Class 2_all

July 23, 2025

1 Python as a calculator

[ ]:
```
2+1
```

[ ]:
```
2**3    # exponentiation
```

[ ]:
```
3/10
```

1.1 Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

Ask Gemini: python variable naming rules and conventions.

[ ]:
```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string

print(counter)
print(miles)
print(name)
```

[ ]:
```
a = 1
b = 2
```

[ ]:
```
# Arithmetic with variables
print(a + b)
print(a + 10)
print(a/b)
```

[ ]:
```
# create variable c
c = a + b
print(c)
```

1.2 Class Exercise

Why does this code not work?

[ ]:
```python
my_variable = 10
print(my_varable)
```

2 Standard Data Types

2.1 Basic data types in Python

We have seen two Python data types:

- `int`, or integer: a number without a fractional part. e.g. 1,2,100…
- `float`, or floating point: a number that has both an integer and fractional part, separated by a point, e.g. 1.2, 1.3, 28.9…
- `str`, or string: a type to represent text. You can use single or double quotes to build a string, e.g. `'Lucy'`, `'Seattle'`.
- `bool`, or boolean: a type to represent logical values. Can only be `True`, or `False`.

2.2 Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]).

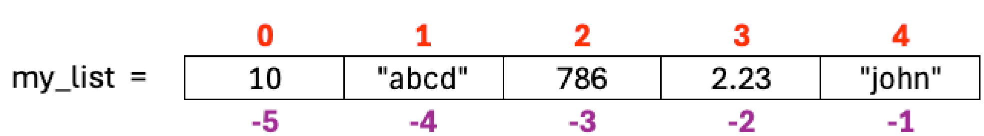
The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

[ ]:
```python
# A list is a compound data type, you can group values together
a = 10
my_list = [ a, 'abcd', 786 , 2.23, 'john']
print(my_list)
```

[ ]:
```python
tinylist = [ 15, 3.34 ]

print(tinylist * 2)        # Prints list two times
print(my_list + tinylist) # Prints concatenated lists
```

2.2.1 Indexing and Slicing

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.



[ ]:
```python
print(my_list)          # Prints complete list
print(my_list[0])       # Prints first element of the list
print(my_list[1:3])     # Prints elements starting from 2nd till 3rd
print(my_list[2:])      # Prints elements starting from 3rd element
print(my_list[2:-1])
print(my_list[-1])      # Prints the last element
print(my_list[-3:-1])
```

[ ]:
```python
# Variables can also be used as indexing
myStart = 2
print(my_list[myStart])

# get the length of a list
len(my_list)
```

2.2.2 Practice

A digital marketing manager tracked their daily ad spend for a campaign over one week. The list ad_spend contains alternating entries of a day of the week and the dollar amount spent on ads for that day.

`ad_spend = ["Monday", 120.0, "Tuesday", 150.5, "Wednesday", 135.0, "Thursday", 160.0, "Friday", 175.5, "Saturday", 190.0, "Sunday", 145.0]`

- Use slicing to create a list, `weekday_spend`, that contains the ad spend data for Monday to Friday.
- Use slicing to create a list, `weekend_spend`, that contains the ad spend data for Saturday and Sunday.
- Print both `weekday_spend` and `weekend_spend`.

Ask Gemini: Can you give me more examples and explainations of python indexing and slicing?

2.2.3 List Manipulation

Replacing list elements is pretty easy. Simply subset the list and assign new values to the subset. You can select single elements or you can change entire list slices at once.

[ ]:
```python
x = ['a', 'b', 'c', 'd']
x[1] = 'r'
x[2:] = ['s', 't']
print(x)
```

Extend a list with the `+` operator.

[ ]:
```python
x = ['a', 'b', 'c', 'd']
y = x + ['e', 'f']
print(y)
```

3 Packages

For example, you may want to calculate the area of a circle. When the radius is `r`, the area is $A = \pi r^2$.

To use the constant `pi`, you'll need the `math` package.

[ ]:
```
# radius
r = 0.3
import math      # Import the math package.
                 # Now you can access the constant pi with math.pi
A = math.pi *r *r
print(A)
```

3.1 Selective import

General imports, like `import` math, make all functionality from the `math` package available to you. However, if you decide to only use a specific part of a package, you can always make your import more selective.

[ ]:
```
from math import pi
A = pi *r *r
print(A)
```

3.2 Practice

Selectively import the square root function `sqrt` from `math` package. Then compute the square root of 10.

4 Numpy

`NumPy` (Numeric Python) is the fundamental package for data science and scientific computing with Python.

[ ]:
```
# the list length represents the length of some rectangles
# the list width represents the width of some rectangles
length = [2, 4, 6, 8]
width = [1.4, 3.2, 1, 6]

# we want to calculate area of the rectangles
length * width
# or double the length of every rectangle
length * 2
```

Numpy array is an alternative to Python list. It makes calculations over the entire arrays.

[ ]:
```
import numpy as np   # you can now refer to numpy as np
# use np.array() to create a numpy array from length.
np_length = np.array(length) # np.array is a function to do the type conversion
```

```
np_length
```

[ ]:
```
# Math on numpy arrays
print(np_length+1)
print(np_length*2)
print(np_length + np_length)

np_width = np.array(width)
print(np_length * np_width)
```

4.1 Practice

You're analyzing the performance of a digital marketing campaign across 5 different countries. Your goal is to calculate the **Return on Investment (ROI)** for each country.

