COP 3035

Intro Programming in Python

Summer 2024

Lecture 20 – part 1

Lab 10 (Optional)
Homework 7 – 07/29/24
Exam 4 – 08/02/24

Lecture 20 – part 2

Review

Review

Object Oriented Programming

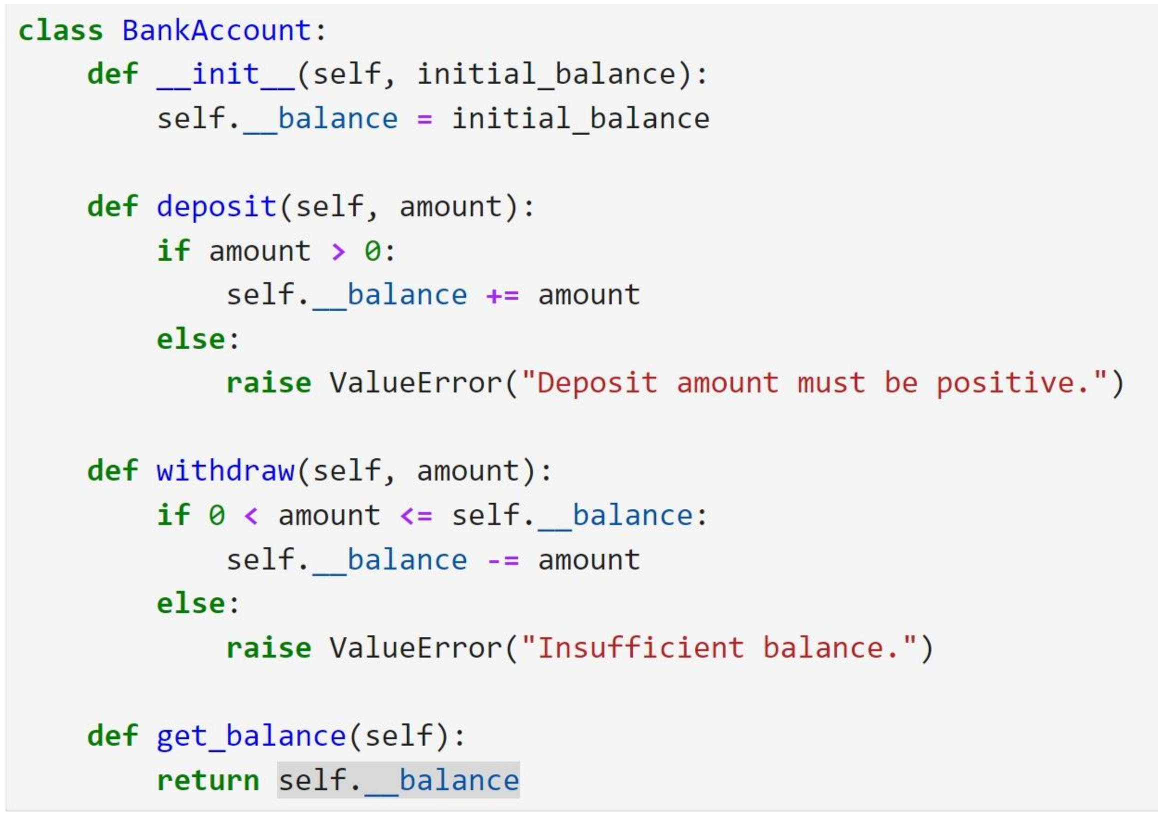
Encapsulation

Integration Exercise

Encapsulation

- In encapsulation, the variables of a class are hidden from other classes and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.
- It promotes more secure code. It ensure data is not changed in unexpected ways.
- Python does not have strict enforcement of access modifiers like private or protected as in other languages. The convention is respected by users and enforced by the Python interpreter.

