Project Based Engineering Instrumentation With CircuitPython

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

Last Update: June 1, 2022

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 chapters moved here

Changes Needed

1. All chapters from TTT need to be moved here

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

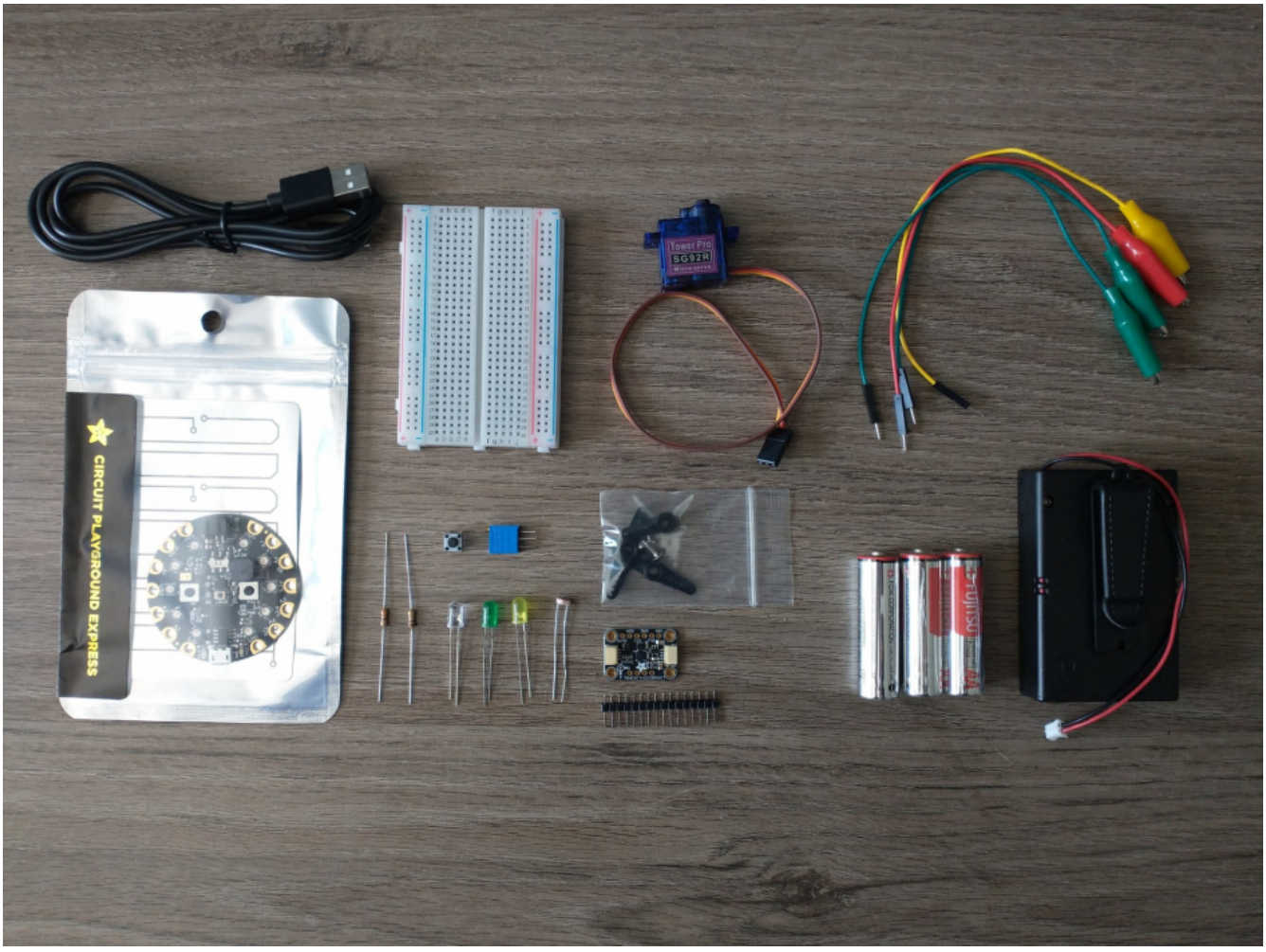
1.1 Parts List

Laptop + internet connectivity

In this class youre going to build some circuits that will enhance your learning experience. Rather than just solving problems by hand youre 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 stil 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%

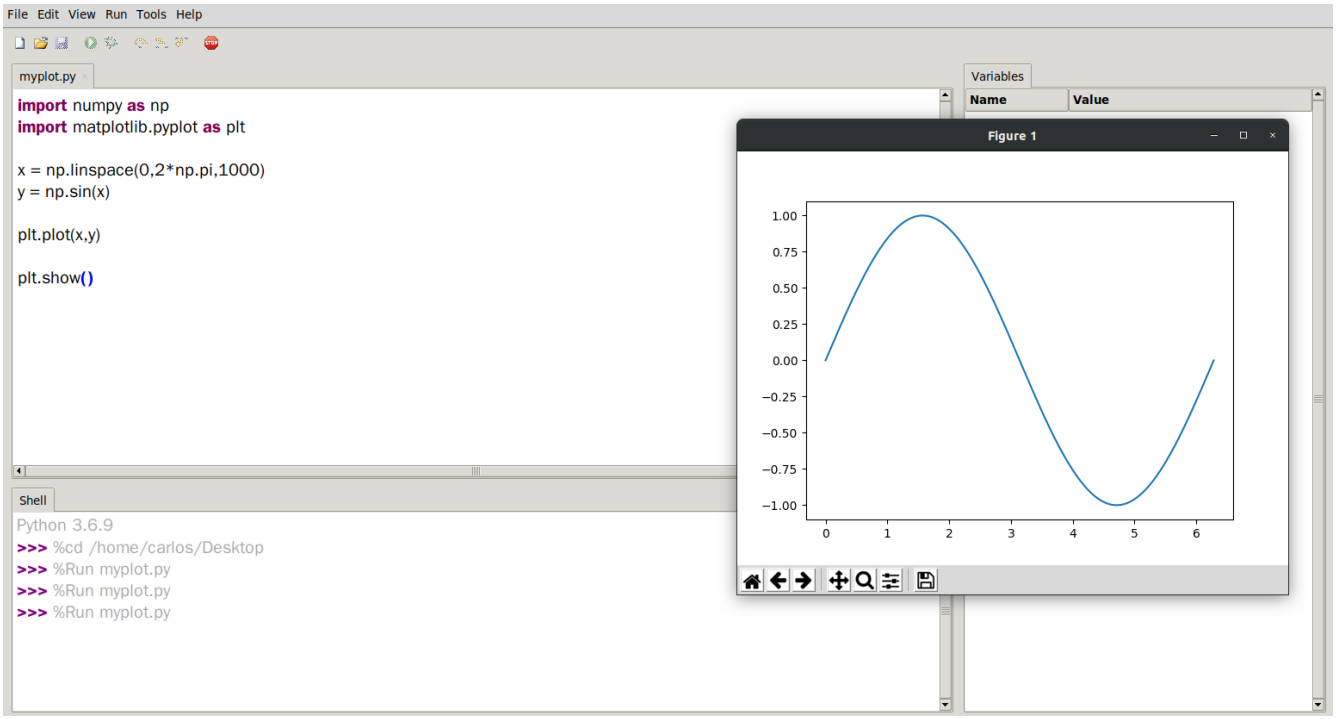
2 Download Python for Desktop

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.