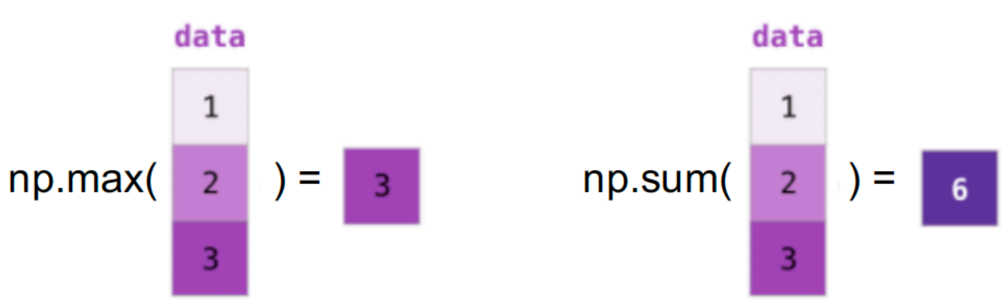
- You ran ads in 5 countries. The **ad spend** (in USD) is: `ad_spend = [1200, 950, 1600, 870, 1320]`. Let's first convert this list into a NumPy array called `spend_usd`.
- Each country generated revenue in its local currency: `revenue_local = [3000, 2200, 4100, 1900, 3400]`.
- The excange rate to USD for each country is `exchange_rates = [1.0, 0.85, 1.1, 0.9, 1.2]  # Local → USD`
- Convert `revenue_local` into USD by multiplying it with `exchange_rates` element-wise. Store the result in a new array called `revenue_usd`.
- Calculate ROI for each country:

$$ROI = \frac{Revenue(USD) - Spend(USD)}{Spend(USD)}$$

- Store the result as a NumPy array called `roi` and print it out.

4.2 Aggregation

[ ]:
```
width = [1.4, 3.2, 1, 6]
print( np.sum(width) )
print( np.mean(width) )
print( np.max(width) )
```



4.3 Numpy array indexing and slicing

To subset both regular Python lists and numpy arrays, you can use square brackets:

[ ]:
```
x = [4 , 9 , 6, 3, 1]
print(x[1])
y = np.array(x)
print(y[1])
print(y[1:3])
print(y[3:])
print(y[-2])
```

For numpy only, you can also use logic values (True, False).

[ ]:
```
my_index = np.array([True, True, False, False, True])
y[my_index]
print(y[my_index])
```

This is most often used with comparative operators to select items:

[ ]:
```
high = y > 5
y[high]
```

Logical operators in Python:

Operator	Description
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to

[ ]:
```
y == 1
y != 1
```

For numpy only, you can also use a list (or numpy array) of integers to select positions.

[ ]:
```
print(y)                # print the entire numpy array
sub_y = y[[0,4,2]]      # get a subset of the numpy array
                        # and assign it to sub_y
print(sub_y)            # print the subset
```

```
# compare with list
print(x)
#t = x[[0,4,2]]          # wrong syntex
```

4.4 Comparing Numpy with basic Python lists

Numpy is great for doing vector arithmetic. If you compare its functionality with regular Python lists, however, some things have changed.

- Numpy arrays cannot contain elements with different types. If you try to build such a list, some of the elements' types are changed to end up with a homogeneous list.
- The typical arithmetic operators, such as + and * have a different meaning for regular Python lists and numpy arrays.
- Element-wise Operations (NumPy only).
- Boolean and subset Indexing (NumPy only).

4.5 2D numpy arrays

Now let's construct our first 2D array. Suppose you're a sales analyst at a company. You have quarterly sales data (in $1,000s) for 2 products (A, B) across 4 regions (North, South, East, West).

[ ]:
```
product_A = [250, 300, 280, 320]
product_B = [180, 200, 190, 210]

# Rows: Products A, B
# Columns: Regions North, South, East, West

list_2d = [[250, 300, 280, 320], [180, 200, 190, 210]]
np_2d = np.array(list_2d)
np_2d
```

0	1	2	3	
250	300	280	320	0
180	200	190	210	1

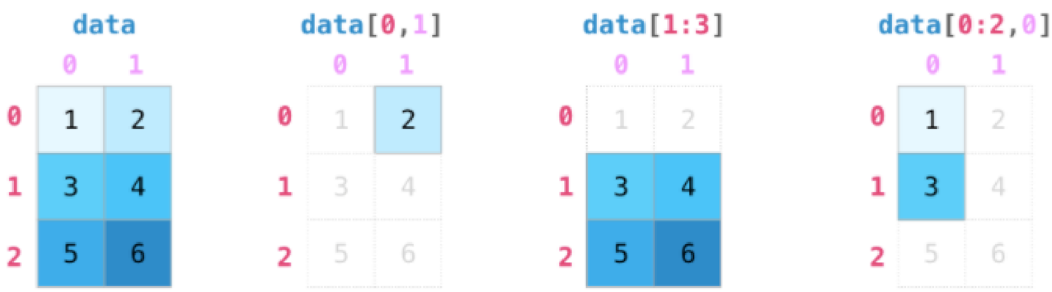
[ ]:
```
np_2d.shape    #2 rows and 4 columns
```