- How? - **Prefix attributes or methods with a double underscore __ to make them private.**



```python
class BankAccount:
    def __init__(self, initial_balance):
        self.__balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
        else:
            raise ValueError("Deposit amount must be positive.")

    def withdraw(self, amount):
        if 0 < amount <= self.__balance:
            self.__balance -= amount
        else:
            raise ValueError("Insufficient balance.")

    def get_balance(self):
        return self.__balance
```

Note: it merely obfuscates their names to discourage direct access (name mangling).

ac1._BankAccount__balance = 2000000

Simplified social media model

1. Base User Class: Define **`User`** with private **`username`** and **`email`**, a class variable **`total_users`**, methods for username access, and a static method for email validation.
2. User Class Extension: Create **`PersonalAccount`** and **`BusinessAccount`** from **`User`**, adding specific attributes (**`birth_date`** for personal and **`business_name`** for business) and polymorphically overriding the post method.
3. Post Classes: Implement a general **`Post`** class with **`content`**, **`author`**, and **`likes`**. Derive **`PersonalPost`** and **`BusinessPost`** for specific post types, adding **`privacy_level`** and **`category`**, respectively.
4. Feed Class: Develop a **`Feed`** class to collect and display posts.
5. Integration and Testing: Instantiate personal and business accounts, create posts, add to feed, and display, ensuring all components integrate well.

Lecture 20 – part 3

Python Modules

Python Modules

- A **module** is a file containing Python definitions and statements.
- The file name is the module name with the suffix **.py** added.
- Modules are used to organize code logically by grouping related functions, classes, and variables. This makes the code easier to understand and use.
- Modules provide their own namespaces, which helps avoid naming conflicts between identifiers.

`module_name.py`

Basic Import syntax:

```
import module_name
```

Selective Import syntax:

```
from module_name import function_name
```

Alias Import syntax:

```
import module_name as mn

from module_name import function_name as fn
```

Creating Your Own Modules

- Simply save your code in a **`.py`** file.
- This file can then be imported into other Python scripts.
- Use the dot notation (**`module_name.function_name`**) to access functions and variables defined in the module.
- Use docstrings **`(””” text ””” `**) to document the module, classes and functions.

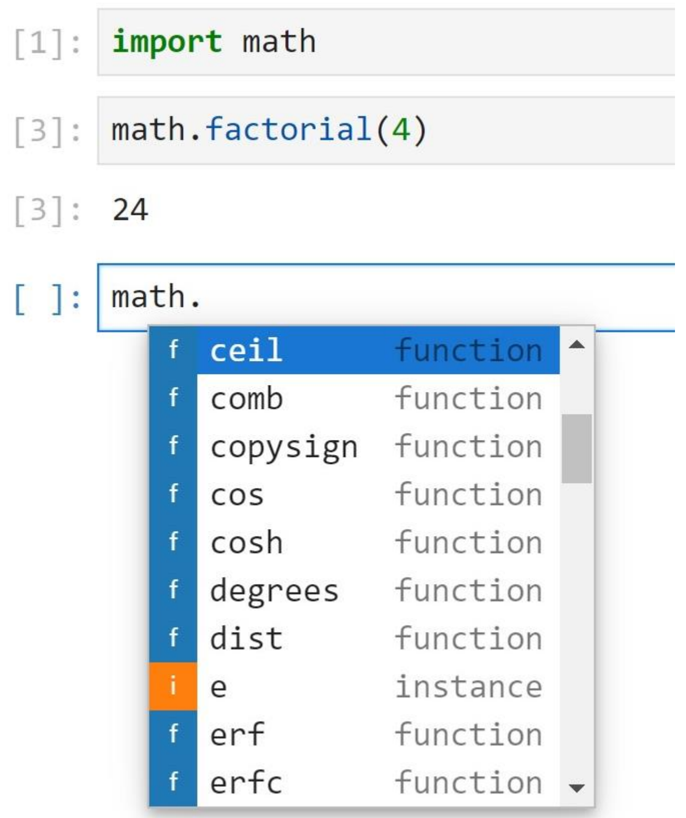
The Python Standard Library

https://docs.python.org/3/library/index.html

Python comes with a rich standard library, which is a collection of modules that provides access to system functionality and standardized solutions.

