the time of this writing the links below are still valid.

1. PURCHASE KIT AT TANGIBLES THAT TEACH
2. Unboxing video on Youtube

When you get your kit familiarize yourself with all of the components. I created an unboxing video on Youtube for you to take a look. Below is also a photo of all the components.



1.2 List of Items in Kit

The kit above comes with the following items. It is possible for you to purchase all items individually but it's possible you may end up getting the wrong component or paying more on shipping. Please exercise caution if you plan on purchasing everything individually.

1. Circuit Playground Bluefruit or Express + Included USB Cable
2. Micro Servo

3. Photocell
4. Two Resistors (330 Ohm and 1K Ohm)
5. Alligator Clips x3
6. External Battery Pack
7. AAA Batteries x3
8. Breadboard
9. Push Button
10. LEDs x2
11. LSM6DS33 + LIS3MDL - 9 DoF IMU

1.3 Assignment

Purchase the required instrumentation kit and create a document with the following items:

1. A receipt of ALL of your purchases that shows you have purchased all items in the kit or just the kit itself from Tangibles that Teach - 50%
2. If you are working in pairs list who you are working with - 50%

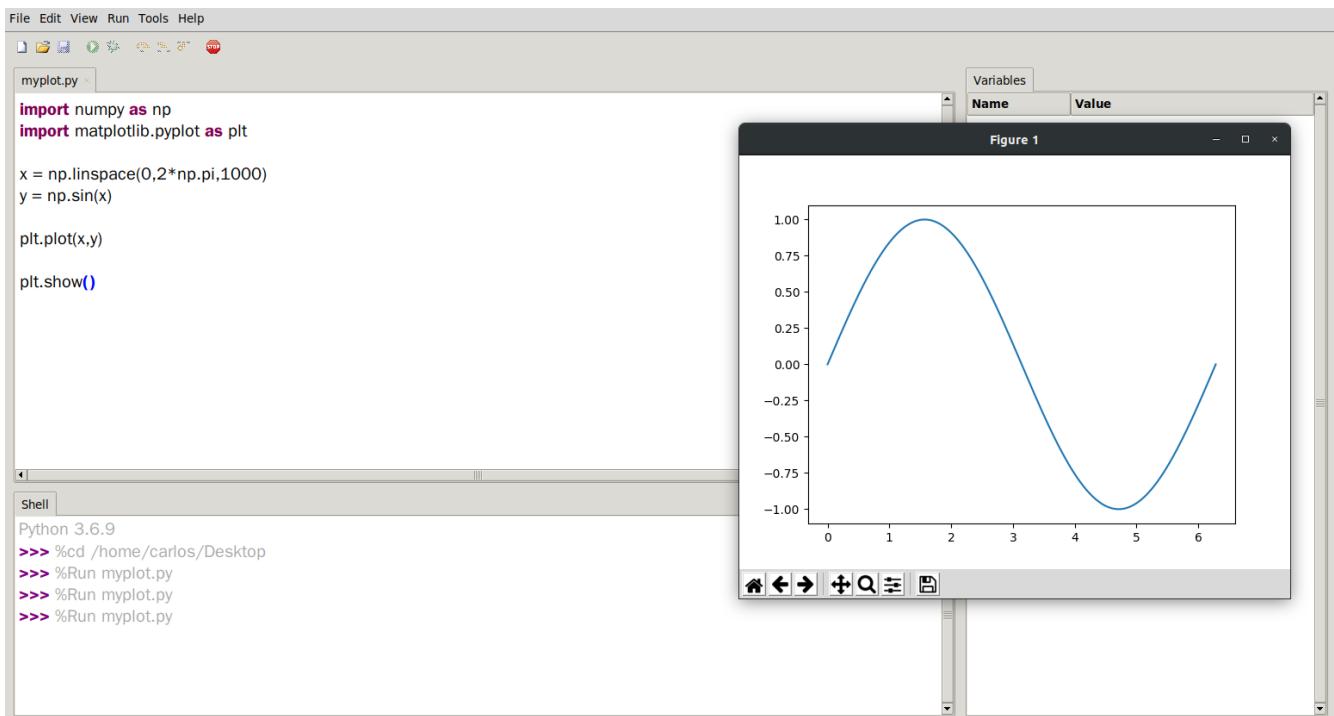
2 Download Python for Desktop

As you learn Instrumentation throughout the semester, you will be tasked with creating computer programs on the Circuit Playground Express (CPX). The CPX itself has its own RAM, CPU, HDD and many sensors. Your CPX is kind of like a mini computer! You can plug the CPX into your computer via USB and access the hard drive (HDD) from your own computer. When you program on the CPX you need to write programs on the CPX itself so that the mini computer can run the program you wrote. The CPX knows how to read multiple different languages but in this class we are going to write everything in the Python language which has been ported to the CPX and called CircuitPython. Since we have to write everything in CircuitPython we need to first learn how to program some things in Python. You can easily download Python by itself but its nice to get whats called an Integrated Development Environment (IDE). This way you can practice writing Python code on your computer while you wait for your purchases to arrive in the mail.

So which IDE can you download and which is recommended? I recommend two IDEs. They are listed below. I recommend getting either one. If you just Google Python download you will find a humongous list of editors (Scratch, Anaconda, Canopy, Eclipse, PyDev, etc). Its easy to get lost when searching for something so broad. Youve been warned.

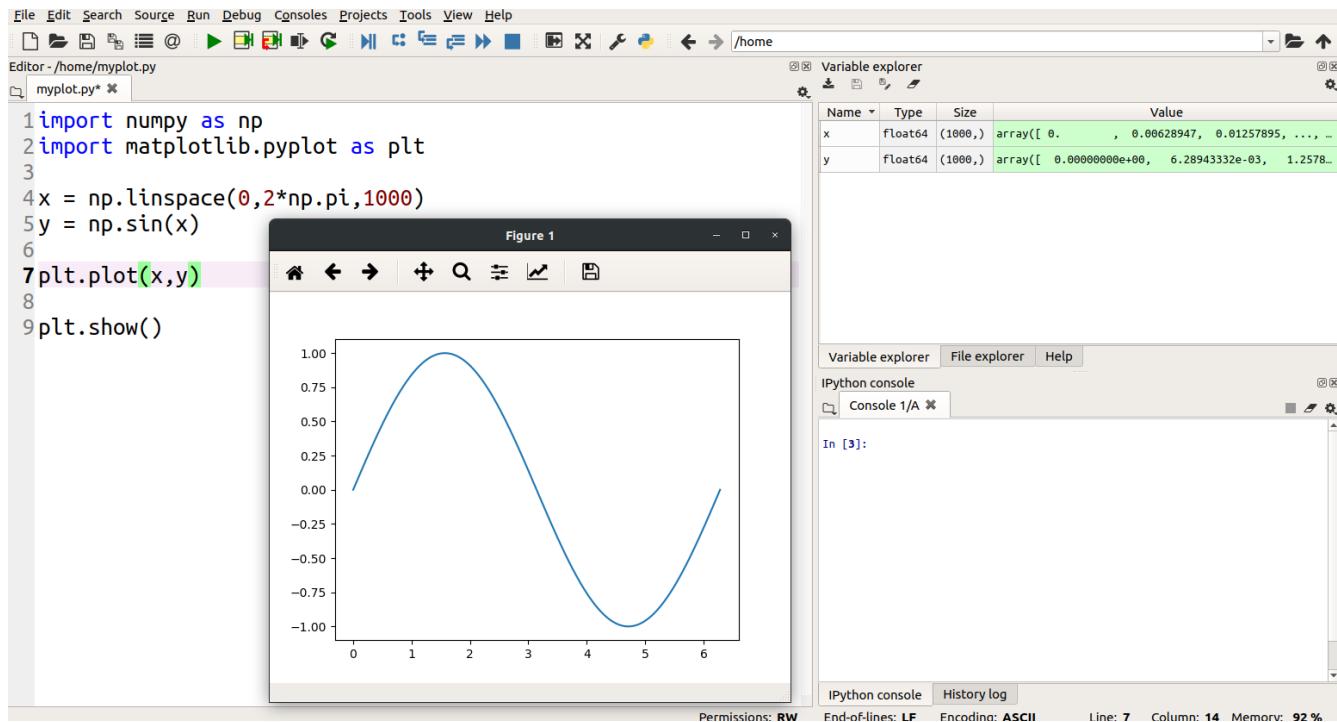
2.1 Thonny

Thonny - <https://thonny.org/> - Youtube video on how to install



2.2 Spyder

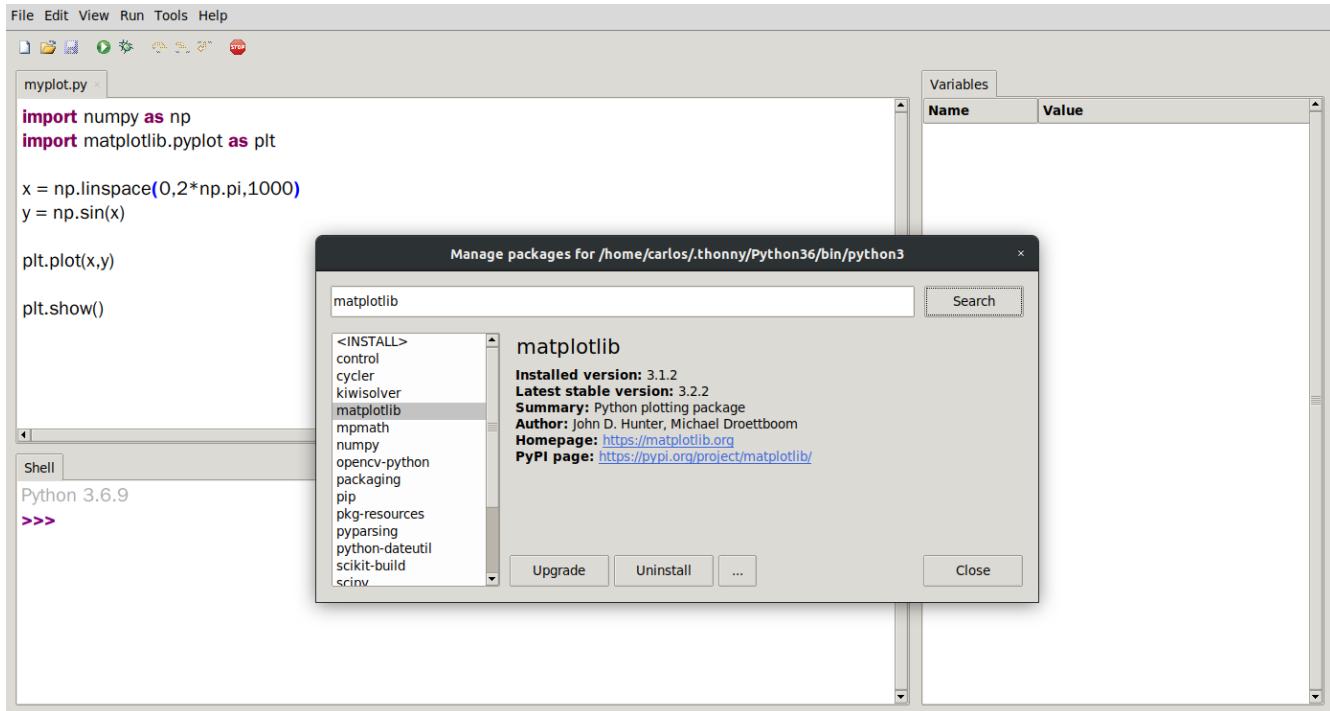
Spyder - <https://www.spyder-ide.org/> - Youtube video on how to install



Well you need to figure out how many bits your computer has. This is a great thing to Google. Type the following: "do I have a 32 bit or 64 bit computer" into Google. Im willing to bet you have a 64 bit computer but you may as well check. Well learn about the difference between 32 and 64 bit computers when we get to the projects on Binary.

2.4 Setting up your IDE

Once you have Thonny or Spyder installed you need to install numpy and matplotlib which are modules within Python that allow us to do some extra things like numerical computation with Python (numpy) and Matlab style Plotting libraries (matplotlib). I explain how to install modules in my Youtube videos above; however, you need to head over to Tools>Manage Packages in Thonny. You can see in the image below I already have version 3.1.2 but I can upgrade to 3.2.2



If numpy or matplotlib is not already included in Spyder then you need to type the following into the Python Console in the lower right hand corner of Spyder which is called the IPython console.

```
!pip install matplotlib
```

If that doesn't work try

```
!pip3 install matplotlib
```

You can see in the output example below that I already have matplotlib installed as it says requirement already satisfied. Assuming you have a valid internet connection it will install the necessary module.

```

IPython console
Console 1/A ✘
In [4]: !pip install matplotlib
Defaulting to user installation because normal site-packages is not
writeable
Requirement already satisfied: matplotlib in ./local/lib/python3.6/
site-packages (3.2.2)
Requirement already satisfied: cycler>=0.10 in ./local/lib/
python3.6/site-packages (from matplotlib) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in ./local/lib/
python3.6/site-packages (from matplotlib) (2.8.1)
Requirement already satisfied: numpy>=1.11 in ./local/lib/python3.6/
site-packages (from matplotlib) (1.19.0)
Requirement already satisfied: kiwisolver>=1.0.1 in ./local/lib/
python3.6/site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: pyparsing!=2.0.4,>=2.1.2,>
=2.1.6,>=2.0.1 in ./local/lib/python3.6/site-packages (from
matplotlib) (2.4.7)
Requirement already satisfied: six in ./local/lib/python3.6/site-
packages (from cycler>=0.10->matplotlib) (1.15.0)

In [5]: |
```

2.5 Scripting

Once you have numpy and matplotlib its time to make a plot. I have a pretty comprehensive youtube video on how to plot in matplotlib but if you prefer text I will walk through a simple example.

```

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0,2*np.pi,1000)
y = np.sin(x)

plt.plot(x,y)

plt.show()
```

The code above will plot a sine wave from 0 to 2pi. The two lines at the top are importing the numpy and matplotlib modules you installed earlier. When they are imported we give them shorter names so its easier to reference them so numpy will now be called np and matplotlib.pyplot will be called plt. The next two lines then create a vector x from 0 to 2pi using 1000 data points. The next line then uses the sine function to create the vector y. Finally x and y are plotted and the figure is instructed to pop up on your screen using the show() function.

