

# Module 5

---

ANISHA JOSEPH

VJEC

A solid orange horizontal bar spanning the width of the slide at the bottom.

# OS Module

---

## Make Directory:

```
File Edit Format Run Opt
import os
os.mkdir("Anisha")
|
```

## Get current directory and change directory

```
File Edit Format Run Options Window Help
import os
print(os.getcwd())
os.chdir("/home/vjec/Desktop")
print(os.getcwd())
|
```

```
>>>
/home/vjec
/home/vjec/Desktop
>>> |
```

# OS Module

---

## Removing Directory:

```
import os
os.rmdir("/home/vjec/Anisha")
```

## List Directories:

```
import os
print(os.listdir())
```

```
>>>
['deja-dup', 'server.class', 'abcd', 'Rational.py', '.mozilla', 'weka.log',
ul', '.ICEauthority', 'pgadmin.log', 'basic.tcl', '.gnome', 'untitled_0ods',
nupg', 'Music', 'eval.odt', '.cache', 'Client.java', '.Xauthority', 'rahul',
etbeans', 'Desktop', 'VR', 'Documents', 'test', '.thunderbird', 'nam_1.14_an
zip', '.bash_history', '1.txt', '.pki', '10CS6111 Advanced Networking lab',
m.jpeg', '.gconf', 'samba-4.5.1', '.adobe', 'simple.tcl', 'area.c', '.xsessi
rrors', '.profile', 'out.nam', 'my_project', 'examples.desktop', 'vali.odt',
ENLEX.odt', 'dhi.pdf', '.dbus', '.dmrc', 'image1.png', '.local', 'NetBeansPr
ts', 'nam_1.14_amd64.deb', 'Rahul', '__pycache__', '.rnd', '.idlerc', 'Untit
4.odt', '.pgpass', 'Videos', 'vote of thanks.odt', 'breezypythongui.py', '.g
```

# OS Module

---

Walking through directory or subdirectory

```
File Edit Format Run Options Win
```

```
import os
tree=os.walk("vjim")
for i in tree:
    print(i)
```

```
import os
tree=os.walk("vjim",topdown=False)
for i in tree:
    print(i)
```

# sys Module

---

Returns list of command line arguments,

```
import sys
print("This is the name of the program:", sys.argv[0])
print("Argument List:", str(sys.argv))
|
```

```
>>>
```

```
This is the name of the program: /home/vjec/mod5.py
Argument List: ['/home/vjec/mod5.py']
.
```

# sys Module

---

Return largest integer a variable can take,

```
# importing the module
import sys
# fetching the maximum value
max_val = sys.maxsize
print(max_val)
|
```

```
>>>
```

```
9223372036854775807
```

```
~~~ |
```

# sys Module

---

Search path for all python modules,

```
File Edit Format Run Options W
# importing the module
import sys
# fetching the maximum value
print(sys.path)
|
```

Version number of current interpreter,

```
File Edit Format Run
import sys
print(sys.version)
```

# Install NumPy

---

```
$sudo apt-get install python3-numpy
```



# Matrix operations – numpy functions

---

Matrices a and b are two arrays.

Addition:  $a+b$  or `np.add(a,b)`

Subtraction:  $a-b$  or `np.subtract(a,b)`

Multiplication:  $C = a \cdot b$  or `np.matmul(a, b)`

Transpose(): `a.transpose()`

# Sample questions

---

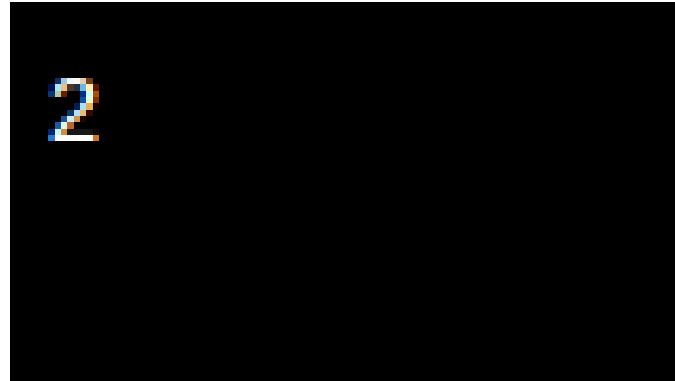
1. Write a Python program to add two matrices and also find the transpose of the resultant matrix.
2. Write a Python program to multiply two matrices.

# Numpy - Random Numbers

---

NumPy offers the random module to work with random numbers.

```
from numpy import random  
  
x = random.randint(100)  
  
print(x)
```



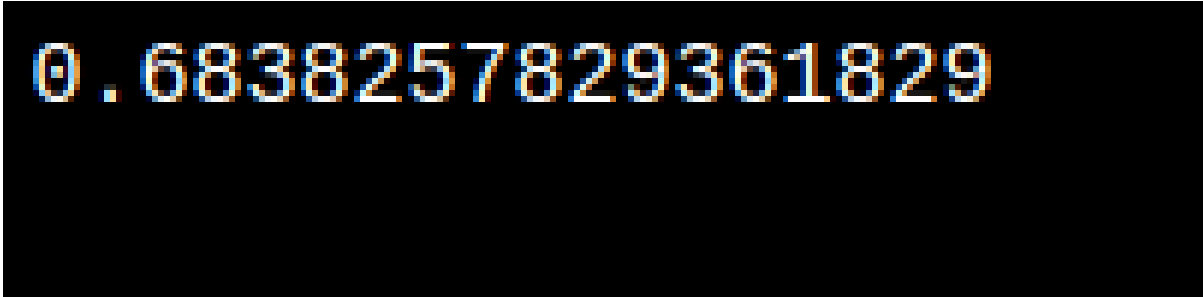
Generate a random integer from 0 to 100:

# Numpy - Random Numbers

---

The random module's **rand()** method returns a random float between 0 and 1.

```
from numpy import random  
  
x = random.rand()  
  
print(x)
```

A terminal window with a black background and yellow text displaying the output of the random number generation.

0.6838257829361829

# Numpy - Random Numbers

---

The **randint()** method takes a size parameter where you can specify the shape of an array.

```
from numpy import random  
  
x=random.randint(100, size=(5))  
  
print(x)
```

---



```
[51 6 5 30 9]
```

# Numpy - Random Numbers

---

Generate a 2-D array with 3 rows, each row containing 5 random integers from 0 to 100:

```
from numpy import random  
  
x = random.randint(100, size=(3, 5))  
  
print(x)
```

```
[[90 99 11 30 34]  
 [66 40 63 36 37]  
 [63 35 89 51 58]]
```

# Numpy - Random Numbers

---

- The `rand()` method also allows you to specify the shape of the array.
- Generate a 1-D array containing 5 random floats:

```
from numpy import random
```

```
x = random.rand(5)
```

```
print(x)
```

```
[0.8251419 0.4053331 0.8831937 0.8171717 0.7623000]
```

# Numpy - Random Numbers

---

Generate a 2-D array with 3 rows, each row containing 5 random numbers:

```
from numpy import random
```

```
x = random.rand(3, 5)
```

```
print(x)
```

```
[[0.14252791 0.44691071 0.59274288 0.73873487 0.22082345]  
 [0.00484242 0.36294206 0.88507594 0.56948479 0.15075563]  
 [0.69195833 0.75111379 0.92780785 0.57986471 0.6203633 ]]
```



# Numpy - Random Numbers

---

- The **choice()** method allows you to generate a random value based on an array of values.
- The **choice()** method takes an array as a parameter and randomly returns one of the values.

```
from numpy import random  
  
x = random.choice([3, 5, 7, 9])  
  
print(x)
```



3

# Numpy - Random Numbers

---

The **choice()** method also allows you to return an *array* of values. Add a size parameter to specify the shape of the array.

```
from numpy import random  
  
x = random.choice([3, 5, 7, 9], size=(3, 5))  
  
print(x)
```

```
[[5 9 7 5 9]  
 [3 7 7 9 7]  
 [3 7 9 9 5]]
```

# matplotlib

---

## **Installation:**

```
$sudo apt-get install python3-matplotlib
```

## **Pyplot:**

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

```
import matplotlib.pyplot as plt
```

# Matplotlib Plotting

---

The **plot()** function is used to draw points (markers) in a diagram.

By default, the **plot()** function draws a line from point to point.

The function takes parameters for specifying points in the diagram.

- Parameter 1 is an array containing the points on the **x-axis**.
- Parameter 2 is an array containing the points on the **y-axis**.

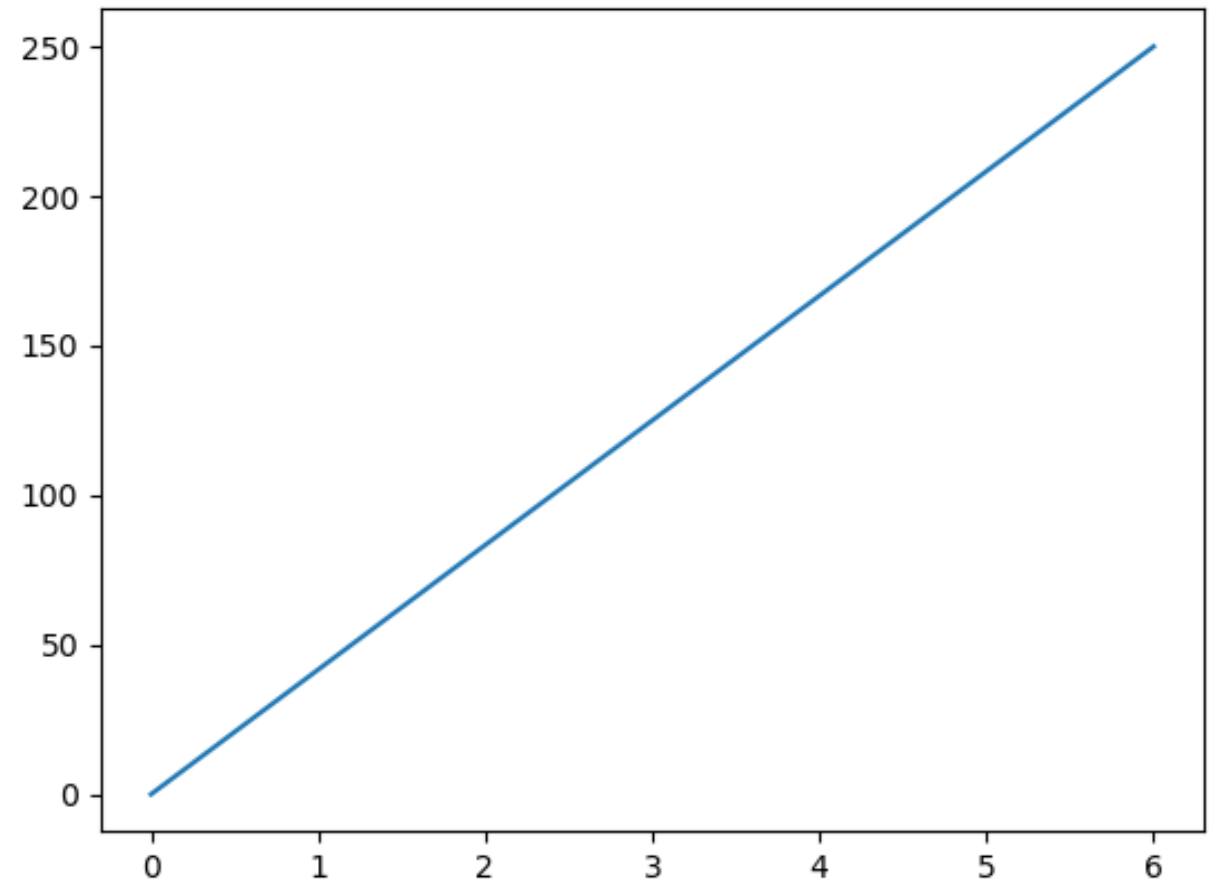
# Example – basic plot

---

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