Module	Description
os	Offers functions to interact with the operating system, such as file and directory operations, executing commands, others.
sys	Provides access to some variables and functions that interact with the Python interpreter, allowing manipulation of the runtime environment.
datetime	For manipulating dates and times, calculating differences, and formatting.
math	Mathematical functions, including trigonometric, logarithmic, and more.
random	Used for generating pseudo-random numbers for various distributions and choosing randomly from sequences.
json	Supports encoding and decoding JSON data, crucial for web data interchange and configuration files.
re	Supports regular expressions for advanced string manipulation and pattern matching.

Example:



```
[1]: import math

[3]: math.factorial(4)

[3]: 24

[ ]: math.
```

Lecture 20 – part 4

Python Packages

Python packages

A <u>module</u> is a <u>single file</u> containing Python code, whereas a <u>package</u> is a <u>collection of modules</u> that are organized in a directory hierarchy.

Parameter	**Module**	**Package**
Definition	It can be a simple Python file (.py extension) that contains collections of functions and global variables.	A package is a collection of different modules with an __init__.py file.
Purpose	Code organization	Code distribution and reuse
Organization	Code within a single file	Related modules in a directory hierarchy
Sub-modules	None	Multiple sub-modules and sub-packages
Required Files	Only Python file (.py format)	__init__.py file and additional Python files
How to Import	import module_name	import package_name.module_name or from package_name import module_name
Example	math, random, os, datetime, csv	Numpy, Pandas, Matplotlib, django

