## Matplotlib intro

- Matplotlib is a python library for creating charts and graphs
- This has the power to convert the plain numbers into visually appealing graphs

- We can make
  1. Line charts
  2. Bar charts
  3. Scatter plots
  4. Histograms
  5. pie charts

- We can even customize the size and pretty much all

# Install the Library :-
→ pip install matplot lib

# checking the version of the installed matplotlib

`Print(matplotlibo__version__)` These __func__ are called dunder functions.

- There is a module called pyplot which exists in the library of matplotlib

- Pyplot provides a user-friendly interface for plotting

So well import it along with matplotlib

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

→ instead of calling the full name we imporit matplotlib.pyplot as plt.

# Creating a basic plot

→ To creak any plot we need x and Y Coordinates here let's just use python List

x = [2023, 2024, 2025, 2026] → for x coordinates

Y = [15, 25, 30, 20] → for Y Coordinates

→ make sure that no. of x coord == no. of Y coord

→ To Creak a basic plot acces the plt and call the plot function. and pass the Coordinats

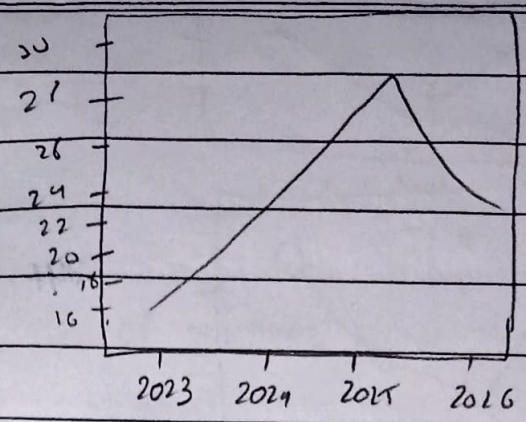
```
Pt.Plot(x,y)
```

The first argument is for the x-axis
The Second argument is for the Y-axis

→ if we run this nothing displays
→ we need to call the show function inorder to view the plot

```
Plt.show()
```

plt.show()




→ if we only pass one argument this argument would be for points on the Y axis

# Default Behaviour for plot:-

1. Start at 0 and place a point at each increment of one
2. each of these ticks they increment by 1 each
3. The points on the x-axis will be provided to us

# Using numpy arrays → Because numpy arrays are faster and give more functionality

```
import numpy as np
x = np.array([2023, 2024, 2025, 2026])
Y = np.array([15, 25, 30, 20])

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

# Plot Customization

→ We can customize the plots which we create, ~~more~~

→ More specially we can customize the lines and markers

→ To do this pass a keyword into the plot function which is called marker

```
plt.plot(x, y, marker = '')
```

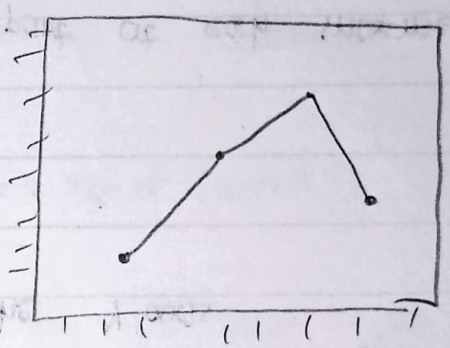
→ marker equals a given character.

→ Depending on the character that we set for markers, our marker will be one of a few symbols

eg

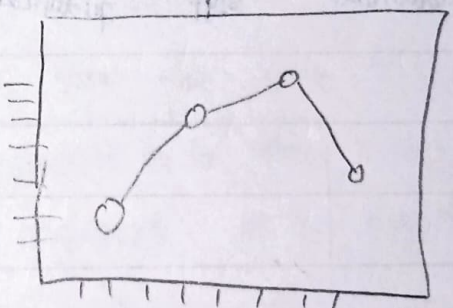
```
plt.plot(x, y, marker = '.')
```

→ Then each set of coordinates has a point



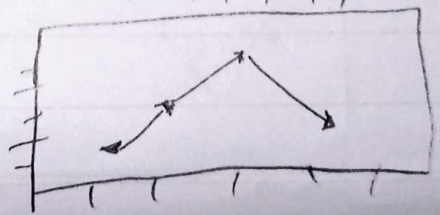
→ Another is circle to get it set the character to o