2.6 Built-In Help Function and dir()

Running code will always create syntax errors. Typing your syntax error into Google will yield so many results you might get lost. Sometimes it helps to know how to learn things just from your computer. For example, type in the commands below in the IPython console or the Shell.

```
import numpy as np
dir(np)
```

```

>>> import numpy as np
>>> dir(np)
['ALLOW_THREADS', 'AxisError', 'BUFSIZE', 'CLIP', 'ComplexWarning', 'DataSource', 'ERR_CALL', 'ERR_DEFAULT', 'ERR_IGNORE', 'ERR_LOG', 'ERR_PRINT', 'ERR_RAISE', 'ERR_WARN', 'FLOATING_POINT_SUPPORT', 'FPE_DIVIDEBYZERO', 'FPE_INVALID', 'FPE_OVERFLOW', 'FPE_UNDERFLOW', 'False_', 'Inf', 'Infinity', 'MAXDIMS', 'MAY_SHARE_BOUNDS', 'MAY_SHARE_EXACT', 'MachAr', 'ModuleDeprecationWarning', 'NAN', 'NINF', 'NZERO', 'NaN', 'PINF', 'PZERO', 'RAISE', 'RankWarning', 'SHIFT_DIVIDEBYZERO', 'SHIFT_INVALID', 'SHIFT_OVERFLOW', 'SHIFT_UNDERFLOW', 'ScalarType', 'Tester', 'TooHardError', 'True_', 'UFUNC_BUFSIZE_DEFAULT', 'UFUNC_PYVALS_NAME', 'VisibleDeprecationWarning', 'WRAP', '_NoValue', '_UFUNC_API', '__NUMPY_SETUP__', '__all__', '__builtins__', '__cached__', '__config__', '__doc__', '__file__', '__git_revision__', '__loader__', '__name__', '__package__', '__path__', '__spec__', '__version__', '_add_newdoc_ufunc', '_distributor_init', '_globals', '_mat', '_pytesttester', 'abs', 'absolute', 'absolute_import', 'add', 'add_docstring', 'add_newdoc', 'add_newdoc_ufunc', 'alen', 'all', 'allclose', 'alltrue', 'amax', 'amin', 'angle', 'any', 'append', 'apply_along_axis', 'apply_over_axes', 'orange', 'arccos', 'arccosh', 'arcsin', 'arcsinh', 'arctan', 'arctan2', 'arctanh', 'argmax', 'argmin', 'argpartition', 'argsort', 'argwhere', 'around', 'array', 'array2string', 'array_equal', 'array_equiv', 'array_repr', 'array_split', 'array_str', 'asanyarray', 'asarray', 'asarray_chkfinite', 'ascontiguousarray', 'asfarray', 'asfortranarray', 'asmatrix', 'asscalar', 'atleast_1d', 'atleast_2d', 'atleast_3d', 'average', 'bartlett', 'base_repr', 'binary_repr', 'bincount', 'bitwise_and', 'bitwise_not', 'bitwise_or', 'bitwise_xor', 'blackman', 'block', 'bmat', 'bool', 'bool8', 'bool_', 'broadcast', 'broadcast_arrays', 'broadcast_to', 'busday_count', 'busday_offset', 'busdaycalendar', 'byte', 'byte_bounds', 'bytes0', 'bytes_', 'c_', 'can_cast', 'cast', 'cbrt', 'cdouble', 'ceil', 'cfloat', 'char', 'character', 'chararray', 'choose', 'clip', 'clongdouble', 'clongfloat', 'column_stack', 'common_typ

```

I included a photo of the output from the dir function. Youll notice there are a ton of functions in numpy. Every function in Python has a

`__doc__`

function. Thats two underscores followed by doc and then another two underscores. If youre ever curious about what a particular function does you can just run the command below again in the IPython console or Shell. In this example Im looking at what *arctan2* does.

```
print(np.arctan2.__doc__)
```

```

>>> print(np.arctan2.__doc__)

arctan2(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None,
subok=True[, signature, extobj])

Element-wise arc tangent of ``x1/x2`` choosing the quadrant correctly.

The quadrant (i.e., branch) is chosen so that ``arctan2(x1, x2)`` is
the signed angle in radians between the ray ending at the origin and
passing through the point (1,0), and the ray ending at the origin and
passing through the point (``x2``, ``x1``). (Note the role reversal: the
``y``-coordinate" is the first function parameter, the ``x``-coordinate"
is the second.) By IEEE convention, this function is defined for
`x2` = +/-0 and for either or both of `x1` and `x2` = +/-inf (see
Notes for specific values).

This function is not defined for complex-valued arguments; for the
so-called argument of complex values, use `angle`.

Parameters
-----
x1 : array_like, real-valued
    `y`-coordinates.
x2 : array_like, real-valued
    `x`-coordinates. If ``x1.shape != x2.shape``, they must be broadcastable to a common
    shape (which becomes the shape of the output).
out : ndarray, None, or tuple of ndarray and None, optional

```

Youll see that arctan2 takes 2 input arguments x1 and x2. I didnt include the entire output but if you continue to scroll through the output it will even include examples on how to use the function.

Another way to learn certain functions is by visiting the appropriate documentation. The Numpy Docs website for example has all the documentation you need for Numpy. Navigating that website you can find the same documentation for arctan2.

As a last resort you can always Google how to compute the inverse tangent 2 function in Python. Note though that there is so much content out there on Google that you could easily get lost. Still, theres also so much information that the answers are out there for just about anything.

So you have three methods for finding out how to program in python. The dir and `__doc__` functions in Python, using the appropriate documentation online and of course Google. Im lumping Youtube in with Google which is also another way to learn information although when I want to find information quickly I just use the documentation. Its the best in my opinion.

2.7 Assignment

Plot the following function:

$$T(t) = 60(1 - e^{-5t}) + 30 \quad (1)$$

Plot the function from 0 to 10 seconds and label the x axis Time (sec) and the y-axis Temperature (F). Add a grid as well. You might need to look up how to do some of these things.

Upload a PDF with all of the photos below included. My recommendation is for you to create a Word document and insert all the photos into the document. Then export the Word document to a PDF and then upload the PDF.

1. Screenshot of your Python IDE - 25%
2. A selfie of you watching one of my python youtube videos and upload that photo - 25%
3. Screenshot your code of the temperature plot above even if it doesnt work - 25%
4. If you got the code to work, hit the Floppy Disk button (SAVE) when your figure pops up and include that in your upload. No screenshots! - 25%

3 Getting Started with the CPX/CPB

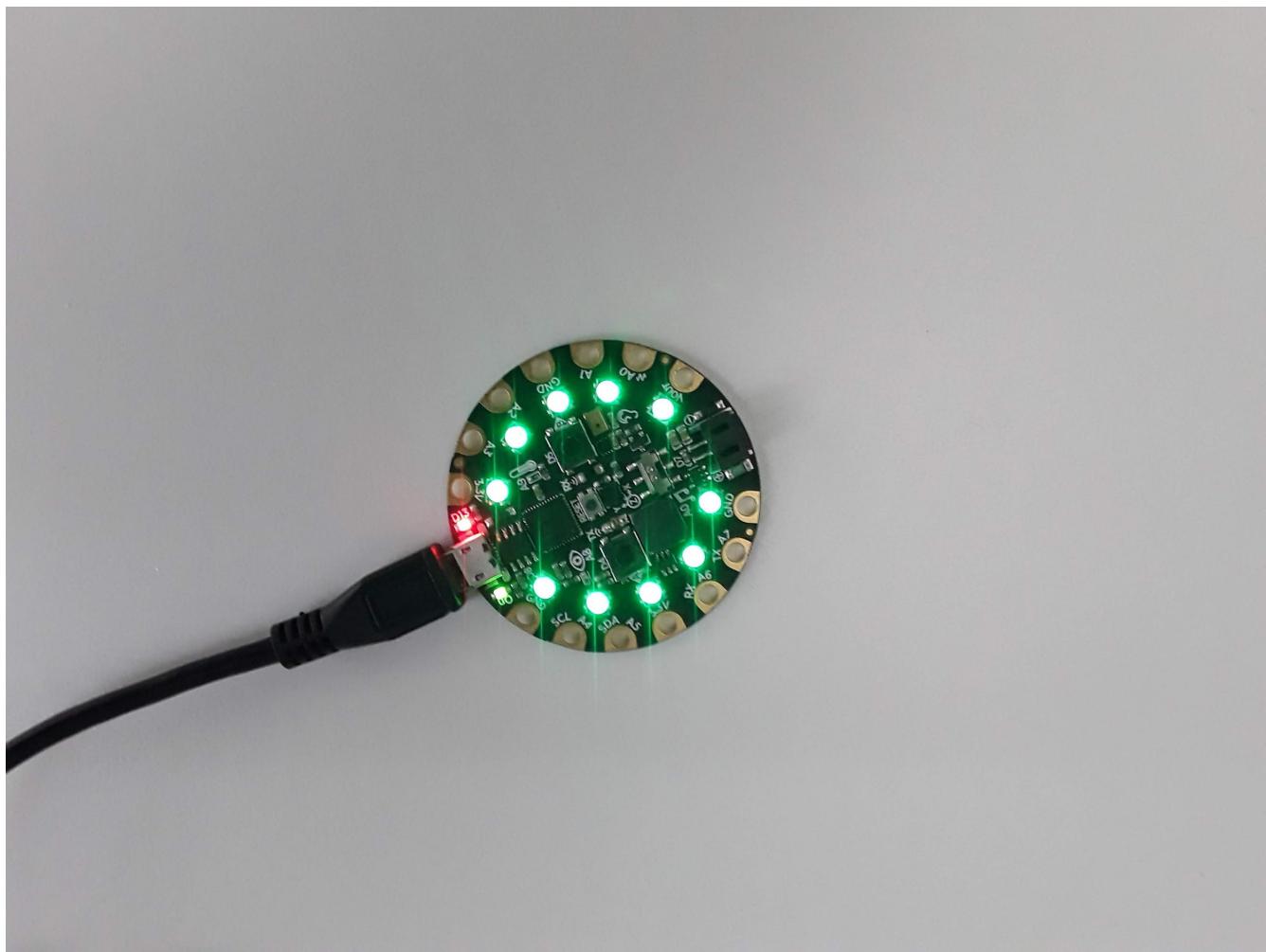
3.1 Parts List

1. Laptop
2. CPX(or CPB)
3. USB Cable (with a data line. Not all USB cables have data lines)

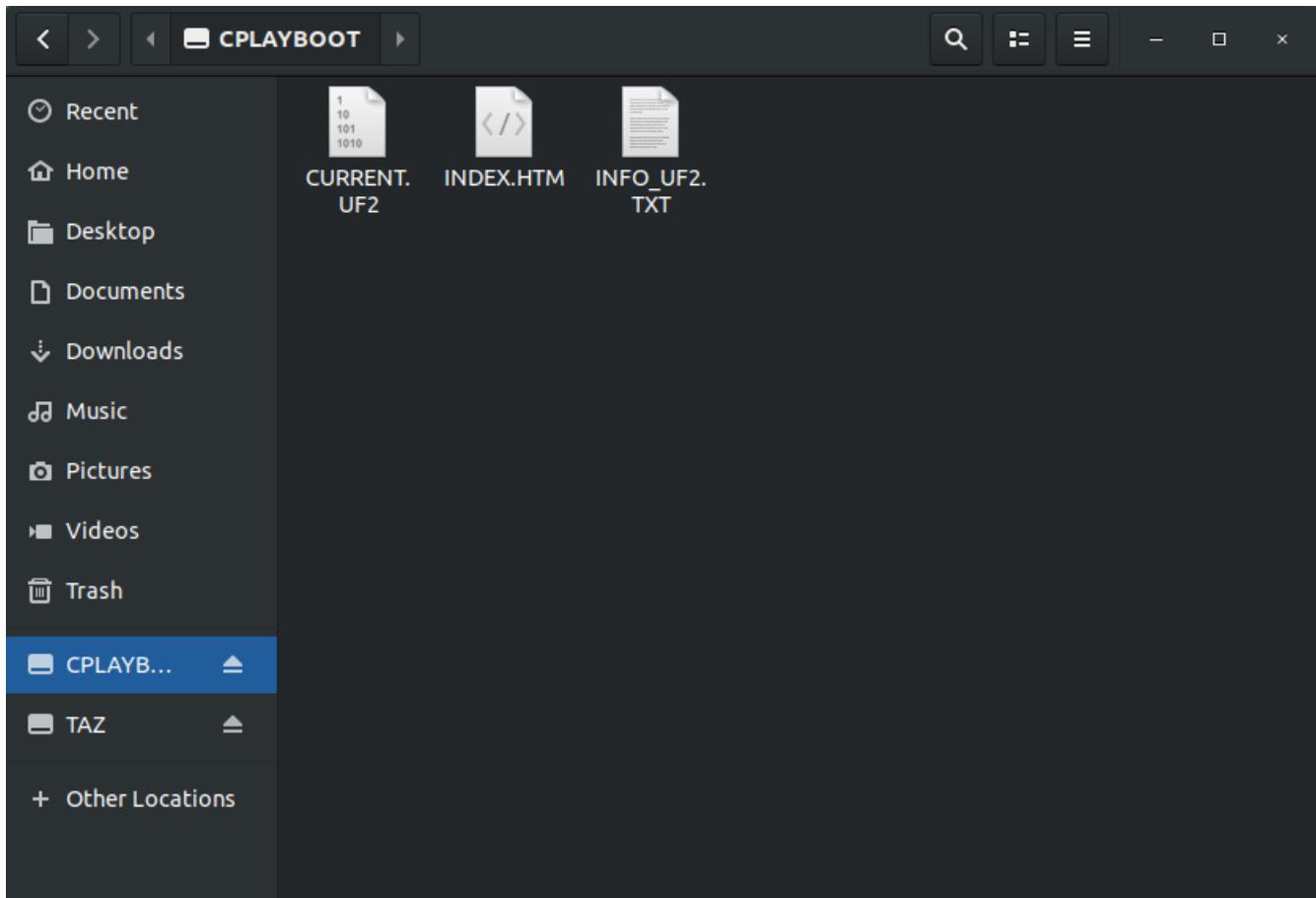
3.2 Setting up your Circuit Playground

By now you hopefully have your Circuit Playground (CPX) and it's time to get your CPX up and running. A very in depth and detailed tutorial can be found on the Adafruit Learn site. The text below is a summary of what you need to do to get the CPX up and running.