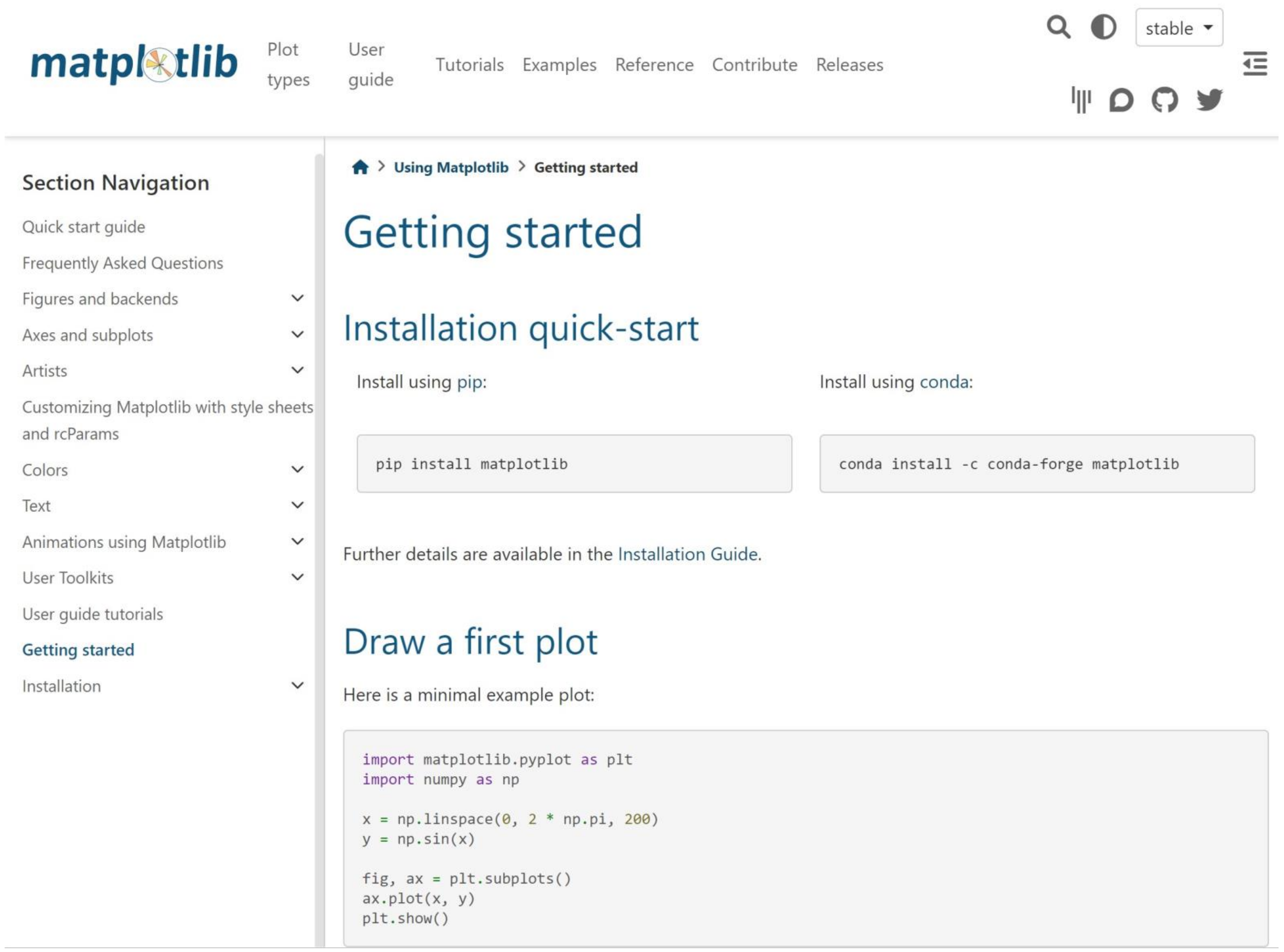
Lecture 20 – part 5

Matplotlib

Matplotlib:
A comprehensive library for creating static, animated, and interactive visualizations in Python.

Installation:
```
pip install matplotlib
```

Importing:
```
import matplotlib.pyplot as plt
```



Getting started

Installation quick-start

Install using pip:

```
pip install matplotlib
```

Install using conda:

```
conda install -c conda-forge matplotlib
```

Further details are available in the Installation Guide.

Draw a first plot

Here is a minimal example plot:

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

x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```

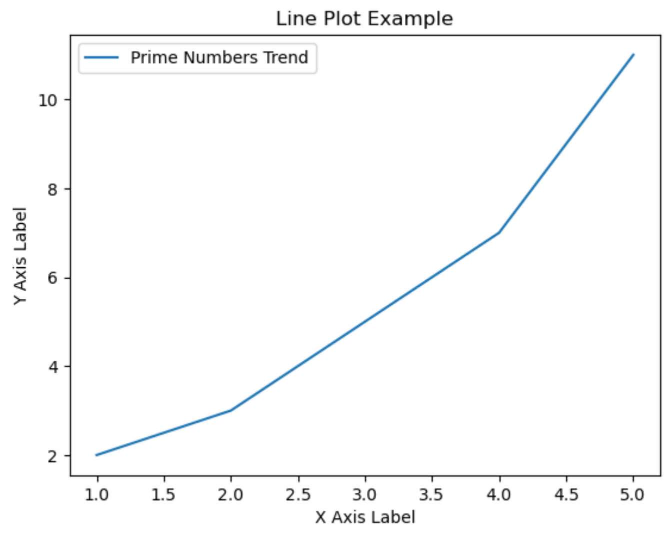
https://matplotlib.org/stable/users/getting_started/