4.5.1 Indexing and Slicing

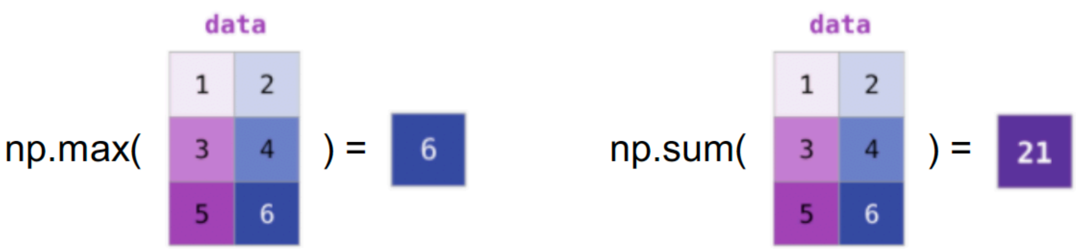
```
[ ]: np_2d[0,2]
```

```
[ ]: # Select the height and weight of the third and fourth player
     np_2d[:,1:3]    # : stands for all the rows
     # product A of the second and third region
     np_2d[0,1:3]
```

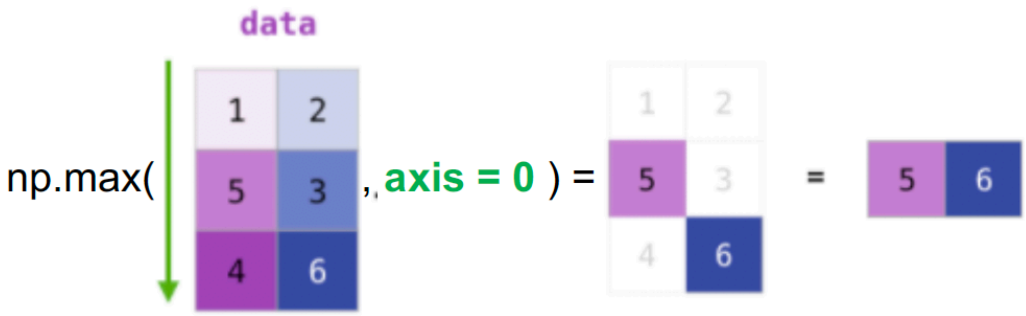
```
[ ]: # If you only want to retrieve the sales for product A
     np_2d[0,:] # it is equivalent to np_2d[0] which omits the column index
     np_2d[0, [0,2,3] ]
```

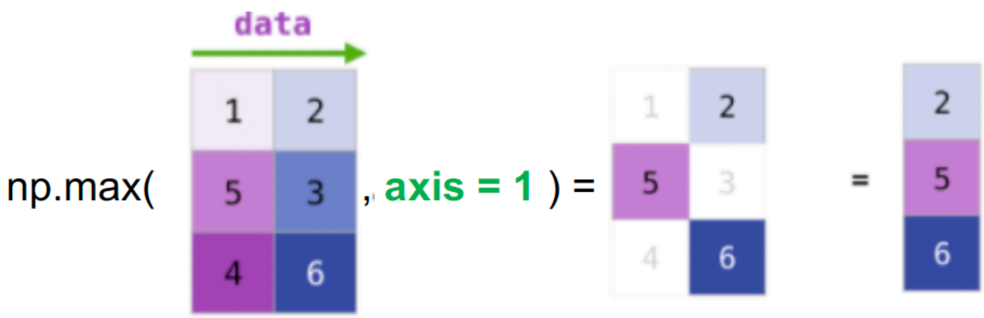


4.5.2 Aggregation



Not only can we aggregate all the values in a 2d numpy array, but we can also aggregate across the rows or columns by using the axis parameter:





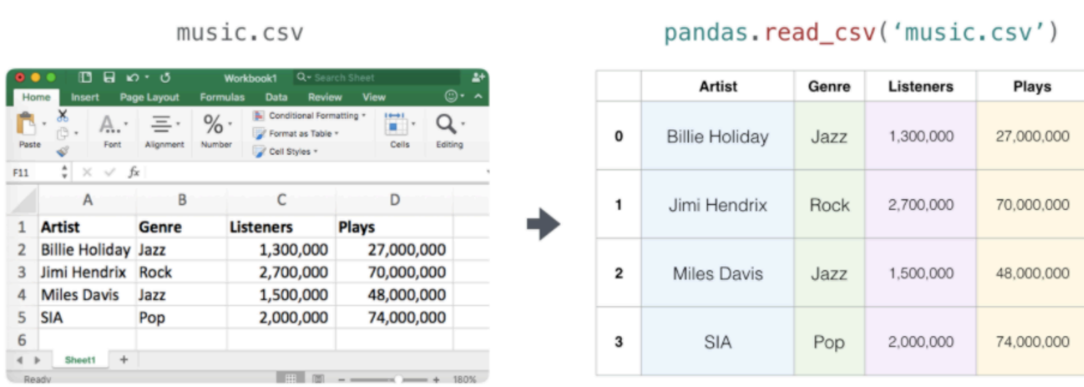
5 Data Representation

Think of all the data types you'll need to crunch and build models around (spreadsheets, images, audio, text...). So many of them are perfectly suited for representation in an n-dimensional array.