```
xpoints = np.array([0, 6])  
ypoints = np.array([0, 250])
```

```
plt.plot(xpoints, ypoints)  
plt.show()
```



# Matplotlib Markers

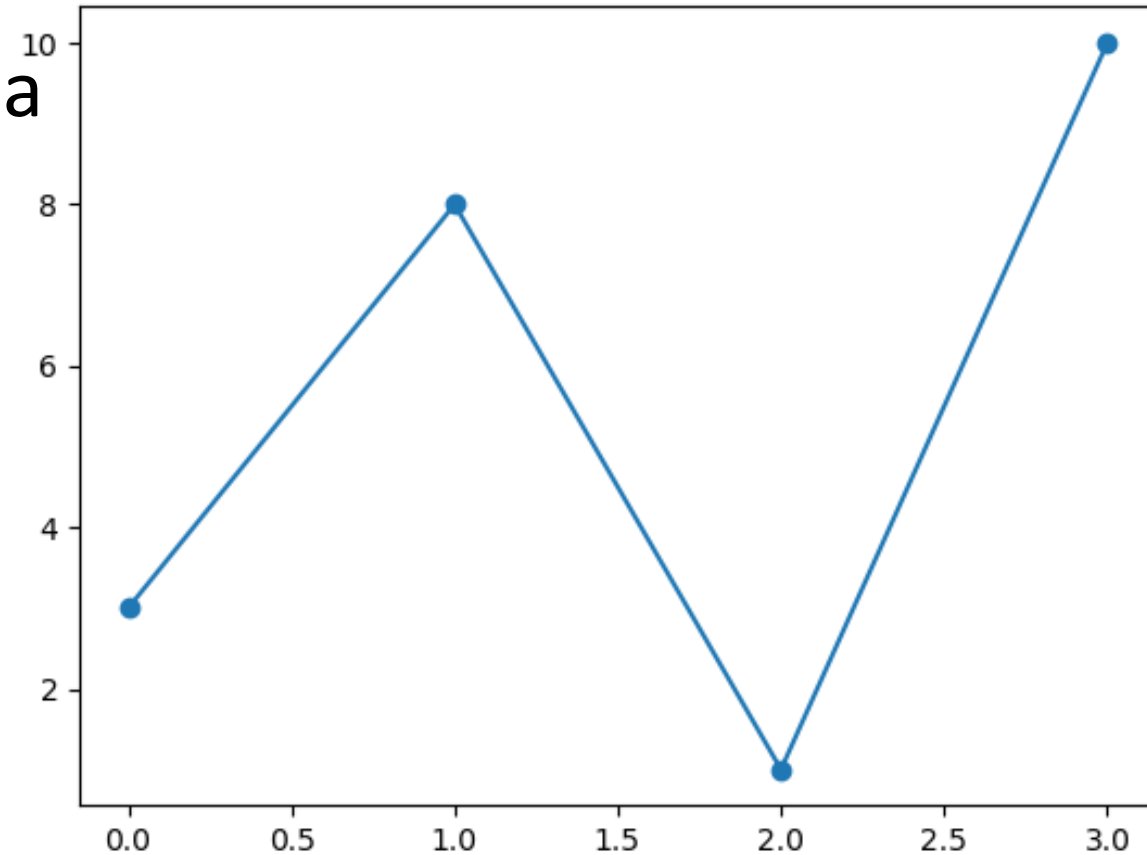
---

You can use the keyword argument `marker` to emphasize each point with a specified marker:

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

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o')
plt.show()
```



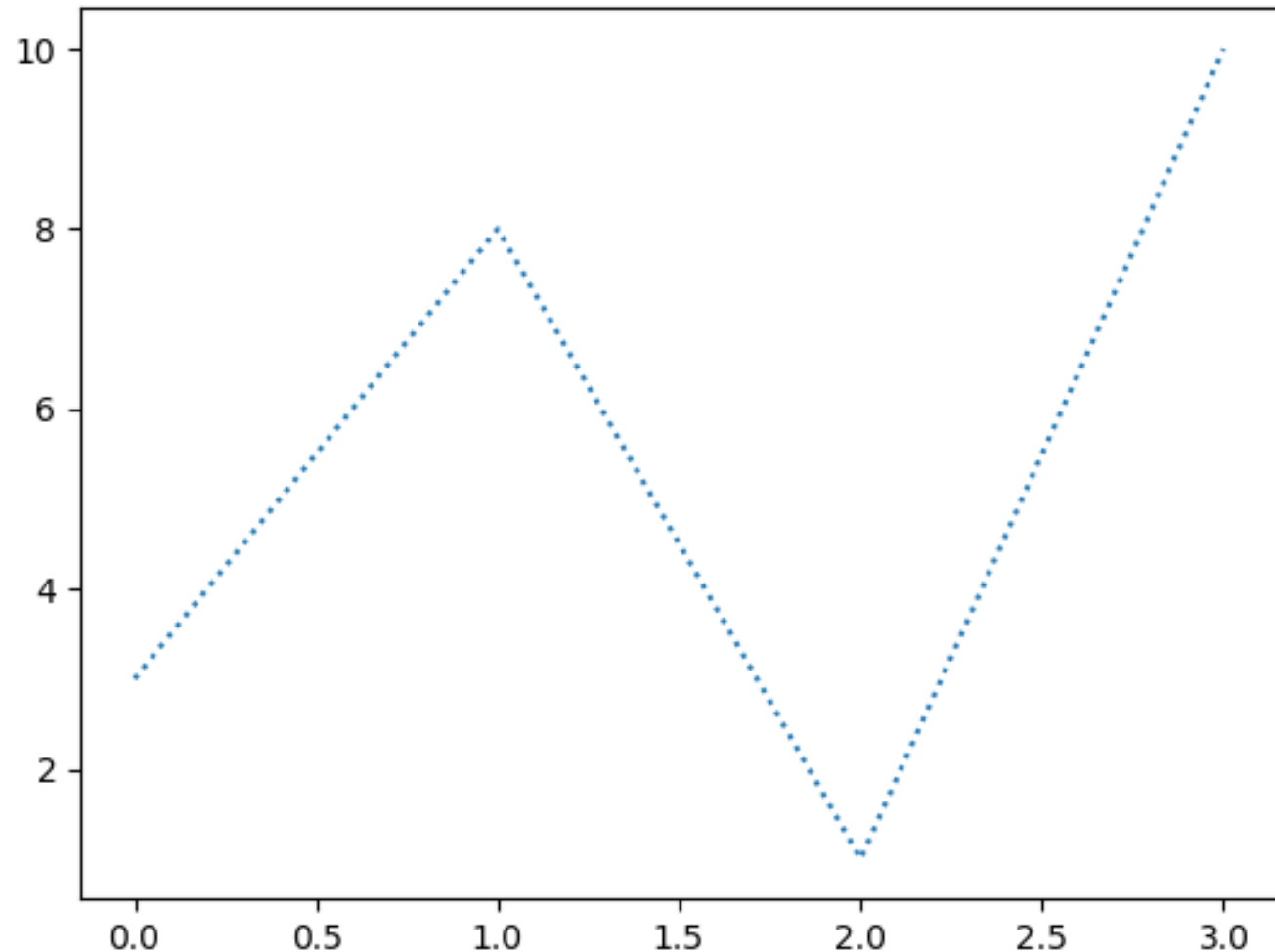
# Matplotlib Line

You can use the keyword argument `linestyle`, or shorter `ls`, to change the style of the plotted line:

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

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```



# Matplotlib Labels and Title

With Pyplot, you can use the `xlabel()` and `ylabel()` functions to set a label for the x- and y-axis.