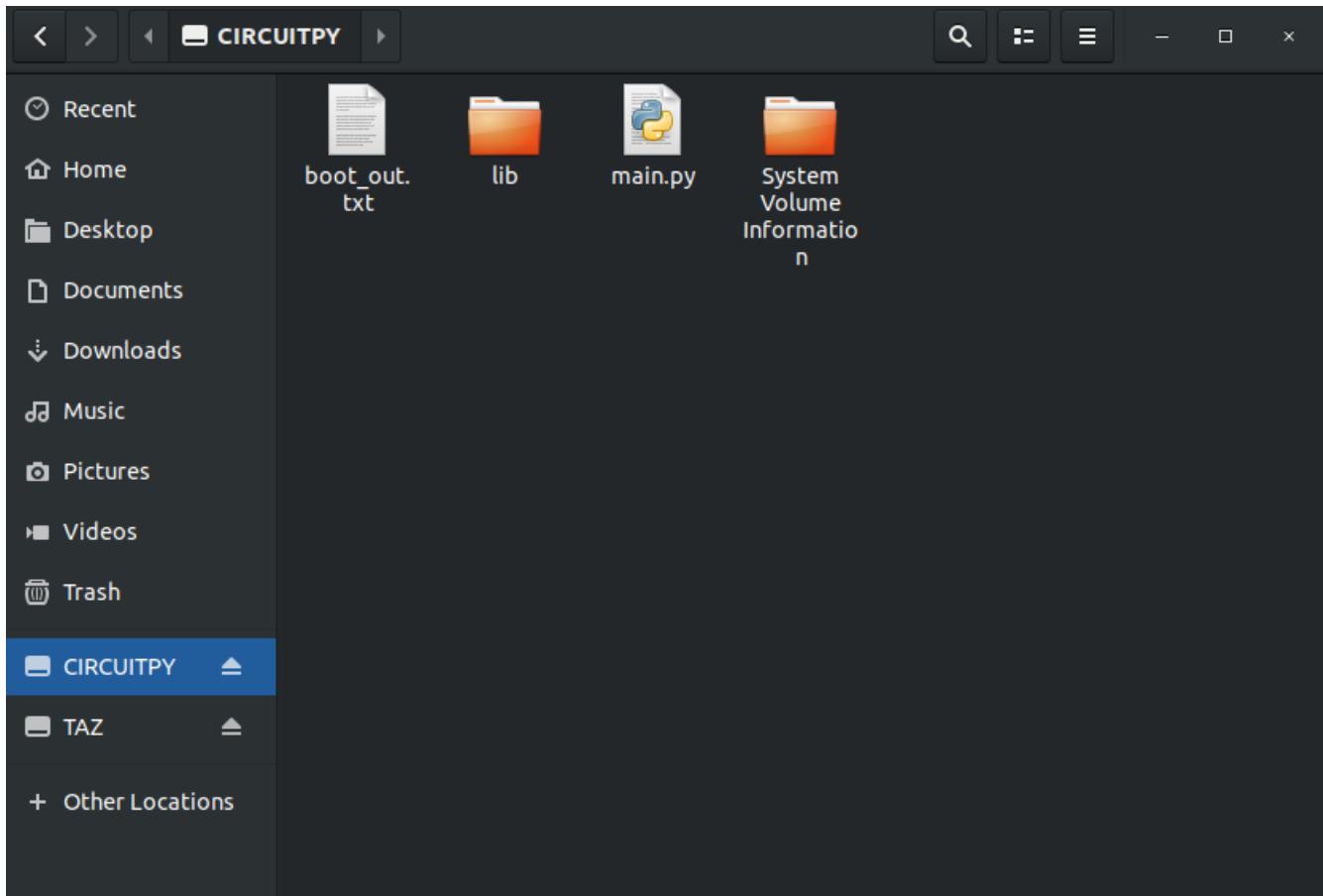
When you get your CPX and plug it into the computer via USB it actually won't run Python just yet. First you need to double click the reset button (the button in the center. It says RESET above the button) and put it into boot mode. All the neopixels (the ring lights on the CPX) will light up green.



Something called CPLAYBOOT will pop up on your computer just like a USB stick or external harddrive. A couple files will be in there but it doesn't matter what they say right now.



You then need to download what's called a UF2 file and transfer it onto the CPX. Note if you purchased a kit with the Bluefruit you need to download a different UF2. Make sure you get the right one. Once the UF2 is downloaded you need to drag the UF2 over to the CPLAYBOOT drive on your computer. After a bit of time a USB drive called CIRCUITPY will pop up as a flash drive on your computer. The CPX is now like a USB stick with 2MB of storage.

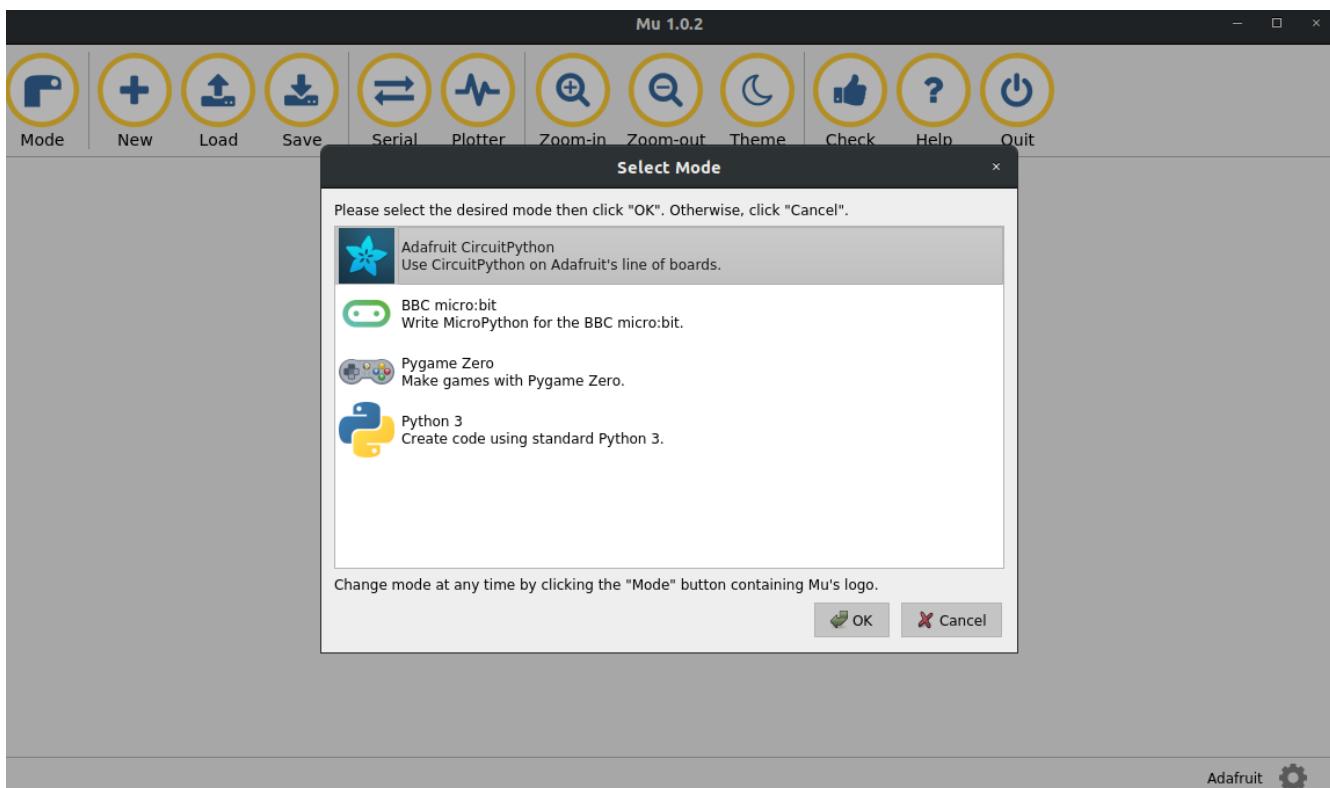


At this point the CPX is like a mini computer. If you put Python code on the flash drive it will run python code. Since I've done this before, there are a number of files already on my CPX. You may have some or none of these. The boot_out.txt file will tell you the version of CircuitPython you have on the CPX. Mine says this:

Adafruit CircuitPython 5.3.0 on 2020-04-29; Adafruit CircuitPlayground Express with samd21g18

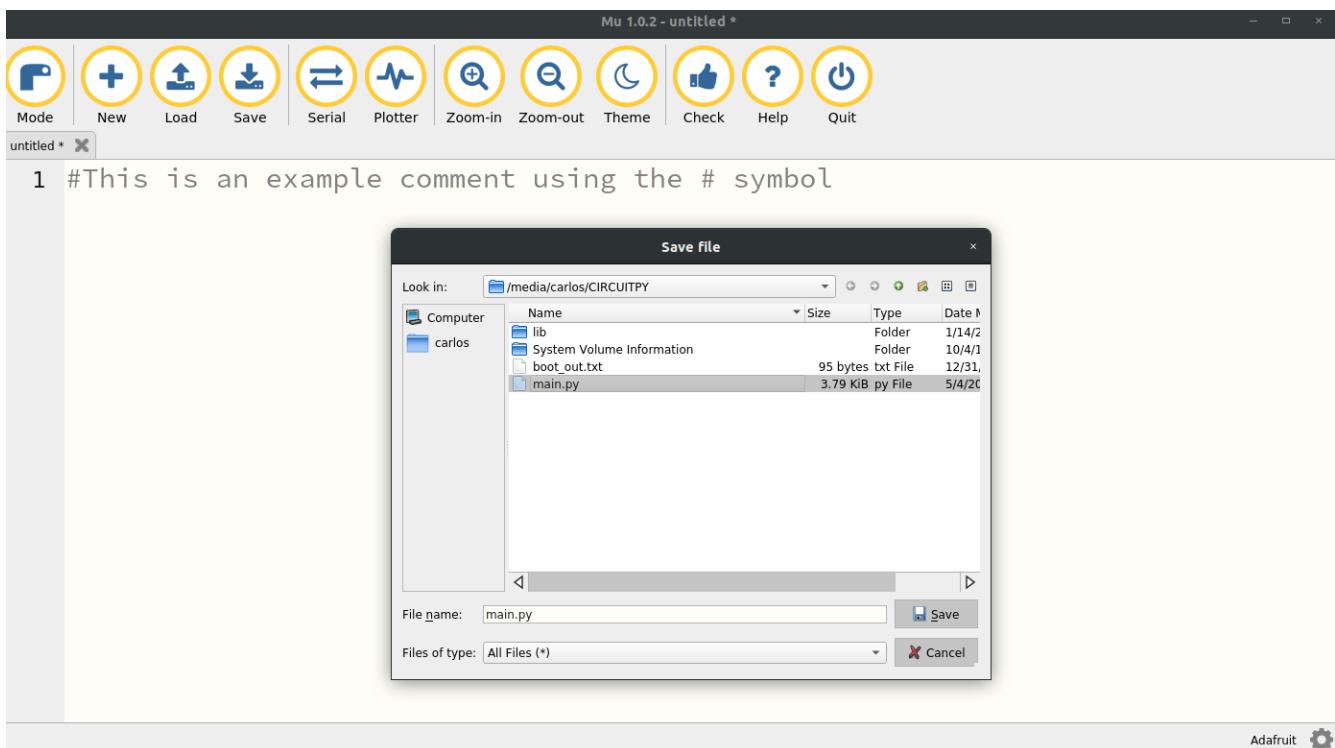
Which means I have CircuitPython version 5.3.0 last updated on April 29th, 2020. The CPX itself is using the ATSAMD21G18 microprocessor. The folder lib is a folder with extra libraries that you may need to install later. The file main.py is the Python file that the CPX is currently running. **The CPX can store as many Python files as possible for a 2MB flash drive but it will only run ONE Python script at a time and that file must be named main.py** The folder *System_Volume_Information* is a file management folder that we will never use.

If you want the CPX to run code you simply need to edit the file *main.py* or if that file does not exist you just need to create it. You could just open Notepad or any other text app (Sublime, TexWrangler, Emacs, Vi, Nano, Gedit, Notepad++, Wordpad, VSCode, etc) but the CPX has a lot of debugging options and it is recommended to use a program called Mu. Mu is a good way to write and debug code on the CPX. Note that Mu is only used to program the CPX. If you want to run Python code on your laptop you need to use Thonny or Spyder (or whatever other IDE you downloaded). If you want to run Python code on your CPX you must use Mu. Once you've downloaded and installed Mu and open it up it will look like this (Note it's possible the software gets updated from the time this book is published. As such be sure to select the Circuitpython Mode for board development).



Make sure to select the Adafruit CircuitPython option. If the software has been updated and that option no longer exists, be sure to select the option that says Circuitpython for board development.

Alright, lets start writing code! If you have a file called *main.py* on your CPX click the Load button and load *main.py* (make sure to load the *main.py* that is stored on your CPX and not somewhere else on your computer.) If a file *main.py* does not exist on your CPX simply click the New button and then Save the file as *main.py* (again make sure you save it to the CPX and not to your computer)



You'll see that I am accessing the CIRCUITPY drive and saving the file as main.py. The file itself is empty and just has a comment using the # symbol. At this point since the file is blank the CPX won't do anything.

Now this is really important. Your CPX is a USB stick that can hold as many Python files as 2MB will allow but it can only run or execute one python script at a time. Furthermore it will only run two types of files. It will run code.py if it exists and if it can't find code.py it will run main.py. If the CPX can't find main.py or code.py it will just not do anything. If you have two versions of main.py or a combination of main.py or code.py it will run one of them and not the other. Make sure you only have one version of main.py or code.py but not both! Some common things to check if your CPX isn't working.

1. Make sure you're using Mu in the right Mode
2. Make sure main.py or code.py is on the CIRCUITPY drive and not somewhere on your computer.
3. Make sure you are editing the right file in Mu. Do you have two versions of main.py?
4. Are you editing using Thonny or Spyder?
5. Are you editing a file on your computer? Make sure you are writing to the CIRCUITPY drive.
6. Unplug the CPX, close Mu and try again.