5.1 Tables and Spreadsheets

Think of the previous Iris data representation or the data representation of basketball players. It would be wonderful if we can refer to a column by its name.

A spreadsheet or a table of values is a two dimensional matrix. The most popular abstraction in python for those is the pandas dataframe, which actually uses NumPy and builds on top of it. If you would like to learn more about Pandas, you can start from here: https://colab.google/articles/pandas



	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

First, we need to upload the dataset to the cloud's virtual machine. The module `files` provides functions for interacting with the Colab file system. Then, it calls the files.upload() function. This function opens a file upload dialog in the Colab notebook, allowing you to select files from your local machine to upload. The uploaded files are stored in the Colab environment's virtual machine. **OR** you can drag/upload the csv file to the file folder on google colab.

[ ]:
```python
from google.colab import files
uploaded = files.upload()
```

Pandas allows us to load a spreadsheet and manipulate it programmatically in python. The central concept in pandas is the type of object called a DataFrame – basically a table of values which has a label for each row and column. Let's load this basic CSV file containing data from music.csv.

[1]:
```python
import pandas as pd
df = pd.read_csv('music.csv')
df # Now the variable df is a pandas DataFrame
```

[1]:
```
           Artist Genre  Listeners       Plays
0  Billie Holiday  Jazz  1,300,000  27,000,000
1    Jimi Hendrix  Rock  2,700,000  70,000,000
2     Miles Davis  Jazz  1,500,000  48,000,000
3             SIA   Pop  2,000,000  74,000,000
```

`DataFrame.to_numpy()` gives a NumPy representation of the underlying data. Note that this can be an expensive operation when your DataFrame has columns with different data types, which comes down to a fundamental difference between pandas and NumPy: NumPy arrays have one dtype for the entire array, while pandas DataFrames have one dtype per column.

For DataFrame of all floating-point values, `DataFrame.to_numpy()` is fast and doesn't require copying data.

5.1.1 Index and Slicing by position

Select via the position of the passed integers using `iloc`.

		0	1	2	3
		Artist	**Genre**	**Listeners**	**Plays**
0	**0**	Billie Holiday	Jazz	1,300,000	27,000,000
1	**1**	Jimi Hendrix	Rock	2,700,000	70,000,000
2	**2**	Miles Davis	Jazz	1,500,000	48,000,000
3	**3**	SIA	Pop	2,000,000	74,000,000

[ ]:
```python
df.iloc[3, :]
df.iloc[0:2, 1:3]
df.iloc[2, :3]
df.iloc[ [1,3,2] , :3]
```

5.1.2 Index and Slicing by label/name

We can also select any column or row using its row and column name, by using `loc`, but `loc` may also be used with a boolean array..

[ ]:
```
df.columns
df.loc[3,'Artist']
df.loc[1:3,'Artist']  # since now 1,2,3 are labels, *both* endpoints are included:
df.loc[2, ['Artist','Plays'] ]
df.loc[:, ['Artist','Plays'] ]

df[ ['Artist','Plays'] ]
df[ 'Artist' ]
type( df['Artist'] ) # a Series, which is a one-dimensional array-like object containing the data and label/name.

# `loc` can be used with boolean arrays to filter rows based on conditions.
df.loc[ df['Genre'] == 'Jazz' , :]
df.loc[ df['Genre'] == 'Jazz', ['Artist', 'Plays'] ]
```

5.1.3 Practice

We will use the Iris data, where each sample is one of three types of flowers that has had the size of its petals and sepals carefully measured. This is perhaps the best known database to be found in the pattern recognition and data mining literature. This dataset contains 150 different iris plants, with the information of its sepal length, sepal width, petal length and petal width.

[2]:
```
df = pd.read_csv('iris.csv')
df
```

[2]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa

```
2            4.7          3.2           1.3          0.2     Iris-setosa
3            4.6          3.1           1.5          0.2     Iris-setosa
4            5.0          3.6           1.4          0.2     Iris-setosa
..           …          …           …          …          …
145          6.7          3.0           5.2          2.3  Iris-virginica
146          6.3          2.5           5.0          1.9  Iris-virginica
147          6.5          3.0           5.2          2.0  Iris-virginica
148          6.2          3.4           5.4          2.3  Iris-virginica
149          5.9          3.0           5.1          1.8  Iris-virginica

[150 rows x 5 columns]
```

Questions:

1. Print out the 3rd iris plant.
2. Make a new variable, sepal_width, containing the entire second column of iris dataframe.
3. What is the petal length of the 95th iris plant?
4. Find all iris plants whose species are "Iris-versicolor".
5. Find out the average petal length of all versicolors using `np.mean()`.

5.2 Images

An image is a matrix of pixels of size (height × width), which can be properly represented by numpy arrays.

If the image is black and white (a.k.a. grayscale), each pixel can be represented by a single number (commonly between 0 (black) and 255 (white)). Want to crop the top left 100 x 100 pixel part of the image? Just tell NumPy to get you image[:100, :100].

If the image is colored, then each pixel is represented by three RGB numbers (ranging from 0-255). A colored image is represented by a 3d numpy array, with the dimensions: (height x width x 3).

6 Next Class: Data Visualization

Data visualization is a key part of any data science workflow. As the cliché goes, a picture is worth a thousand words.