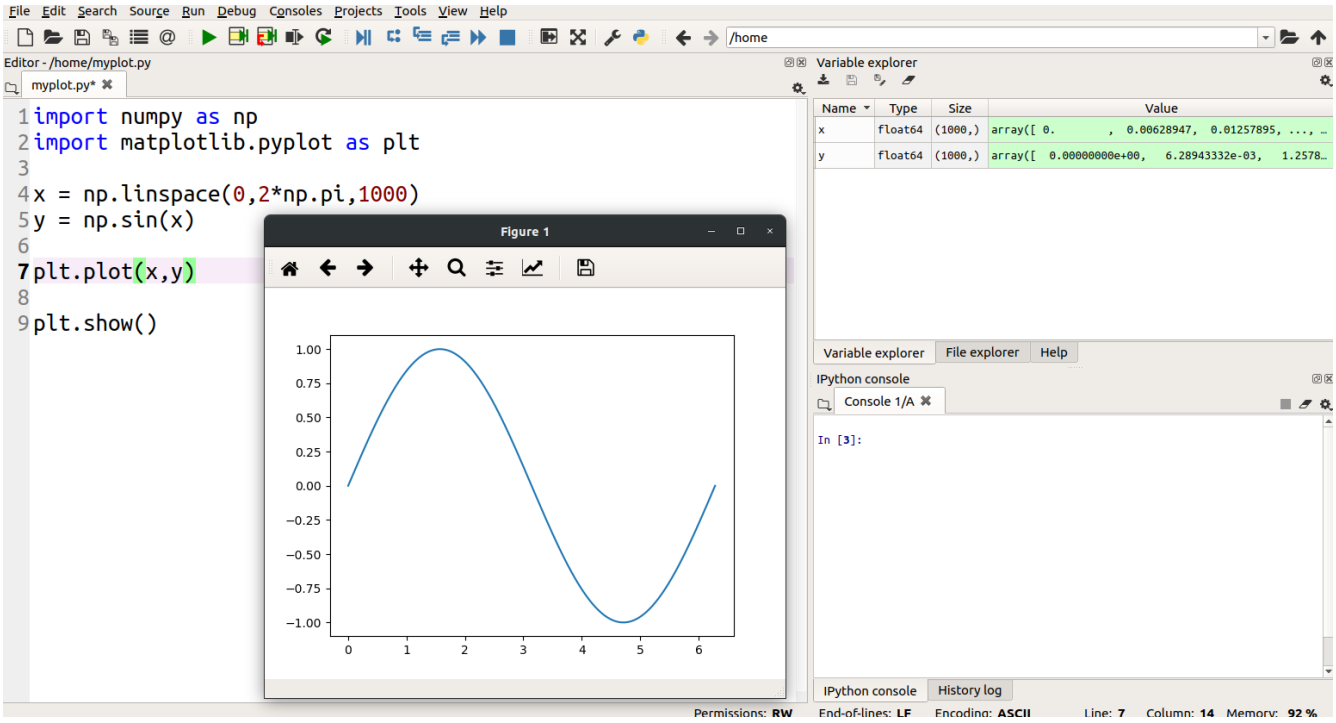
2.1 Thonny

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



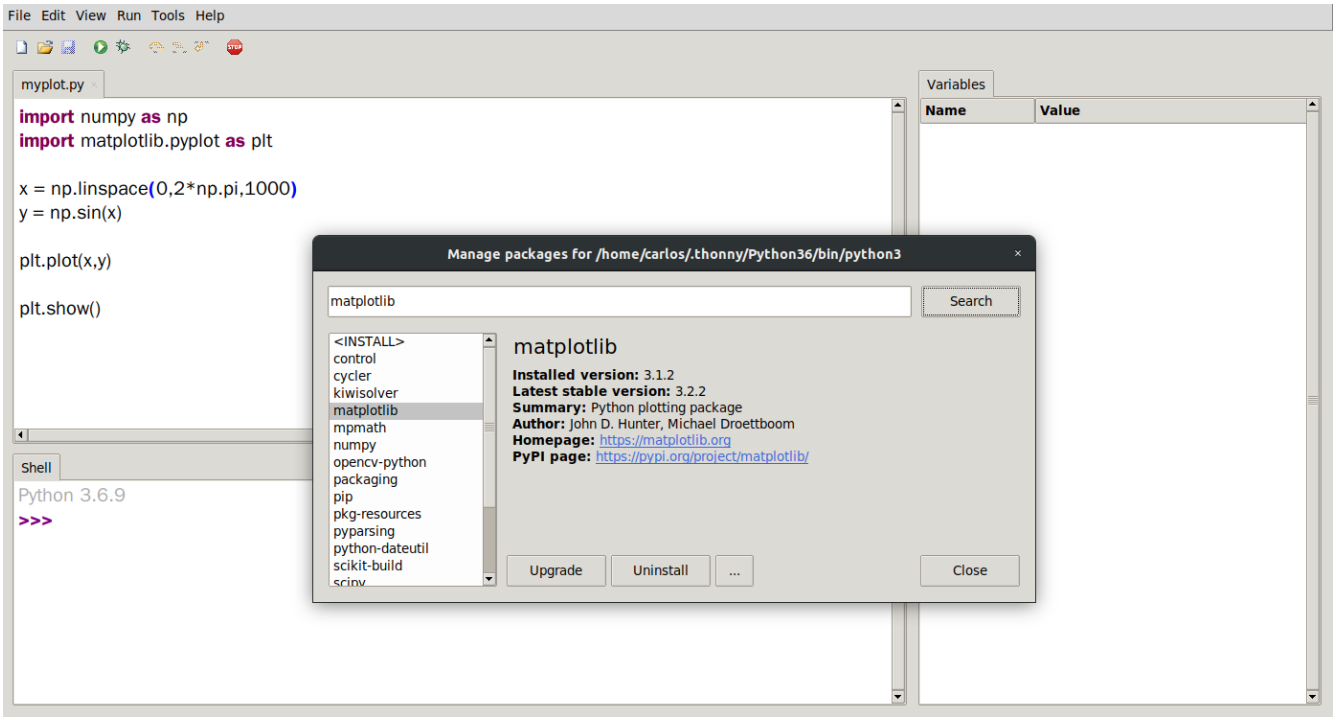
2.3 Other Options

It is possible to use Google Colab if you want to collaborate on Python projects or even get apps for your phone (Pydroid or Pythonista depending on Android or iPhone). You'll need to download 32 bit or 64 bit but which one?

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



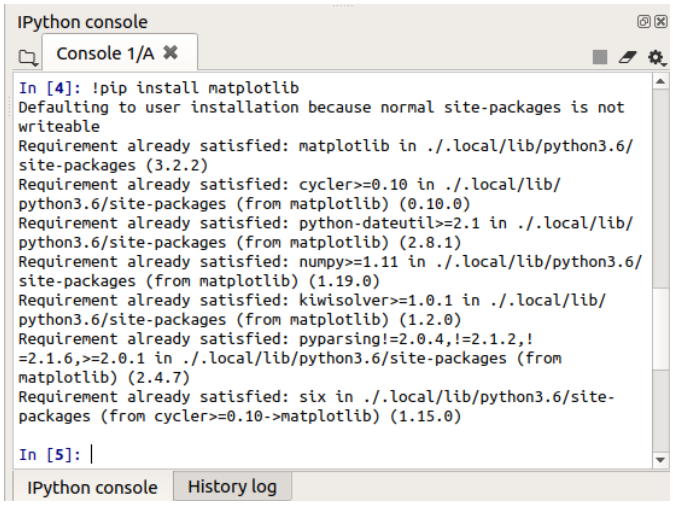
If numpy or matplotlib is not already included in Spyder then you need to type the following into the Python Console in the lower right hand corner of Spyder which is called the IPython console.

```
!pip install matplotlib
```

If that doesnt 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.



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_SUPP
ORT', 'FPE_DIVIDEBYZERO', 'FPE_INVALID', 'FPE_OVERFLOW', 'FPE_UNDERFLOW', 'False_', 'Inf', 'Infinit
y', 'MAXDIMS', 'MAY_SHARE_BOUNDS', 'MAY_SHARE_EXACT', 'MachAr', 'ModuleDeprecationWarning', '
NAN', 'NINF', 'NZERO', 'NaN', 'PINF', 'PZERO', 'RAISE', 'RankWarning', 'SHIFT_DIVIDEBYZERO', 'SHIFT_I
NVALID', 'SHIFT_OVERFLOW', 'SHIFT_UNDERFLOW', 'ScalarType', 'Tester', 'TooHardError', 'True_', 'UFU
NC_BUFSIZE_DEFAULT', 'UFUNC_PYVALS_NAME', 'VisibleDeprecationWarning', 'WRAP', '_NoValue', '_U
FUNC_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', 'arange', 'arccos', 'arcc
osh', 'arcsin', 'arcsinh', 'arctan', 'arctan2', 'arctanh', 'argmax', 'argmin', 'argpartition', 'argsort', 'argw
here', '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', 'binco
unt', 'bitwise_and', 'bitwise_not', 'bitwise_or', 'bitwise_xor', 'blackman', 'block', 'bmat', 'bool', 'bool8',
 'bool_', 'broadcast', 'broadcast_arrays', 'broadcast_to', 'busday_count', 'busday_offset', 'busdaycalen
dar', 'byte', 'byte_bounds', 'bytes0', 'bytes_', 'c_', 'can_cast', 'cast', 'cbrt', 'cdouble', 'ceil', 'cfloat', 'c
har', '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 commo
n 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:

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

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

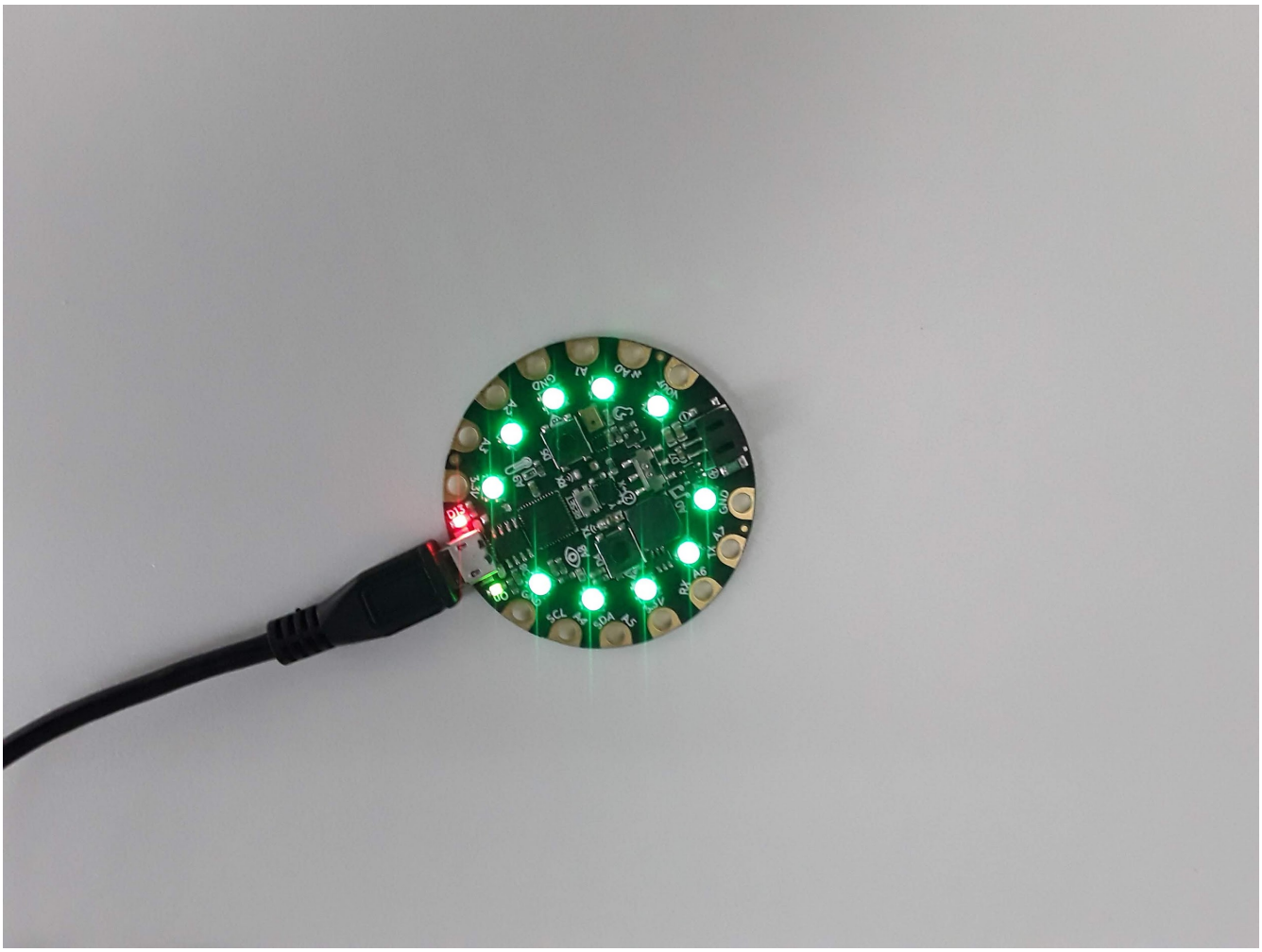
3.1 Parts List

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

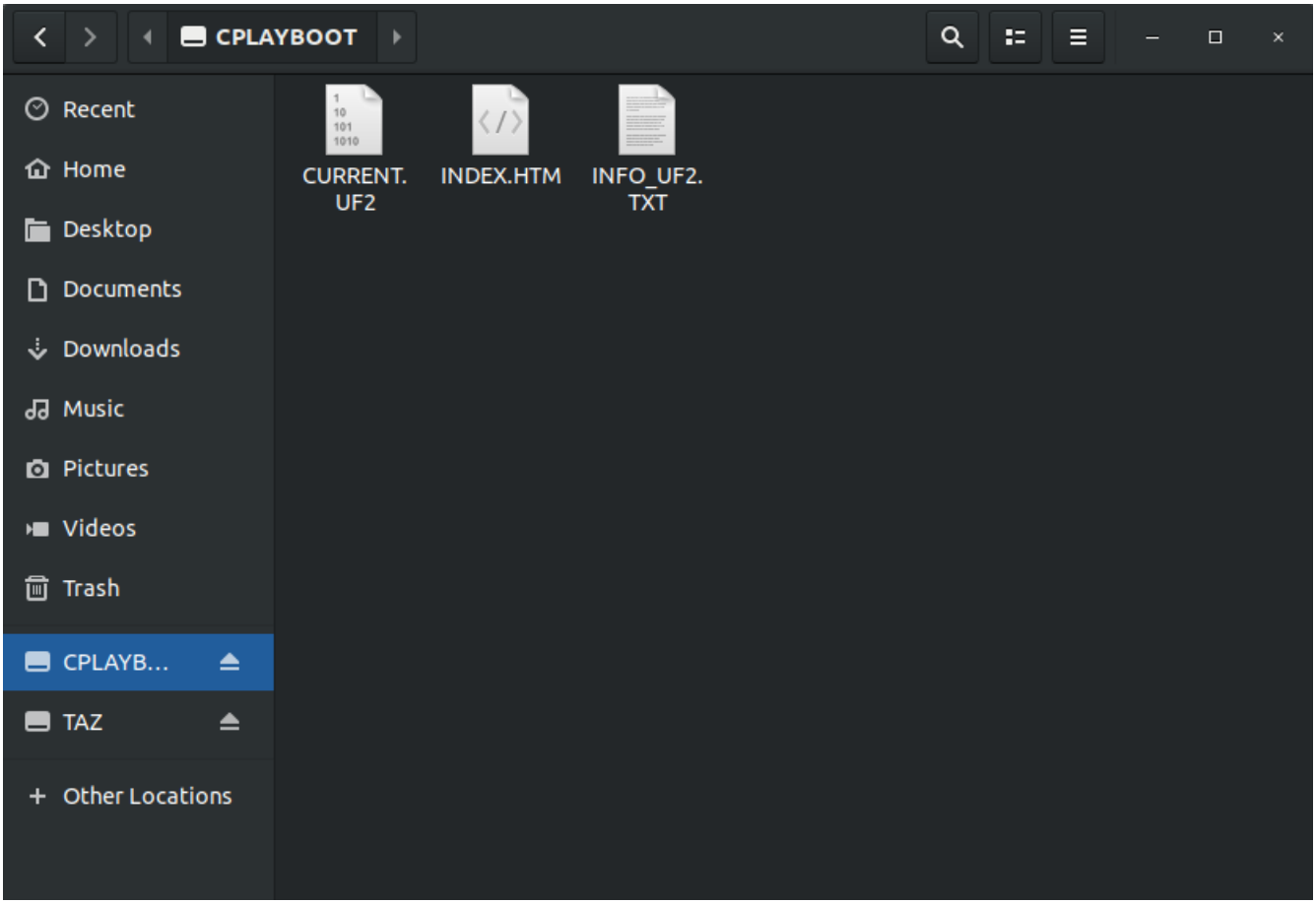
3.2 Setting up your Circuit Playground

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.

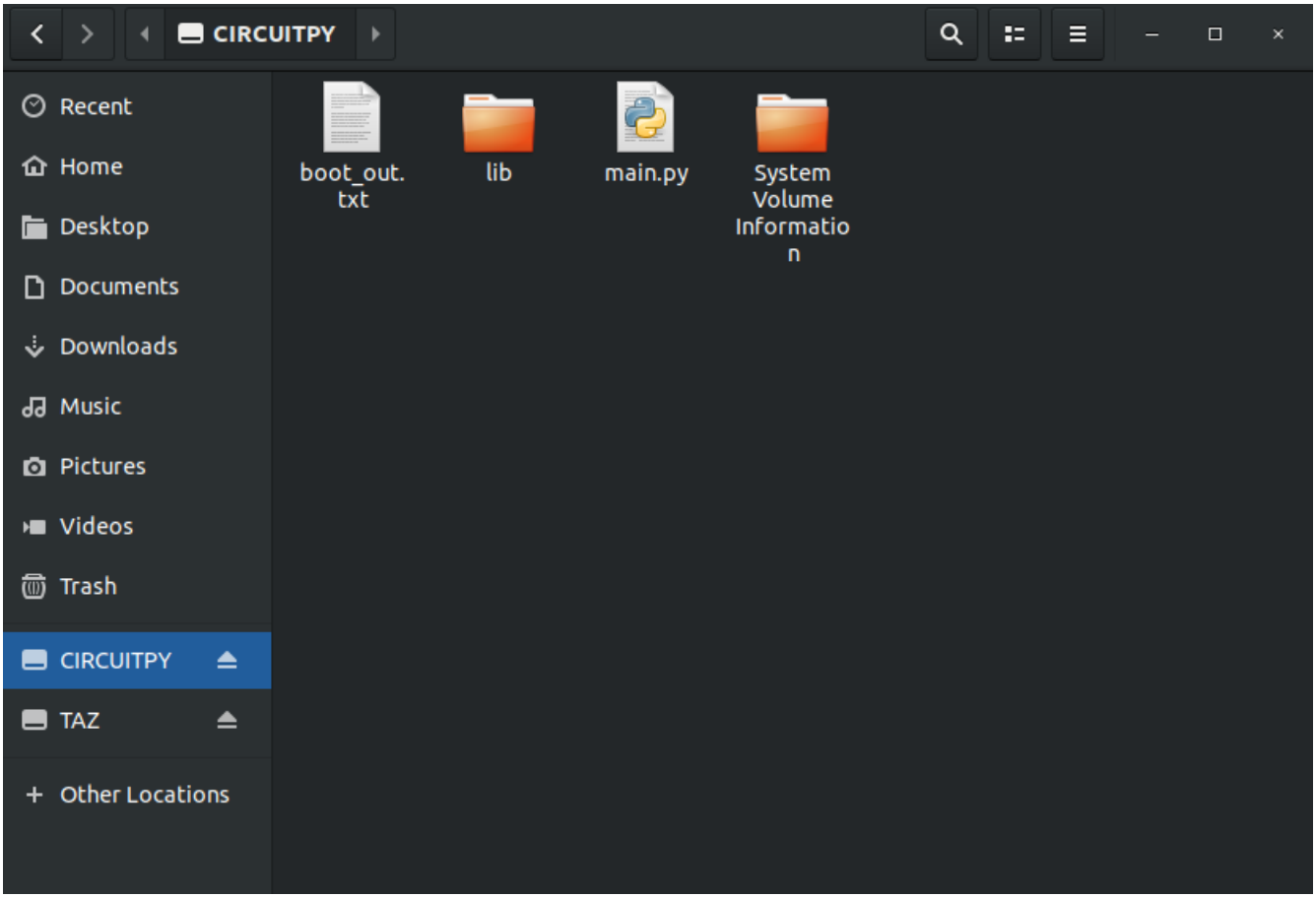
When you get your CPX and plug it into the computer via USB it actually won't run Python just yet. First you need to double click the reset button (the button in the center. It says RESET above the button) and put it into boot mode. All the neopixels (the ring lights on the CPX) will light up green.



