

# Python

## Unit 2

By

Gaurav Siddharth

# Python Decision making and Loops

---

- Write conditional statements using:

If statement, if else statement, elif statement and Boolean expressions, While loop, For loop, Nested Loop, Infinite loop, Break statement, Continue statement, Pass statement.

- Use for and while loops along with useful built-in functions to iterate over and manipulate lists, sets, and dictionaries
- Plotting data, Programs using decision making and loops

---

## Decision control instructions

# if condition

---

- By default, statements in the python script are executed sequentially from the first to the last. This sequential process can be altered by using decision control statements.

- Syntax-

if [Boolean expression]:

    statement1

    statement2

...

- The 'if' block starts from the new line after ':' and all the statements under the 'if' condition starts with an increased indentation.
- The 'if' block gets executed only if the Boolean expression evaluates to True.
- To end the block or 'if', decrease the indentation.
- If multiple statements have to be executed within the 'if' block, then they must be written with the same level of indent.
- If all the statements are not in the same indentation, then it will raise an Indentation Error.

## if condition Cont.

---

- The statements with the same indentation level as 'if' condition will not be consider in the 'if' block.
- Don't mix tabs and spaces. They may appear ok on screen, but would be reported as error. In new version of the software, it will not result in error.
- Note that if we have multiple 'if' condition, then each 'if' block can contain statement with different indentation.

Eg.

```
if [Boolean expression]:
```

```
    statements
```

```
if [Boolean expression]:
```

```
    statements
```

```
if [Boolean expression]:
```

```
    statements
```

# else Condition

- Along with the 'if' condition, the 'else' condition can be optionally used to define an alternate block of statements to be executed if the boolean expression in the 'if' condition evaluates to False.

- Syntax:

if [Boolean expression]:

statement1

statement2

...

else:

statement1

statement2

...

- Make sure that the indented block for 'else' condition starts after the ':' symbol written after else, similar to 'if' condition.
- Note that you cannot have multiple 'else' blocks, and it must be the last block for a particular 'if' condition.
- 'else' block can have different indent compared to 'if' block.

# elif Condition

---

- The 'elif' condition is used to include multiple conditional expressions after the 'if' condition or between the 'if' and 'else' conditions.

- Syntax:

if [Boolean expression]:

    [statements]

elif [Boolean expression]:

    [statements]

elif [Boolean expression]:

    [statements]

else:

    [statements]

- The 'elif' block is executed if the specified condition evaluates to True.
- If the 'if' condition evaluates to False, then it will evaluate the 'elif' blocks and execute the 'elif' block whose expression evaluates to True. If multiple 'elif' conditions become True, then the first 'elif' block will be executed.

## elif Condition Cont.

---

- All the 'if' , 'elif' , and 'else' conditions must start from the same indentation level, otherwise it will raise the Indentation Error. However, their block can have different indent.
- Any non-zero number (positive, negative, integer, float) is treated as True, and 0 as False.
- Condition is built using relation operators <, >, <=, >=, ==, !=



# Nested if, elif, else Conditions

- In nested 'if', 'elif' , and 'else' condition, the inner condition must be with increased indentation than the outer condition, and all the statements under the one block should be with the same indentation.

- Syntax:

if [Boolean expression]:

    if [Boolean expression]:

        [statements]

    else:

        if [Boolean expression]:

            [statements]

        elif [Boolean expression]:

            [statements]

elif [Boolean expression]:

    [statements]

else:

    [statements]

# Receiving Input

---

- The way `print( )` function is used to output values on screen, `input( )` built-in function can be used to receive input values from keyboard.
- `Input( )` function returns a string, i.e. if 23 is entered it returns '23'. So if we wish to perform arithmetic on the number entered, we need to convert the string into int or float.

## pass Statement

---

- `pass` statement is intended to do nothing on execution. Hence it is often called a no-op instruction.

---

## Repetition control instructions

# While Loop

---

- Python uses the ‘while’ and ‘for’ keywords to constitute a conditional loop, by which repeated execution of a block of statements is done until the specified boolean expression evaluates to true. When it turns out to be False, the program will exit the loop.

- Syntax:

while [Boolean expression]:

    statement1

    statement2

...

- All the statements in the body of the loop must start with the same indentation, otherwise it will raise a Indentation Error.
- Use the ‘break’ keyword to exit the ‘while’ loop at some condition. Use the ‘if’ condition to determine when to exit from the ‘while’ loop.
- Use the ‘continue’ keyword to start the next iteration and skip the statements after the ‘continue’ statement on some conditions.

## While Loop Cont.

---

- The 'else' block can follow the 'while' loop. The 'else' block will be executed when the boolean expression of the 'while' loop evaluates to False.

- Syntax:

while [Boolean expression]:

statement1

statement2

...

else:

statement1

statement2

...

- If the 'while' loop is terminated abruptly using a break statement then the 'else' block will not get executed.

# While Loop Cont.

---

- A 'while' loop can be used in following three situations:
  1. Repeat a set of statements till a condition remains True.
  2. Repeat a set of statements a finite number of times.
  3. Iterate through a string, list and tuple.

# For Loop

- 'For' loop is used to iterate over elements of a sequence such as string, tuple, list, set or dictionary. It has two forms:

- Syntax:

```
for x in sequence:
```

```
    statement1
```

```
    statement2
```

```
    ...
```

- Syntax:

```
for x in sequence:
```

```
    statement1
```

```
    statement2
```

```
    ...
```

```
else:
```

```
    statement1
```

```
    statement2
```

```
    ...
```

## For Loop Cont.

- Similar to 'while' loop, break and continue statements can be used in 'for' loop.
- 'else' block is optional. If present, it is executed if loop is not terminated abruptly using break.
- A 'for' loop can be used in following two situations:
  1. Iterate through a string, list, tuple, set or dictionary.
  2. Repeat a set of statements a finite number of times.
- The body of the 'for' loop is executed for each member element in the sequence. Hence, it doesn't require explicit verification of a boolean expression controlling the loop (as in the 'while' loop).
- 'for' loop iterates over the dictionary using the items( ) method.
- To repeat a set of statements a finite number of times a built-in function range( ) is used.
- The range class is an immutable sequence type. The range ( ) function generates a sequence of integers and cannot generate a sequence of floats.



## For Loop Cont.

---

- Syntax of range

`range(start, stop, step)`

start (inclusive) to stop (exclusive)

- Nested loops for both 'while' and 'for' loop is possible.
- If while iterating through a collection using a 'for' loop and we wish get an index of the item as well, then we should use the built-in `enumerate( )` function.