Data exploration should really be part of your data analysis from the very beginning, as there is a lot of value and insight to be gained from just looking at your data. Summary statistics often don't tell the whole story. Furthermore, the impact of an effective visualization is difficult to match with words and will go a long way toward ensuring that your work gets the recognition it deserves.

When visualizing data, the most important factor to keep in mind is the purpose of the visualization. This is what will guide you in choosing the best plot type. It could be that you are trying to compare two quantitative variables to each other. Maybe you want to check for differences between groups. Perhaps you are interested in the way a variable is distributed. Each of these goals is best served by different plots and using the wrong one could distort your interpretation of the data or the message that you are trying to convey.

6.1 Introduction to Matplotlib

Matplotlib is the leading visualization library in Python. This tutorial is intended to help you get up-and-running with matplotlib quickly. We will go over how to create the most commonly used plots, when you would want to use each one, and highlight the parameters that you are most likely to adjust.

- Basic Plot: Line Chart, Scatter Plot
- Distribution Plot: Histogram, Boxplot

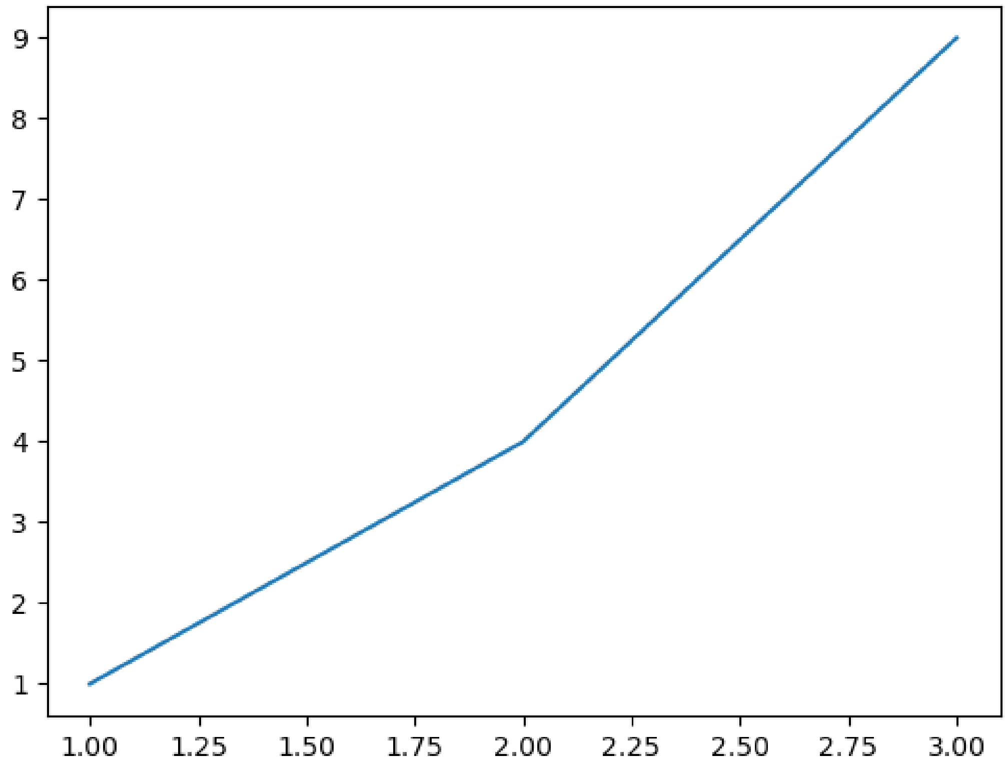
[3]:
```
from matplotlib import pyplot as plt
```

6.1.1 Simple Line Chart

Plotting with default settings

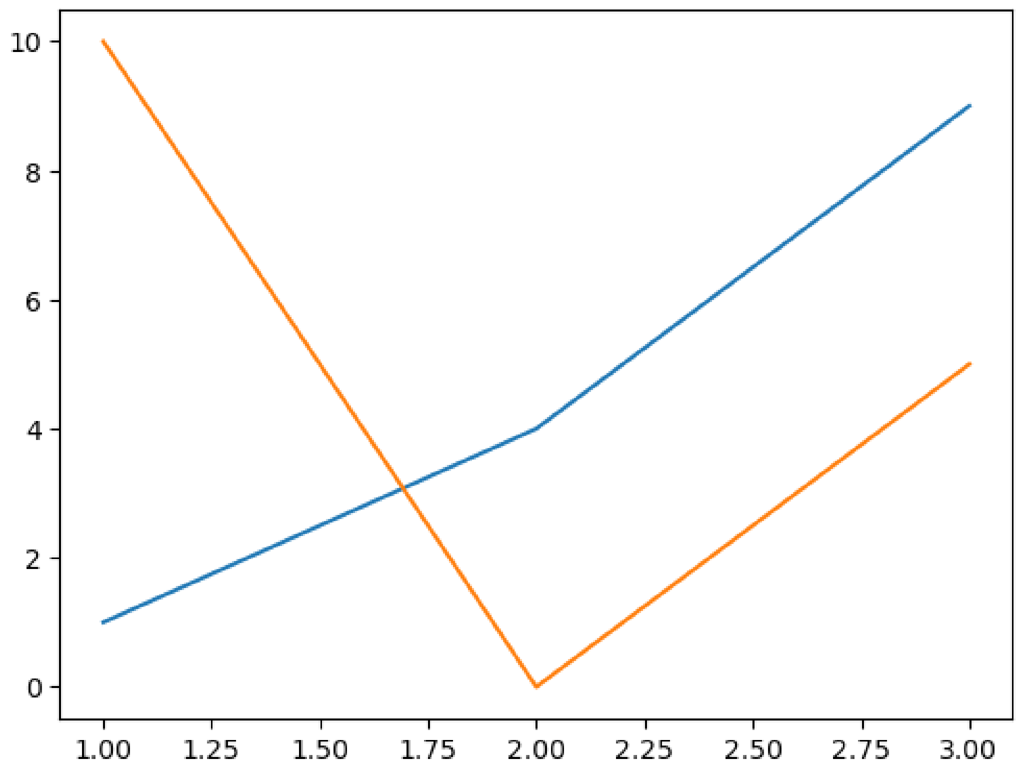
[4]:
```
x = [1, 2, 3]
y = [1, 4, 9]
```

```
plt.plot(x,y)

plt.show()
```



