Topics:

- Good Coding Practices
- Writing Functions
- Python Resources
- Taylor Series
- Search Engines and Generative AI
- Plotting

Good Coding Practices

What are good coding practices? These tips have come from class discussion in years past.

Strategize your problem-solving

- Plan out your problem-solving approach
- Write your code in separate steps, and review these steps to get a better understanding
- chunk your code into different small sections

Give your code style and readability

- Write comments to explain your code
- Name your variables sensibly, not just a, b, c
- Make your code readable to others
- Use a case when naming variables (e.g., `snake_case` or `camelCase`) or some type of consistent style

Make your code efficient

- Make parts of your code more reusable by writing functions for repeated calculations

Debug your code

- Run your code many times as you work, so you can spot where bugs are located
- Test your code as you go using print statements to see if the outputs are what you expect

Keep track of relevant information

- comment physical units into your code wherever physical quantities are represented
- Cite any sources in your notebook. One option is to use comments at the top/bottom of your code chunk

Writing Functions

When writing a new function, it's good practice to write details about the function's purpose, inputs, and outputs. This is for your own reference so that you are able to more easily write and use your function, but also for others who may be collaborating with you. This includes peers and instructors, who can more easily understand what you are trying to do with your code when there are comments describing it.

You can use the template below to include appropriate commentary and define a function.

```
# function_name (input_name1, input_name2, input_name3)
# purpose of function
#
# input_name1: purpose, type
# input_name2: purpose, type
# input_name3: purpose, type
#
# output: output_name
# output purpose, type

def function_name(input_name1, input_name2, input_name3):
    body of function_name
    ...
    return output_name
```

Python Resources

Here are some resources you can use to help you learn Python. Depending on the type of help you are looking for, some resources may be more helpful than others.

Python Education Subreddit: Community, general advice, and a place you can ask/browse questions about using Python.

W3Schools: Tutorials on Python programming from a low-level perspective (data structures, syntax, etc).

GeeksforGeeks: Tutorials that are more focused on concepts and examples of how to implement different tools. Highly recommended!

Programiz: More tutorials :)

Stack Overflow: One of the best debugging resources. A link isn't posted here, because the best way to use it is to Google/Bing an error message or a description of what you're stuck on and add the words "stack overflow" to your search.

Kaggle Lessons: Python walkthrough/crash course that you can work through at your own pace.

Python Tutor: Copy and paste your code in here to see a visualization of how the computer runs your code -- this is a great debugging tool.

Official Python Documentation: Go here to look under the hood of the Python programming language. There are also several links to more resources and tutorials here.

There are also docs for specific libraries, like **numpy** and **matplotlib.pyplot**

Taylor Series

Here is a helpful **conceptual video on Taylor Series**.

Search Engines and Generative AI

Some tips class discussions in years past are included below.

For using search engines like Google, and AI chatbots like ChatGPT:

- Use unique keywords when framing your query
- Likely someone has asked your question before on Google -- if there are only a small number of results you could try rewording it
- Google keywords alone for more general information
- When using ChatGPT, be more specific, let it know whether it answered your question, and answer it's follow-up questions
- With ChatGPT, you can copy your whole code into it and ask it to explain what each part does
- Ask questions about a specific method -- for example, search for "numpy array functions" instead of just "lists"
- Be aware, you can tell ChatGPT that it is wrong, but it still might not correct itself properly
- Sometimes ChatGPT can produce unnecessarily complex code even for a simple query

Plotting

Here are a few of the key pieces of plotting code offered by Matplotlib:

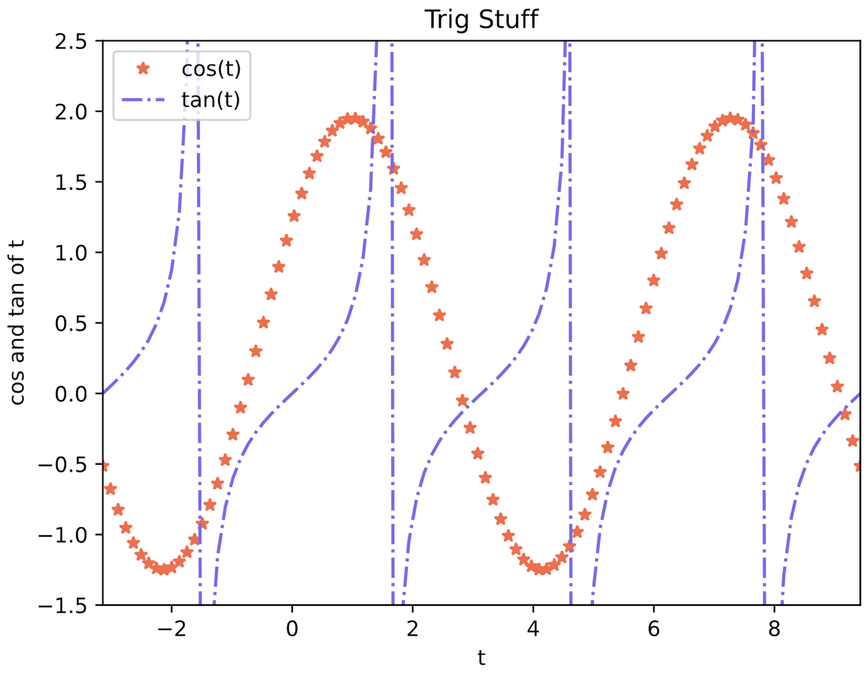
Function/Attribute	Purpose	How to Use	Syntax
`import` statement	gain access to `plt` functions and attributes	Use it on it's own line, and run it once before doing any plotting. It's best to have it near the top of your notebook with other import statements so you can run it once and forget about it.	`import matplotlib.pyplot as plt`
`plot`	creation of axes and plotted values	Use it on it's own line and provide `x` and `y` inputs, plus other optional attributes.	`plt.plot(x, y, ...)`
`legend`	adding a legend to a plot	Use it on it's own line with parentheses, but no inputs required. Previous uses of `plot` should have labels (see below). The `loc` input is optional, and allows you to specify where the legend is placed.	`plt.legend(loc="center left")`
`label`	create a label to appear in the legend	Use it as an input to `plot`, and don't forget to use `legend` later on.	`plt.plot(x, y, ..., label="mylabel")`
`color`	specify the color of the plot	Use it inside `plot` as an optional input. Options are the names of colors, found here.	`plt.plot(x, y, ..., color="red")`