# Infinite loop

---

- 'while' loop is used to create an infinite loop.
- In Boolean expression of 'while' loop write any 'non-zero' number or 'True' to create an infinite loop.

# Useful

---

- `end = ' '` in `print(i, end=' ')` prints a space after printing `i` in each iteration.
- Unicode for characters:
  - 0-9 is 48-57
  - A-Z is 65-90
  - a-z is 97-122.

# Some of the Python Built in functions

Function	Accepts	Return Type
ascii()	Varies	str
bin()	number: int	str
bytes()	Varies	bytes
chr()	i: int i>=0 i<=1114111	str
hex()	number: int	str
int()	Varies	int
oct()	number: int	str
ord()	c: str len(c) == 1	int
str()	Varies	str

# Examples: Python Built in functions

---

```
>>> ascii("abcdefg")
```

```
““abcdefg””
```

```
>>> ascii((1, 2, 3))
```

```
'(1, 2, 3)'
```

```
>>> bin(400)
```

```
'0b110010000'
```

```
>>> [bin(i) for i in [1, 2, 4, 8, 16]]
```

```
['0b1', '0b10', '0b100', '0b1000', '0b10000']
```

```
>>> bytes((104, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100))
```

```
b'hello world'
```

```
>>> bytes(range(97, 123))
```

```
b'abcdefghijklmnopqrstuvwxyz'
```

# Examples: Python Built in functions

---

```
>>> chr(97)
```

```
'a'
```

```
>>> chr(0b01100100) # Binary literal (int)
```

```
'd'
```

```
>>> hex(100)
```

```
'0x64'
```

```
>>> [hex(i) for i in [1, 2, 4, 8, 16]]
```

```
['0x1', '0x2', '0x4', '0x8', '0x10']
```

```
>>> int('11')
```

```
11
```

```
>>> int('11', base=2)
```

```
3
```

## Examples: Python Built in functions

---

```
>>> ord("a")
```

```
97
```

```
>>> [ord(i) for i in "hello world"]
```

```
[104, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100]
```

# Problem

---

- Write a program that prints numbers from 1 to 10 using an infinite loop. All numbers should get printed in the same line.



# Solution

---

```
i = 1
while 1 :
    print(i, end = ' ')
    i += 1
    if i > 10 :
        break
```

# Problem

---

- Write a program using while loop that receives 3 sets of values of p, n and r and calculates simple interest for each. where p is principle amount, n is time period, r is rate of interest,

# Solution

---

```
i = 1
while i <= 3 :
    p = float(input('Enter value of p: '))
    n = int(input('Enter value of n: '))
    r = float(input('Enter value of r: '))
    si = p * n * r / 100
    print('Simple interest = Rs. ' + str (si))
    i = i + 1
```

# Problems

- a) `j = 1`  
    `while j <= 10 :`  
        `print(j)`  
        `j++`
- b) `while true :`  
    `print('Infinite loop')`
- c) `lst = [10, 20, 30, 40, 50]`  
    `for count = 1 to 5 :`  
        `print(lst[ i ])`
- d) `i = 15`  
    `not while i < 10 :`  
        `print(i)`  
        `i -= 1`
- e) `for x in range(65, 91) :`  
    `print(chr(x), end=' ')`

---

## Plotting Data

# Plot

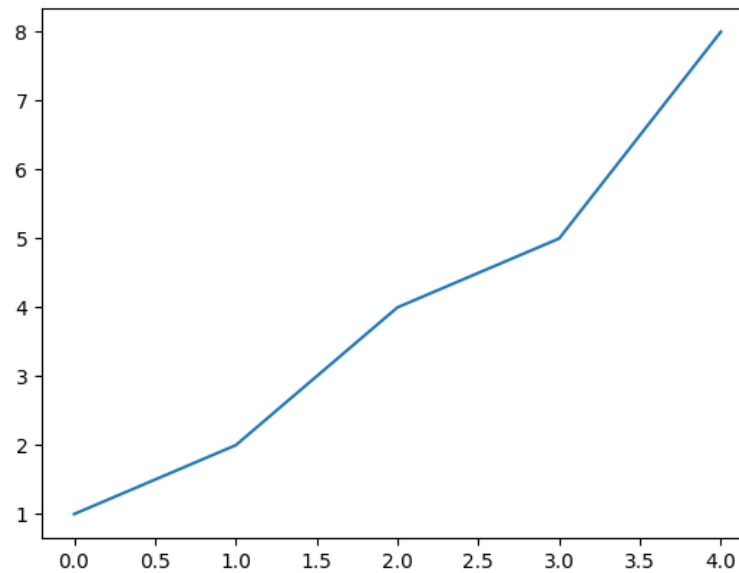
# Importing relevant modules

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

```
x=[1, 2, 4, 5, 8]
```

```
plt.plot(x)
```

```
plt.show()
```



# Plotting one data w.r.t. other

# Importing relevant modules

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

```
x=[1, 2, 4, 5, 8]
```

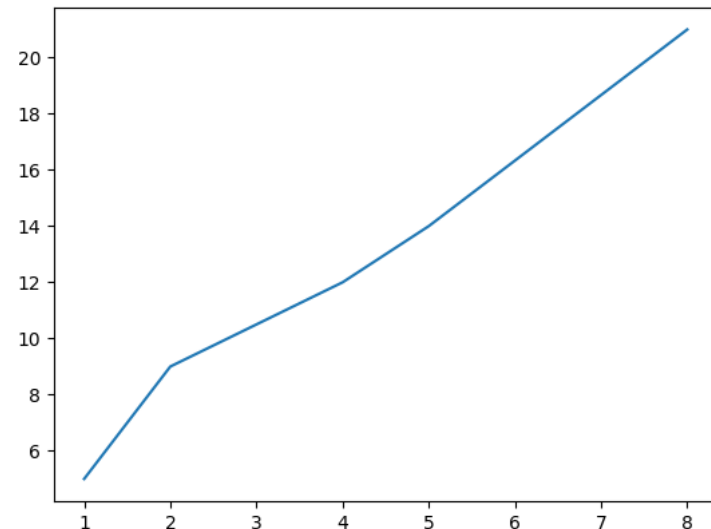
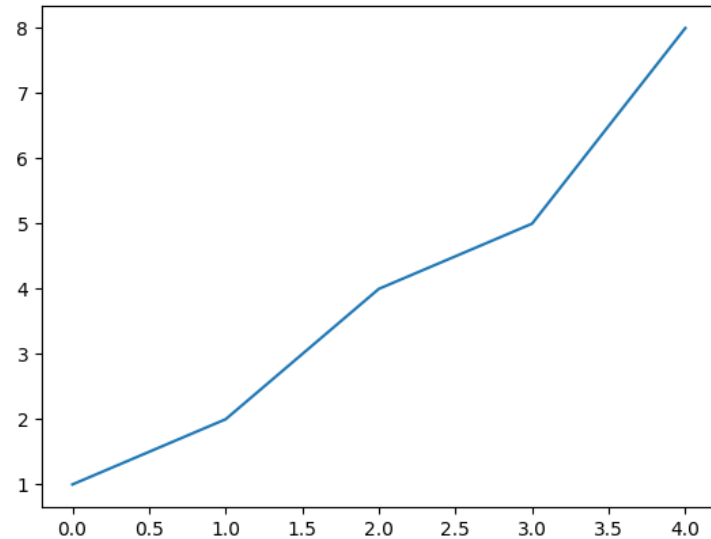
```
plt.plot(x)
```

```
plt.show()
```

```
y=[5, 9, 12, 14, 21]
```