Plotting multiple lines on a single chart

[5]:
```
x = [1, 2, 3]
y = [1, 4, 9]
z = [10, 0, 5]

plt.plot(x,y)
plt.plot(x,z)
plt.show()
```

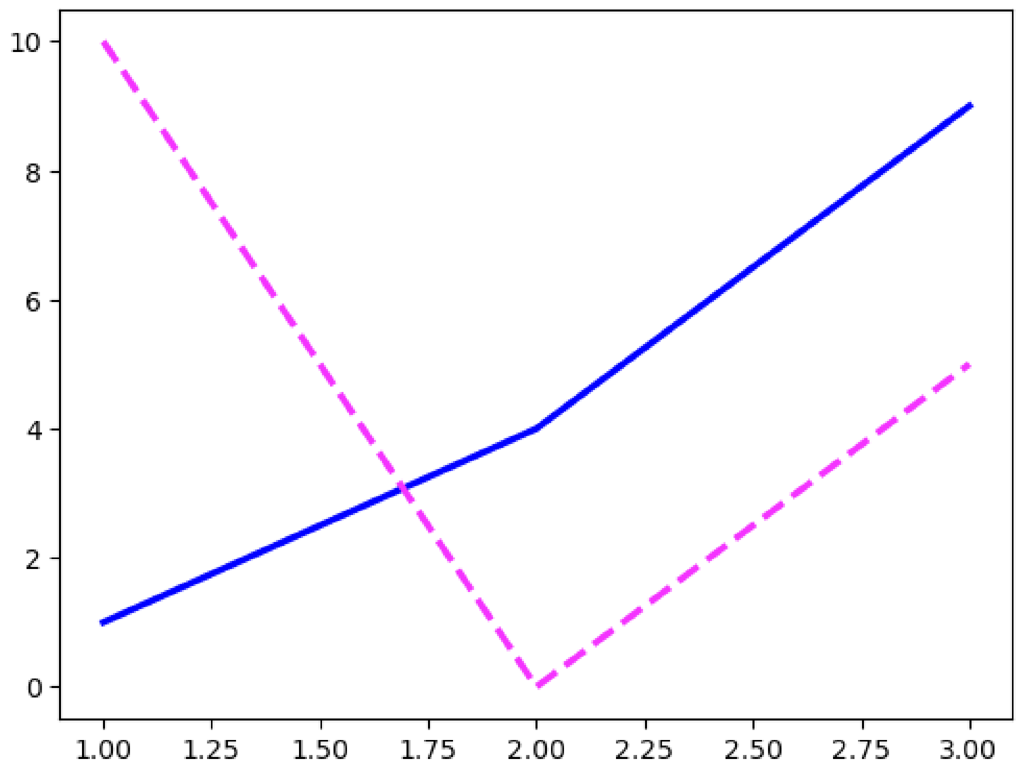


Changing colors, line widths and line styles We want to have the y in blue and the z in red and a slighty thicker line for both of them. We also want to change the line styles.

The third argument in the function call is a character that represents the type of symbols used for the plotting. The full list of available symbols can be seen in the documentation of `plt.plot`, or in Matplotlib's online **documentation**. https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