---

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

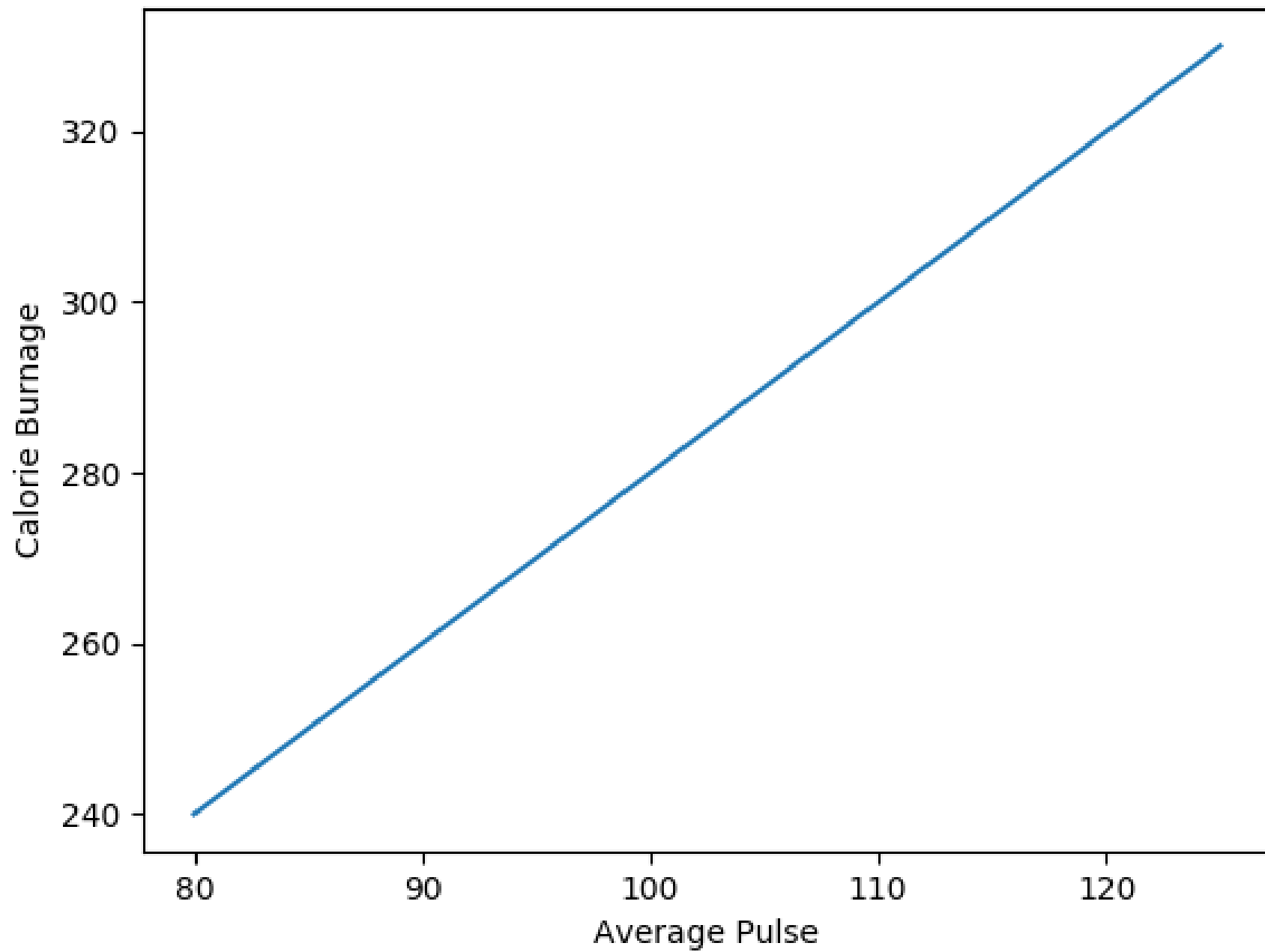
plt.plot(x, y)

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```



Sports Watch Data



# Matplotlib Adding Grid Lines

With Pyplot, you can use the `grid()` function to add grid lines to the plot.

---

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

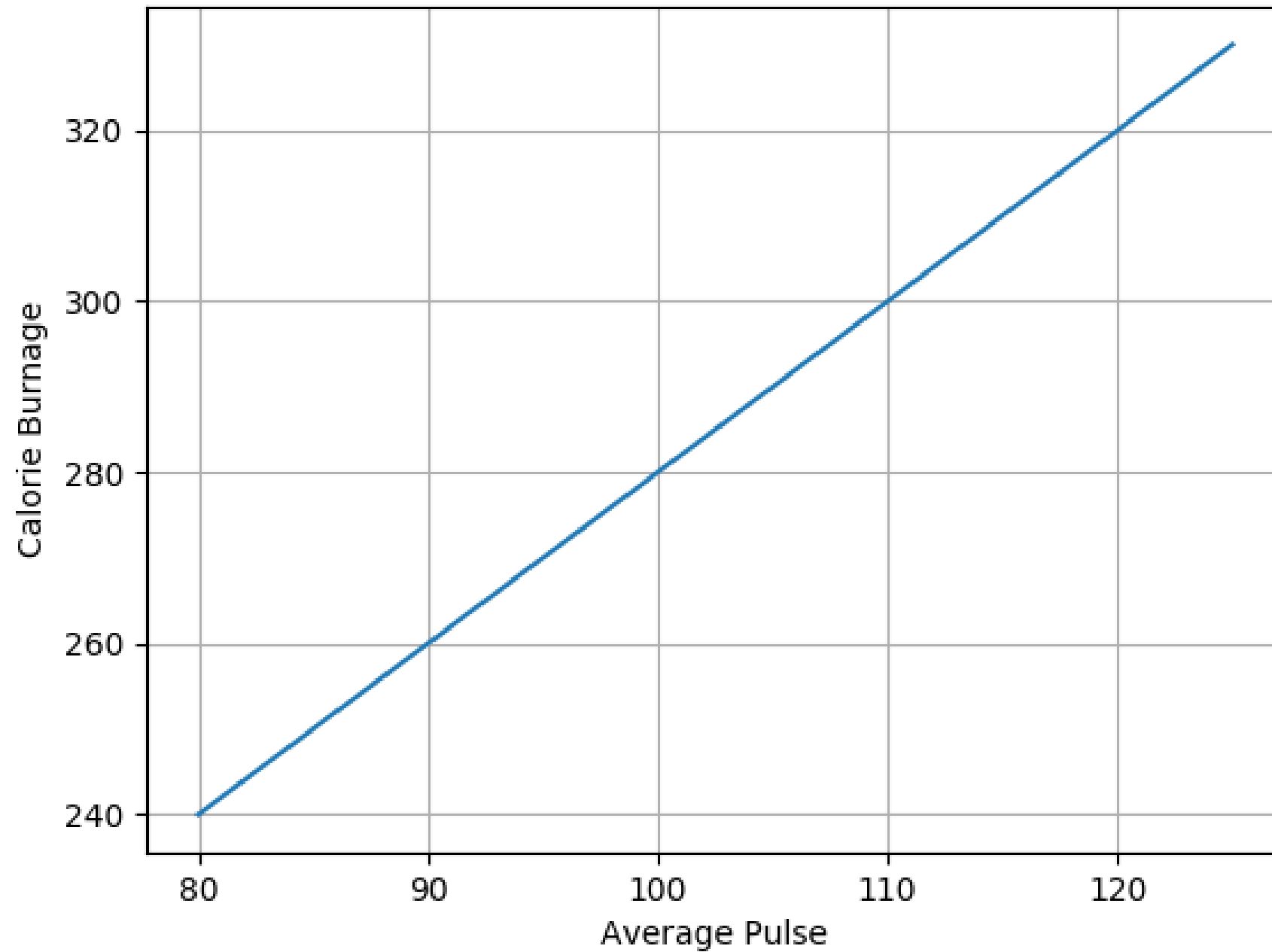
plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid()

plt.show()
```

Sports Watch Data



# Matplotlib Subplot

---

With the **subplot()** function you can draw multiple plots in one figure:  
Parameters are number of rows, number of columns, index of the current plot.

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

```
#plot 1:
```

```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
```

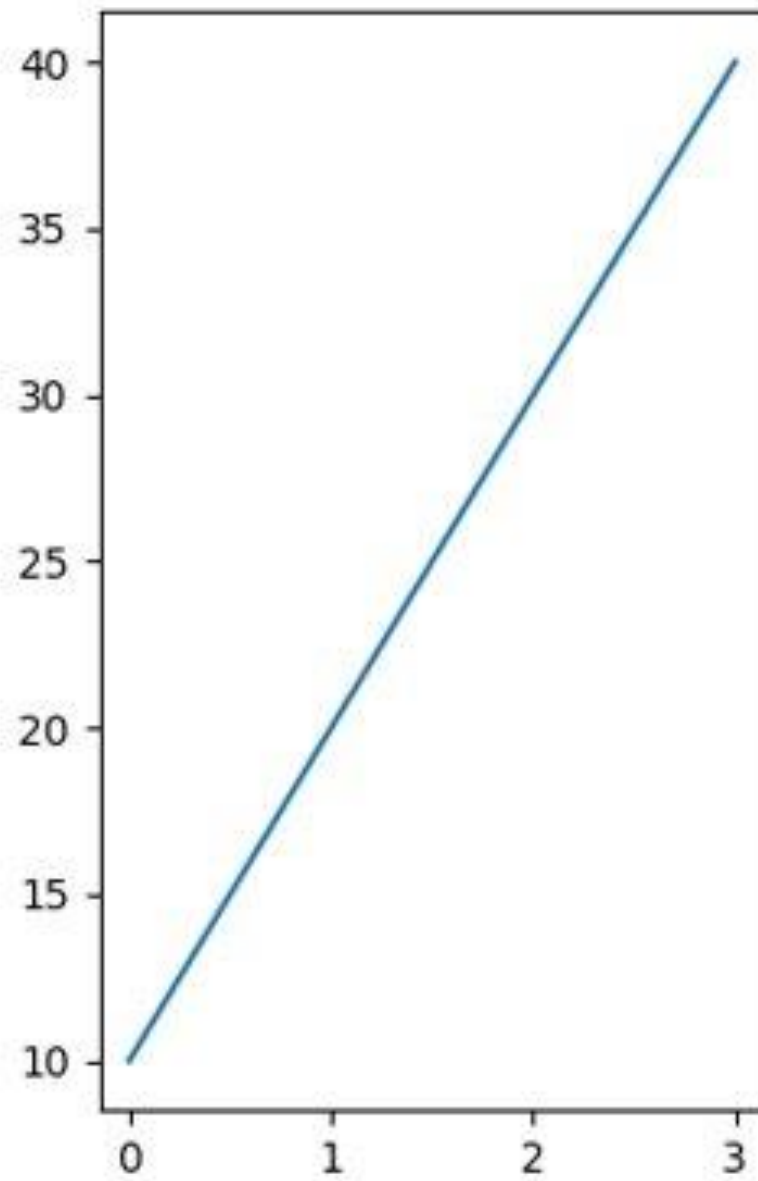
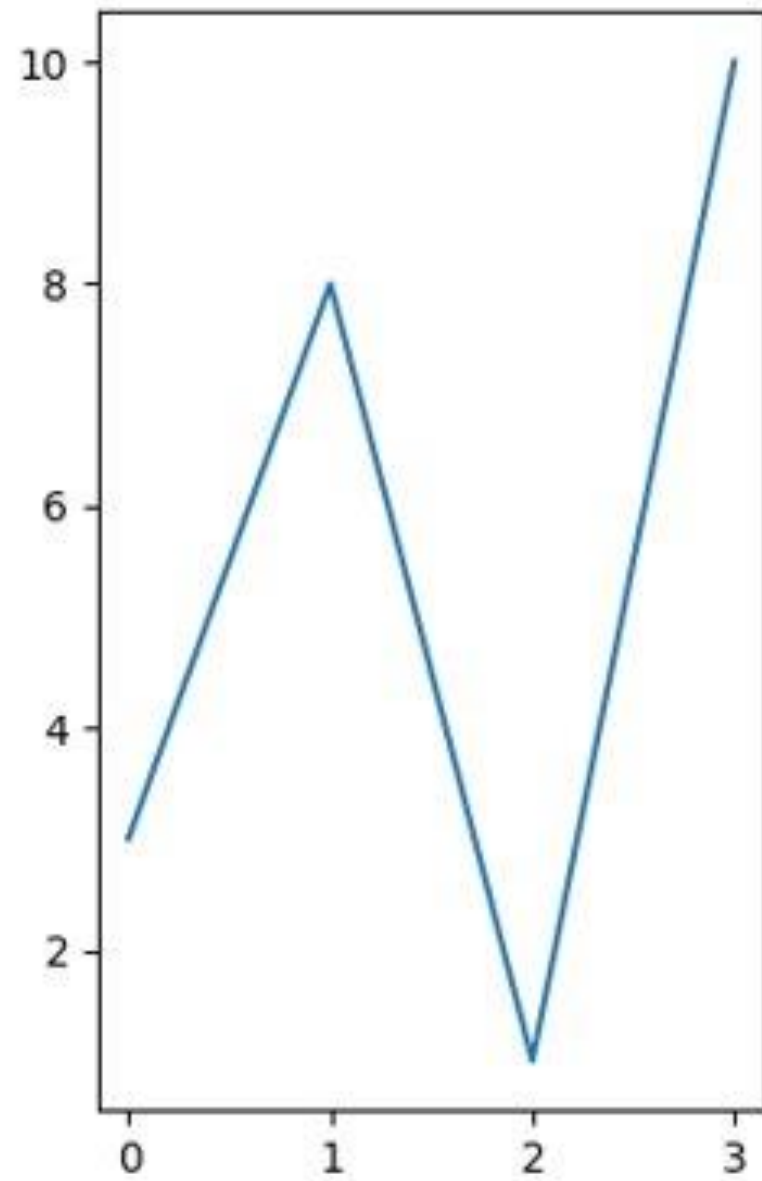
```
plt.subplot(1, 2, 1)
plt.plot(x,y)
```