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

```
plt.show()
```



# Plotting multiple data in single plot

# Importing relevant modules

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

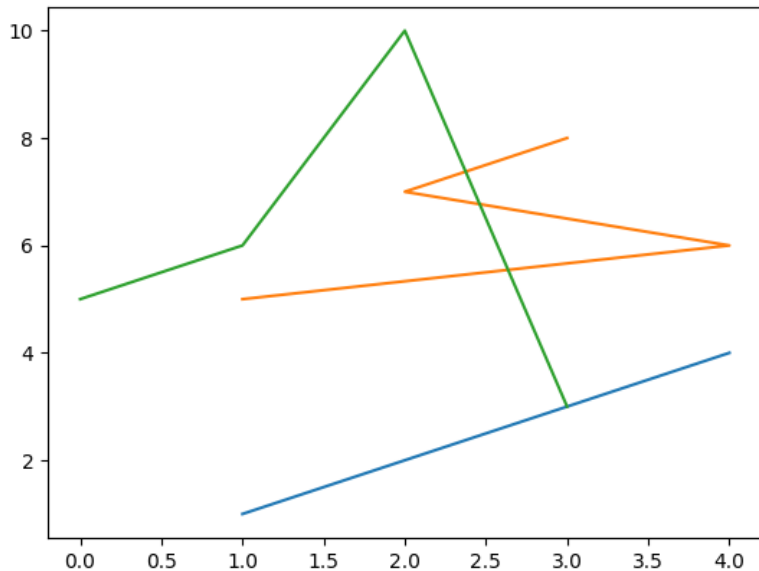
```
plt.plot([1,2,3,4], [1,2,3,4])
```

```
plt.show()
```

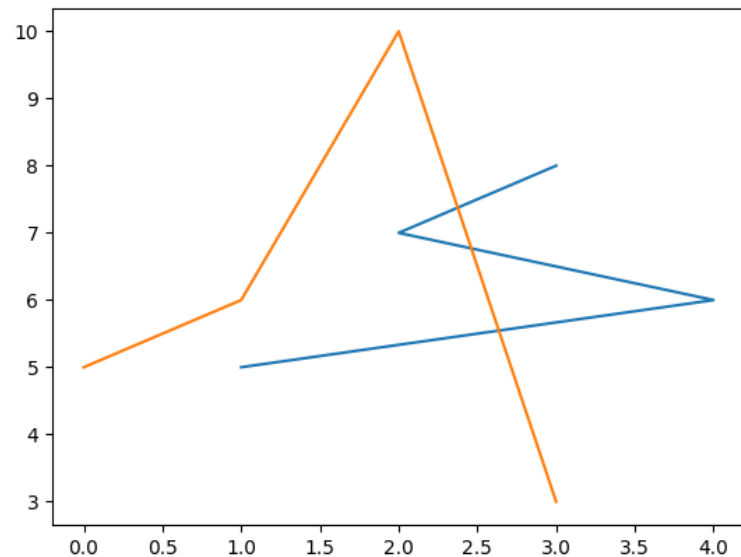
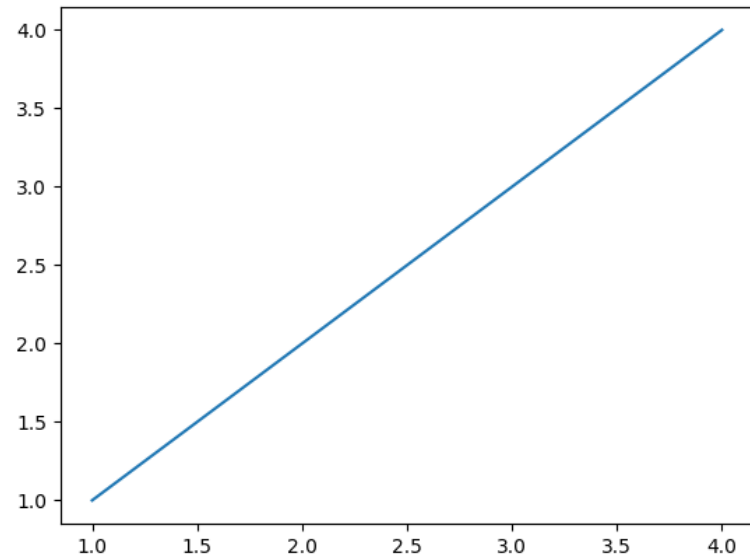
```
plt.plot([1,4,2,3], [5,6,7,8])
```

```
plt.plot([5,6,10,3])
```

```
plt.show()
```



Just have single `plt.show()` at the end.