Something called CPLAYBOOT will pop up on your computer just like a USB stick or external harddrive. A couple files with be in there but it doesnt matter what they say right now.



You then need to download whats 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.

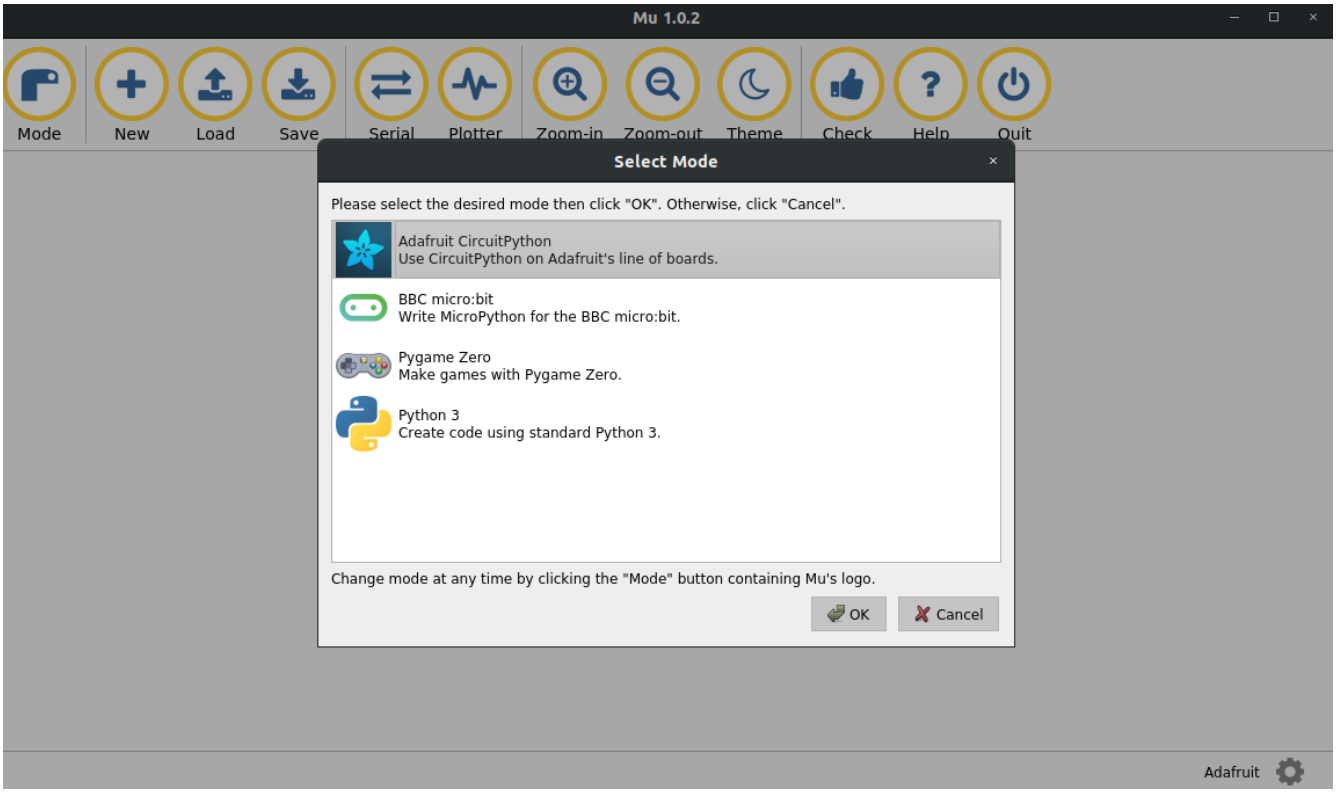


At this point the CPX is like a mini computer. If you put Python code on the flash drive it will run python code. Since Ive done this before, there are a number of files already on my CPX. You may have some or none of these. The boot_out.txt file will tell you the version of CircuitPython you have on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

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

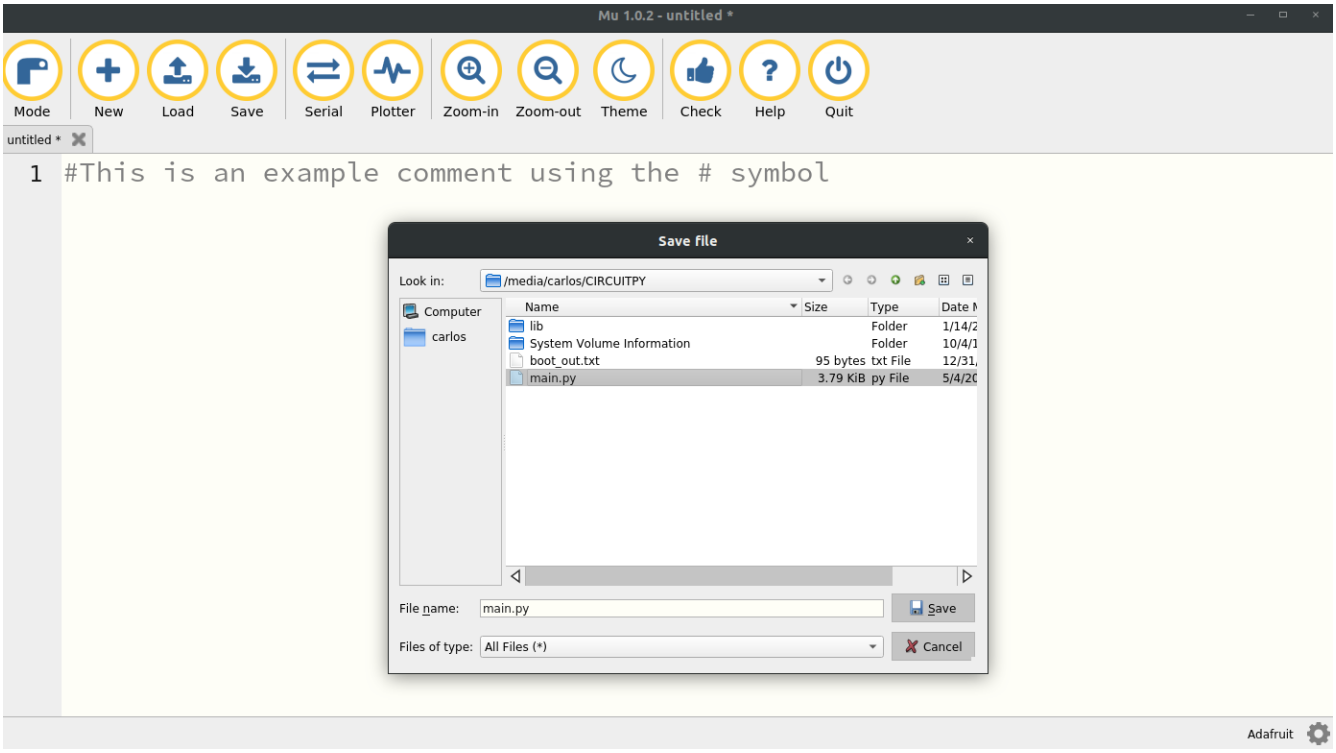
Which means I have CircuitPython version 5.3.0 last updated on April 29th, 2020. The CPX itself is using the ATSAMD21G18 microprocessor. The folder lib is a folder with extra libraries that you may need to install later. The file main.py is the Python file that the CPX is currently running. **The CPX 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 alot of debugging options and it is recommended to use a program called Mu. Mus 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 youve downloaded and installed Mu and open it up it will look like this (Note it's possible the 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)



Youll 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 wont do anything.

