

# **Project Based Engineering Instrumentation With CircuitPython**

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

Last Update: May 30, 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/](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

<b>1</b>	<b>Part 1 Purchase Equipment and Download Python IDE for Desktop</b>	<b>5</b>
1.1	Parts List . . . . .	5
1.2	List of Items in Kit . . . . .	5
1.3	Part 1 Assignment . . . . .	6
<b>2</b>	<b>Part 2 Download Python for Desktop</b>	<b>6</b>
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 Part 2 . . . . .	11
<b>3</b>	<b>Getting Started with the CPX/CPB</b>	<b>12</b>
3.1	Parts List . . . . .	12
3.2	Setting up your Circuit Playground . . . . .	12

# 1 Part 1 Purchase Equipment and Download Python IDE for Desktop

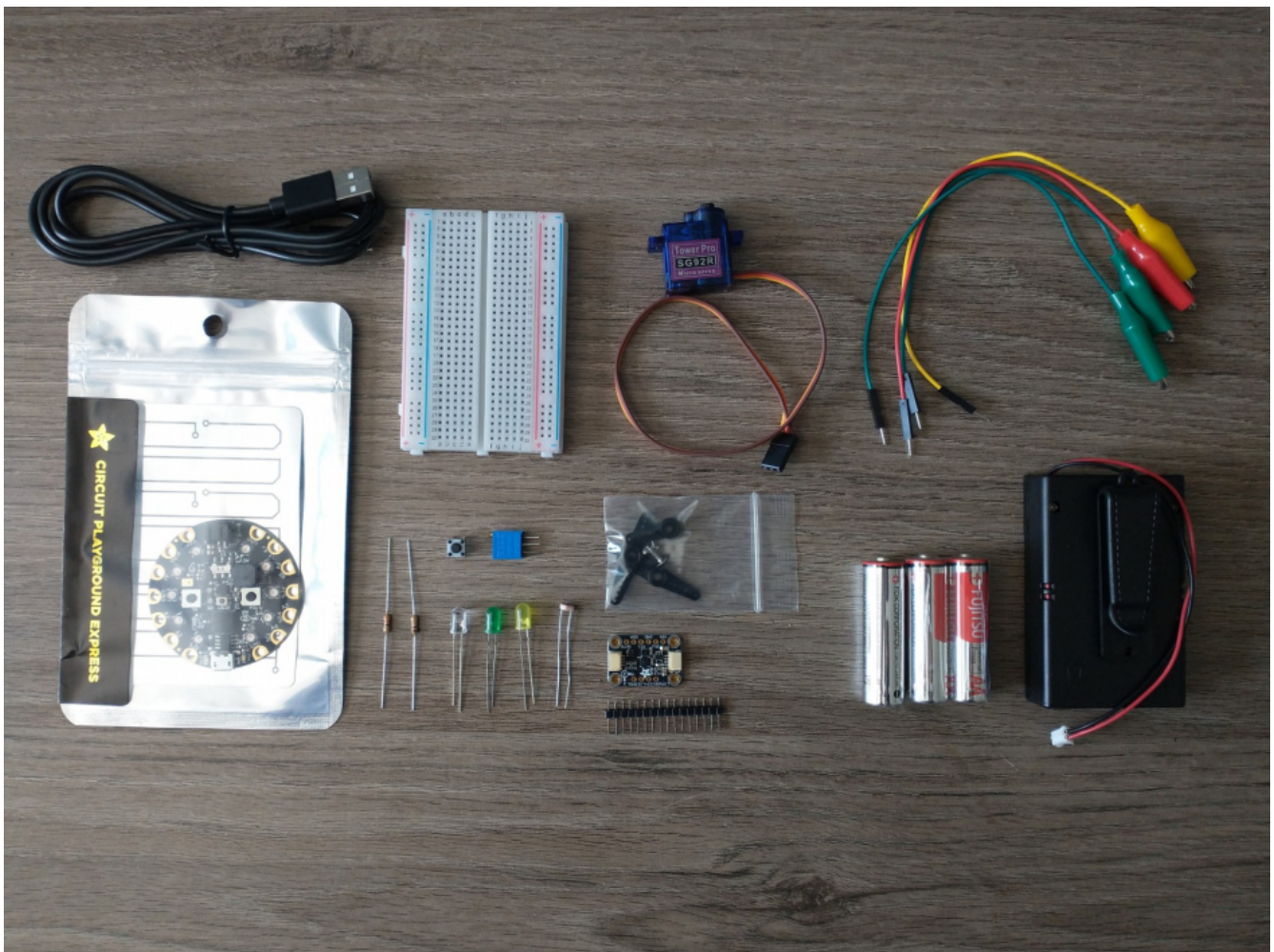
## 1.1 Parts List

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 Part 1 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%

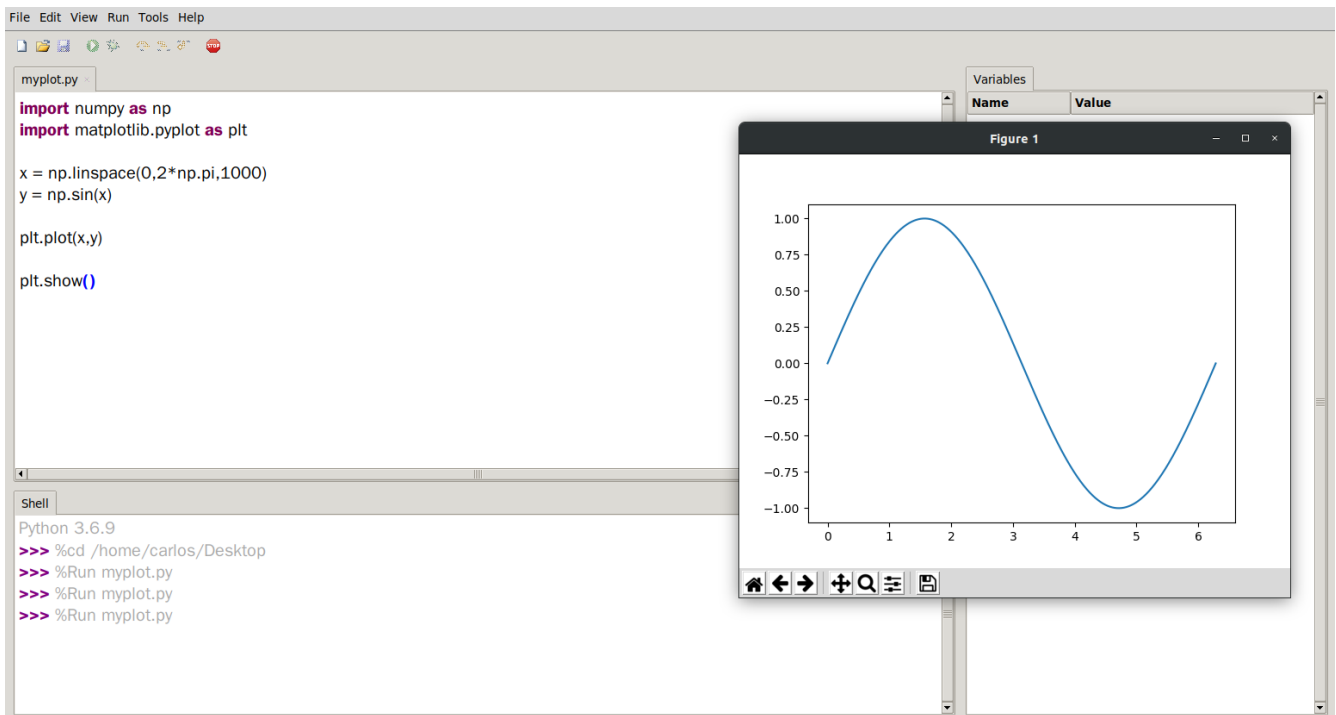
## 2 Part 2 Download Python for Desktop

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.