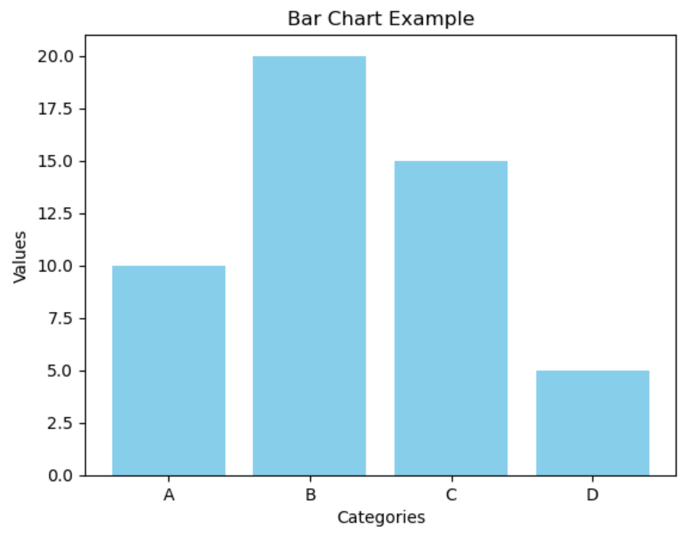
Basic plotting

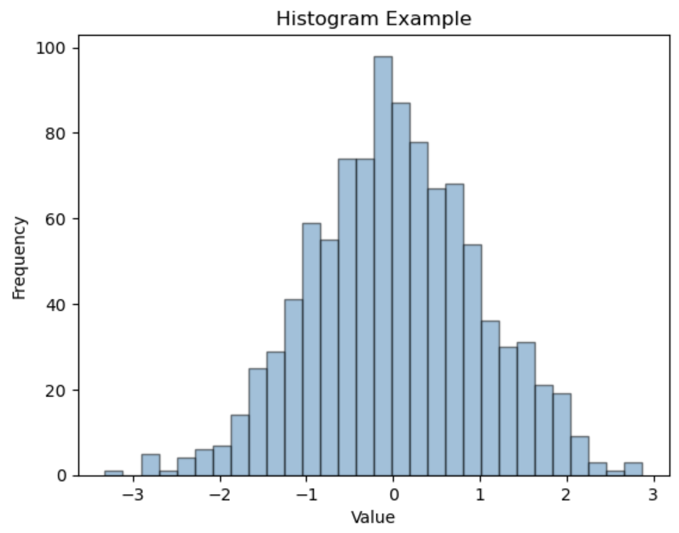
Line Plot : Basic syntax and customization (color, linestyle, marker).

Bar Chart: Comparing data side-by-side.

Histogram: Visualizing distributions.







https://matplotlib.org/stable/users/explain/quick_start.html

matplotlib.pyplot.plot()

The matplotlib.pyplot.plot() function is quite flexible, and its syntax can vary depending on how you want to customize your plot.

Syntax:

```
plot([x], y, [fmt], *, data=None, **kwargs)
plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

x, y: These are arrays or sequences of values. x is optional; if not provided, the default will be range(len(y)).

fmt: A format string, optional, that specifies color, marker, and line type in a shorthand form. For example, **'ro-'** means red circles connected by lines.

NOTE: The ****kwargs** parameter lets you pass in numerous other options to customize markers, lines, and more.

Common Parameters

data:	An optional parameter that allows specifying the data source (Dictionary).
color:	Specifies the color of the line. You can use named colors, hex codes, or RGB/A tuples.
label:	Sets the label for this line, which will appear in the legend.
linewidth or **lw**:	Sets the width of the line.
markersize or **ms**:	Determines the size of the markers.
linestyle or **ls**:	Defines the style of the line, such as solid, dashed, or none.
marker:	Chooses the marker style for the points, like circles, squares, etc.

```
plt.plot(y)     # Assumes x as range(len(y))
plt.plot(x, y)  # Plots y vs x
plt.plot(x, y, 'bo-')  # Blue circles with solid lines
plt.plot(x, y, color='green', marker='o', linestyle='dashed’,
         linewidth=2, markersize=12)
```

Line Styles

Symbol	Name
-	Solid
--	Dashed
-.	Dash-dot
:	Dotted
None	No line
''	No line

Markers

Symbol	Name
.	Point
,	Pixel
o	Circle
v	Triangle Down
^	Triangle Up
<	Triangle Left
>	Triangle Right
1	Tri Down
2	Tri Up
3	Tri Left
4	Tri Right

Symbol	Name
8	Octagon
s	Square
p	Pentagon
*	Star
h	Hexagon1
H	Hexagon2
+	Plus
x	X
D	Diamond
d	Thin Diamond
`	`
_	Hline

Color