So let's get the CPX to do something simple like blink an LED. I have an entire Github devoted to Microelectronics. Specifically I have a folder with all of my Circuit Playground files. The easiest program to run is the blink.py script. I've attached a screenshot of the script below.

```
1 import board
2 import digitalio
3 import time
4
5 led = digitalio.DigitalInOut(board.D13)
6 led.direction = digitalio.Direction.OUTPUT
7
8 while True:
9     print(time.monotonic())
10    led.value = True
11    time.sleep(0.5)
12    led.value = False
13    time.sleep(0.5)
14    led.value = True
15    time.sleep(0.1)
16    led.value = False
17    time.sleep(0.1)
```

We will talk about what this code is doing later on. For now copy and paste these 17 or so lines of code and paste them into Mu specifically the *main.py* script. It will hopefully look like this.

```

1 #This is an example comment using the # symbol
2
3 import board
4 import digitalio
5 import time
6
7 led = digitalio.DigitalInOut(board.D13)
8 led.direction = digitalio.Direction.OUTPUT
9
10 while True:
11     print(time.monotonic())
12     led.value = True
13     time.sleep(0.5)
14     led.value = False
15     time.sleep(0.5)
16     led.value = True
17     time.sleep(0.1)
18     led.value = False
19     time.sleep(0.1)

```

Adafruit

Make sure to save. You can click *Zoom-In* and *Zoom-out* to zoom in and out to change the font size and see more of the output. If the blink code is working you will see a red led labeled D13 on the CPX blink back and forth. D13 stands for digital pin 13. Youll notice there are analog pins labeled A5 and A6 among other numbers. Lets talk about the code a bit more and explain why its doing what its doing. The three lines at the top of the code are *import* commands to import different modules just like we did for *numpy* and *matplotlib*. In this case the modules being imported are *board*, *digitalio* and *time*. The module *board* is used to import the layout of the CPX so we can access different pins on the CPX. The module *digitalio* stands for digital input output which means we are inputting and outputting digital signals. Since we can combine this module with the *board* module we will be able to output digital signals to different pins on the CPX board. Remember that PCB stands for printed circuit board so *board* implies we are accessing pins on the CPX PCB. Hopefully that makes sense. The final module we are importing is *time* which acts just like the *time* module on your desktop computer. It will let us access the CPXs internal clock.

Moving along, lines 7 and 8 create an LED object using the *board* module and *digitalio*. Its a long line of code that basically says, create a variable called *led* that lets us output a digital signal to pin D13. We also set the direction of the LED to output since we only want to write to the LED.

Lines 10 through 19 kick off an infinite loop that never ends. The line that says *while True:* means loop while *True*. Well *True* is always true which means it will loop forever. The colon at the end of the line tells Python that the loop condition statement ends and to begin looping from 11 through 19.

Line 11 specifically says *print(time.monotonic())*. First the *print()* function is used to print things so that you and I can see it. Rather than just seeing a blinking LED we want to see the time printed. The *time.monotonic()* is using the module *time* which we imported and using a function from that module called *monotonic()* which calls the internal clock of the CPX. So how do you see the output from the print statement? Hit the *Serial* button on Mu and you will hopefully see some output. Heres what it looks like on my machine.

The screenshot shows the Mu 1.0.2 IDE interface. The top bar displays "Mu 1.0.2 - main.py". Below the bar is a toolbar with icons for Mode, New, Load, Save, Serial, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The main code editor window contains the following Python code:

```

1 import board
2 import digitalio
3 import time
4
5 led = digitalio.DigitalInOut(board.D13)
6 led.direction = digitalio.Direction.OUTPUT
7
8 while True:
9     print(time.monotonic())
10    led.value = True
11    time.sleep(0.5)
12    led.value = False
13    time.sleep(0.5)
14    led.value = True
15    time.sleep(0.1)
16    led.value = False
17    time.sleep(0.1)
18
19

```

The output window below the code editor shows three lines of text: "4742.87", "4744.06", and "4745.27". The bottom right corner of the interface has the Adafruit logo.

In this case you can see the time printed every time it goes through the loop. You may even see an error. This *Serial* button is great for debugging because it will tell you the error in your code. For example, in the picture below I have an error in my code and the *Serial* output is letting me know.

The screenshot shows the Mu 1.0.2 IDE interface. The top bar displays "Mu 1.0.2 - main.py". Below the bar is a toolbar with icons for Mode, New, Load, Save, Serial, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The main code editor window contains the following Python code:

```

1 #This is an example comment using the # symbol
2
3 import board
4 import digitalio
5 import timed
6
7 led = digitalio.DigitalInOut(board.D13)
8 led.direction = digitalio.Direction.OUTPUT
9
10 while True:
11     print(time.monotonic())
12     led.value = True
13     time.sleep(0.5)
14     led.value = False
15     time.sleep(0.5)
16     led.value = True
17     time.sleep(0.1)

```

The output window below the code editor shows the following text:

```

main.py output:
Traceback (most recent call last):
  File "main.py", line 5, in <module>
ImportError: no module named 'timed'

Press any key to enter the REPL. Use CTRL-D to reload.

```

The bottom right corner of the interface has the Adafruit logo.

In this case I have an error on line 5. Its saying there is no module named *timed*. The reason that module doesn't exist is because the module is actually *time* not *timed*. You can use the *Serial* monitor to check on your

program and see what errors you may have. Ok so there are two more lines of code to discuss. They are `led.value = True` and `time.sleep(0.5)`.

These lines of code are repeated throughout the while loop and do two things. First, the `led.value` either sets the value of the LED to True which turns the light on or False which turns the light off. The LED is digital which means the signal can either be on or off. Theres no in between for digital signals. The `time.sleep()` function tells the CPX to pause for half a second. You can change the number in the parentheses if you want to change length of time the code pauses. Note that the CPX completely pauses. That is no code runs during a sleep.

If youve gotten the LED to blink youre all set for this lab. However, Id like to you learn a few more things about documentation. Just like Python on your desktop you can lookup the documentation on the CPX itself. For example, Ive added a print statement to print the directory of time.

```
Mu 1.0.2 - main.py
Mode New Load Save Serial Plotter Zoom-in Zoom-out Theme Check Help Quit
main.py X
1 #This is an example comment using the # symbol
2
3 import board
4 import digitalio
5 import time
6
7 print(dir(time))
8
9 led = digitalio.DigitalInOut(board.D13)
10 led.direction = digitalio.Direction.OUTPUT
11
12 while True:
13     #print(time.monotonic())
14     led.value = True
15     time.sleep(0.5)
16     led.value = False
17     time.sleep(0.5)
18     led.value = True
Adafruit CircuitPython REPL
02:55:01
6295.07
soft reboot

Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
main.py output:
['__class__', '__name__', 'localtime', 'mktime', 'monotonic', 'monotonic_ns', 'sleep', 'struct_time', 'time']

Adafruit
```

In this case Ive added `print(dir(time))` to line 7. The output shows that the `time` module has 9 functions including `monotonic()`. Unfortunately CircuitPython does not have `__doc__` functions built in which means if you want to learn about a specific function, you need to visit the documentation for CircuitPython. Heres the specific documentation for the time module. A lot of example code from the Adafruit Learning System is also on Github.

Finally, in order to keep practicing using Python on your desktop I want you to modify the code above to run on your desktop computer. Youre going to have to modify a few things. First, make sure to open Thonny or Spyder depending on which version of Python IDE you downloaded. Then, only import the `time` module. All the other modules dont exist on your desktop. Also, get rid of all the lines of code that blink the LED. We just want to print time in a while loop. Finally, the time module on your desktop uses a function called `time.time()` (unless you have Python3 installed) so when you print time make sure to use that module instead. Again visit the documentation for time for Python if you want to learn more. Thonny by default will load Python version 3 but its possible you may have Python version 2 so make sure you look up the documentation for the appropriate version of Python. After searching through the documentation you can use a function called `asctime()` on your desktop. This is the output I get in Thonny when I add a sleep of 1 second. The exercise for you is to do something similar.

>>> %Run myprint.py

```
Thu Jul  2 10:42:37 2020
Thu Jul  2 10:42:38 2020
Thu Jul  2 10:42:39 2020
Thu Jul  2 10:42:40 2020
Thu Jul  2 10:42:41 2020
Thu Jul  2 10:42:42 2020
Thu Jul  2 10:42:43 2020
Thu Jul  2 10:42:44 2020
```

3.3 TL;DR

1. Plug in your CPX and double tap it to go into reset mode. CPLAYBOOT will mount to your computer.
2. Download the UF2 file.
3. Drag the UF2 file to your CPLAYBOOT drive. After a few seconds CIRCUITPY will mount.
4. You need to then download Mu
5. Open Mu and make sure to select the mode Adafruit CircuitPython
6. Open main.py in Mu from the CIRCUITPY drive in Mu.
7. If code.py is on your CPX delete it
8. Copy the blink.py script into main.py
9. Once you have the script running, modify the script to run on your Desktop using Spyder or Thonny

There is an accompanying youtube video to help you see me perform the 8 steps above.

3.4 Assignment

Upload a PDF with all of the photos and text below included. My recommendation is for you to create a Word document and insert all the photos and text into the document. Then export the Word document to a PDF. For videos I suggest uploading the videos to Google Drive, turn on link sharing and include a link in your PDF.

1. Include a screenshot of your computer showing the CIRCUITPY drive on your computer - 25%
2. Copy and paste your blink.py code and write a paragraph explaining any changes you made to get it to work - 25%
3. Take a video of you and your CPX blinking the LED. State your name, wave at the camera and show the CPX blinking (You must use software like OBS studio which records your screen and yourself. No cell phone videos allowed). - 25%

4. Modify the blink.py code to run on your desktop computer in Spyder or Thonny. Copy and paste your desktop version of the code and screenshot the output. Include the code and the output in your submission - 25%

4 Troubleshooting Guide

Is your CPX/CPB broken or not running code? Read below.

Now this is really important. Your CPX is a USB stick that can hold as many Python files as 2MB will allow but it can only run or execute one python script at a time. Furthermore it will only run two types of files. It will run code.py if it exists and if it can't find code.py it will run main.py If the CPX can't find main.py or code.py it will just not do anything. If you have two versions of main.py or a combination of main.py or code.py it will run one of them and not the other. Make sure you only have one version of main.py or code.py but not both! Some common things to check if your CPX isn't working.

1. Make sure you're using Mu in the right Mode
2. Make sure main.py or code.py is on the CIRCUITPY drive and not somewhere on your computer.
3. Make sure you are editing the right file in Mu. Do you have two versions of main.py?
4. Are you editing using Thonny or Spyder?
5. Are you editing a file on your computer? Make sure you are writing to the CIRCUITPY drive.
6. Unplug the CPX, close Mu and try again.
7. What if my CIRCUITPY drive doesn't work anymore
8. First, try and reset the CPX to CPLAYBOOT and reflash the UF2 to see if that fixes it.
9. If that doesn't work sometimes you just need to completely erase the CIRCUITPY drive so head over to this troubleshooting guide <https://learn.adafruit.com/adafruit-circuit-playground-express/troubleshooting> and follow some of the steps they tell you.
10. As a last resort you can try to download these UF2s and hopefully it will fix all errors and mistakes - CPX, CPB

5 External LEDs and Push Buttons

5.1 Parts List

1. Laptop
2. CPX + USB Cable
3. 2 Alligator clips
4. Push Button
5. Breadboard
6. LED (Light Emitting Diode) (x3 in case you fry some)
7. Resistor (300 to 1000 Ohms)

5.2 Learning Objectives

1. The VOUT and 3.3V pin are always "ON" even when code is not running on the CPX. So long as your CPX is plugged in via USB or a Lipo Battery
2. LED are Light Emitting **Diodes** which means current only flows in one direction
3. LEDs need resistors in series otherwise they will get too hot and burn up
4. Breadboard pinout diagrams
5. Analog pins can be controlled by simply using the digitalio module
6. LEDs can be hooked up to analog pins and set to blink by changing the board pin

In this project we're going to use the same blink code as before but modify it to blink an external LED. The purpose of this lab is to familiarize yourself with the pins on the CPX and create a simple circuit using the 5V pin on the CPX and one of the Analog pins. Your laptop has a battery with something between 10 to 20V. There are DC to DC converters in your laptop that provide 5V to your USB ports. These USB ports can be used to power your CPX as you have done in the past few labs.