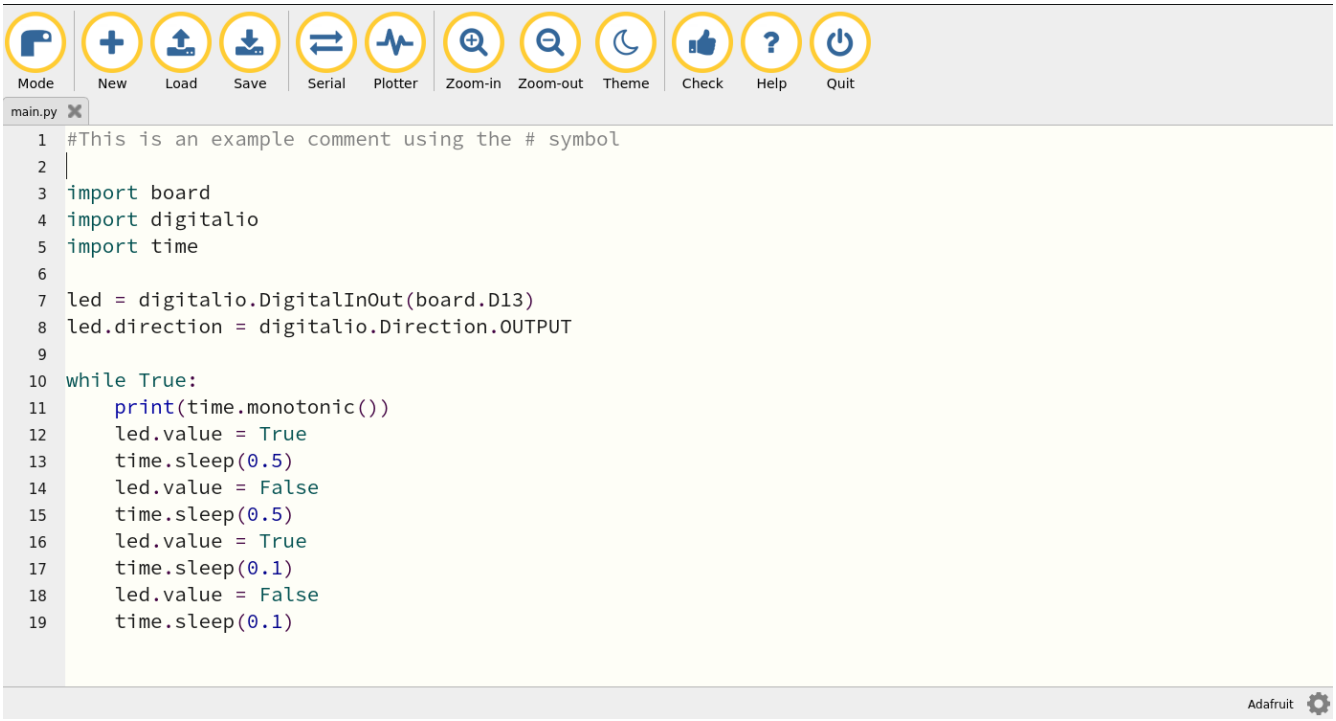
Now this is really important. Your CPX is a USB stick that can hold as many Python files as 2MB will allow but it can only run or execute one python script at a time. Furthermore it will only run two types of files. It will run code.py if it exists and if it can't find code.py it will run main.py If the CPX can't find main.py or code.y 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.by 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 lets 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. Ive 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.

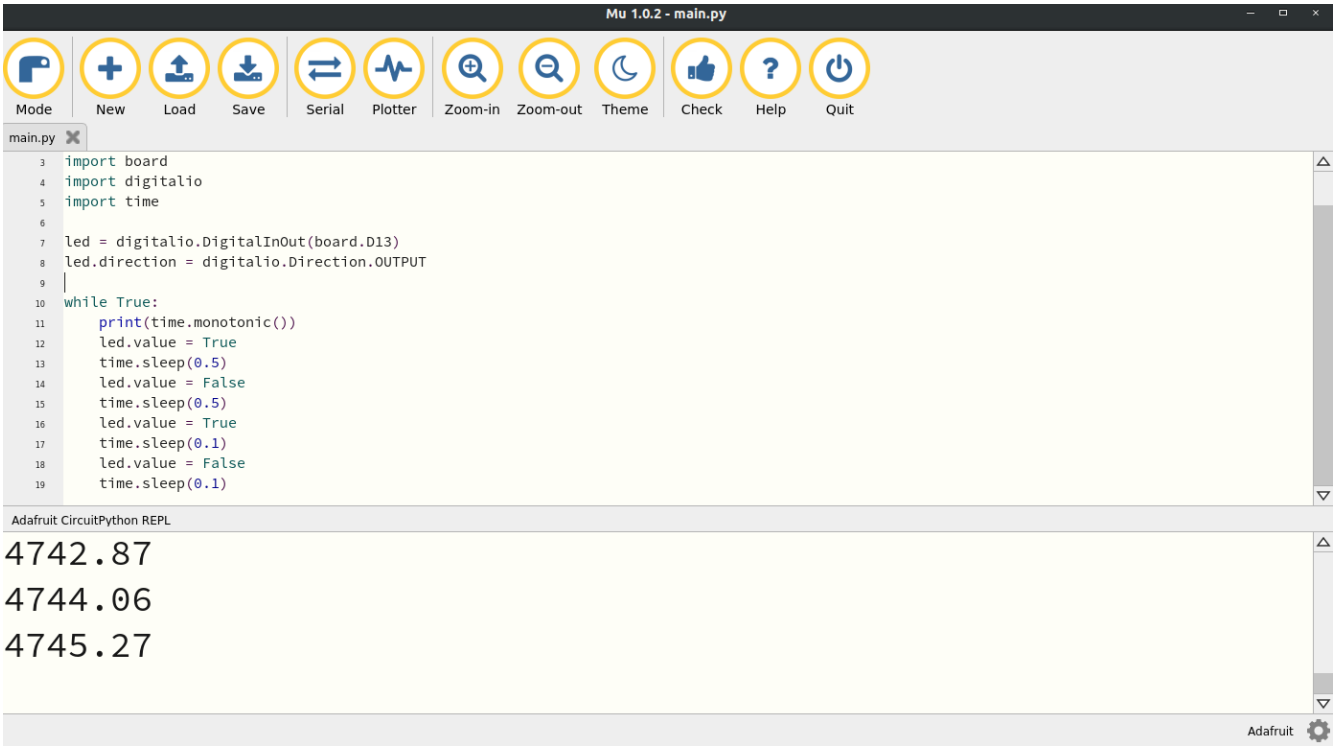


Make sure to save. You can click *Zoom-In* and *Zoom-out* to zoom in and out to change the font size and see more of the output. If the blink code is working you will see a red led labeled D13 on the CPX blink back and forth. D13 stands for digital pin 13. Youll notice there are analog pins labeled A5 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.

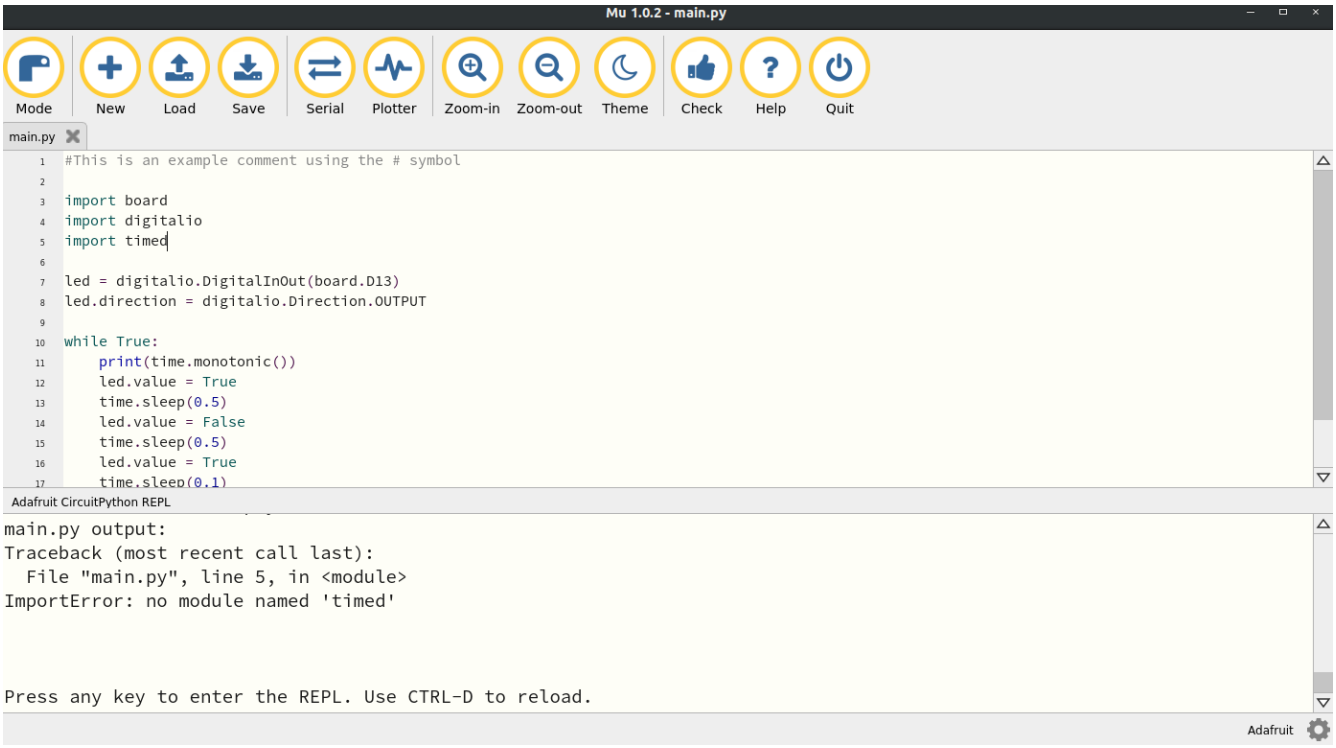
Moving along, lines 7 and 8 create an LED object using the *board* module and *digitalio*. Its a long line of code that basically says, create a variable called *led* that lets us output a digital signal to pin D13. We also set the direction of the LED to output since we only want to write to the LED.