```
color = ‘cyan’
color=‘c’
color=‘#00FFFF’
color=(0,1,1)
```

Color	Short Name	Hex Code	RGB Tuple (*)
Black	k	#000000	(0, 0, 0)
White	w	#FFFFFF	(255, 255, 255)
Red	r	#FF0000	(255, 0, 0)
Green	g	#008000	(0, 128, 0)
Blue	b	#0000FF	(0, 0, 255)
Cyan	c	#00FFFF	(0, 255, 255)
Magenta	m	#FF00FF	(255, 0, 255)
Yellow	y	#FFFF00	(255, 255, 0)
Light Blue	-	#ADD8E6	(173, 216, 230)
Orange	-	#FFA500	(255, 165, 0)
Purple	-	#800080	(128, 0, 128)
Brown	-	#A52A2A	(165, 42, 42)
Pink	-	#FFC0CB	(255, 192, 203)
Gray	-	#808080	(128, 128, 128)
Lime	-	#00FF00	(0, 255, 0)

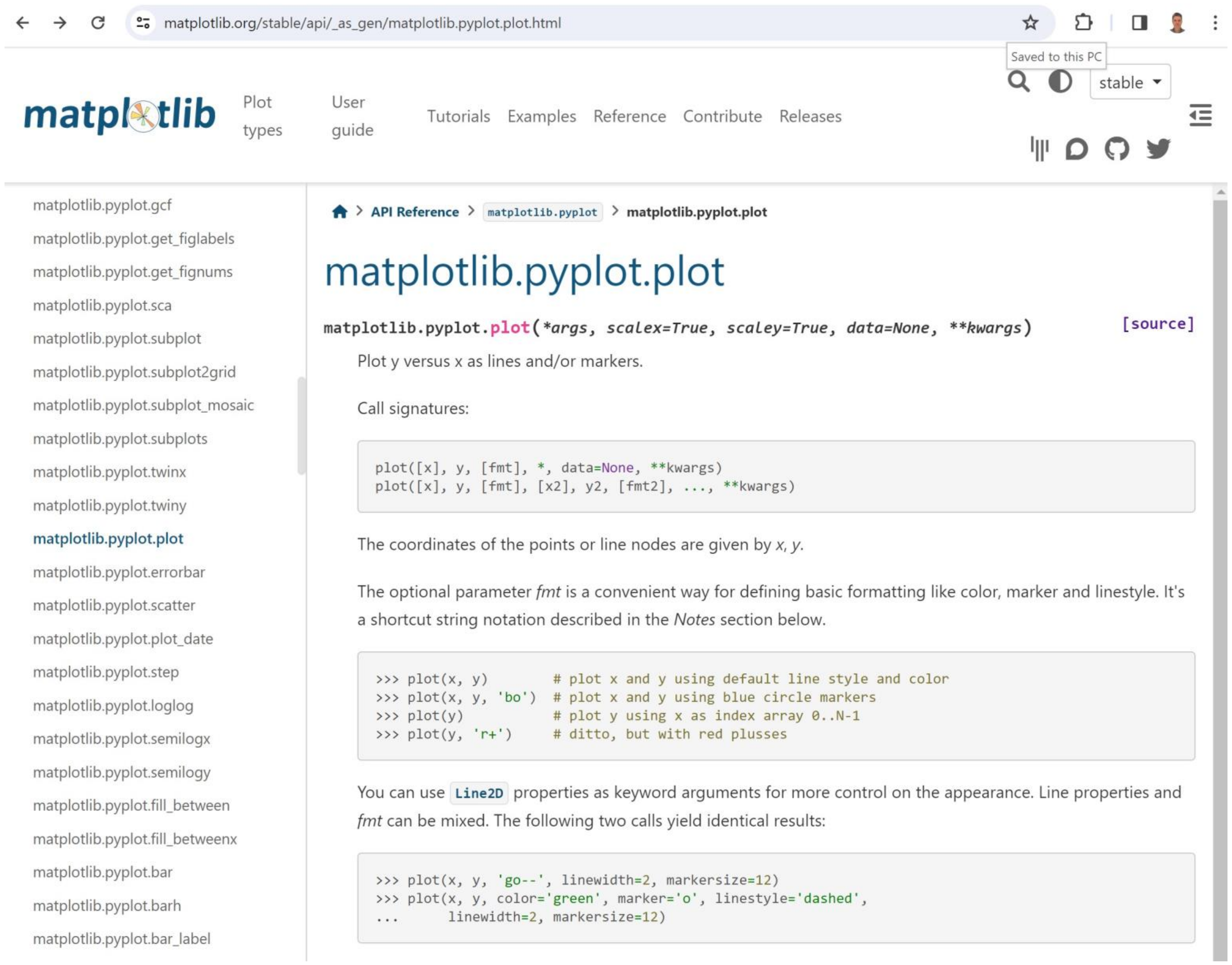
* Note: You need to convert values to [0,1] by dividing by 255.

Example using the data parameter:

```
import matplotlib.pyplot as plt

# Sample data as a dictionary
data = {
    'x': range(1, 11),
    'y1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'y2': [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
}

# Plotting with explicit formatting
plt.plot('x', 'y1', data=data, marker='o', color='red', linestyle='-', label='Ascending')
plt.plot('x', 'y2', data=data, marker='^', color='blue', linestyle='--', label='Descending')

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

API Reference:

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html


matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html
Saved to this PC
stable
matplotlib
Plot types
User guide
Tutorials
Examples
Reference
Contribute
Releases
matplotlib.pyplot.gcf
matplotlib.pyplot.get_figlabels
matplotlib.pyplot.get_fignums
matplotlib.pyplot.sca
matplotlib.pyplot.subplot
matplotlib.pyplot.subplot2grid
matplotlib.pyplot.subplot_mosaic
matplotlib.pyplot.subplots
matplotlib.pyplot.twinx
matplotlib.pyplot.twiny
matplotlib.pyplot.plot
matplotlib.pyplot.errorbar
matplotlib.pyplot.scatter
matplotlib.pyplot.plot_date
matplotlib.pyplot.step
matplotlib.pyplot.loglog
matplotlib.pyplot.semilogx
matplotlib.pyplot.semilogy
matplotlib.pyplot.fill_between
matplotlib.pyplot.fill_betweenx
matplotlib.pyplot.bar
matplotlib.pyplot.barh
matplotlib.pyplot.bar_label
API Reference > matplotlib.pyplot > matplotlib.pyplot.plot

matplotlib.pyplot.plot

`matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)` [source]

Plot y versus x as lines and/or markers.

Call signatures:

```
plot([x], y, [fmt], *, data=None, **kwargs)
plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

The coordinates of the points or line nodes are given by *x*, *y*.

The optional parameter *fmt* is a convenient way for defining basic formatting like color, marker and linestyle. It's a shortcut string notation described in the *Notes* section below.

```
>>> plot(x, y)        # plot x and y using default line style and color
>>> plot(x, y, 'bo')  # plot x and y using blue circle markers
>>> plot(y)           # plot y using x as index array 0..N-1
>>> plot(y, 'r+')     # ditto, but with red plusses
```

You can use `Line2D` properties as keyword arguments for more control on the appearance. Line properties and *fmt* can be mixed. The following two calls yield identical results:

```
>>> plot(x, y, 'go--', linewidth=2, markersize=12)
>>> plot(x, y, color='green', marker='o', linestyle='dashed',
...      linewidth=2, markersize=12)
```

Exercises

1. Draw a plot of a simple list [1,2,3,4,5]. Then add a title and a grid.
2. Draw a plot using two lists X and Y. Add title, grid, axis titles and a line label.
3. Draw three plots for $y_1=x$, $y_2=x^2$ and $y_3=x^3$, for the first 50 numbers. Use different colors and markers.
4. Generate a line plot with three different mathematical functions:

   y1 = sin(x) ,

   y2 = cos(x) and

   y3 = 2sin(x)cos(x),

   for x ranging from 0 to 2π.

   Use different line styles and colors for each function. Include a title and legends. Label the axis.

5. Complete the tutorial for matplotlib.pyplot at:
https://matplotlib.org/stable/tutorials/pyplot.html#sphx-glr-tutorials-pyplot-py