# Plotting multiple data in a plot of our choice

# Importing relevant modules

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

```
plt.figure(1)
```

```
plt.plot([1,2,3,4], [1,2,3,4])
```

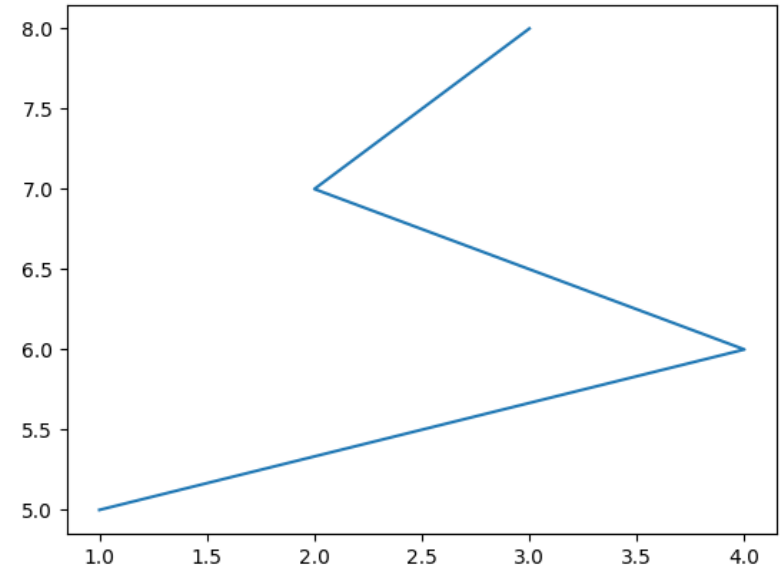
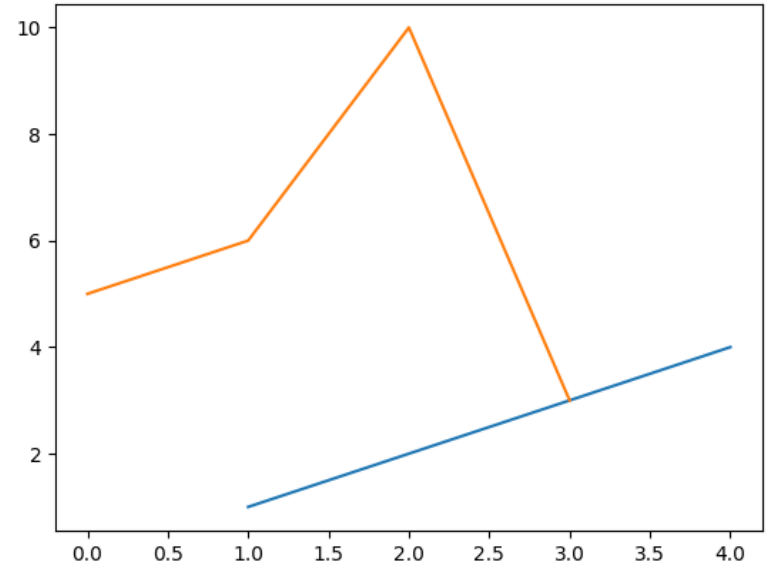
```
plt.figure(2)
```

```
plt.plot([1,4,2,3], [5,6,7,8])
```

```
plt.figure(1)
```

```
plt.plot([5,6,10,3])
```

```
plt.show()
```



# Plotting multiple data with different x-axis values

# Importing relevant modules

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

```
plt.figure(1)
```

```
plt.plot([1,2,3,4], [1,2,3,4])
```

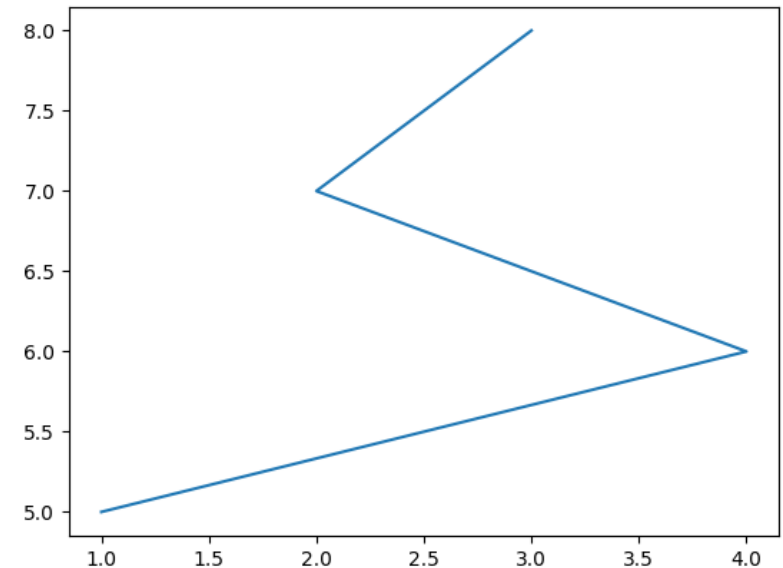
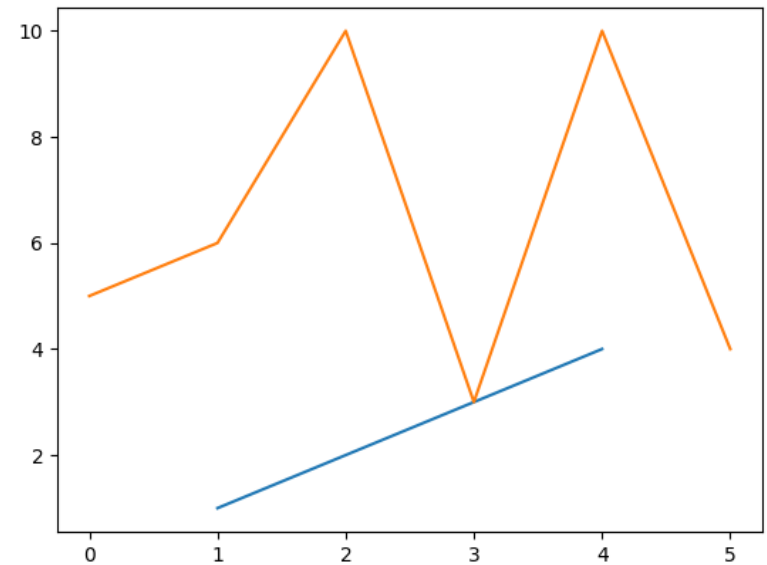
```
plt.figure(2)
```

```
plt.plot([1,4,2,3], [5,6,7,8])
```

```
plt.figure(1)
```

```
plt.plot([5,6,10,3,10,4])
```

```
plt.show()
```