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.



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.

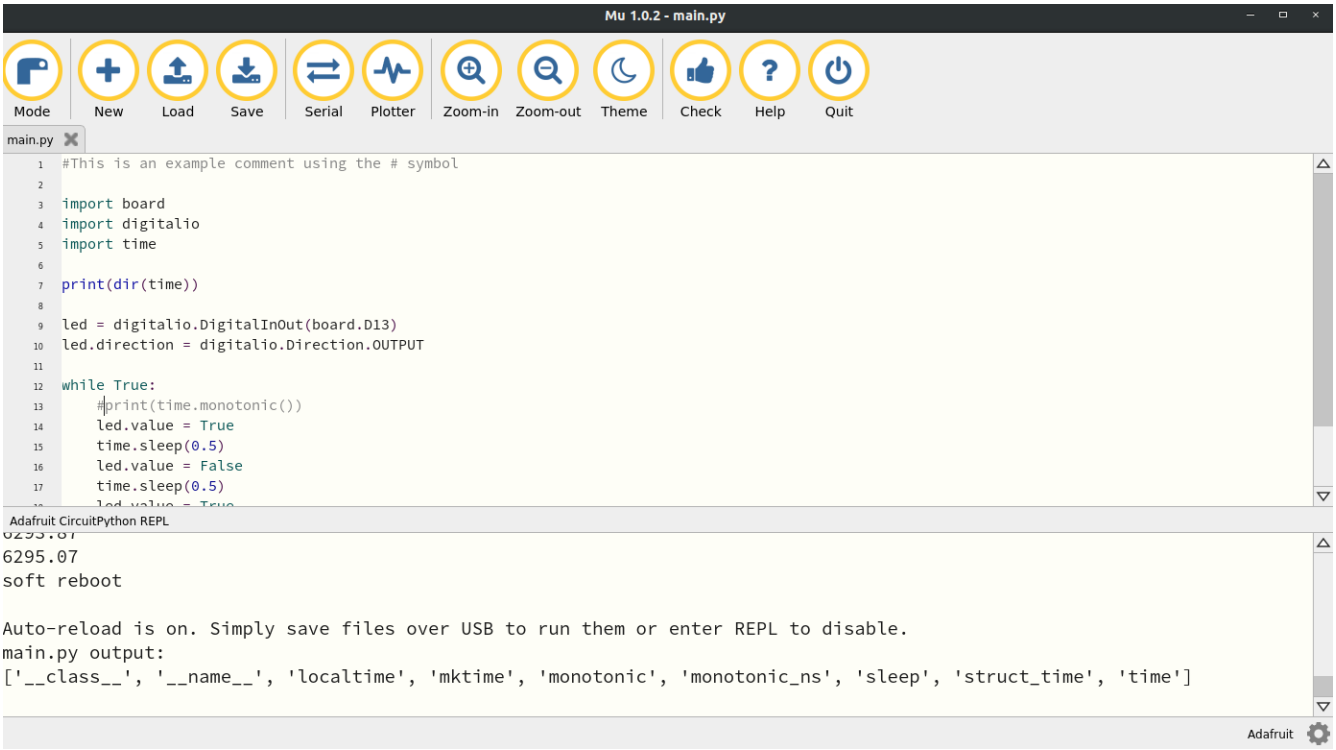


In this case I have an error on line 5. Its saying there is no module named *timed*. The reason that module doesnt exist is because the module is actually *time* not *timed*. You can use the *Serial* monitor to check on your

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.



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.by 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 doesnt work anymore
8. First, try and reset the CPX to CPLAYBOOT and reflash the UF2 to see if that fixes it.
9. If that doesnt 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 were 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 youll see the D13 LED blink as before. You wont 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 doesnt work it may be because those two wires are backwards.**

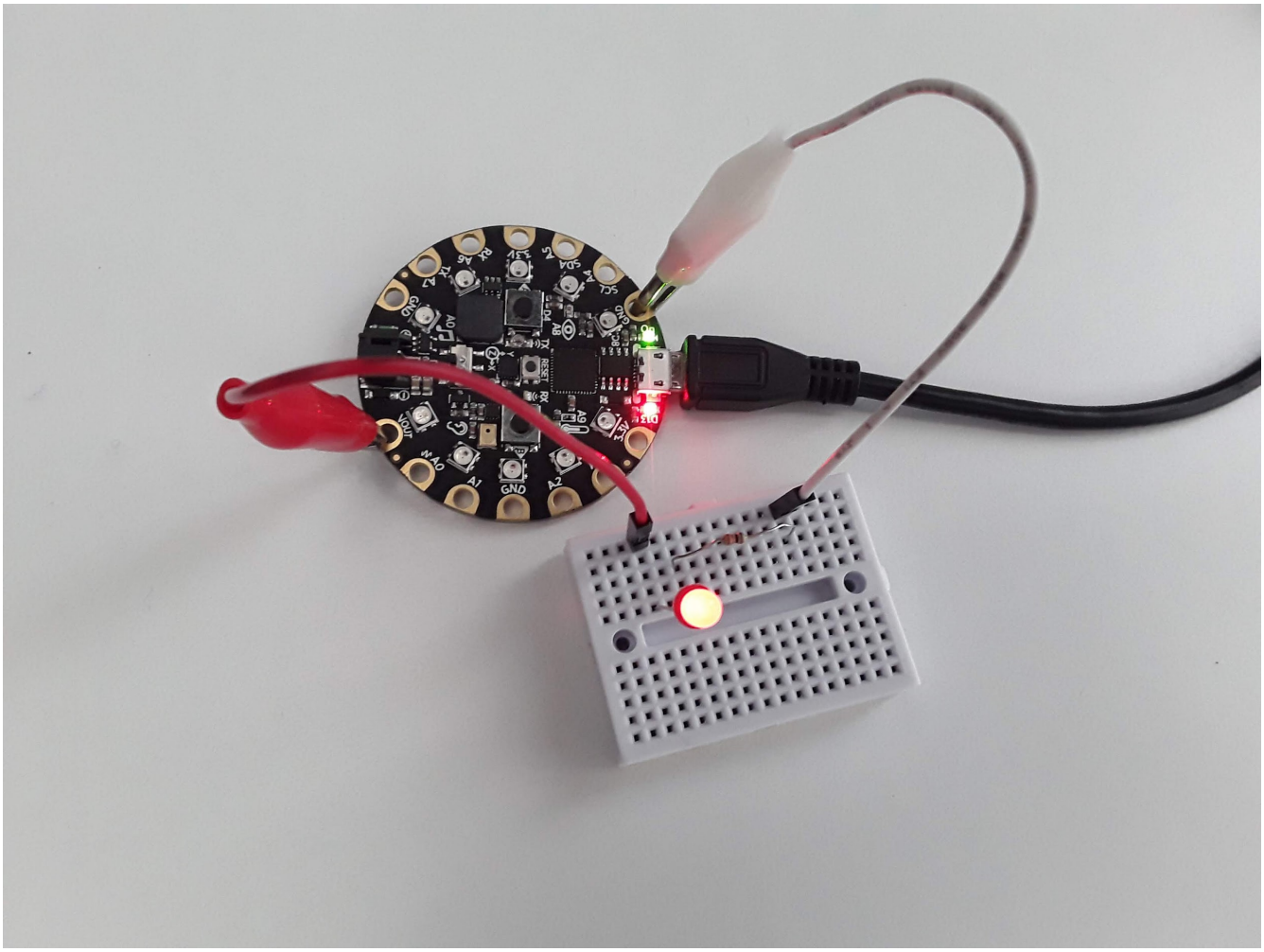
The CPX itself uses 3.3V logic which means when it converts numbers to binary a 0 (False) represents 0 volts and a 1 (True) represents 3.3V. The CPX has ports that are labeled various things. GND stands for ground and you need to hook the negative end of your circuit to this and it also has VOUT which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

Youre going to need to use a breadboard so if youre 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