If you purchased the optional battery pack you can also power the CPX using 3 AA batteries. These batteries nominally have 1.5 V but fully charged it's actually something like 1.8 V. So 1.8 times 3 is 5.4V which is enough to power the CPX. If you have the battery pack and some AA batteries, give it a try. If you still have the blink code from the last project on board you'll see the D13 LED blink as before. You won't be able to see the serial print()

output as before but that code will be running which is why D13 is blinking. **I have noticed that some of the battery packs have power and ground wires swapped. If the battery pack doesn't work it may be because those two wires are backwards.**

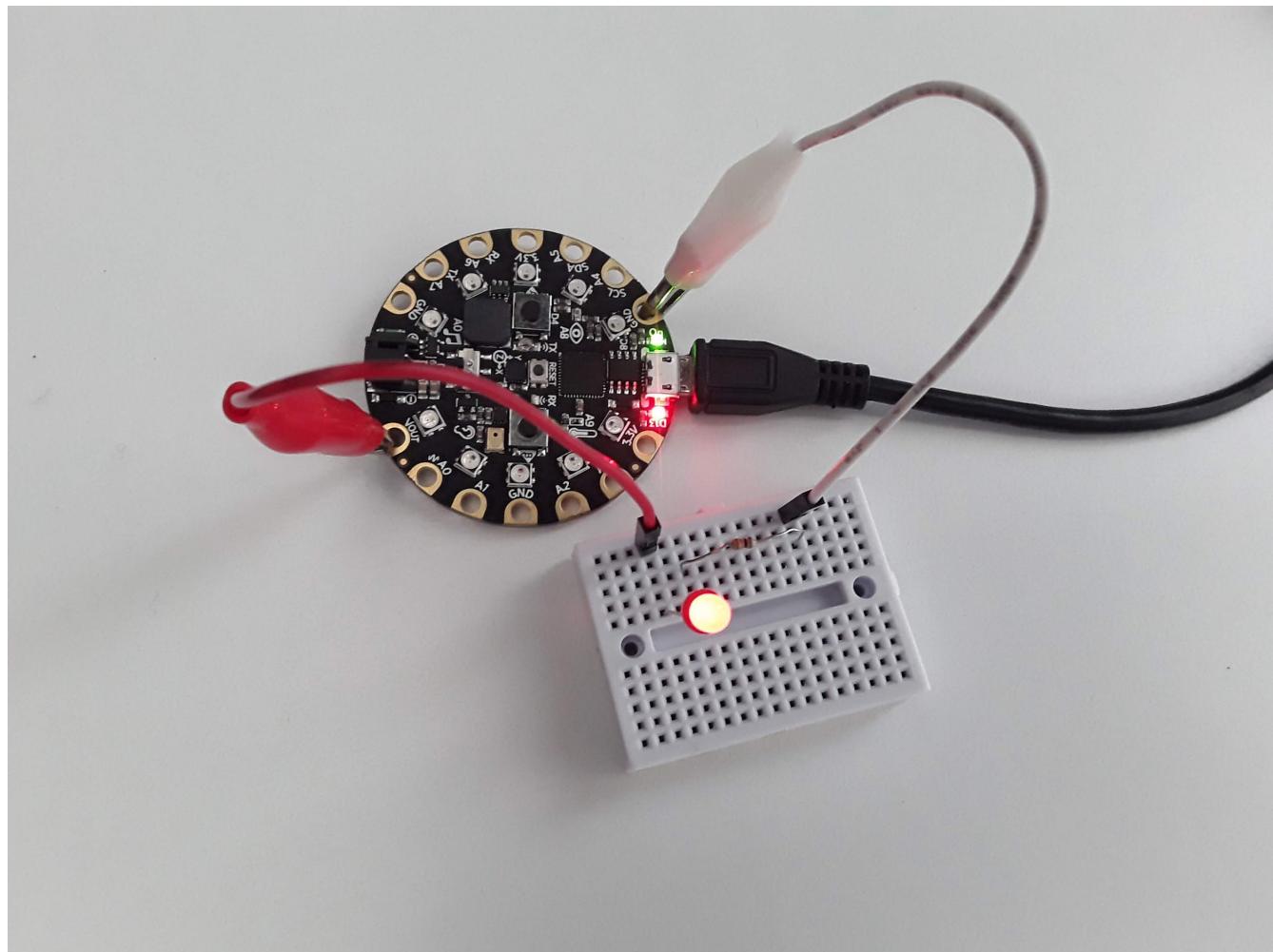
The CPX itself uses 3.3V logic which means when it converts numbers to binary a 0 (False) represents 0 volts and a 1 (True) represents 3.3V. The CPX has ports that are labeled various things. GND stands for ground and you need to hook the negative end of your circuit to this and it also has VOUT which supplies 5V to any circuit you build. Hook the positive end of your circuit to the VOUT pin. There is also a port labeled 3.3V and obviously that outputs 3.3V

You're going to need to use a breadboard so if you're not familiar with how breadboards work I would recommend watching this video on how breadboards work. Your lab today specifically involves an external LED. You can read about LEDs more online if you wish. **Remember that the long leg of the LED is the positive end and the short leg is the negative end.** The task today is to wire an LED up to the CPX in the following ways

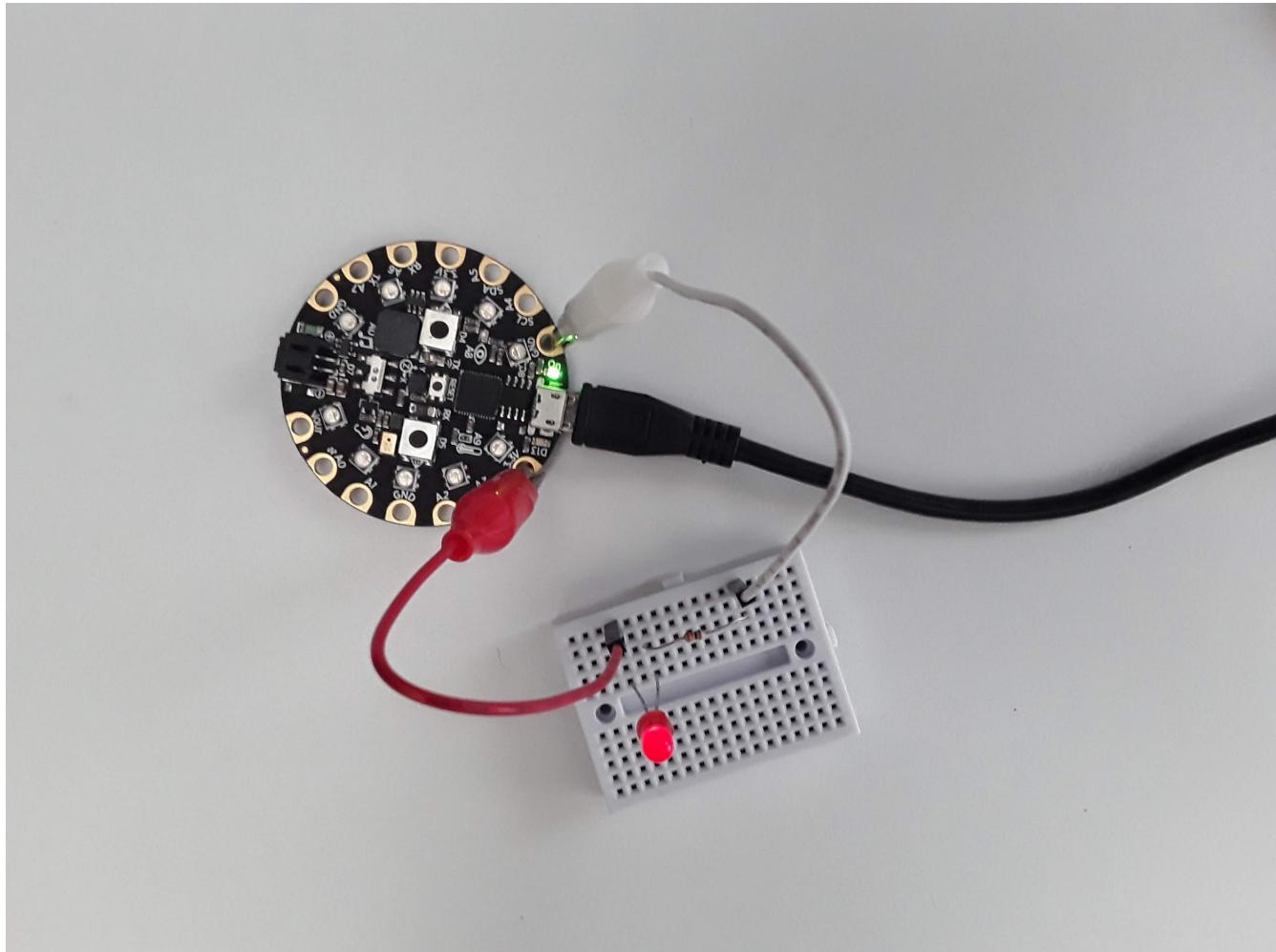
Whenever you modify a circuit on the breadboard, always be sure to remove power from the CPX. You can damage multiple components if you're not careful.

5.3 LED with no Code

For this part we are going to light up the LED without the use of any code on the CPX. First, wire up the circuit with the positive end connected to 5V. This is how my circuit looks. Make sure to use a resistor between 300 and 1000 Ohms. An LED does not have that much internal resistance so you need a resistor in series with an LED to reduce the amount of current flowing through the LED or the entire LED will fry. If you use a resistor that has too much impedance the LED just won't turn on because the voltage/current through the LED will be below the activation voltage of the circuit.

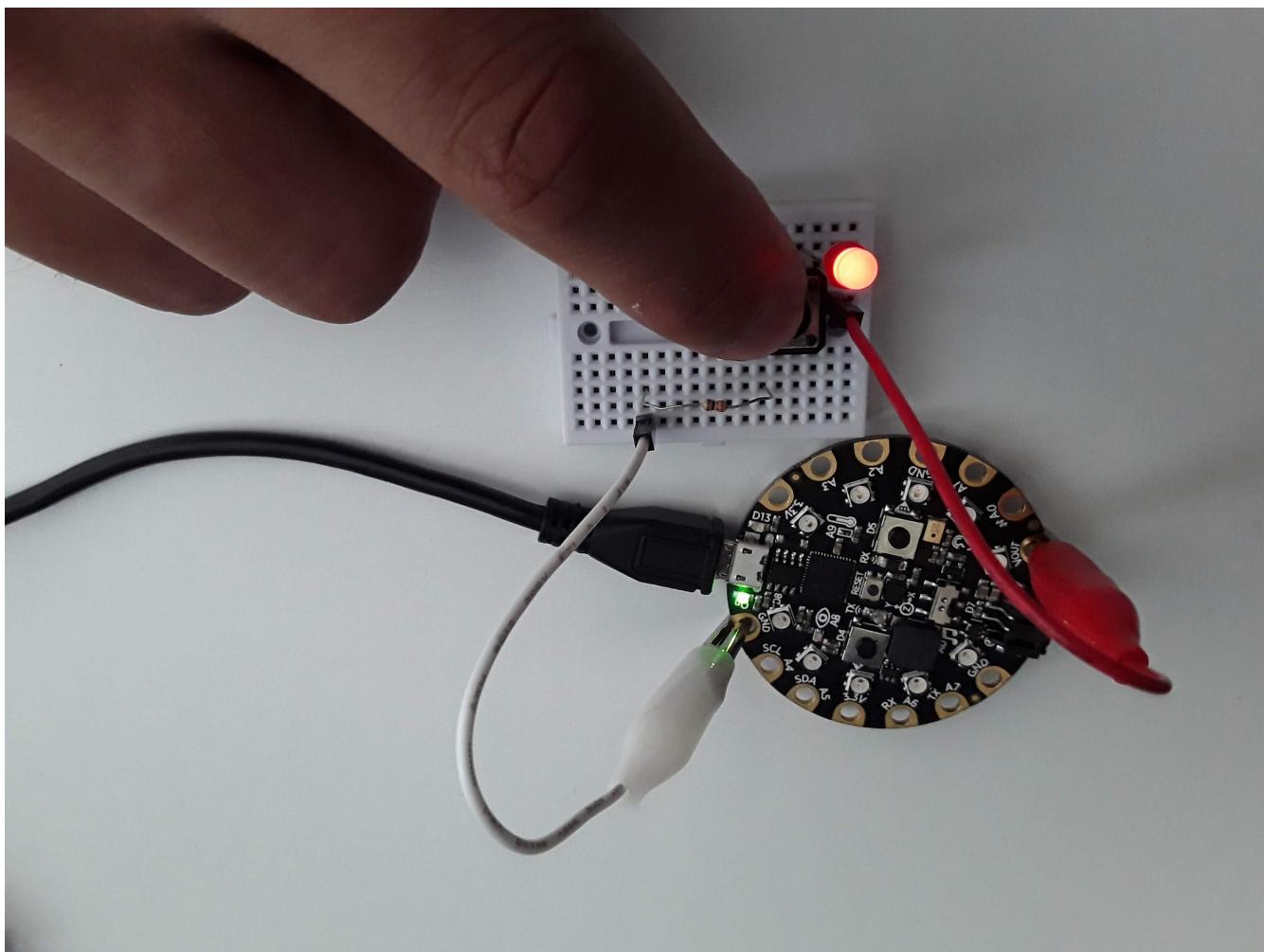


Once you have that circuit working, wire up the circuit again with the 3.3V output. Do you notice anything different when you hook up the circuit with different pins? Heres my circuit. Do you notice something different about the intensity of the LED? Why is it different?



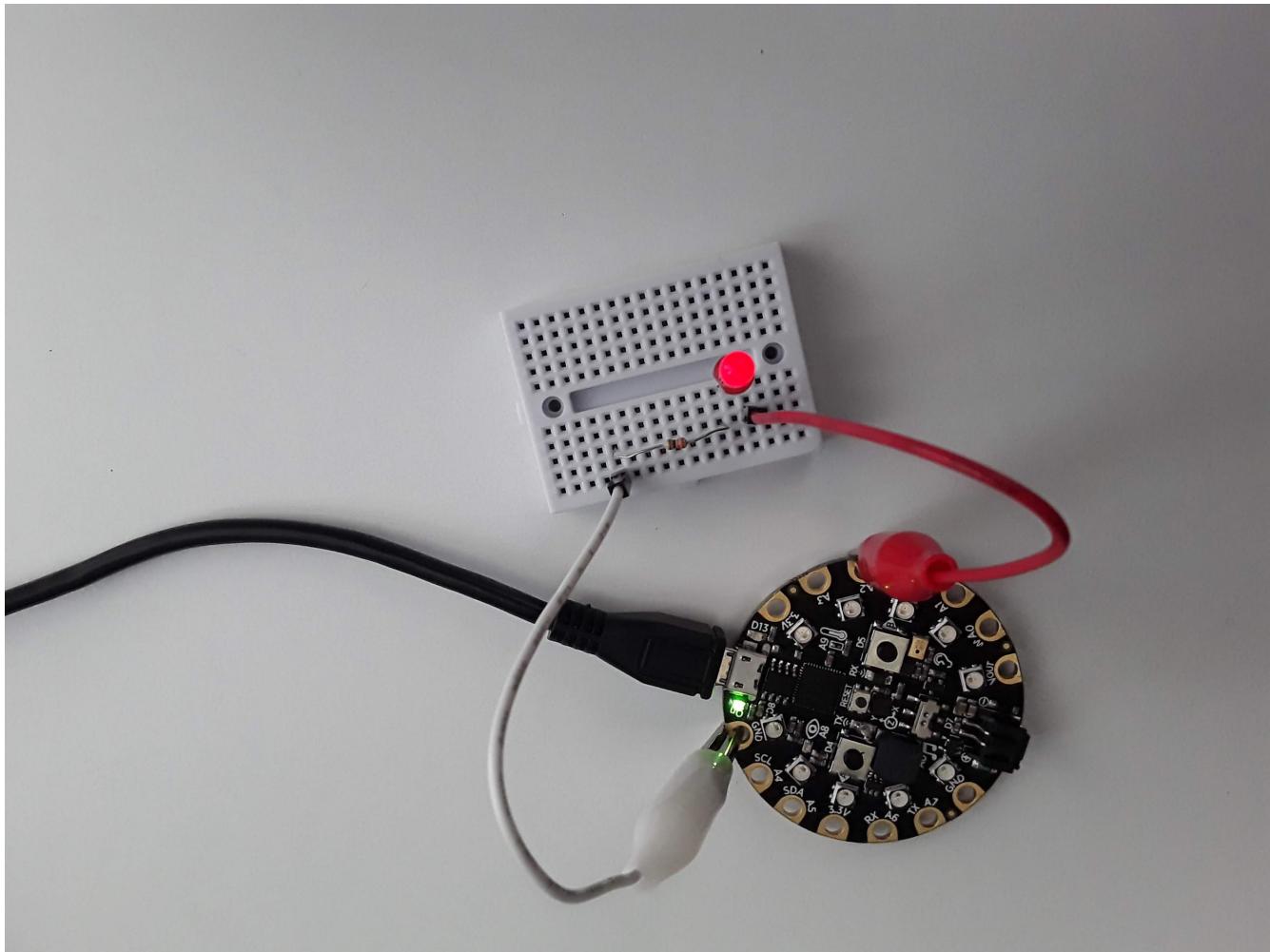
5.4 LED with a push button

Now that we understand breadboards a bit, were now going to manually blink the LED using a push button placed onto the breadboard and have it act like a switch. Therefore, when the button is pressed, the LED will turn on and when the button is released the LED will turn off. The button just acts like a wire so you can plug in the button anywhere in the circuit.