```
#plot 2:
```

```
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
```

```
plt.subplot(1, 2, 2)
plt.plot(x,y)
```

```
plt.show()
```



# Matplotlib Scatter

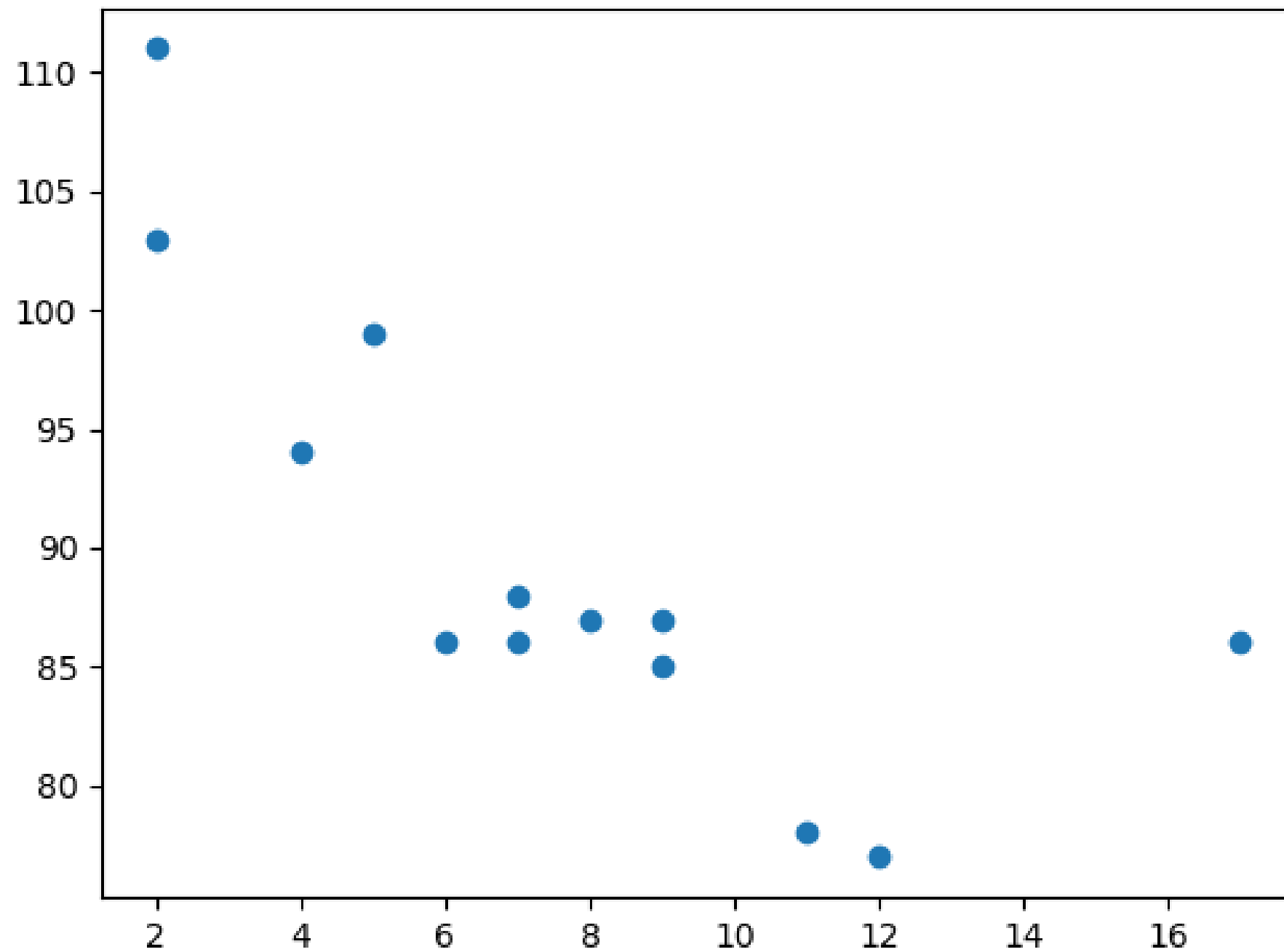
---

The **scatter()** function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

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

x = np.array([5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6])
y = np.array([99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86])

plt.scatter(x, y)
plt.show()
```