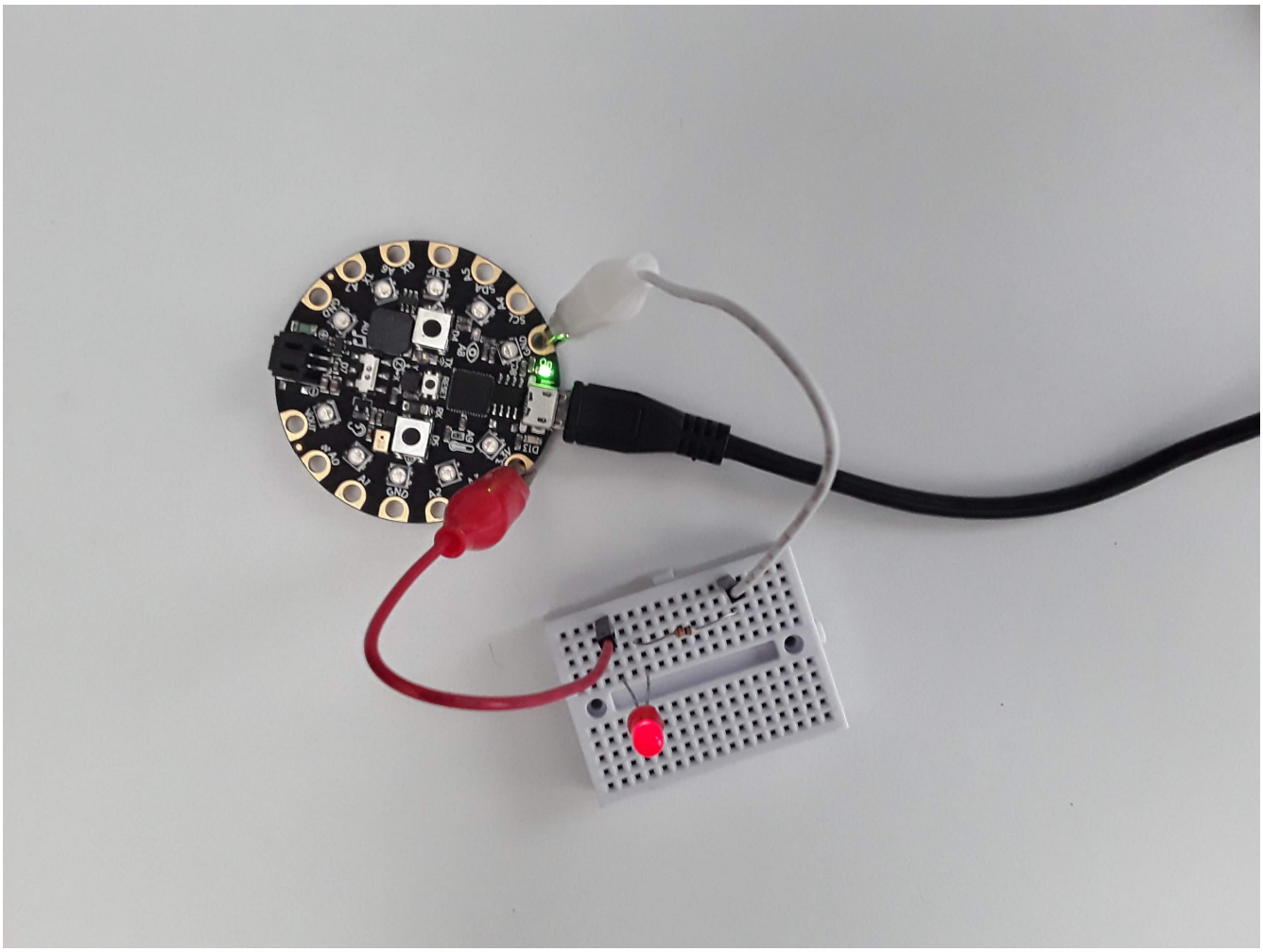
Whenever you modify a circuit on the breadboard, always be sure to remove power from the CPX. You can damage multiple components if youre not careful.

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 wont turn on because the voltage/current through the LED will be below the activation voltage of the circuit.

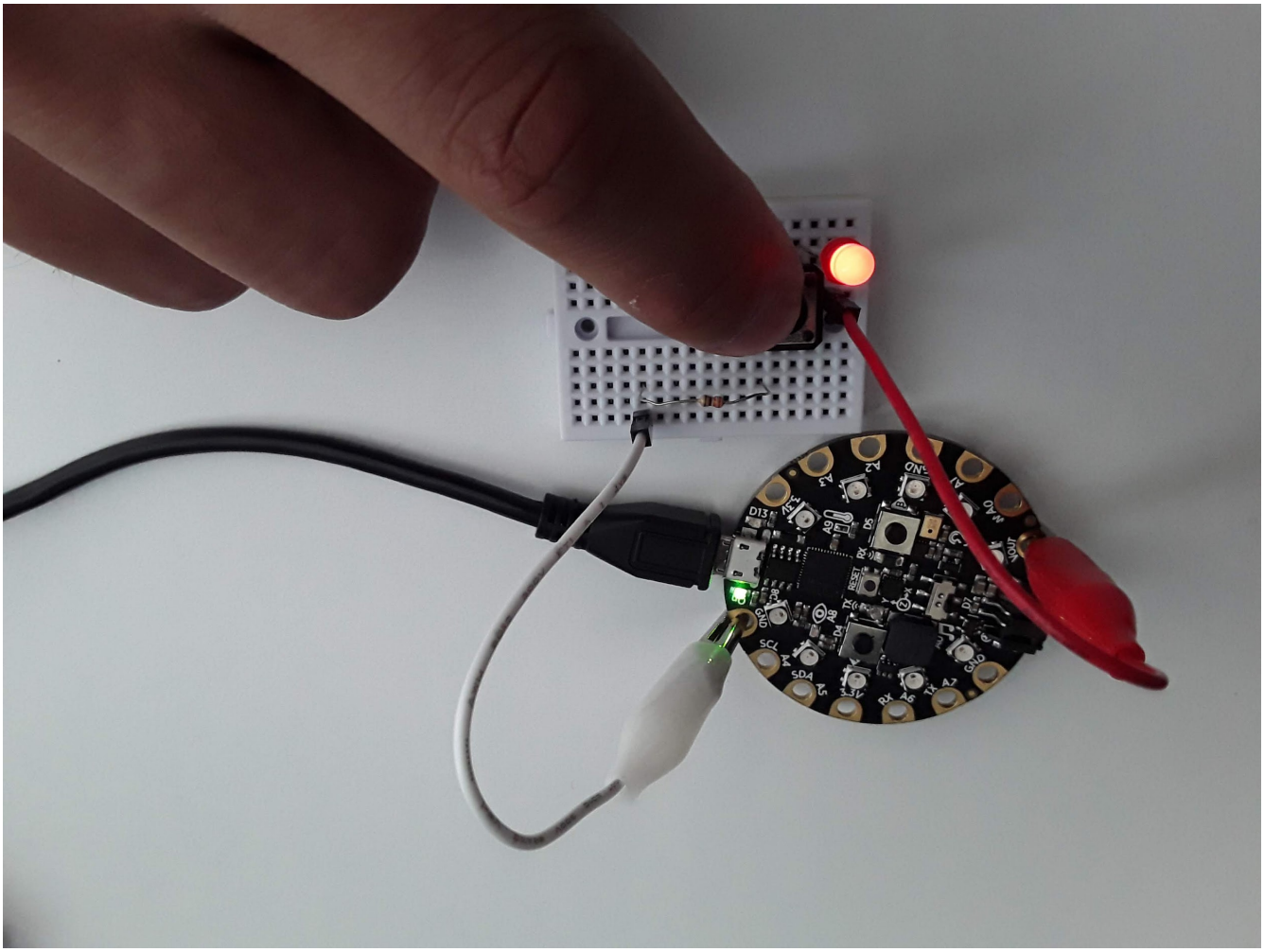


Once you have that circuit working, wire up the circuit again with the 3.3V output. Do you notice anything different when you hook up the circuit with different pins? Heres my circuit. Do you notice something different about the intensity of the LED? Why is it different?