# Plotting data with certain features

Color and Line Width

# Importing relevant modules

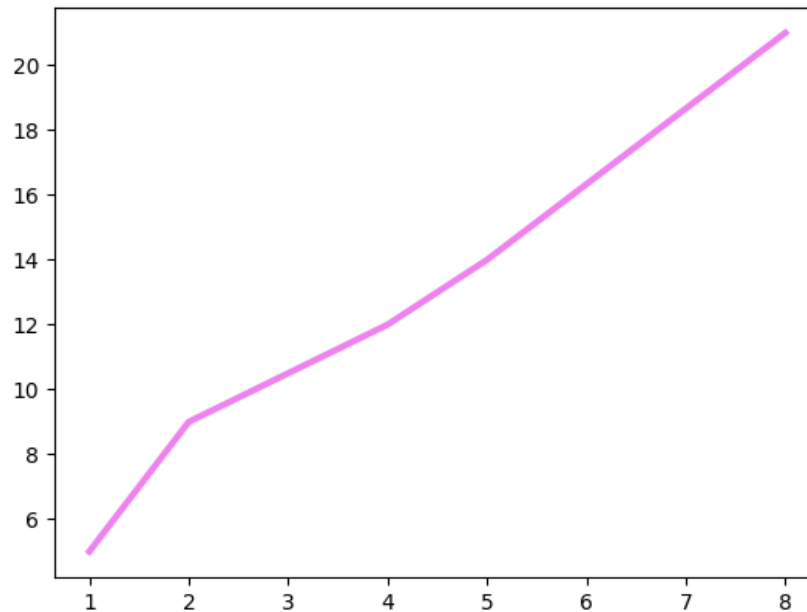
```
import matplotlib.pyplot as plt
```

```
x=[1, 2, 4, 5, 8]
```

```
y=[5, 9, 12, 14, 21]
```

```
plt.plot(x, y, c="violet", linewidth=3)
```

```
plt.show()
```



# Plotting data with certain features Cont.

# Importing relevant modules

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

```
x=[1, 2, 6, 10, 17]
```

```
y=[5, 9, 12, 14, 21]
```

```
plt.plot(x, y, c="yellow", linewidth=3, label="Data 1")
```

```
x2=[1,3 ,7, 13, 21]
```

```
y2=[0, 8, 12, 25, 20]
```

```
plt.plot(x2, y2, c="green", linewidth=4, label="Data 2")
```

```
plt.xlabel("X-axis")
```

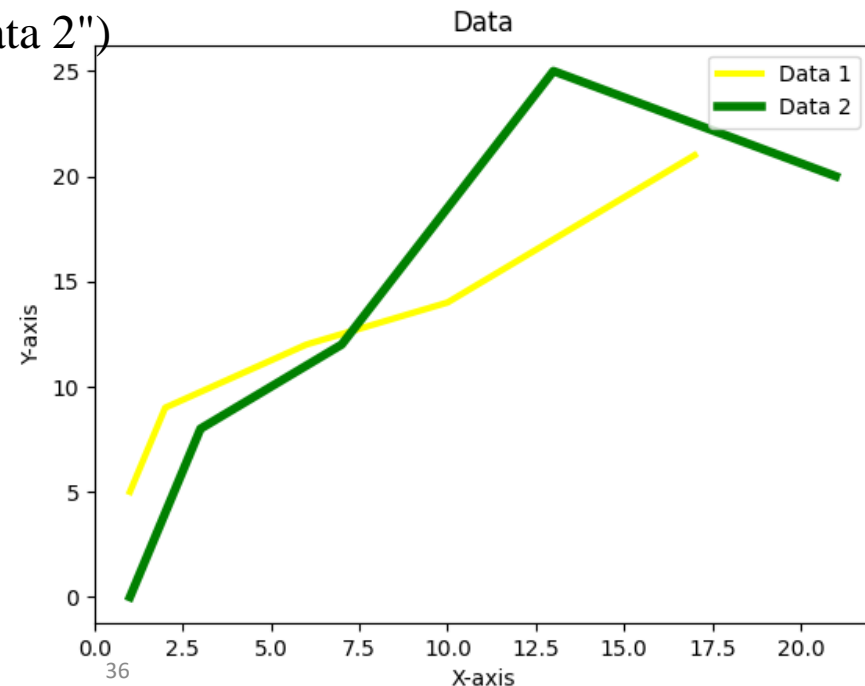
```
plt.ylabel("Y-axis")
```

```
plt.title("Data")
```

```
plt.legend()
```

```
plt.show()
```

Label, Title and Legend



# Plotting data with certain features Cont.

# Importing relevant modules

# Importing relevant modules

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

```
x=[1, 2, 6, 10, 17]
```

```
y=[5, 9, 12, 14, 21]
```

```
plt.plot(x, y, c="yellow", linewidth=3, label="Data 1")
```

```
x2=[1,3 ,7, 13, 21]
```

```
y2=[0, 8, 12, 25, 20]
```

```
plt.plot(x2, y2, c="green", linewidth=4, label="Data 2", linestyle="dashed",
```

```
marker='o', markerfacecolor="grey", markersize=10)
```

```
plt.xlabel("X-axis")
```

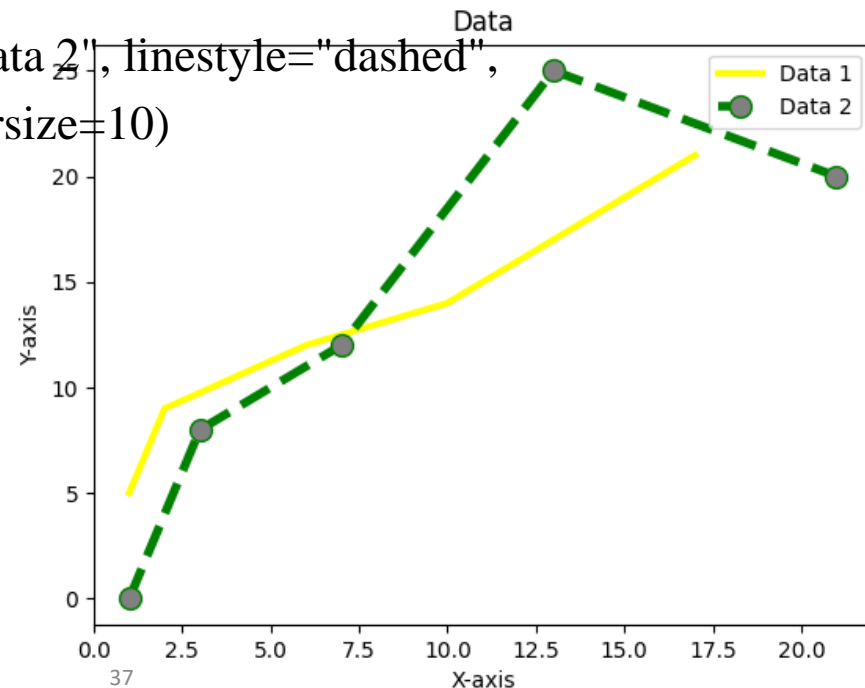
```
plt.ylabel("Y-axis")
```

```
plt.title("Data")
```

```
plt.legend()
```

```
plt.show()
```