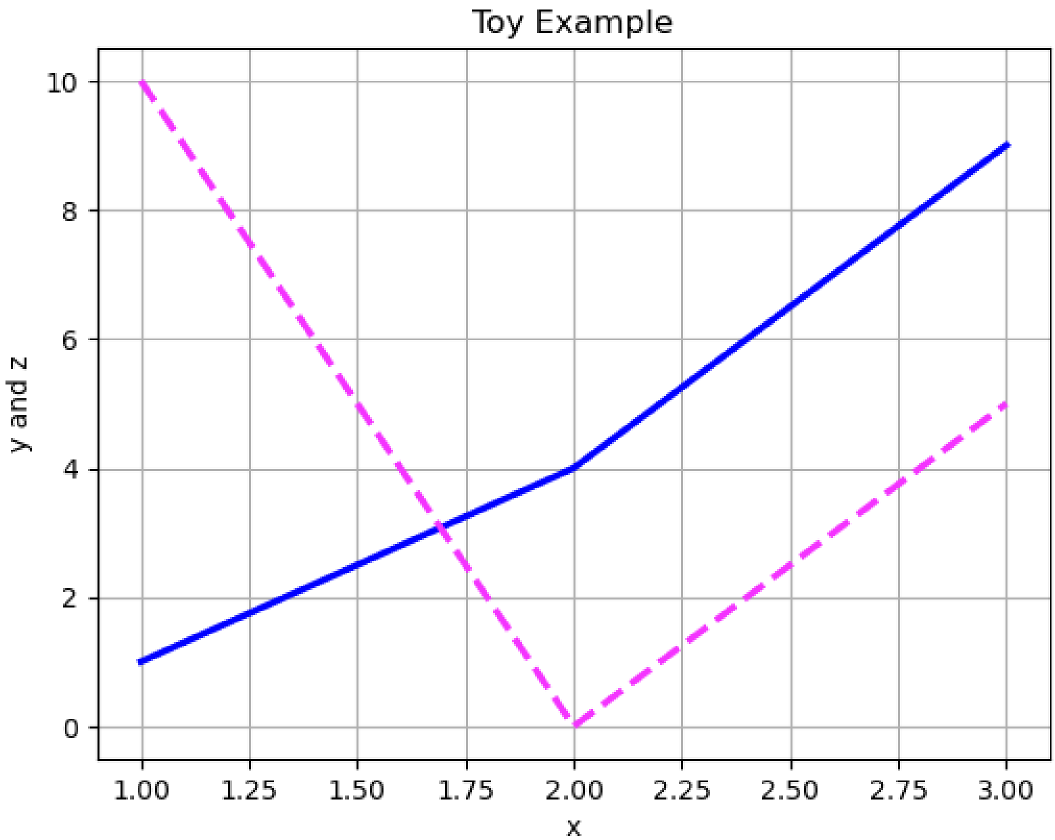
Ask Gemini:, How can I change the line color and line styles?

[7]:
```
x = [1, 2, 3]
y = [1, 4, 9]
z = [10, 0, 5]

plt.plot(x,y, "-", color = 'b', linewidth = 2.5)
plt.plot(x,z, "--", color = '#F433FF', linewidth = 2.5)
plt.show()
```



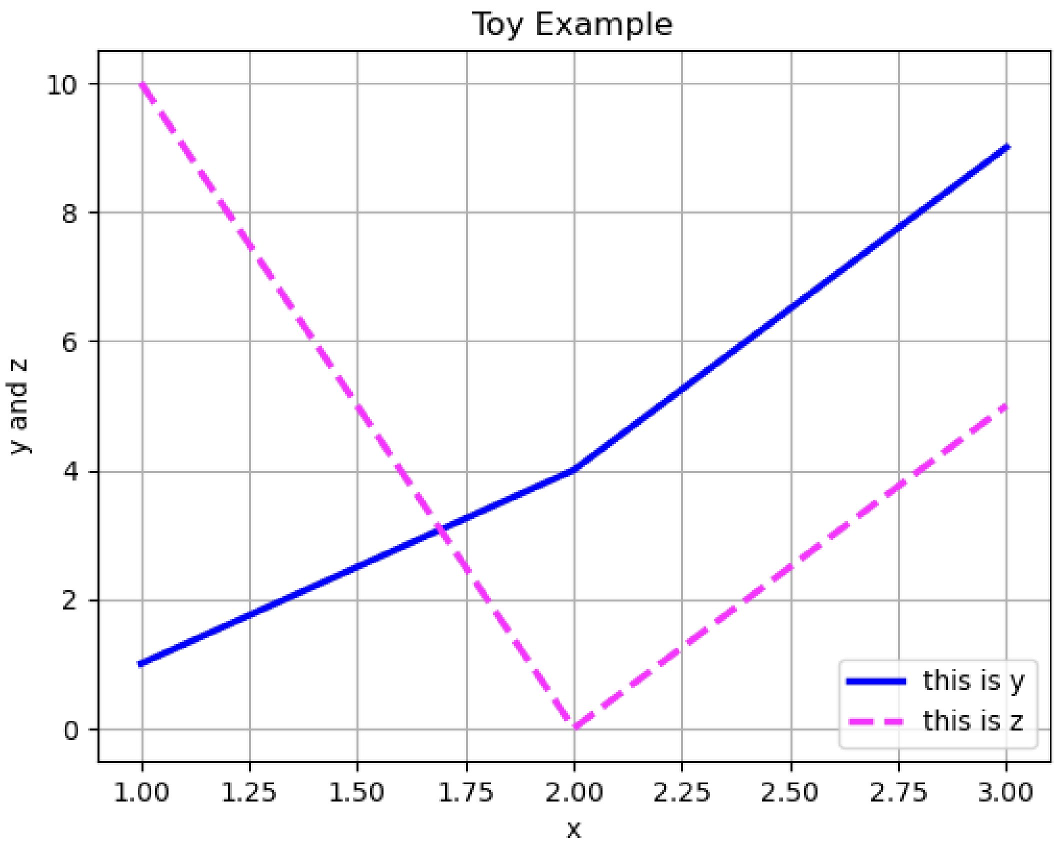
Adding titles and axis labels

[11]:
```
x = [1, 2, 3]
y = [1, 4, 9]
z = [10, 0, 5]

plt.plot(x,y, "-", color = 'b', linewidth = 2.5)
plt.plot(x,z, "--", color = '#F433FF', linewidth = 2.5)

plt.title("Toy Example")
plt.xlabel("x")
plt.ylabel("y and z")

# Optional: Add a grid
plt.grid(True)

plt.show()
```