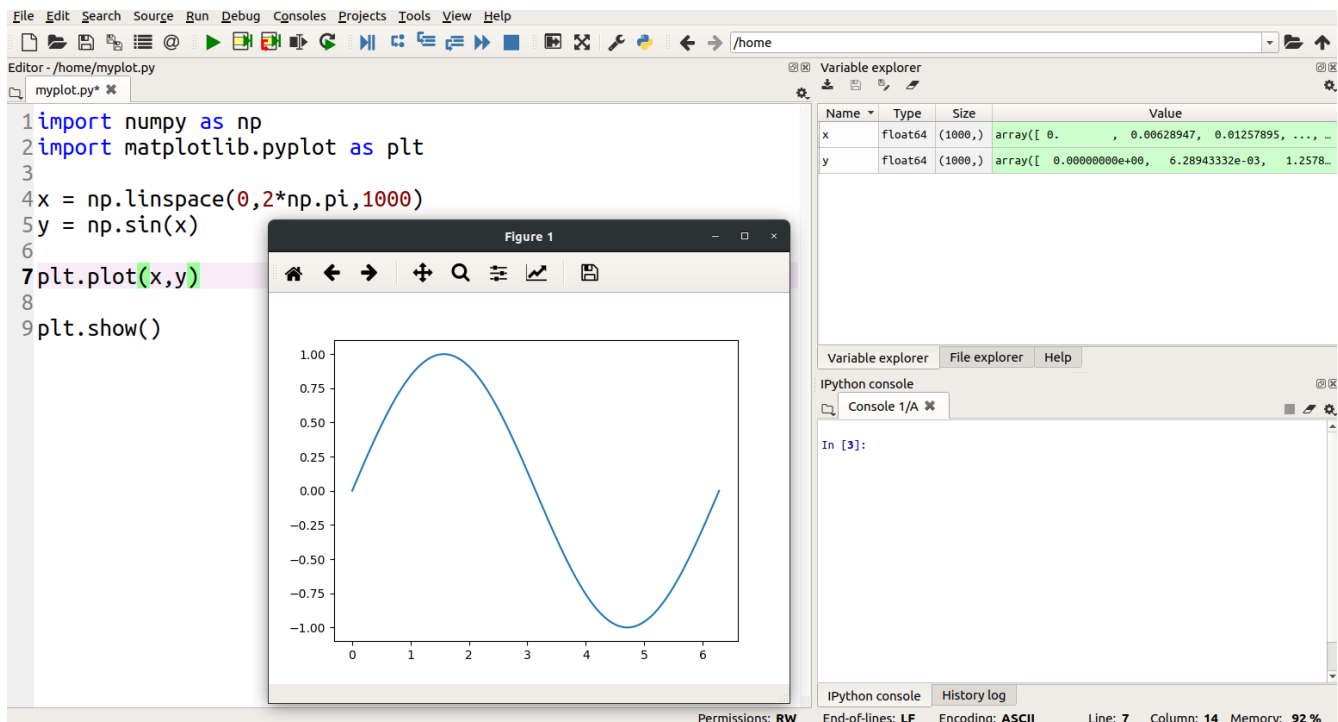
### 2.1 Thonny

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



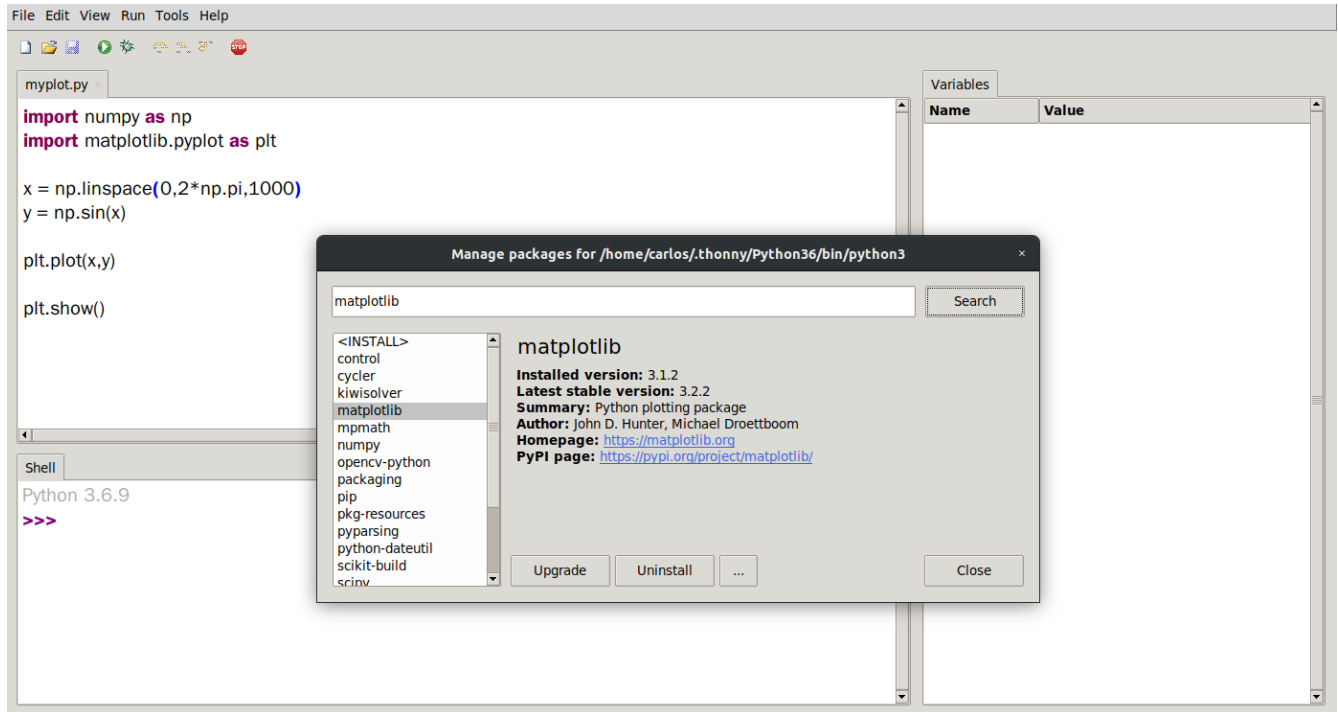
## 2.3 Other Options

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.

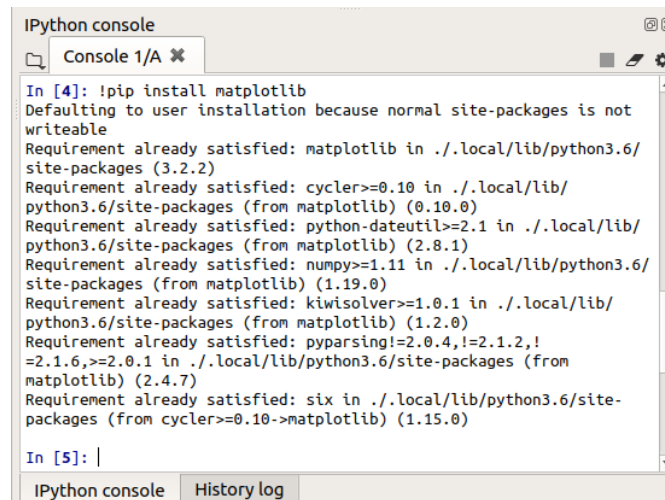
```
!pip install matplotlib
```

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.





```
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]: |

IPython console  History log
```

## 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', 'Infini
ty', '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. You'll notice there are a ton of functions in numpy. Every function in Python has a

```
__doc__
```

function. That's two underscores followed by `doc` and then another two underscores. If you're 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 I'm 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
```

You'll see that `arctan2` takes 2 input arguments `x1` and `x2`. I didn't 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, there's 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. I'm 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. It's the best in my opinion.

## 2.7 Assignment Part 2

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 doesn't 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

Laptop

CPX(or CPB) + USB Cable

### 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.