Line style, Marker and it's face color and size



# Plotting data with certain features Cont.

# Importing relevant modules

# Importing relevant modules

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

```
x=[1, 2, 6, 10, 17]
```

```
y=[5, 9, 12, 14, 21]
```

```
plt.plot(x, y, c="yellow", linewidth=3, label="Data 1")
```

```
x2=[1,3 ,7, 13, 21]
```

```
y2=[0, 8, 12, 25, 20]
```

```
plt.plot(x2, y2, c="green", linewidth=4, label="Data 2", linestyle="dashed",
```

```
marker='o', markerfacecolor="grey", markersize=10)
```

```
plt.xlabel("X-axis")
```

```
plt.ylabel("Y-axis")
```

```
plt.ylim(2, 21)
```

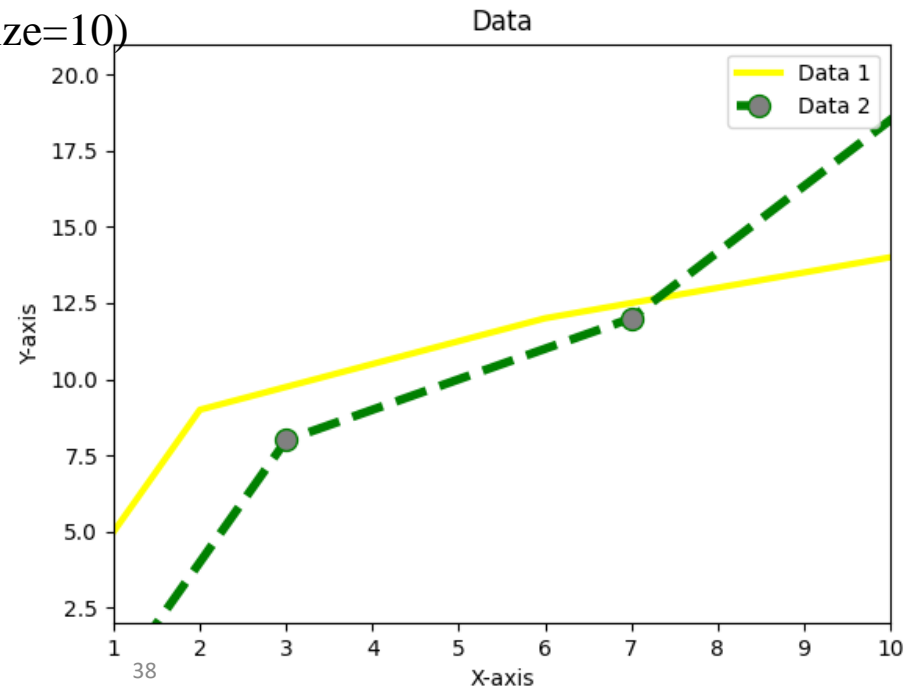
```
plt.xlim(1, 10)
```

```
plt.title("Data")
```

```
plt.legend()
```

```
plt.show()
```

Y and X limit



# Plotting Data involving 'for' loop

# Importing relevant modules

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

```
p=1000
```

```
r=0.05
```

```
t=20
```

```
values=[]
```

```
for i in range(t+1):
```

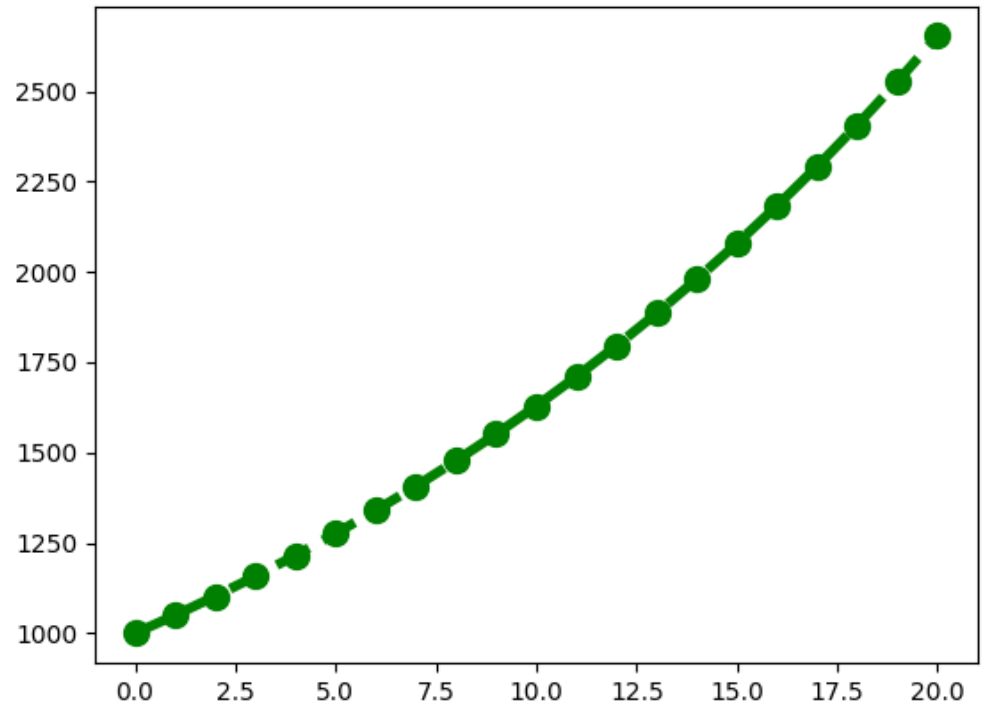
```
    values.append(p)
```

```
    p+=p*r
```























```
plt.plot(values, c="green", linewidth=4, label="Data 2", linestyle="dashed",
```

```
        marker='o', markersize=10)
```

```
plt.show()
```