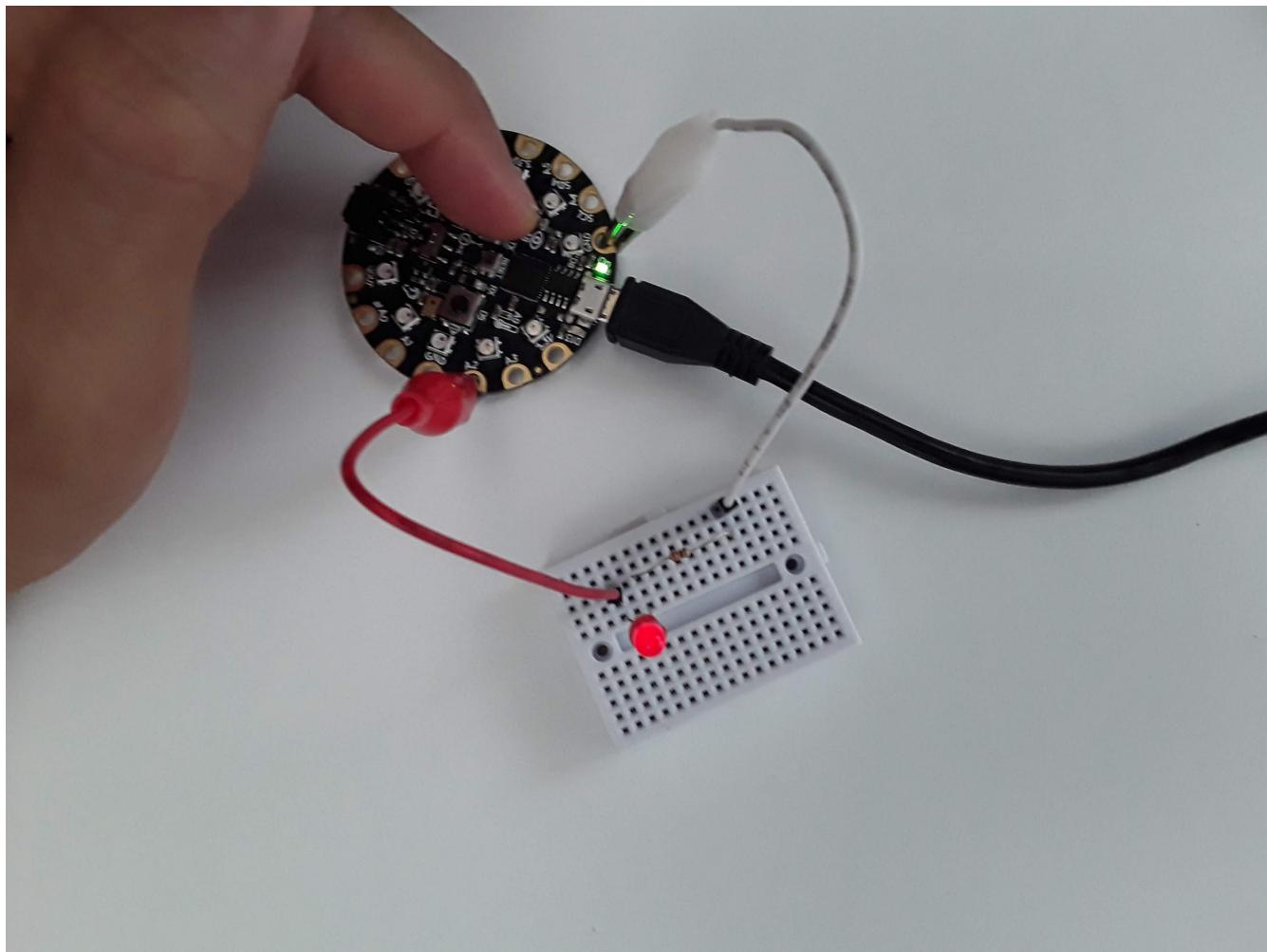
5.5 LED with code

Next I want you to remove the button from the circuit and wire up the LED like you had it when the positive end was connected to VOUT or 3.3V. Except this time I want you to hook the positive end of the circuit to pin A2. Then edit your blink code to blink pin A2. Take a look at the blink code. Right now the code is blinking pin D13. How do you think you need to change the code to blink pin A2? Heres what my circuit looks like for this one. I wont include code for this one since you just need to change one line of code.



5.6 LED with CPX button

Finally, I want to use one of the buttons on the CPX to blink the LED hooked up to pin A2. For this code to work you first need to detect a button press and then tell the program to change the light from True to False depending on what its current status is. This one is a bit more difficult so Ill include the code here and discuss the code itself. Here is my circuit (identical) to the previous one with Button A on the CPX pressed down.



Alright so how do we detect a button press? Well the documentation on this is not so straight forward. What we want to do is detect the INPUT of a digital signal and then do something if we detect that signal. Heres the code I created to get it to work.

```
1 import board
2 import digitalio
3 import time
4
5 buttonA = digitalio.DigitalInOut(board.BUTTON_A)
6 buttonA.direction = digitalio.Direction.INPUT
7 buttonA.pull = digitalio.Pull.DOWN
8
9 led = digitalio.DigitalInOut(board.A2)
10 led.direction = digitalio.Direction.OUTPUT
11 led.value = True
12
13 while True:
14     print('Button value is ',buttonA.value)
15     led.value = False
16     if buttonA.value == True:
17         print('Button Value is ',buttonA.value)
18         led.value = True
19         while buttonA.value == True:
20             print('Waiting for you to let go....')
21             # Wait for all buttons to be released.
22             time.sleep(0.1)
23     time.sleep(0.1)
```

The code above is an image. You could type exactly what you see and hope you don't type any errors (which is highly unlikely) or you could look for my code on my Github. Have you bookmarked this link yet? I recommend you do so!!! So you're making a code about Buttons....hmmm. Click that Github link. Where do you think the code about Buttons is located? Anywho, let's talk about the code.

The first 3 lines are exactly the same as before. Line 5 through 7 are similar to creating the led variable except were using *BUTTON_A* as the board value and setting the direction to *INPUT*. Finally were setting the pull direction to *DOWN*. This means the button acts like a pull down resistor and when its pressed the value of the button goes *HIGH*.

Lines 9-11 are the same as before. We create an led variable and tell the CPX that the led is hooked up to pin A2. We then start the while loop on line 13. First if you click the Serial button youll see the text Button value is False. Its False because *buttonA.value* is not being pressed. On line 15 the led is set to False (turned off). On line 16 the button value is checked using an if statement as to whether or not the button is pressed (True). If the button is pressed, the user will be notified that the button is pressed on line 17 and the led will be set to True (on)

in line 18. Lines 19-22 are while loop that will notify the Serial monitor that you must let go of the button before the code can continue to the main while loop. The `time.sleep` functions are there to make sure a human can operate the button without code running faster than a human can press a button. When I press the button down here is the output I get from the *Serial* monitor.

The screenshot shows the Mu 1.0.2 IDE interface. The top bar displays system information: Thu 11:36, CPU 15% Mem 6.3 GB Swap 1.1 GB, en: en, and various icons for battery, signal, and network. The title bar says "Mu 1.0.2 - main.py". The menu bar includes Activities, File (with a dropdown for mu), Edit, View, Insert, Run, Tools, Help, and a gear icon for settings. Below the menu is a toolbar with icons for Mode (button), New, Load, Save, Serial, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The main workspace shows two tabs: "main.py" and "push_button_external_LED.py". The code in "push_button_external_LED.py" is:

```

7 buttonA.pull = digitalio.Pull.DOWN
8
9 led = digitalio.DigitalInOut(board.A2)
10 led.direction = digitalio.Direction.OUTPUT
11 led.value = True
12
13 while True:
14     print('Button value is ',buttonA.value)
15     led.value = False
16     if buttonA.value == True:
17         print('Button Value is ',buttonA.value)
18         led.value = True
19         while buttonA.value == True:
20             print('Waiting for you to let go....')
21             # Wait for all buttons to be released.
22             time.sleep(0.1)
23     time.sleep(0.1)

```

The Adafruit CircuitPython REPL window below the code editor shows the output of the script:

```

Button value is False
Button value is False
Button value is False
Button value is False
Button value is True
Button Value is True
Waiting for you to let go....

```

Here youll see 4 lines that say “Button value is False” and then two lines that say “Button value is True” followed by 5 lines that say “Waiting for you to let go...”. See if you can get this code to work and play with it and modify it as you see fit. By the way, the LED connected to pin D13 has this exact same circuitry, an LED a resistor, its all just soldered to the PCB so you dont have to build it using a breadboard. Hopefully now you have some appreciation for buttons and LEDs!!

5.7 Assignment

Once you've done the exercise above, upload a PDF with all of the photos and text below included. My recommendation is for you to create a Word document and insert all the photos and text into the document. Then export the Word document to a PDF. For videos I suggest uploading the videos to Google Drive, turn on link sharing and include a link in your PDF.

1. Include a photo of your circuit with your LED turned on using VOUT (make sure your face is in the photo) - 20%
2. Include a photo of your circuit with your LED turned on using 3.3V (make sure your face is in the photo) - 20%
3. Include a video of you turning your LED on and off with the push button (again make sure your face is in the video for enough time to say who you are and say hello) - 20%
4. Include a video of your LED blinking on and off automatically by modifying the blink code (make sure your face is in the video for enough time to say who you are and say hello) - 20%
5. Include a video of your LED blinking by pressing a button on the CPX (make sure your face is in the video for enough time to say who you are and say hello) - 20%

6 Using the CPX/CPB as a Data Acquisition System (DAQ)

6.1 Parts List

1. Laptop
2. CPX/CPB
3. USB Cable

6.2 Learning Objectives

1. Record real time measurements from the CircuitPlayground using the Serial monitor
2. Learn how to use the typing module to type data directly into a spreadsheet or text file
3. Learn how to log data on the on board memory of the CircuitPlayground

6.3 Extra Help

This is a pretty hard project to do with multiple different methods. After youve read through this document I suggest you watch a youtube I created on Logging Data with the Circuit Playground Express. I have also made video where I log temperature and accelerometer data using on board memory with the CircuitPlayground Bluefruit.

6.4 Getting Started

Taking data is the core of any instrumentation project. Data Acquisition Systems or DAQ for short come in all shapes and sizes. Believe it or not the CPX can be used as a crude and cheap DAQ. The CPX can easily take temperature data and monitor the temperature in a greenhouse or take humidity readings of a plant to monitor soil content. Before we learn about the different sensors on board the CPX, we want to make sure we can store that data later rather than just having it spit data out via the serial monitor. For starters though lets get the CPX to print out something simple like button presses since weve touched on that already. The code Im using is shown below and can also be found on Github.

```
1 import board
2 import digitalio
3 import time
4
5 buttonA = digitalio.DigitalInOut(board.D4)
6 buttonA.direction = digitalio.Direction.INPUT
7 buttonA.pull = digitalio.Pull.DOWN
8
9 while True:
10     print(int(buttonA.value))
11     time.sleep(0.1)
```

The code is pretty similar to what I had in the past. I import board, digitalio, and time. I create a buttonA object using the digitalio library to record button presses. I then enter into a while loop print the buttonA.value.

The difference here is that I use the int() function to convert the buttonA.value to an integer. The reason why I do this is because buttonA.value is a boolean. It is either True or False. An integer though is a number and thus a value of False is 0 and True is 1. If you open the serial monitor and push the A button down a few times youll see some zeros and 1s.

The screenshot shows the Mu 1.0.2 IDE interface. At the top is a toolbar with various icons: Mode, New, Load, Save, Serial, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. Below the toolbar, there are three tabs: main.py, push_button_external_LED.py, and button_presses.py. The button_presses.py tab is active, displaying the following Python code:

```
1 import board
2 import digitalio
3 import time
4
5 buttonA = digitalio.DigitalInOut(board.D4)
6 buttonA.direction = digitalio.Direction.INPUT
7 buttonA.pull = digitalio.Pull.DOWN
8
9 while True:
10     print(int(buttonA.value))
11     time.sleep(0.1)
```

Below the code editor is the Adafruit CircuitPython REPL window, which shows the following output:

```
1
0
0
1
0
1
1
```

In the bottom right corner of the REPL window, there is an Adafruit logo with a gear icon.

Mu also has a really neat builtin plotter. Youll see next to the Serial button there is a button called Plotter. If you click that button now nothing will pop up on the screen. Unfortunately in order to plot using the Plotter you need to modify the print() statement to this:

```
print((int(buttonA.value),))
```

Notice the extra parentheses and the comma. Now if you click Plotter youll see something like this. Youll notice that the print statement now has commas in it and the Plotter is recording button presses.