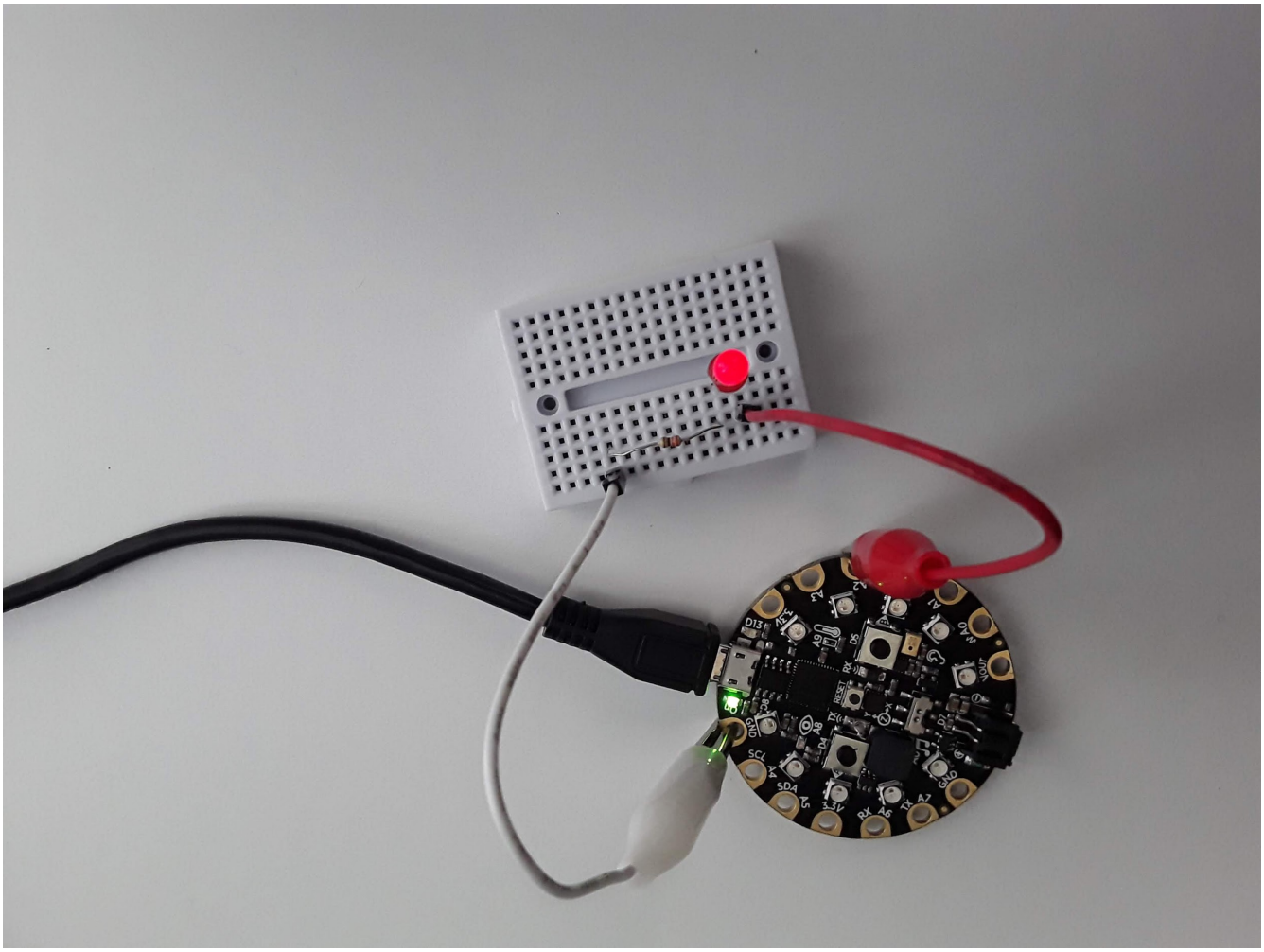
5.4 LED with a push button

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.



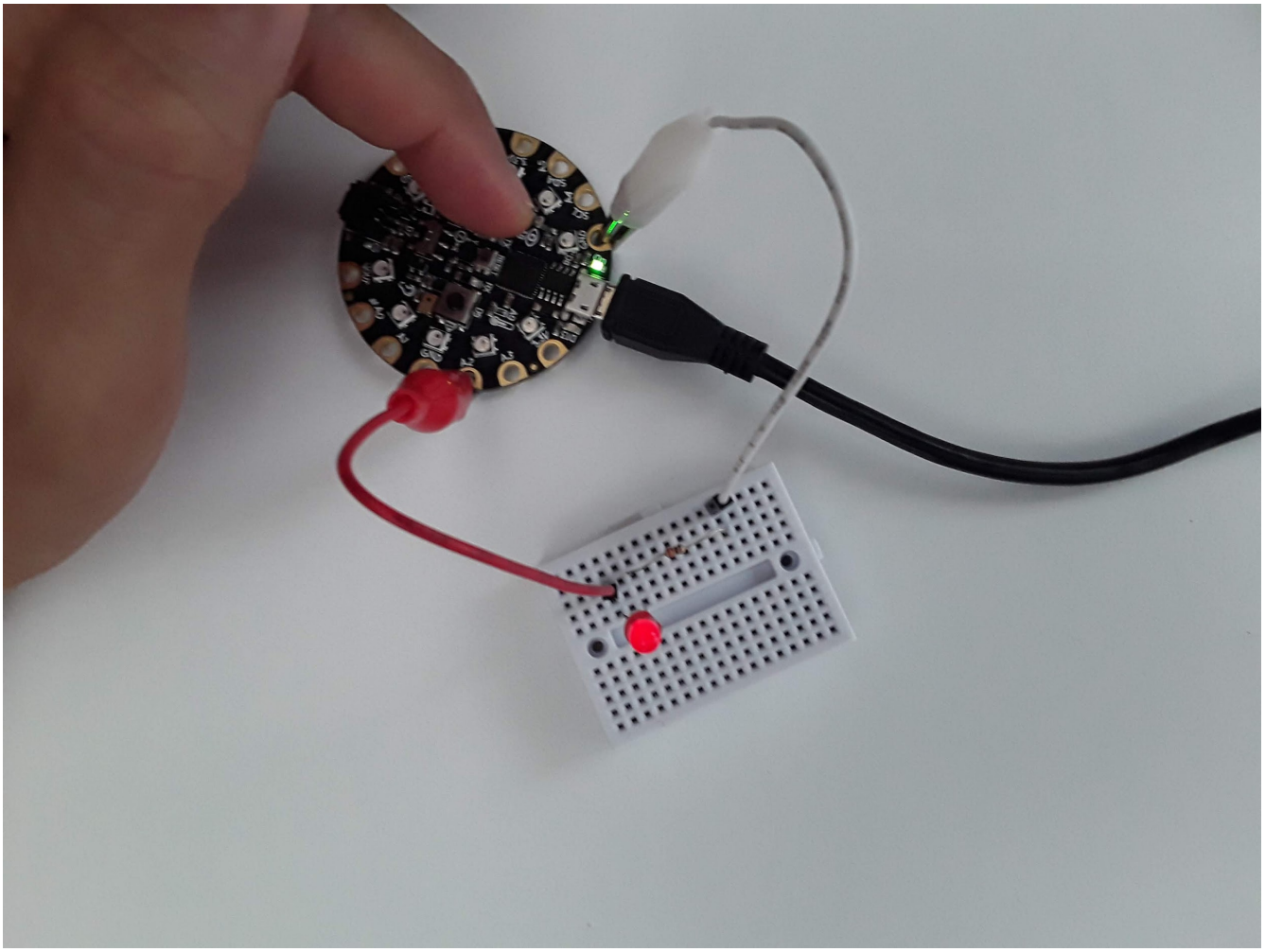
5.5 LED with code

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.

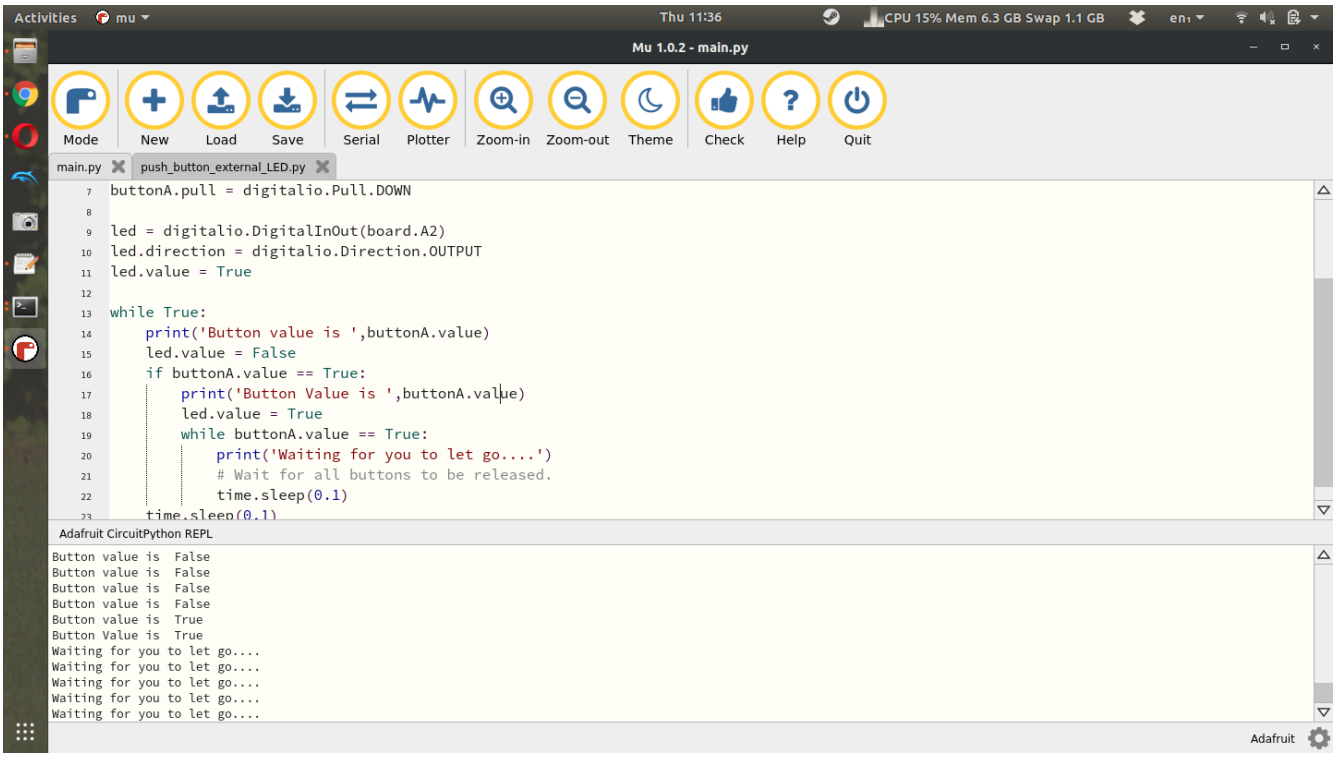
```
1  import board
2  import digitalio
3  import time
4
5  buttonA = digitalio.DigitalInOut(board.BUTTON_A)
6  buttonA.direction = digitalio.Direction.INPUT
7  buttonA.pull = digitalio.Pull.DOWN
8
9  led = digitalio.DigitalInOut(board.A2)
10 led.direction = digitalio.Direction.OUTPUT
11 led.value = 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.



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%