# Matplotlib Markers

Marker	Symbol	Description	Marker	Symbol	Description
“.”		Point	“8”		Octagon
“,”		Pixel	“s”		Square
“o”		Circle	“p”		Pentagon
“v”		Down Triangle	“P”		Filled Plus
“^”		Up Triangle	“*”		Star
“<”		Left Triangle	“h”		Hexagon 1
“>”		Right Triangle	“H”		Hexagon 2
“1”		Down Tri	“+”		Plus
“2”		Up Tri	“x”		X
“3”		Left Tri	“X”		Filled x
“4”		Right Tri	“D”		Diamond



## Character code for color

Character	Color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

**Line Style-** It is defined by string and can be "solid", "dotted", "dashed" or "dashdot".

# List of plot functions to customise plots

Functions	Description
<code>grid([b, which, axis])</code>	Configure the grid lines.
<code>legend(*args, **kwargs)</code>	Place a legend on the axes.
<code>savefig(*args, **kwargs)</code>	Save the current figure.
<code>show(*args, **kw)</code>	Display all figures.
<code>title(label[, fontdict, loc, pad])</code>	Set a title for the axes.
<code>xlabel(xlabel[, fontdict, labelpad])</code>	Set the label for the x-axis.
<code>xticks([ticks, labels])</code>	Get or set the current tick locations and labels of the x-axis.
<code>ylabel(ylabel[, fontdict, labelpad])</code>	Set the label for the y-axis.
<code>yticks([ticks, labels])</code>	Get or set the current tick locations and labels of the y-axis.

# List of functions to have different types of plots

Functions	Description
<code>plot(*args[, scalex, scaley, data])</code>	Plot x versus y as lines and/or markers.
<code>bar(x, height[, width, bottom, align, data])</code>	Make a bar plot.
<code>boxplot(x[, notch, sym, vert, whis, ...])</code>	Make a box and whisker plot.
<code>hist(x[, bins, range, density, weights, ...])</code>	Plot a histogram.
<code>pie(x[, explode, labels, colors, autopct, ...])</code>	Plot a pie chart.
<code>scatter(x, y[, s, c, marker, cmap, norm, ...])</code>	A scatter plot of x versus y.

# Pandas Plot function

- Pandas objects Series and Data Frame come equipped with their own `.plot()` methods.
- If we have a Series or Data Frame type object, we can call the plot method by writing: `s.plot()` or `df.plot()`.
- The `plot()` method of Pandas accepts a considerable number of arguments that can be used to plot a variety of graphs.

<b>kind =</b>	<b>Plot type</b>
line	Line plot (default)
bar	Vertical bar plot
barh	Horizontal bar plot
hist	Histogram
box	Boxplot
area	Area plot
pie	Pie plot
scatter	Scatter plot

# Plotting Data using data frame

## Line plot

# Importing relevant modules

```
import pandas as pd
```

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

```
df=pd.read_csv("D:\L&T\IIIT Ranchi\code\Data.csv")
```

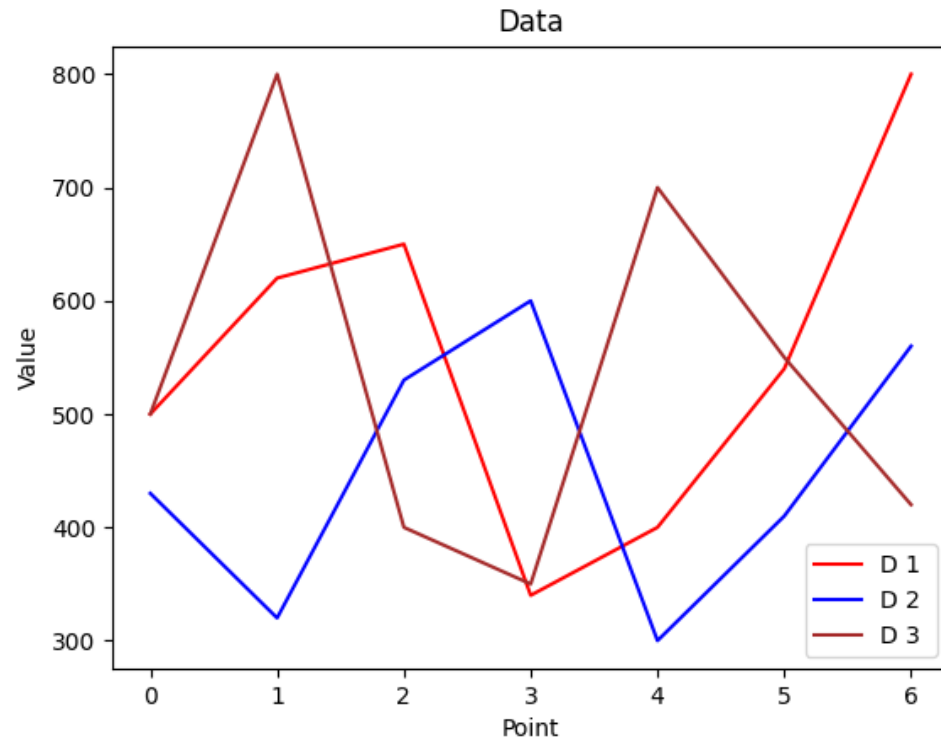
```
df.plot(kind='line', color=['red','blue','brown'])
```

```
plt.title('Data')
```

```
plt.xlabel('Point')
```

```
plt.ylabel('Value')
```

```
plt.show()
```



```
# Importing relevant modules
```

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

```
import pandas as pd
```

```
name=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```