Adding legends

[12]:
```
x = [1, 2, 3]
y = [1, 4, 9]
z = [10, 0, 5]

plt.plot(x,y, "-", color = 'b', linewidth = 2.5, label = 'this is y')
plt.plot(x,z, "--", color = '#F433FF', linewidth = 2.5, label = 'this is z')

plt.title("Toy Example")
plt.xlabel("x")
plt.ylabel("y and z")

plt.legend()

# Optional: Add a grid
plt.grid(True)

plt.show()
```



6.1.2 Scatter Plot

In many cases this is the least aggregated representation of your data. Displays relationship between two numerical variables.

The scatter() function makes a scatter plot with markers of varying size and/or color. The full list of available options can be seen in the documentation of `plt.scatter` **documentation**.

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

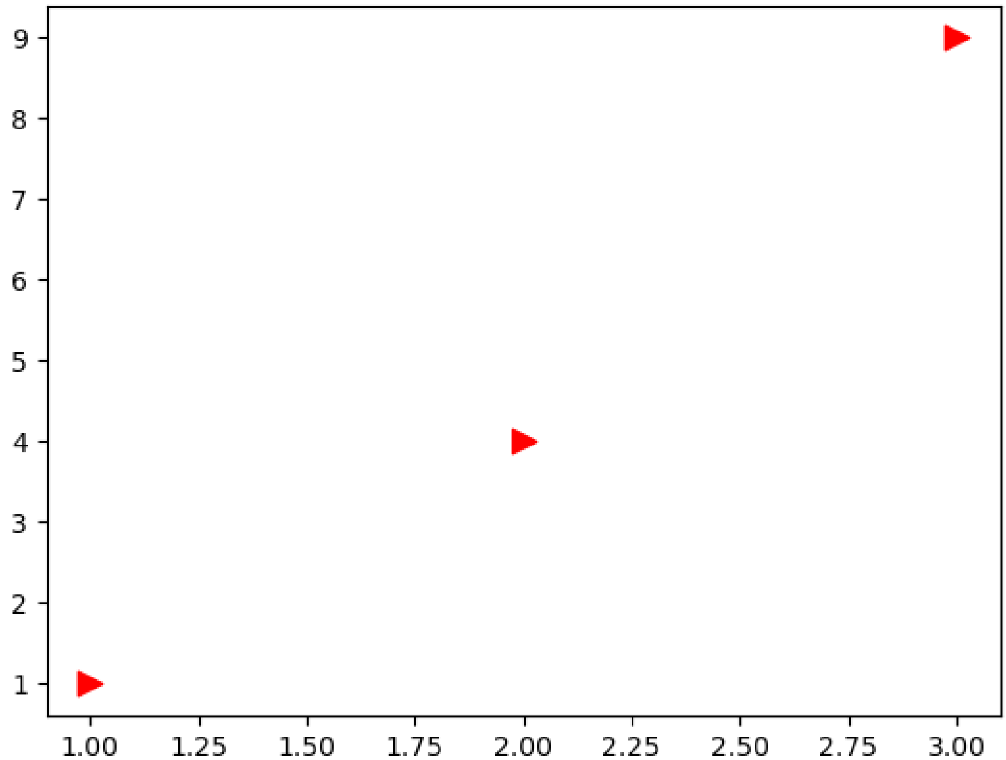
Function template:

`matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=None, edgecolors=None, hold=None, data=None, **kwargs)`

[ ]:
```
x = [1, 2, 3]
y = [1, 4, 9]


plt.scatter(x, y)     # plot with the default setting
plt.show()
```

Changing colors, marker style and marker sizes

[13]:
```
x = [1, 2, 3]
y = [1, 4, 9]

plt.scatter(x, y, s = 80, c = 'r', marker = '>')
plt.show()
```



Practice We will use the Iris data, where each sample is one of three types of flowers that has had the size of its petals and sepals carefully measured.

[14]:
```
df = pd.read_csv('iris.csv')
df
```

[14]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
..	…	…	…	…	…
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica

```
147            6.5           3.0            5.2            2.0  Iris-virginica
148            6.2           3.4            5.4            2.3  Iris-virginica
149            5.9           3.0            5.1            1.8  Iris-virginica

[150 rows x 5 columns]
```

Question:

1. We want to find out the relationship between sepal length and width by constructing a scatter plot. Put the sepal length as the x-axis and sepal width as the y-axis. Set the marker color to 'g'. Be sure to include proper x/y-axis labels.
2. What insights you can gain from this scatter plot?

[16]:
```
import seaborn as sns

# Create a scatter plot with sepal length as x-axis, sepal width as y-axis,
# and color-coded by species using Seaborn
sns.scatterplot(data=df, x="SepalLengthCm", y="SepalWidthCm", hue="Species",
                size="PetalLengthCm", sizes=(20, 400),  alpha=0.5,␣
 ↪palette='Set2') # change colormap

plt.xlabel("sepal length")
plt.ylabel("sepal width")
plt.show()
```