# Matplotlib Bars

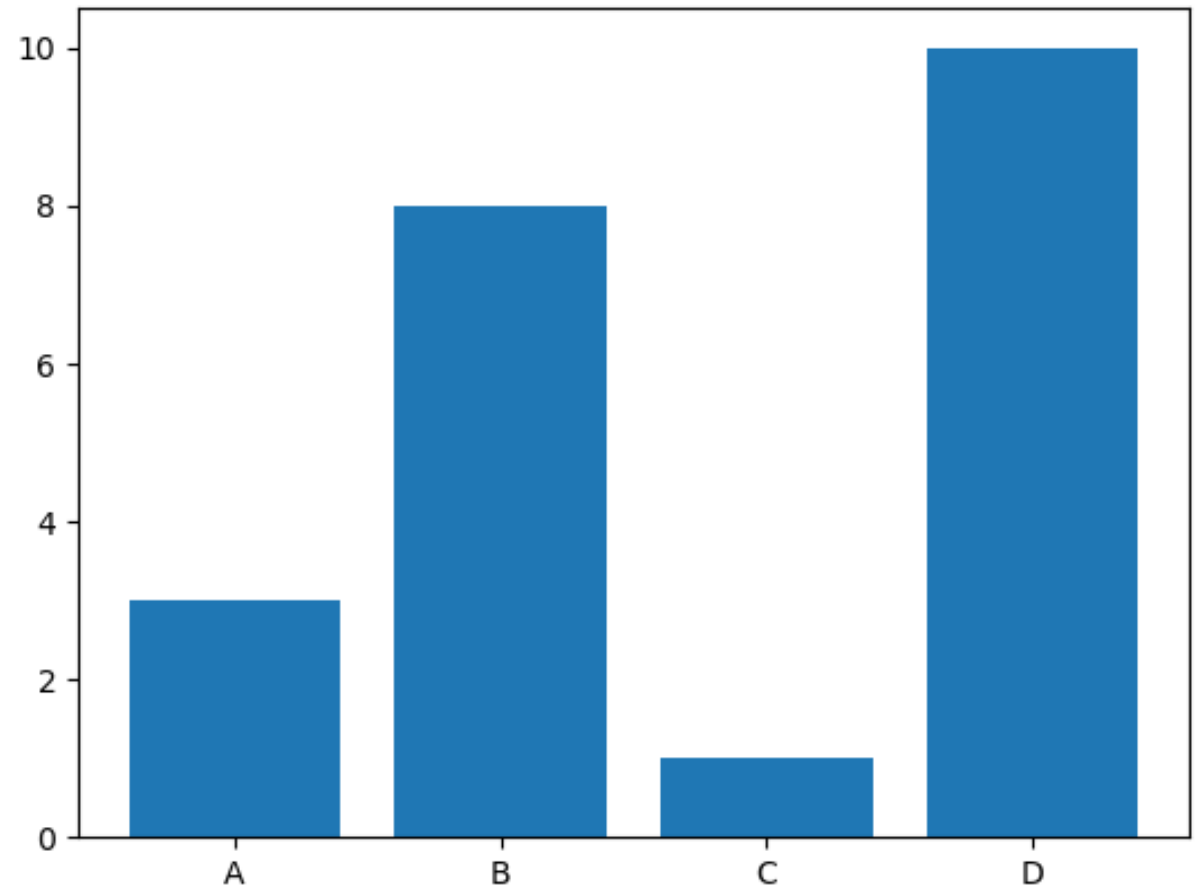
---

With Pyplot, you can use the **bar()** function to draw bar graphs:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y)
plt.show()
```





# Matplotlib Histograms

---

A histogram is a graph showing *frequency* distributions.

It is a graph showing the number of observations within each given interval.

Example: Say you ask for the height of 250 people, you might end up with a histogram like this:

In Matplotlib, we use the **hist()** function to create histograms.

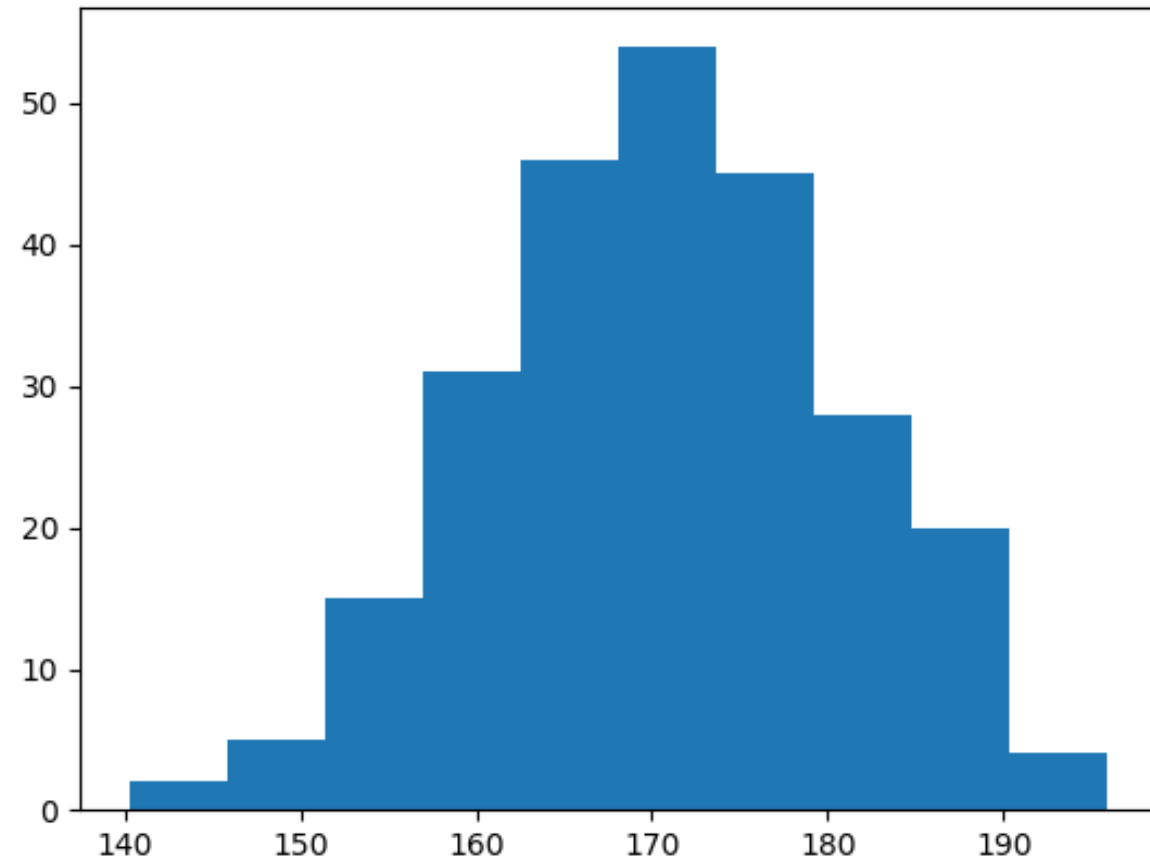
# Matplotlib Histograms

---

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

x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```



# Matplotlib Pie Charts

---

With Pyplot, you can use the **pie()** function to draw pie charts:

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

y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```



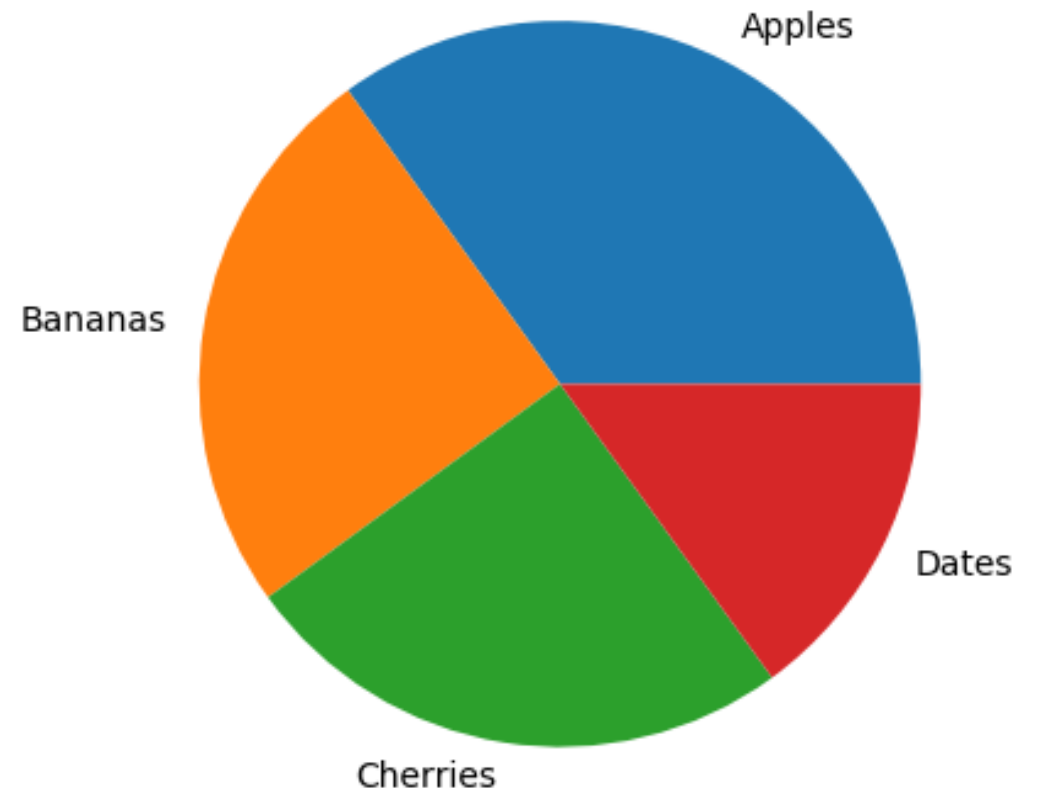
```
import matplotlib.pyplot as plt
import numpy as np
```

```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
```

```
plt.pie(y, labels = mylabels)
plt.show()
```

## Labels

Add labels to the pie chart with the label parameter. The label parameter must be an array with one label for each wedge:



## Start Angle

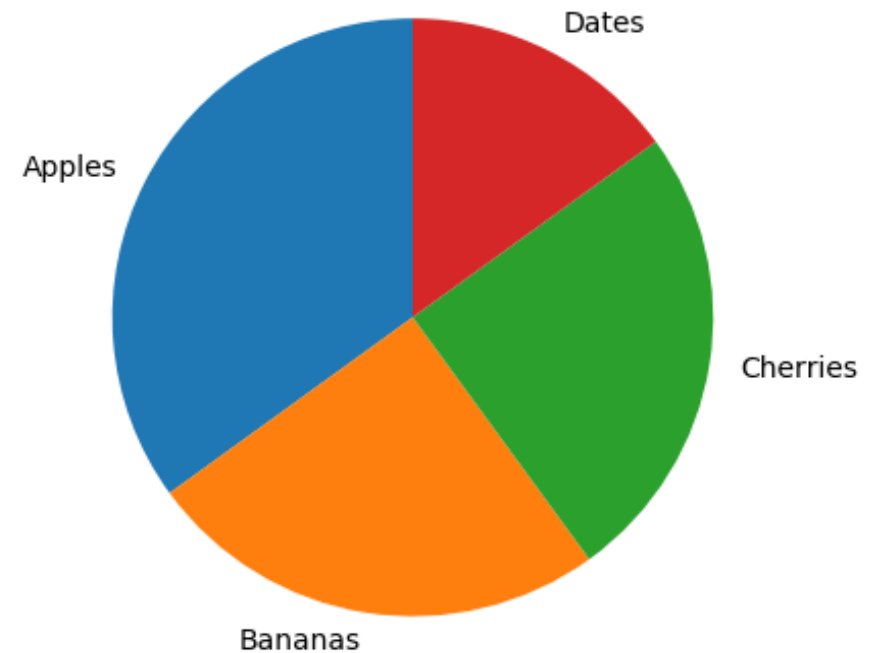
As mentioned the default start angle is at the x-axis, but you can change the start angle by specifying a startangle parameter.

The startangle parameter is defined with an angle in degrees, default angle is 0:

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

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels, startangle = 90)
plt.show()
```



# For more: Reference

---

[https://www.w3schools.com/python/matplotlib\\_line.asp](https://www.w3schools.com/python/matplotlib_line.asp)

# Ticks and Tick Labels

---

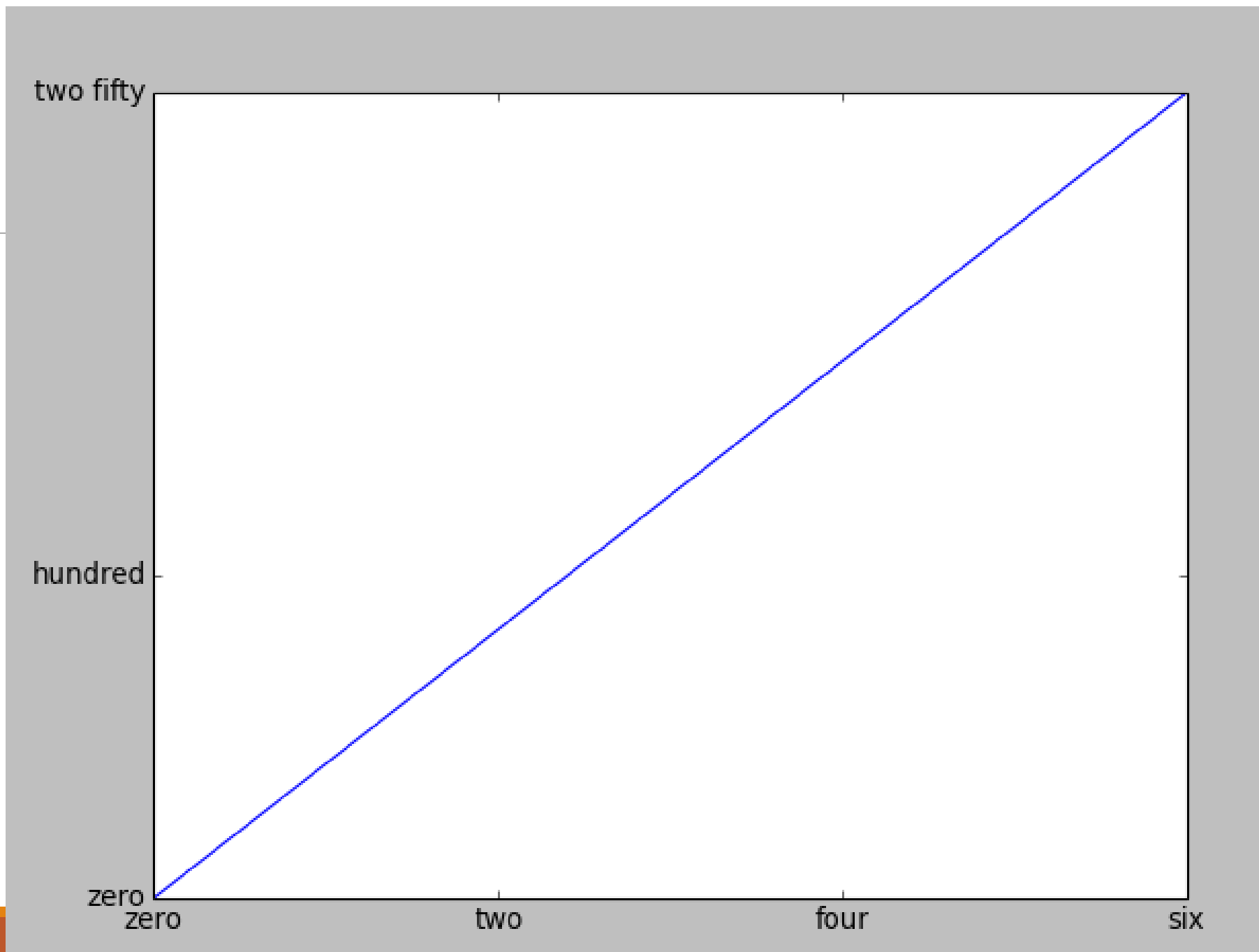
- **Ticks** are the markers denoting data points on axes.
- The **xticks()** and **yticks()** function takes a list object as argument. The elements in the list denote the positions on corresponding axis where ticks will be displayed.
- **labels** corresponding to tick marks can be set by **set\_xlabel()** and **set\_ylabel()** functions respectively.
- **plt.axes(\*args, emit=True, \*\*kwargs)**: For setting the axes for our plot with parameter *rect* as *[left, bottom, width, height]* for setting axes position. ***none***: It gives a new full window axes.
- **ax.set\_title('sine')** : To give title as Sine