Mu 1.0.2 - main.py

Mode New Load Save Serial Plotter Zoom-in Zoom-out Theme Check Help Quit

main.py push_button_external_LED.py button_presses.py

```
1 import board
2 import digitalio
3 import time
```

Adafruit CircuitPython REPL

```
(0,)
(0,)
(0,)
(0,)
(0,)
(0,)
(0,)
(0,)
(0,)
(0,)
(0,)
(0,)
(0,)
(0,)
(0,)
(0,)
(0,)
(0,)
```

Adafruit CircuitPython Plotter

The problem with this is we still can't save the recorded data anywhere. Before we get into saving data let's first edit the print statement again to get rid of the Plotter by removing the extra parentheses and add `time.monotonic()` that way we can keep track of when a button was pressed. My print statement looks like this now:

```
print(time.monotonic(), int(buttonA.value))
```

Looking at the serial monitor now you'll see that time is being printed alongside the button presses.

Mu 1.0.2 - main.py

Mode New Load Save Serial Plotter Zoom-in Zoom-out Theme Check Help Quit

main.py push_button_external_LED.py button_presses.py

```
1 import board
2 import digitalio
3 import time
4
5 buttonA = digitalio.DigitalInOut(board.D4)
6 buttonA.direction = digitalio.Direction.INPUT
7 buttonA.pull = digitalio.Pull.DOWN
8
9 while True:
10     print(time.monotonic(), int(buttonA.value))
11     time.sleep(0.1)
```

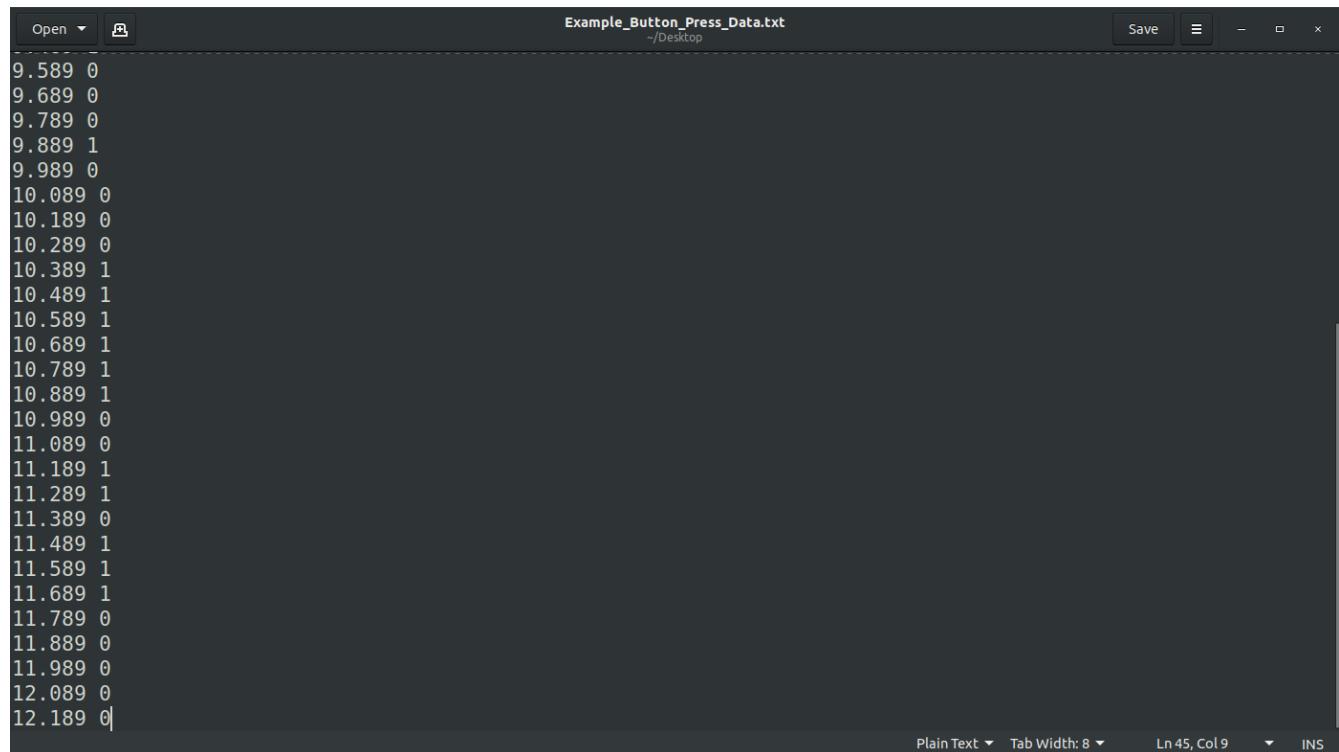
Adafruit CircuitPython REPL

```
777.944 0
778.044 1
778.144 1
778.244 0
778.344 0
778.444 1
778.544 0
778.644 0
778.744 1
```

Now we are in a position where we can record some data and save it to our computer. There are 3 ways to record data. I call the first, Method1 and you basically just copy and paste from the serial monitor, Method2 where you have the CPX/CPB type data into a spreadsheet and Method3 where you log data internally onto the CPX/CPB itself.

6.5 Method 1 - Copying Serial Monitor Data

If you open up the serial monitor you can see the data output. If you unplug the CPX while its taking data the code will stop. **Note: In newer versions, unplugging your CPX will result in a loss of data. If this happens try pressing CTRL+C after you click the REPL window.** With the code stopped you can select all the data in the Serial monitor and then copy and paste the data into a text file on your computer. You can actually copy this into a new file on Thonny and just save it as a *.txt file. Once you have the data in a text file you can proceed to plotting in Python on your desktop which I discuss in the last section. Heres some example data in Gedit which is a simple text editing program.



The screenshot shows a Gedit text editor window. The title bar reads "Example_Button_Press_Data.txt" and "Desktop". The main pane contains the following text:

```
9.589 0
9.689 0
9.789 0
9.889 1
9.989 0
10.089 0
10.189 0
10.289 0
10.389 1
10.489 1
10.589 1
10.689 1
10.789 1
10.889 1
10.989 0
11.089 0
11.189 1
11.289 1
11.389 0
11.489 1
11.589 1
11.689 1
11.789 0
11.889 0
11.989 0
12.089 0
12.189 0|
```

The status bar at the bottom shows "Plain Text" and "Tab Width: 8".

6.6 Method 2 - Automatically Populate a Spreadsheet

The downside with the above method of course is if you have a ton of data to record you could lose the data or run into a massive copy and paste issue. The second option is to use this module called keyboard which takes control of your keyboard on your desktop computer and actively types your data into a spreadsheet. The code is very extensive but Ill include the simple one here so we can discuss it. Below are the first 30 lines of code. The first 6 lines of code are just comments since I heavily adopted this code from the Adafruit Learn System. My version of this code can be found on my Github. Lines 8 - 14 are import commands as weve seen previously. The regular import modules board, time and digitalio are imported but we are also importing the Keyboard module so that the CPX can takeover our keyboard. Lines 16-22 create two buttons. First we create buttonA attached to pin D4 and then a switch attached to pin D7. If you look on the CPX there is a switch labeled D7. Before you copy this code onto the CPX make sure you move the switch towards the ear looking symbol. Lines 26-28 created the keyboard object. We are going to call it layout for this example code.

```

1 # Circuit Playground Express Data Logger
2 # Log data to a spreadsheet on-screen
3 # Open Spreadsheet beforehand and position to start (A,1)
4 # Use slide switch to start and stop sensor readings
5 # Time values are seconds since board powered on (relative time)
6 # Adapted from Adafruit Forums
7
8 import time
9 import digitalio
10 import board
11 import usb_hid
12 from adafruit_hid.keyboard import Keyboard
13 from adafruit_hid keycode import Keycode
14 from adafruit_hid.keyboard_layout_us import KeyboardLayoutUS
15
16 buttonA = digitalio.DigitalInOut(board.D4)
17 buttonA.direction = digitalio.Direction.INPUT
18 buttonA.pull = digitalio.Pull.DOWN
19
20 switch = digitalio.DigitalInOut(board.D7)
21 switch.direction = digitalio.Direction.INPUT
22 switch.pull = digitalio.Pull.UP
23
24 # Set the keyboard object!
25 # Sleep for a bit to avoid a race condition on some systems
26 time.sleep(1)
27 kbd = Keyboard(usb_hid.devices)
28 layout = KeyboardLayoutUS(kbd) # US is currently only option.
29
30 print("Time\tButton Value") # Print column headers

```

The next 30 lines are shown below. Lines 32-35 define a function. Functions in Python have a pretty standard structure. The keyword def is used to denote that the next line is a definition for a function. The name of the function is *slow_write()*. The input to the function is string which ironically enough is a string object. Line 33-35 define what the function does. Line 33 sets up a for loop where the code loops through each character in the string. Everytime it gets to a new character it will use your keyboard to type that character using the layout.write(c) command. The time.sleep(0.02) is just to slow down the keyboard so your computer can keep up. That function is defined above the standard while True: statement on line 37 but is called on line 42. Youll see there is a *slow_write(output)* on line 42. In this case output is a string and its sent to the function *slow_write()*. So in this case we have a function that can write a string so we just need to take data and then write it using our keyboard. Line 38 is an if statement that will only be true if the switch on pin D7 is pushed towards the music note on the CPX. If the switch is not thrown the code will move to the else statement on line 52 and tell the user that you need to flip the switch. If the switch is thrown line 40 will take data for us. First it will record the time.monotonic() and store it as a floating point number using the %0.1f designation which means that it will store 1 decimal as a %floating point number for f.

```

32  def slow_write(string):    # Typing should not be too fast for
33      for c in string:       # the computer to be able to accept
34          layout.write(c)
35          time.sleep(0.02)   # use 1/5 second pause between characters
36
37  while True:
38      if switch.value == True:
39          # Turn on the LED to show we're logging
40          output = "%0.1f\t%d" % (time.monotonic(), int(buttonA.value))
41          print(output)        # Print to serial monitor
42          slow_write(output)  # Print to spreadsheet
43
44          kbd.press(Keycode.DOWN_ARROW)  # Code to go to next row
45          time.sleep(0.01)
46          kbd.release_all()
47          for _ in range(2):
48              kbd.press(Keycode.LEFT_ARROW)
49              time.sleep(0.005)
50              kbd.release_all()
51          #time.sleep(0.025)  # Wait a bit more for Google Sheets
52      else:
53          print('Not logging. Flip Switch to start logging')
54      # Change 0.1 to whatever time you need between readings
55      time.sleep(1.0)

```

The second number in the string is an integer or a base 10 (decimal) integer designated by the %d part of the format. The integer is int(buttonA.value). Youll see a \t in between the formatted numbers which is a tab. The tab is there to tab between cells in a spreadsheet. Line 41 will print the output string to the Serial monitor and it will also type the contents of the string. Very important here. When you flip the switch on the CPX your keyboard will start typing in whatever active window is selected. If you dont have a spreadsheet opened and active (selected), the keyboard will just begin typing in whatever window is open. Make sure you have a spreadsheet program open and ready to go. Lines 44-51 tell the keyboard to hit the *DOWN_ARROW* on your keyboard to move to the next row and the *LEFT_ARROW* twice to move back to the first column. Line 55 is a sleep to only log data once a second. I ran this code for a bit and had it type into LibreOffice Calc which is a free spreadsheet program. Google Sheets or Microsoft Excel will also work just fine.

Example_Data.csv - LibreOffice Calc

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	573.2	0														
2	574.4	0														
3	575.6	0														
4	576.8	1														
5	578	1														
6	579.2	1														
7	580.5	0														
8	581.7	0														
9	582.9	0														
10	584.1	1														
11	585.3	1														
12	586.5	1														
13	587.8	0														
14	589	0														
15																
16																
17																
18																
19																
20																
21																
22																
23																
24																
25																
26																
27																
28																
29																
30																
31																
32																
33																
34																
35																
36																
37																
38																

Sheet 1 of 1 | Default | English (USA) | Average: ; Sum: 0 | 100% |

You'll see that the first column is time with 1 decimal point and the second column is the button press values. At this point you must click *Save As...* and save the document as a CSV which stands for Comma Separated Value. Once you have the file saved you can proceed to plotting in Python on your Desktop.

6.7 Method 3 - Logging Data Directly to on board memory

The problem with the above 2 methods is that you need a laptop to log data in the field. It would be nice if you could use the optional battery pack and just have the CPX log data on the CPX itself. This is the most complex way but in my opinion the best way. In order to get this to work you need to allow the drive on the CPX to have read/write permissions. This requires you to load a piece of software called boot.py and put it on the CPX. I have this software on my Github. The software is shown below. The first 10 lines are probably very familiar. Import some modules and then create a switch object. Line 13 is where all of the storage permissions are changed. If the flip is switched towards the A button, the storage module is used to allow you to write to the CPX. The problem here is that if you do this, you won't be able to edit code. I'll explain the procedure here in a minute. As always, the relevant Adafruit tutorial is on the Adafruit Learn System if you want to read more about it. Again make sure you store this file onto the CIRCUITPY drive and save it as boot.py

```

1 import board
2 import digitalio
3 import storage
4
5 # For Gemma M0, Trinket M0, Metro M0/M4 Express, ItsyBitsy M0/M4 Express
6 #switch = digitalio.DigitalInOut(board.D2)
7 # switch = digitalio.DigitalInOut(board.D5) # For Feather M0/M4 Express
8 switch = digitalio.DigitalInOut(board.D7) # For Circuit Playground Express
9 switch.direction = digitalio.Direction.INPUT
10 switch.pull = digitalio.Pull.UP
11
12 # If the switch pin is connected to ground CircuitPython can write to the drive
13 storage.remount("/", switch.value)
14 if (switch.value):
15     print('Storage changed')
16 else:
17     print('Change the switch')

```

In addition to storing the file boot.py youll need to edit your main.py script to only log data when the switch is moved towards the B button. The software to record button presses on disk is shown below and as always on my Github. In this software we again see the standard commands. Lines 1-3 import all the modules we need and then 5-15 create a switch, a button and an LED. In this case were using the LED soldered to the board. Line 17-20 check to see if the user has flipped the switch. If the switch is False the storage module on boot.py will allow the drive to act like a data logger and it will open a file called *Test_Data.txt* for writing (w). If the switch is True then the user will be notified that the file has not been opened for writing. Lines 22 through 33 include the infinite while loop. Line 23 turns the LED on and line 24 prints out the current time and the button value in integer form. If the switch value is False the program will create an output string by converting all numbers to strings using the str function. Notice that there is a *str("\n")* at the end of the output variable which tells the computer to write a new line of data to the file. Lines 28 and 29 write the output to the file from line 18 and then flush the output which means the CPX waits for the data to be fully written before moving on. It also turns the LED off so we know the CPX took data even when we arent looking at the Serial monitor. If the switch value is true it means that the we never opened the data file and thus we tell the user we arent logging data and its time to flip the switch and hit reset.

```

1 import time
2 import board
3 import digitalio
4
5 switch = digitalio.DigitalInOut(board.D7)
6 switch.direction = digitalio.Direction.INPUT
7 switch.pull = digitalio.Pull.UP
8
9 buttonA = digitalio.DigitalInOut(board.D4)
10 buttonA.direction = digitalio.Direction.INPUT
11 buttonA.pull = digitalio.Pull.DOWN
12
13 led = digitalio.DigitalInOut(board.D13)
14 led.direction = digitalio.Direction.OUTPUT
15 led.value = True
16
17 if switch.value == False:
18     file = open('Test_Data.txt', 'w')
19 else:
20     print('Not opening file for writing')
21
22 while True:
23     led.value = True
24     print(time.monotonic(), int(buttonA.value))
25     if switch.value == False:
26         print('Writing Data to Disk')
27         output = str(time.monotonic()) + " " + str(int(buttonA.value)) + str('\n')
28         file.write(output)
29         file.flush()
30         led.value = False
31     else:
32         print('Not logging data. Flip the switch and then hit reset')
33     time.sleep(1) #sleep for so many seconds between measurements

```

So here is the flow of what you want to do for method 3.