Function/Attribute	Purpose	How to Use	Syntax
`marker`	specify the shape of each data point on the plot	Use it inside `plot` as an optional input. Options can be found here.	`plt.plot(x, y, ..., marker=".")`
`linestyle`	specify type of line connecting each data point	Use it inside `plot` as an optional input. Options are "None" for no line, or several line options found here.	`plt.plot(x, y, ..., linestyle="None")`
`xlim` and `ylim`	specify the bounds of the plot window	Use it on it's own line with two inputs: lower bound and upper bound	`plt.xlim(-50, 2000)`
`xlabel` and `ylabel`	create axis labels on the plot	Use it on it's own line with a "string" input indicating the desired axis label.	`plt.xlabel("Time (seconds)")`
`title`	create a title for your plot	Use it on it's own line with a "string" input indicating the desired title.	`plt.title("My Gorgeous Plot")`
`show`	ensure the plot displays in the output, sometimes not needed but always a good idea	Use it on it's own line with empty parentheses.	`plt.show()`

Below, you can see an example of these functionalities in use.

```
import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(-np.pi, 3*np.pi, 100)

cost = 1.6 * np.cos(t-1) + 0.35
tant = 0.4 * np.tan(t)

# you can use plt.plot multiple times to create multiple plots on the same set of axes
# the optional attributes can be left out and/or input in whatever order you please
plt.plot(t, cost, color="tomato", linestyle="None", marker="*", label="cos(t)")
plt.plot(t, tant, linestyle="dashdot", label="tan(t)", color="mediumslateblue")

# the functions below all modify the axes and plots created above
plt.legend(loc="upper left")
plt.xlim(-np.pi, 3*np.pi)
plt.ylim(-1.5, 2.5)
plt.xlabel("t")
plt.ylabel("cos and tan of t")
plt.title("Trig Stuff")
```

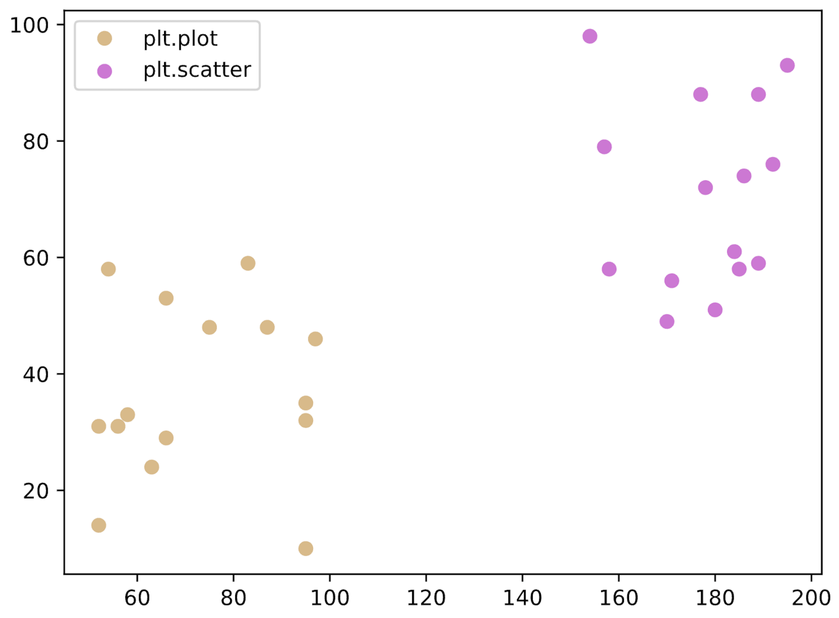
```
# this makes the plot show up!
plt.show()
```



Another useful function is `plt.scatter`. This creates a scatter plot similar to when you set `linestyle="None"` in a regular `plt.plot` function call. See below for an example.

Additional note: when using `linestyle="None"` with `plt.plot`, you must also specify a `marker` shape. Without this, you might not see anything at all when you create your plot.

```
# we can use np.random to generate some numbers for plotting
x1 = np.random.randint(50, 100, 15)
y1 = np.random.randint(10, 60, 15)

x2 = np.random.randint(150, 200, 15)
y2 = np.random.randint(40, 100, 15)

# creating a scatter plot with plt.plot
plt.plot(x1, y1, linestyle="None", marker="o", color="burlywood", label="plt.plot")

# creating a scatter plot with plt.scatter -- by default the points are shaped like
marker="o"
plt.scatter(x2, y2, color="orchid", label="plt.scatter")

plt.legend()
plt.show()
```


plt.plot
plt.scatter
100
80
60
40
20
60
80
100
120
140
160
180
200