# Example

---

```
import matplotlib.pyplot as plt
import numpy as np
#create x and y axis points
xpoints = np.array([0, 6])
ypoints = np.array([0, 250])
#Draw figure
fig = plt.figure()
#Set axes
ax=plt.axes()
#plot x and y axis points
plt.plot(xpoints, ypoints)
#Set x_ticks and x_ticklabels
ax.set_xticks([0,2,4,6])
ax.set_xticklabels(['zero','two','four','six'])
# set y_ticks and y_ticklabels
ax.set_yticks([0,100,250])
ax.set_yticklabels(['zero','hundred','two fifty'])
plt.show()
```





# Legend

---

A legend is an area describing the elements of the graph. In the matplotlib library, there's a function called **legend()** which is used to Place a legend on the axes.

The attribute **Loc** in **legend()** is used to specify the location of the legend.

Default value of loc is **loc="best"** (upper left). The strings '**upper left**', '**upper right**', '**lower left**', '**lower right**' place the legend at the corresponding corner of the axes/figure.

# Legend

---

The Following are some more attributes of function **legend()** :

**shadow**: [None or bool] Whether to draw a shadow behind the legend. It's Default value is None.

**markerscale**: [None or int or float] The relative size of legend markers compared with the originally drawn ones. The Default is None.

**numpoints**: [None or int] The number of marker points in the legend when creating a legend entry for a Line2D (line). The Default is None.

**fontsize**: The font size of the legend. If the value is numeric the size will be the absolute font size in points.

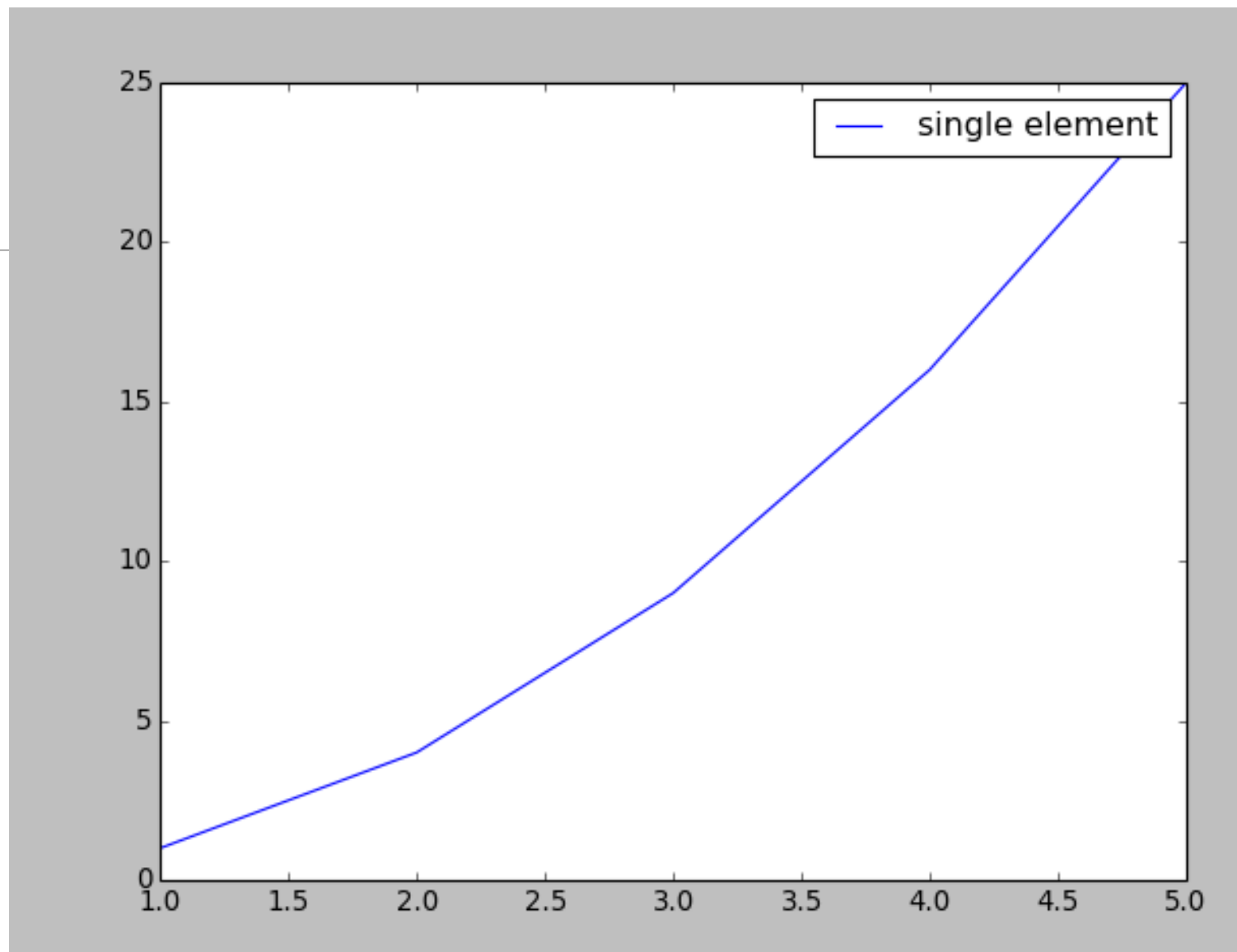
**facecolor**: [None or "inherit" or color] The legend's background color.

**edgecolor**: [None or "inherit" or color] The legend's background patch edge color.

# Example

---

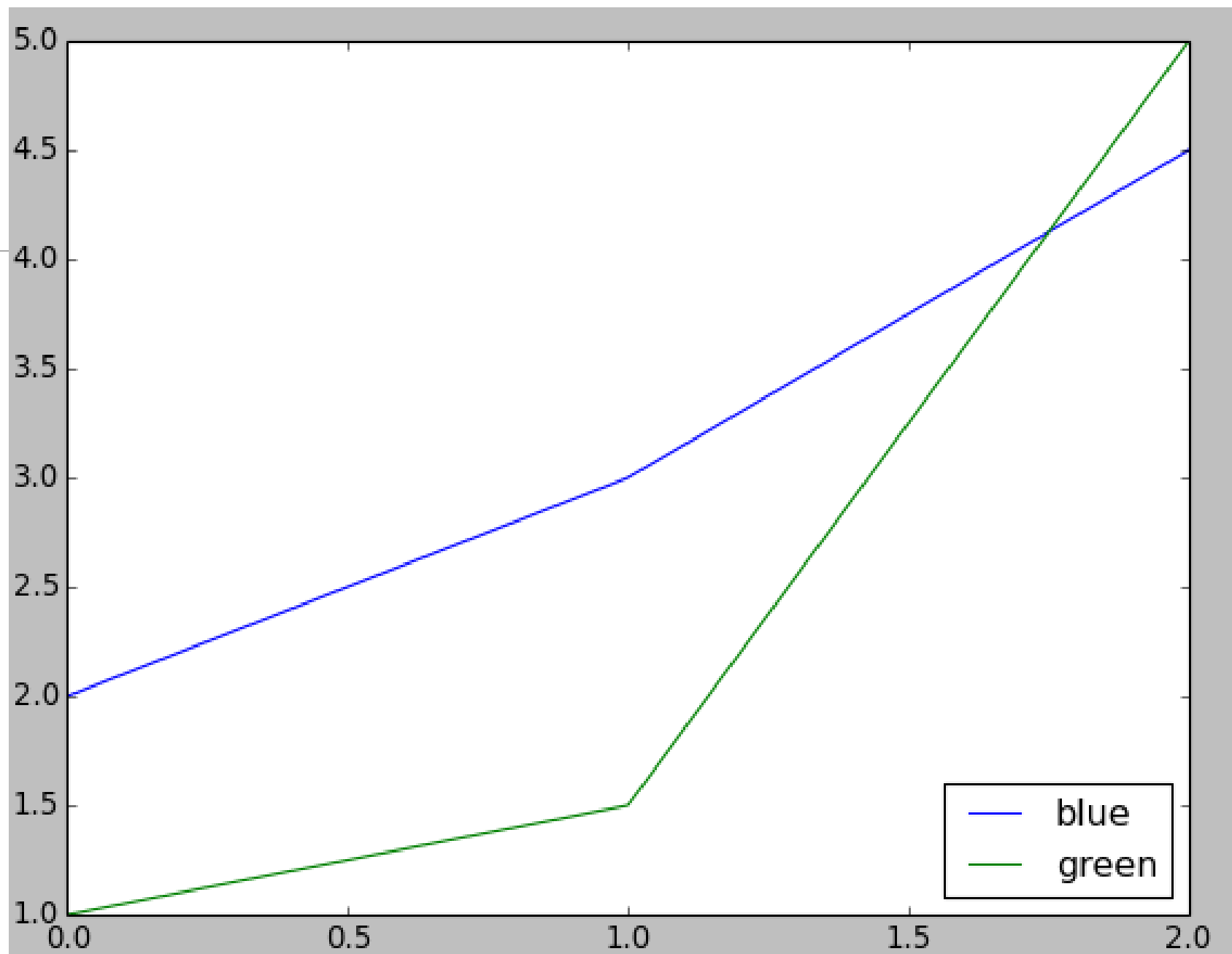
```
import numpy as np
import matplotlib.pyplot as plt
# X-axis values
x = [1, 2, 3, 4, 5]
# Y-axis values
y = [1, 4, 9, 16, 25]
# Function to plot
plt.plot(x, y)
# Function add a legend
plt.legend(['single element'])
# function to show the plot
plt.show()
```