```
plt.plot(x, y, marker = 'o')
```



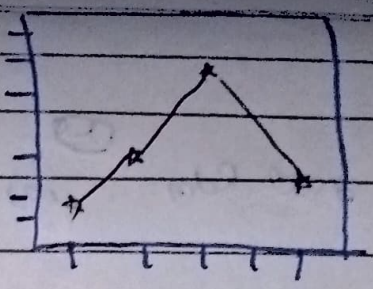
→ To get a Triangle marker make the character to v

```
plt.plot(x, y, marker = 'v')
```



→ To get a star set the character to *

plt.plot(x, y, marker = '*')



#To change or set the size of marker

→ use the markersize keyword

→ The size can be anything our wish

plt.plot(x, y, marker = '.', markersize = 20)

→ It can also be abbreviated to ms meaning markersize

# To change the color of the marker

→ use the markerfacecolor = "" #to change face of the marker

→ You can use

- RGB values
- Name of the colour
- hexodecimal values
- HSL valyes

→ You can also make this shorter 'mfc' meaning markerface color.

→ Using shortforms is good but not readable so don't try to use them.

→ You can also change the edge color of the marker use the markeredgecolor = "" #to change the color of edge

→ the short form is mec

# Customizing the Lines

## ① Line style:-

- You can change the style of the line shown
- The default is solid

→ By the keyword `linestyle = " "`

eg `linestyle = "dashed"` eg `linestyle = "dotted"` eg `linestyle = "dashdot"`

eg `linestyle = None` → only point no line

## ② line width

Key word :- Linewidth

- Normal value is 1

## ③ line color

Key word `color = " "`

- Select any color you want ← instructions similar to earlier

# More than one line to Customize

eg
```
x = np.array([2023, 2024, 2025, 2026])
y1 = np.array([15, 25, 30, 20])
y2 = np.array([17, 23, 39, 5])

plt.plot(x, y1, Customizations)

plt.plot(x, y2,)

plt.show()
```

→ Here the second line wont have the customizations similar to ①

so you can copy paste it 😁

⇒ Create a dictionary with Key-value pairs

```
line_style = dict(Customizations)
plt.plot(x, y1, **line_style)
plt.plot(x, y2, **line_style)
```

→ It would be like to unpack the entire dictionary as if each key-value pair is passed as individual keyword arguments

## Labels :-

# → Setting a title for the chart :-

→ The `plt.title(" ")` is the function which is used to set the ~~title f~~ title for the plot

→ pass the title as a string

# Customization of title :-

→ To do this we can pass in some keyword arguments.

① → font size :- change the font size of the title

② → family :- To change the font family

③ fontweight:- ="bold",

→ This makes the font to be bold

④ color:-

→ used to set the color of the Title

→ Labels:-

→ These are like the Quantites which are present at the coordinates

→ X Label → set the Label for x-axis

→ Y Label → sets the Label for Y-axis

eg `PLt.xLabel("Years")`

→ You can change the fontsize, font family, fontweight and color same as title

→ The coordinates of the graphs ie ticks are incrementing eachtime by -5 to not make it. do like this we can do

`PLt.xticks(x)` → which makes the ticks only at the elements of x

# Customizing the ticks

→ To do this call the tick_params() function re the ticks parameter function

plt.tick_params()

→ To select a certain axis use the keyword argument of axes

axis = "x" → only x axis is selected
axis = "y" → only Y-axis is selected
axis = "both" → Both the axes are selected

→ To customize use the plural keyword argument

plt.tick_params (axis = "both",
colors = "Blue")