1. Unplug the CPX
2. Flip the switch towards the A button.
3. Plug in the CPX and save the boot.py and main.py files. Remember you can only save Python scripts when the switch is flipped towards the A button.
4. When you are ready to start recording data, flip the switch towards the B button. If you're looking at the Serial monitor, the software will throw an error. Just ignore it and hit the reset button. When your computer recognizes the CPX you can turn the Serial monitor on and off.
5. When you are done taking data simply slide the switch over towards the A button and hit reset again. This is what my Serial monitor looks like when I do this. You'll see that I was writing to disk for like 25 seconds and then I flipped the switch back towards the A button.

The screenshot shows the Mu 1.0.2 IDE interface. The top menu bar displays "Mu 1.0.2 - main.py". Below the menu is a toolbar with icons for Mode, New, Load, Save, Serial, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The main workspace contains five tabs: "main.py", "push_button_external_LED.py", "button_presses.py", "record_button_presses_typing.py", and "boot.py". The "main.py" tab is active, showing the following Python code:

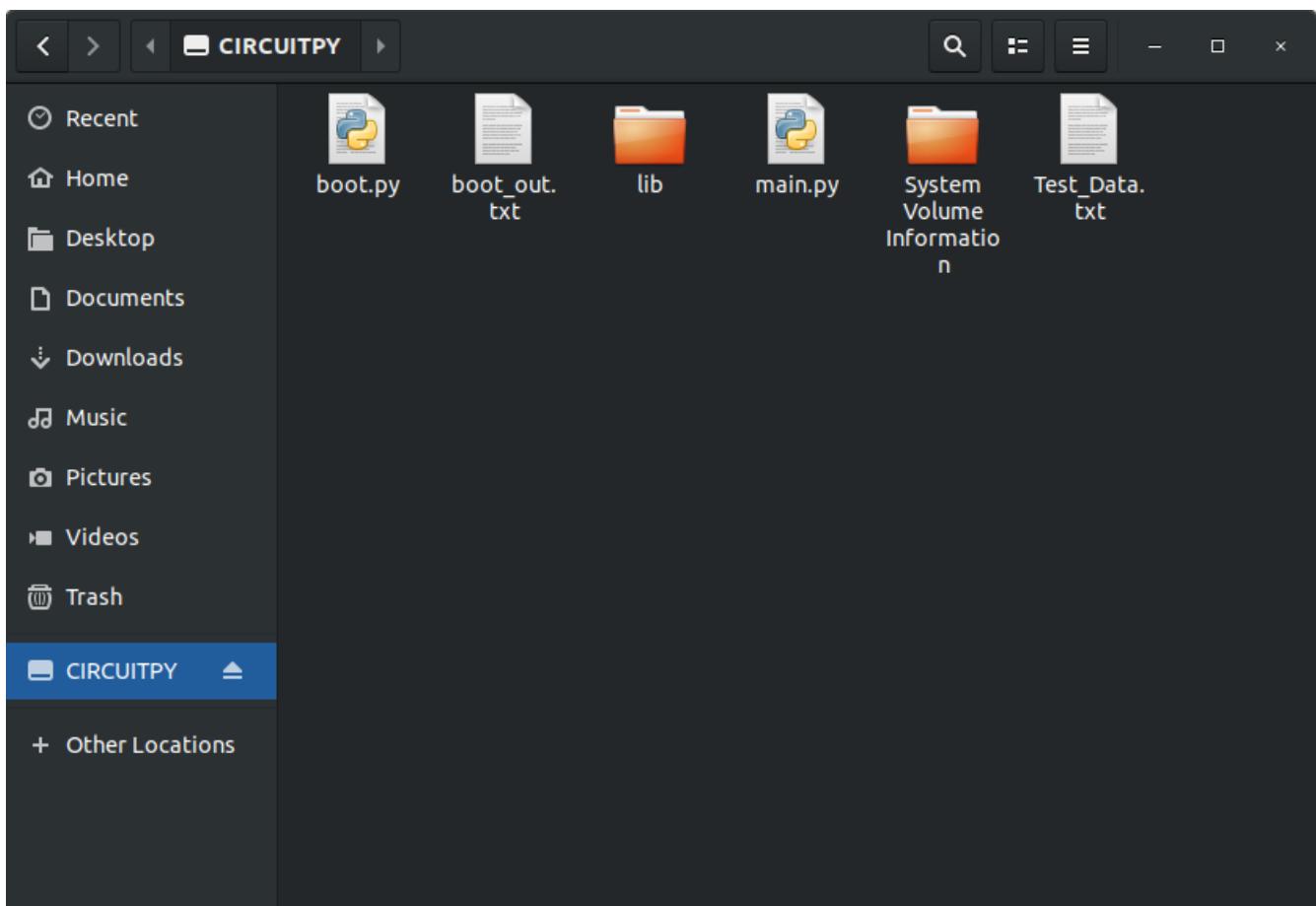
```
13 led = digitalio.DigitalInOut(board.D13)
14 led.direction = digitalio.Direction.OUTPUT
15 led.value = True
16
17 if switch.value == False:
18     file = open('Test_Data.txt', 'w')
19 else:
20     print('Not opening file for writing')
21
```

Below the code, the Adafruit CircuitPython REPL window shows the following output:

```
Writing Data to Disk
23.364 0
Writing Data to Disk
24.502 0
Writing Data to Disk
25.639 0
Not logging data. Flip the switch and then hit reset
26.64 0
Not logging data. Flip the switch and then hit reset
27.641 0
Not logging data. Flip the switch and then hit reset
28.642 0
Not logging data. Flip the switch and then hit reset
```

In the bottom right corner of the workspace, there is an "Adafruit" logo with a gear icon.

With the switch flipped and data taken, open your folder manager and take a look at the CIRCUITPY drive. This is what mine looks like. Youll see I have two Python files and a file *Test_Data.txt* with all my data in it.



If you open the *Test_Data.txt* file you will hopefully see data in it.

A screenshot of a text editor window titled "Example_Button_Press_Data.txt" located at "/Desktop". The window has a dark theme. The text area contains a series of binary-like data points separated by spaces, representing button press events. The data starts with "9.589 0" and continues through "12.189 0".

```
9.589 0
9.689 0
9.789 0
9.889 1
9.989 0
10.089 0
10.189 0
10.289 0
10.389 1
10.489 1
10.589 1
10.689 1
10.789 1
10.889 1
10.989 0
11.089 0
11.189 1
11.289 1
11.389 0
11.489 1
11.589 1
11.689 1
11.789 0
11.889 0
11.989 0
12.089 0
12.189 0
```

At this point you can copy this text file over to your desktop computer and proceed to the Python plotting portion. Ok so lets recap method 3 one more time.

1. Unplug CPX (or remove power)
2. Slide switch to A
3. Plug in CPX (or provide battery power)
4. Slide switch to B
5. Reset
6. Take data for however long you want
7. Slide switch to A
8. Remove power if you're on battery power
9. Plug CPX into computer if not already connected
10. Transfer data file to computer

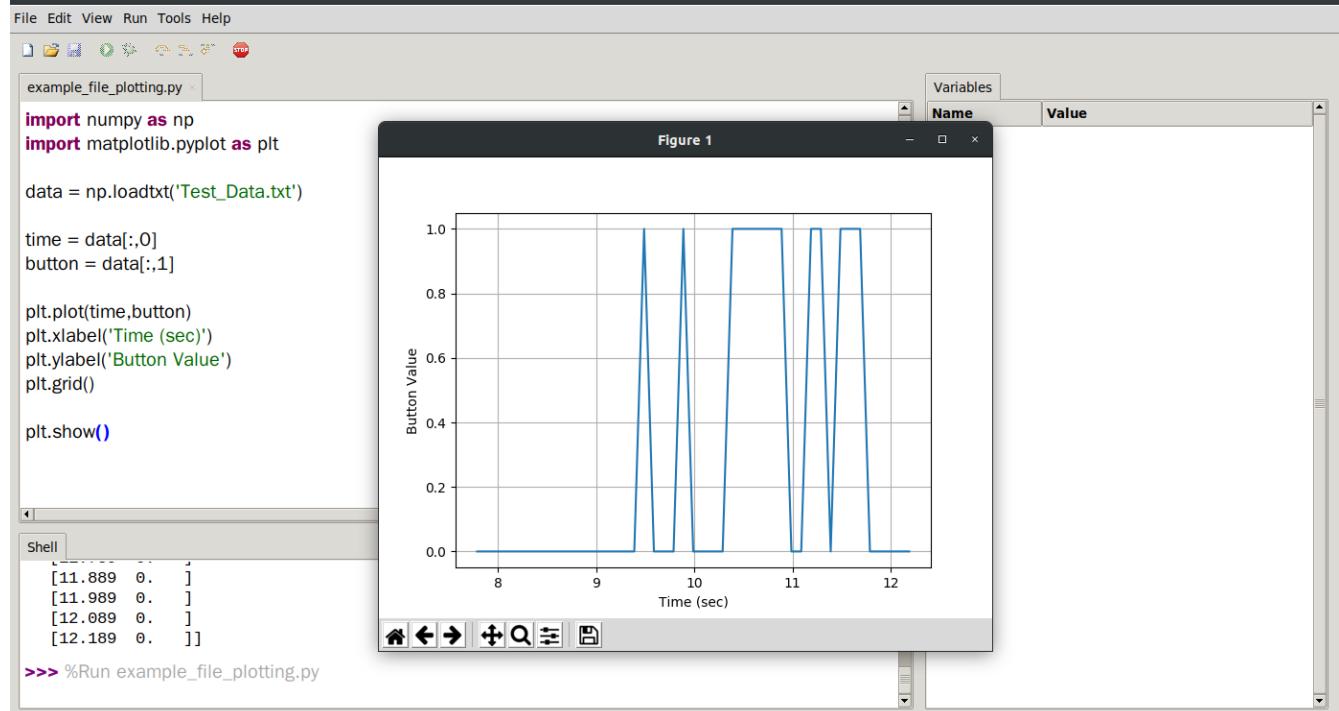
6.8 Plotting Logged Data

Alright so there you have it. I have explained 3 methods to datalogging. Here are the methods again in summary.

1. Print data to Serial and copy and paste
2. Use the Keyboard module to save data to a spreadsheet
3. Access the storage of your CPX and write data to a text file on the CPX

All methods will work but some will obviously have their pros and cons. I suggest you get comfortable with 1 method and use that for the remainder of the semester. Whatever option you choose though will provide you with a data file that you can read in Python on your desktop computer to plot. The simplest way to import data is by using the loadtxt function from the module numpy. Here is some very simple code to plot data from a text file. I also have a YouTube video explaining how to plot a text file if you'd rather watch something.

When you plot make sure your *Test_Data.txt* file is in the same folder as your plotting script in Thonny or Spyder. Here's my example code (this code is not on GitHub but you only need 3 or 4 lines of code to plot).



The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, and Run. The main window has several panes:

- Code Editor:** Displays the Python script `example_file_plotting.py`. The code imports numpy and matplotlib.pyplot, loads data from `Test_Data.txt`, and plots the time versus button value.
- Variables:** A table showing variables with their names and values.
- Figure 1:** A plot titled "Figure 1" showing "Button Value" on the y-axis (0.0 to 1.0) versus "Time (sec)" on the x-axis (8 to 12). The plot shows a series of sharp spikes at approximately 9.5s, 10.0s, 10.5s, 11.0s, and 11.5s, where the button value jumps to 1.0 and then drops back to 0.0.
- Shell:** A terminal-like pane showing the output of running the script. It displays the first few lines of the data from `Test_Data.txt`.
- Run:** A button labeled "Run" with a play icon.

In this example lines 1 and 2 import numpy and matplotlib. Line 4 imports data from the *Test_Data.txt* file and then 6 and 7 save the first and second columns into time and button. The remaining lines plot the data and create x and y labels as well as a grid. Hopefully now you are well versed in taking data and plotting in Python.

6.9 Assignment

Upload a PDF with all of the photos and text below included. My recommendation is for you to create a Word document and insert all the photos and text into the document. Then export the Word document to a PDF. For videos I suggest uploading the videos to Google Drive, turn on link sharing and include a link in your PDF.

1. Use method 1, 2 or 3 and save time and button presses to a text file
2. Include a video explaining which method you are using to record button presses and show yourself pressing the CPX button a few times and recording data. Make sure to wave and introduce yourself - 30%
3. Copy and Paste your CPX code used to log data - 20%
4. Copy and paste your Python desktop code used to plot your data - 20%
5. Include a plot of your button presses with time on the x-axis and button presses on the y-axis (no screenshots) - 30%