# Example

---

```
# importing modules
import numpy as np
import matplotlib.pyplot as plt
# Y-axis values
y1 = [2, 3, 4.5]
# Y-axis values
y2 = [1, 1.5, 5]
# Function to plot
plt.plot(y1)
plt.plot(y2)
# Function add a legend
plt.legend(["blue", "green"], loc="lower right")
# function to show the plot
plt.show()
|
```



# CSV file format

---

- **CSV** (Comma Separated Values) is a simple **file format** used to store tabular data, such as a spreadsheet or database.
- A CSV file stores tabular data (numbers and text) in plain text.
- Each line of the file is a data record.
- Each record consists of one or more fields, separated by commas.
- The use of the comma as a field separator is the source of the name for this file format.



# Working with CSV files- pandas

---

**Install pandas:**

```
$sudo apt install python3-pandas
```

A simple way to store big data sets is to use **CSV files** (comma separated files).

```
import pandas as pd
df = pd.read_csv('sample.csv', encoding= 'unicode_escape')
print(df.to_string()) |
```

# pandas - DataFrame

**Pandas DataFrame:** It is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Pandas DataFrame consists of three principal components, the **data**, **rows**, and **columns**.

The diagram illustrates a Pandas DataFrame with the following structure:

	<i>Name</i>	<i>Team</i>	<i>Number</i>	<i>Position</i>	<i>Age</i>
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Annotations in the diagram:

- Columns:** A blue label at the top with arrows pointing to the column headers: *Name*, *Team*, *Number*, *Position*, and *Age*.
- Rows:** An orange label on the left with arrows pointing to the row indices: 0, 1, 2, 3, 4, 5, and 6.
- Data:** A purple label at the bottom with a bracket pointing to the data cells of the rows, specifically highlighting the values in the *Team*, *Number*, *Position*, and *Age* columns for rows 2 through 6.

# Creating a dataframe using List:

---

DataFrame can be created using a single list or a list of lists.

```
# import pandas as pd
import pandas as pd

# list of strings
lst = ['Geeks', 'For', 'Geeks', 'is',
       'portal', 'for', 'Geeks']

# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
print(df)
```

```
0      Geeks
1        For
2      Geeks
3         is
4    portal
5        for
6      Geeks

[7 rows x 1 columns]
...
```

# Creating DataFrame from dict of ndarray/lists:

---

```
# Python code demonstrate creating  
# DataFrame from dict ndarray / lists  
# By default addresses.
```

```
import pandas as pd
```

```
# initialise data of lists.
```

```
data = {'Name': ['Tom', 'nick', 'krish', 'jack'],  
        'Age': [20, 21, 19, 18]}
```

```
# Create DataFrame
```

```
df = pd.DataFrame(data)
```

```
# Print the output.
```

```
print(df)
```

	Age	Name
0	20	Tom
1	21	nick
2	19	krish
3	18	jack

[4 rows x 2 columns]

## Column Selection:

```
# Import pandas package
import pandas as pd
```

```
# Define a dictionary containing employee data
data = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age': [27, 24, 22, 32],
        'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification': ['Msc', 'MA', 'MCA', 'Phd']}
```

```
# Convert the dictionary into DataFrame
df = pd.DataFrame(data)
```

```
# select two columns
print(df[['Name', 'Qualification']])
```

	Name	Qualification
0	Jai	Msc
1	Princi	MA
2	Gaurav	MCA
3	Anuj	Phd

[4 rows x 2 columns]

# Row Selection:

---

Pandas provide a unique method to retrieve rows from a Data frame. [DataFrame.loc\[\]](#) method is used to retrieve rows from Pandas DataFrame.

Rows can also be selected by passing integer location to an [iloc\[\]](#) function.

# DataFrame functions

---

**abs()** : Return a Series/DataFrame with absolute numeric value of each element.

```
>>> s = pd.Series([-1.10, 2, -3.33, 4])
>>> s.abs()
0      1.10
1      2.00
2      3.33
3      4.00
dtype: float64
```

# DataFrame functions

---

**agg**([func, axis]) : Aggregate using one or more operations over the specified axis.

```
>>> df = pd.DataFrame([[1, 2, 3],
...                    [4, 5, 6],
...                    [7, 8, 9],
...                    [np.nan, np.nan, np.nan]],
...                    columns=['A', 'B', 'C'])
```

Aggregate these functions over the rows.

```
>>> df.agg(['sum', 'min'])
```

	A	B	C
sum	12.0	15.0	18.0
min	1.0	2.0	3.0



# DataFrame functions

---

[append](#)(other[, ignore\_index, ...]): Append rows of *other* to the end of caller, returning a new object.

```
>>> df = pd.DataFrame([[1, 2], [3, 4]], columns=list('AB'), index=['x', 'y'])
>>> df
   A  B
x  1  2
y  3  4
>>> df2 = pd.DataFrame([[5, 6], [7, 8]], columns=list('AB'), index=['x', 'y'])
>>> df.append(df2)
   A  B
x  1  2
y  3  4
x  5  6
y  7  8
```

# DataFrame functions

---

[apply](#)(func[, axis, raw, result\_type, args]): Apply a function along an axis of the DataFrame.

```
>>> df = pd.DataFrame([[4, 9]] * 3, columns=['A', 'B'])
>>> df
```

	A	B
0	4	9
1	4	9
2	4	9

Using a numpy universal function (in this case the same as `np.sqrt(df)`):

```
>>> df.apply(np.sqrt)
```

	A	B
0	2.0	3.0
1	2.0	3.0
2	2.0	3.0

# DataFrame functions

---

[count](#)([axis, level, numeric\_only]) : Count non-NA cells for each

```
>>> df = pd.DataFrame({"Person":  
...                     ["John", "Myla", "Lewis", "John", "Myla"],  
...                     "Age": [24., np.nan, 21., 33, 26],  
...                     "Single": [False, True, True, True, False]})
```

```
>>> df
```

	Person	Age	Single
0	John	24.0	False
1	Myla	NaN	True
2	Lewis	21.0	True
3	John	33.0	True
4	Myla	26.0	False

# DataFrame functions

---

[count](#)([axis, level, numeric\_only]) : Count non-NA cells for each column or row.

Constructing DataFrame from a dictionary:

```
>>> df = pd.DataFrame({"Person":  
...                     ["John", "Myla", "Lewis", "John", "Myla"],  
...                     "Age": [24., np.nan, 21., 33, 26],  
...                     "Single": [False, True, True, True, False]})  
>>> df
```

	Person	Age	Single
0	John	24.0	False
1	Myla	NaN	True
2	Lewis	21.0	True
3	John	33.0	True
4	Myla	26.0	False

Notice the uncounted NA values:

```
>>> df.count()  
Person      5  
Age         4  
Single      5  
dtype: int64
```

Counts for each **row**:

```
>>> df.count(axis='columns')  
0      3  
1      2  
2      3  
3      3  
4      3  
dtype: int64
```

# DataFrame functions

---

**DataFrame.groupby**([by, axis, level, ...]): Group DataFrame using a mapper or by a Series of columns.

```
>>> df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',  
...                               'Parrot', 'Parrot'],  
...                   'Max Speed': [380., 370., 24., 26.]})  
>>> df
```

	Animal	Max Speed
0	Falcon	380.0
1	Falcon	370.0
2	Parrot	24.0
3	Parrot	26.0

```
>>> df.groupby(['Animal']).mean()
```

	Max Speed
Animal	
Falcon	375.0
Parrot	25.0

# DataFrame functions

```
>>> arrays = [['Falcon', 'Falcon', 'Parrot', 'Parrot'],
...           ['Captive', 'Wild', 'Captive', 'Wild']]
>>> index = pd.MultiIndex.from_arrays(arrays, names=('Animal', 'Type'))
>>> df = pd.DataFrame({'Max Speed': [390., 350., 30., 20.]},
...                   index=index)
>>> df
```

Animal	Type	Max Speed
Falcon	Captive	390.0
	Wild	350.0
Parrot	Captive	30.0
	Wild	20.0

```
>>> df.groupby(level=0).mean()
```

Animal	Max Speed
Falcon	370.0
Parrot	25.0

```
>>> df.groupby(level="Type").mean()
```

Type	Max Speed
Captive	210.0
Wild	185.0

# DataFrame functions

[`DataFrame.max`](#)([axis, skipna, level, ...]): Return the maximum of the values over the requested axis.

---

## Examples

```
>>> idx = pd.MultiIndex.from_arrays([
...     ['warm', 'warm', 'cold', 'cold'],
...     ['dog', 'falcon', 'fish', 'spider']],
...     names=['blooded', 'animal'])
>>> s = pd.Series([4, 2, 0, 8], name='legs', index=idx)
>>> s
```

blooded	animal	
warm	dog	4
	falcon	2
cold	fish	0
	spider	8

Name: legs, dtype: int64

```
>>> s.max()
8
```



# DataFrame functions

[`DataFrame.min`](#)([axis, skipna, level, ...]): Return the minimum of the values over the requested axis.

---

## Examples

---

```
>>> idx = pd.MultiIndex.from_arrays([
...     ['warm', 'warm', 'cold', 'cold'],
...     ['dog', 'falcon', 'fish', 'spider']],
...     names=['blooded', 'animal'])
>>> s = pd.Series([4, 2, 0, 8], name='legs', index=idx)
>>> s
```

blooded	animal	
warm	dog	4
	falcon	2
cold	fish	0
	spider	8

Name: legs, dtype: int64

```
>>> s.min()
0
```

# DataFrame functions

---

[DataFrame.mean](#)([axis, skipna, level, ...]): Return the mean of the values over the requested axis.

**For more functions:**

**Reference:** <https://pandas.pydata.org/docs/reference/frame.html>

# Working with CSV files

---

## Load CSV files to Python Pandas

---

```
import pandas as pd
df = pd.read_csv('sample.csv', encoding= 'unicode_escape')
print(df.to_string())
|
```

### `read_csv()` : parameters

- 1. filepath\_or\_buffer:** It is the location of the file which is to be retrieved using this function. It accepts any string path or URL of the file.
- 2. sep:** It stands for separator, default is ',' as in csv(comma separated values).

# read\_csv() : parameters(continue)

---

- 3. header:** It accepts int, list of int, row numbers to use as the column names and start of the data. If no names are passed, i.e., header=None, then, it will display first column as 0, second as 1, and so on.
- 4. usecols:** It is used to retrieve only selected columns from the csv file.
- 5. nrows:** It means number of rows to be displayed from the dataset.
- 6. index\_col:** If None, there are no index numbers displayed along with records.
- 7. squeeze:** If true and only one column is passed, returns pandas series.
- 8. skiprows:** Skips passed rows in new data frame.
- 9. names:** It allows to retrieve columns with new names.

# read\_csv() : parameters(continue)

---

---

```
import pandas as pd
df = pd.read_csv('sample.csv', encoding= 'unicode_escape', sep=',')
print(df.to_string())
|
```

# read\_csv() : parameters(continue)

---

Read the csv file sep='|' :

---

```
import pandas as pd
df = pd.read_csv('sample.csv', encoding= 'unicode_escape', sep=',')
print(df.to_string())
```

**Read the csv file with header parameter:**

The row 0 seems to be a better fit for the header.

Note: Row numbering starts from 0 including column header

```
# Read the csv file
df = pd.read_csv("data1.csv")
df.head()
```

	Ranking	Name	Count 1	Count 2	Dates
0	Rank	State	Population	National Share (%)	Date
1	1	Uttar Pradesh	19,98,12,341	16.51%	25-02-2021
2	2	Maharashtra	11,23,74,333	9.28%	14-04-2021
3	3	Bihar	10,40,99,452	missing	19-02-2021
4	4	West Bengal	9,12,76,115	7.54%	24-02-2021

```
# Read the csv file with header parameter  
df = pd.read_csv("data1.csv", header=1)  
df.head()
```

	Rank	State	Population	National Share (%)	Date
0	1	Uttar Pradesh	19,98,12,341	16.51%	25-02-2021
1	2	Maharashtra	11,23,74,333	9.28%	14-04-2021
2	3	Bihar	10,40,99,452	missing	19-02-2021
3	4	West Bengal	9,12,76,115	7.54%	24-02-2021
4	5	Madhya Pradesh	7,26,26,809	6%	13-02-2021



# Renaming column headers

```
# Read the csv file with names parameter
df = pd.read_csv("data.csv", names=['Ranking', 'ST Name', 'Pop',
'NS', 'D'])
df.head()
```

	Ranking	ST Name	Pop	NS	D
0	Rank	State	Population	National Share (%)	Date
1	1	Uttar Pradesh	19,98,12,341	16.51%	25-02-2021
2	2	Maharashtra	11,23,74,333	9.28%	14-04-2021
3	3	Bihar	10,40,99,452	missing	19-02-2021
4	4	West Bengal	9,12,76,115	7.54%	24-02-2021

# Writing to CSV file

---

Pandas **DataFrame** provides **to\_csv()** method to write/export DataFrame to CSV comma-separated delimiter file along with header and index.

```
df.to_csv('filename.csv')
```

## Write DataFrame to CSV without Header

```
df.to_csv('filename.csv', header=False)
```

## Writing Using Custom Delimiter

```
df.to_csv('filename.csv', header=False, sep='|')
```

## Writing to CSV ignoring Index

```
df.to_csv('filename.csv', index=False)
```

# Writing to CSV file

---

## Writing to CSV ignoring Index

```
df.to_csv('filename.csv', index=False)
```

## Export Selected Columns to CSV File

```
column_names = ['Courses', 'Fee', 'Discount']
```

```
df.to_csv('filename.csv', index=False, columns=column_names)
```

## Change Header Column Names While Writing

```
column_names = ['Courses', 'Course_Fee', 'Course_Duration', 'Course_Discount']
```

```
df.to_csv('filename.csv', index=False, header=column_names)
```

# Writing to CSV file

---

## **Write DataFrame to CSV by Encoding**

```
df.to_csv(file_name, sep='\t', encoding='utf-8')
```

## **Append DataFrame to existing CSV File**

```
df.to_csv("c:/tmp/courses.csv", header=False, sep='|', index=False, mode='a')
```

# Writing to CSV file

```
# importing pandas as pd
import pandas as pd

# list of name, degree, score
nme = ["aparna", "pankaj", "sudhir", "Geeku"]
deg = ["MBA", "BCA", "M.Tech", "MBA"]
scr = [90, 40, 80, 98]

# dictionary of lists
dict = {'name': nme, 'degree': deg, 'score': scr}

df = pd.DataFrame(dict)

# saving the dataframe
df.to_csv('file1.csv')
```

# Clean and Update the CSV file

---

CSV Data Cleaning Checks

Missing Values

Outliers

Duplicate Values

1. Cleaning Missing Values in CSV File:

In Pandas, a missing value is usually denoted by NaN , since it is based on the [NumPy package](#) it is the special floating-point NaN value particular to NumPy.

# Clean and Update the CSV file

---

## 1. Dropping Missing Values:

**df.dropna()** – Drop all rows that have any NaN values

**df.dropna(how='all')** – Drop only if ALL columns are NaN

**df.dropna(thresh=2)** – Drop row if it does not have at least two values that are not NaN

**df.dropna(subset=[1])** – Drop only if NaN in specific column

# Dropping Missing Values:

```
#importing pandas
import pandas as pd

#Importing dataset
df = pd.read_csv('IMDB-Movie-Data.csv')

#Size of original dataset
print(df.shape)

#Dropping the missing rows.
df_dropped = df.dropna(how = 'any')
```



# Clean and Update the CSV file

---

## 2. Replacing Missing values:

Pandas module has the **.fillna()** method, which accepts a value that we want to replace in place of **NaN** values. We just calculated the mean of the column and passed it as an input argument to **fillna()** method.

**.sum()** uses to find sum of all values along the index axis. We are going to skip the NaN values in the calculation of the sum.

```
import pandas as pd

#importing the dataset
df = pd.read_csv('IMDB-Movie-Data.csv')

#Creating a copy of dataframe
df_new = df

df_new['Metascore'] = df_new['Metascore'].fillna((df_new['Metascore'].mean()))

#printing the dataframes after replacing null values
print(df_new.isna().sum())
print(df.isna().sum())
```

# Sample questions

---

Q. Given a file “auto.csv” of automobile data with the fields index, company, body-style, wheel-base, length, engine-type, num-of-cylinders, horsepower, average-mileage, and price, write Python codes using Pandas to

- 1) Clean and Update the CSV file
- 2) Print total cars of all companies
- 3) Find the average mileage of all companies
- 4) Find the highest priced car of all companies.

# Sample questions

---

Q. Write Python program to write the data given below to a CSV file.

<b>SN</b>	<b>Name</b>	<b>Country</b>	<b>Contribution</b>	<b>Year</b>
1	Linus Torvalds	Finland	Linux Kernel	1991
2	Tim Berners-Lee	England	World Wide Web	1990
3	Guido van Rossum	Netherlands	Python	1991

Q. What are the important characteristics of CSV file format.

# Sample questions

---

Q. Given the sales information of a company as CSV file with the following fields month\_number, facecream, facewash, toothpaste, bathingsoap, shampoo, moisturizer, total\_units, total\_profit. Write Python codes to visualize the data as follows

- 1) Toothpaste sales data of each month and show it using a scatter plot
- 2) Face cream and face wash product sales data and show it using the bar chart

Calculate total sale data for last year for each product and show it using a Pie chart.

# Flask

---

- **Flask** is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries.
- It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.
- Flask is a web application framework written in Python. Armin Ronacher, who leads an international group of Python enthusiasts named Pocco, develops it.
- Flask is based on Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects.

# Install virtualenv for development environment

---

```
$sudo apt-get install virtualenv
```

To activate corresponding environment, on **Linux/OS X**, use the following –

```
venv/bin/activate
```

Hello.py

---

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World'

if __name__ == '__main__':
    app.run()
```



# Flask – Application

---

Importing flask module in the project is mandatory. An object of Flask class is our **WSGI** application.

Flask constructor takes the name of **current module** (**\_\_name\_\_**) as argument.

The **route()** function of the Flask class is a decorator, which tells the application which URL should call the associated function.

**app.route(rule, options)**

The **rule** parameter represents URL binding with the function.

The **options** is a list of parameters to be forwarded to the underlying Rule object.

# Flask – Application

---

In the above example, **‘/’ URL** is bound with **hello\_world()** function. Hence, when the home page of web server is opened in browser, the output of this function will be rendered.

Finally the **run()** method of Flask class runs the application on the local development server.

**app.run(host, port, debug, options)**

All parameters are optional,

**host:** Hostname to listen on. Defaults to 127.0.0.1 (localhost). Set to ‘0.0.0.0’ to have server available externally and **port:** Defaults to 5000 (TCP/UDP)

**debug:** Defaults to false. If set to true, provides a debug information

**options:** To be forwarded to underlying Werkzeug server.

# Micro services using Flask

---

The framework very easy to work with for developing RESTful services working with JSON.

The simplicity of both the framework and the language itself allows you to write small, concise request handler functions.

```
@app.route('/item/<item_type>', methods=['POST', 'PUT'])
def update_item(item_type):
    is_update = request.method == 'PUT'
    result = engine.process(request.json, update=is_update)
    if result:
        return jsonify(result)
    else:
        return 'Nope :(', 400
```

# Micro services using Flask

---

You can also easily add extra request processing logic around your endpoints. For example, if you have one doing some expensive operation, you could memoize the results for some time instead of repeating it on every call.

```
@app.route('/hard/work')
@cache.memoize(timeout=30 * 60)
def expensive_operation():
    return look_busy_while_doing_this()
```

# Micro services using Flask

---

## Caching View Functions

To cache view functions you will use the [cached\(\)](#) decorator. This decorator will use request.path by default for the cache\_key.:

```
@cache.cached(timeout=50)
def index():
    return render_template('index.html')
```