```
height=[115,120.1,135.5,111,138.2,117,122.1,124.5,132.7]
```

```
weight=[50.2,45.6,48.2,55.9,67.5,40.1,58.7,51.6,60.2]
```

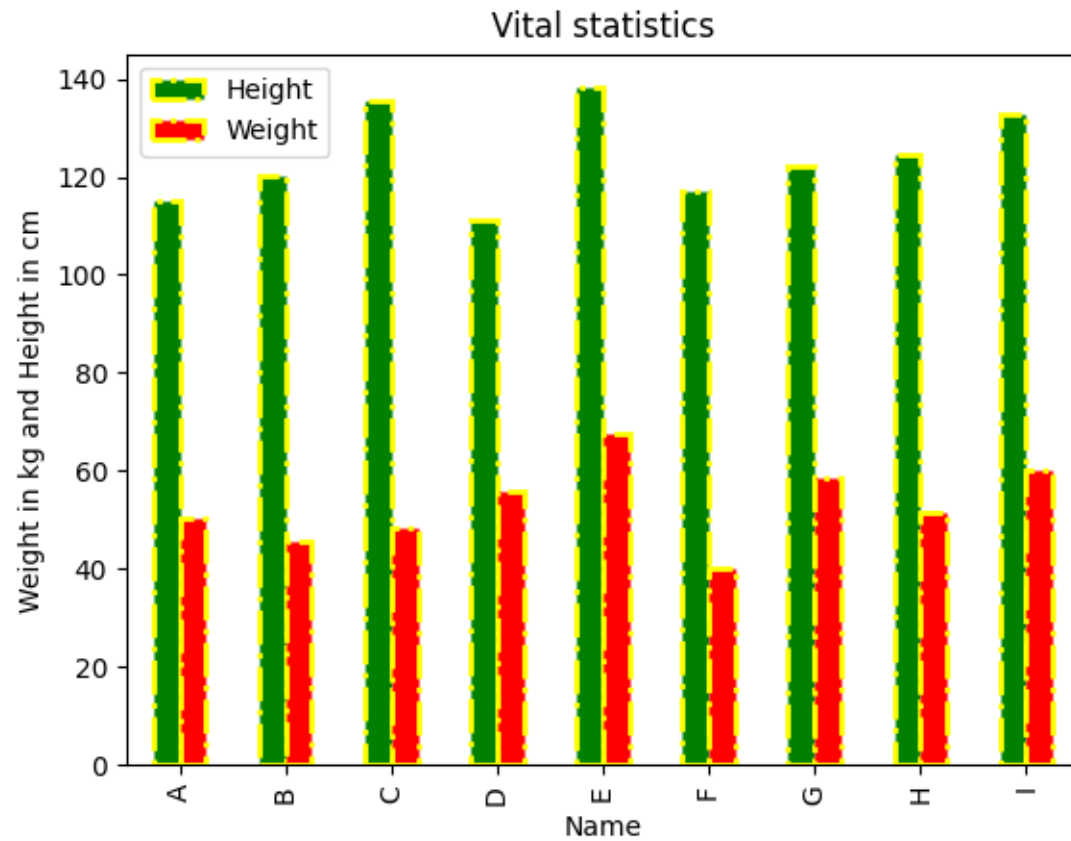
```
df=pd.DataFrame({"Name":name,"Height":height,"Weight":weight})
```

```
df.plot(kind='bar',x= 'Name', title='Vital statistics', color=['green', 'red'],
```

```
    edgecolor='yellow', linewidth=2, linestyle='dashdot')
```

```
plt.ylabel('Weight in kg and Height in cm')
```

```
plt.show()
```



# Problem

---

- Write a program using for loop and break statement to go through the elements in a given list and check whether all the numbers in a list are multiple of 5, if yes then print 'all the numbers are multiples of 5' else print 'all the numbers are not multiples of 5'.

Given list is [10, 35, 55, 66, 20, 45]



# Solution

---

```
for x in [10, 20, 30, 3, 40, 50] :  
    if x % 10 != 0 :  
        print('all the numbers are not multiples of 5')  
        break  
else :  
    print('all the numbers are multiples of 5')
```

# Importing relevant modules

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

```
import pandas as pd
```

```
Data={'name':['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I'],
```

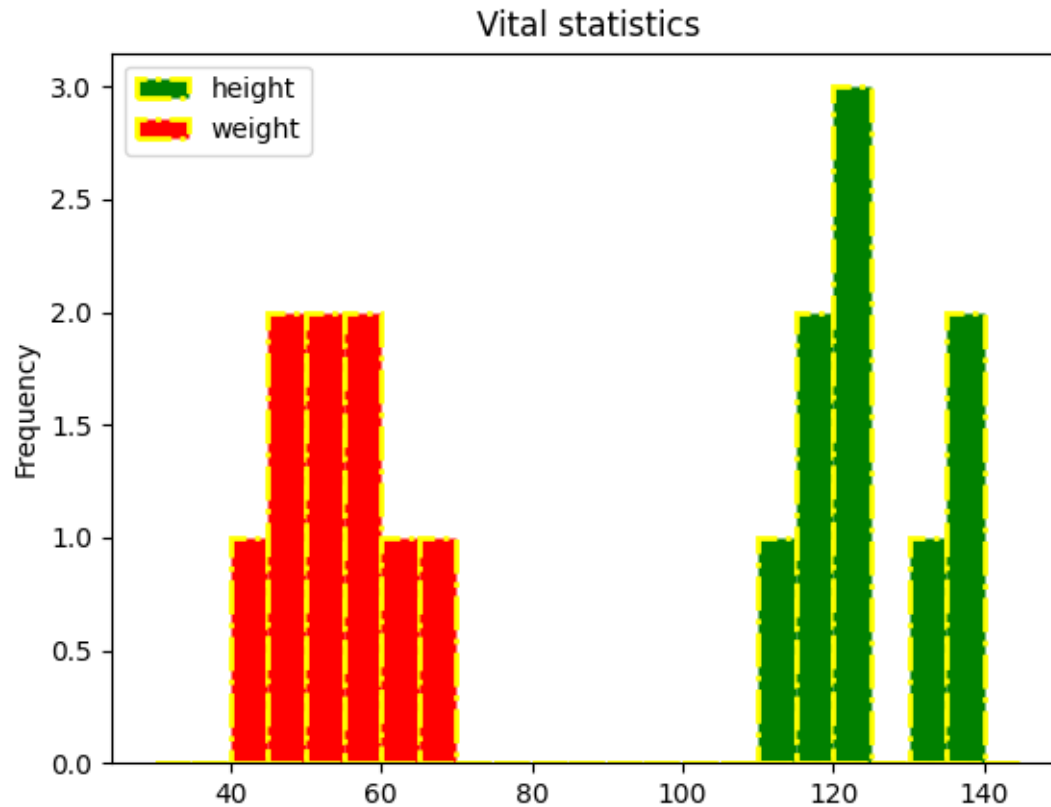
```
'height':[115,120.1,135.5,111,138.2,117,122.1,124.5,132.7],
```

```
'weight':[50.2,45.6,48.2,55.9,67.5,40.1,58.7,51.6,60.2]}
```

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

```
df.plot(kind='hist',x= 'name', bins=range(30,150,5), title='Vital statistics', color=['green',  
'red'], edgecolor='yellow', linewidth=2, linestyle='dashdot')
```

```
plt.show()
```



**fill**- Specify True or False to fill in `df.plot()` function.

**hatch**- To fill each hist with pattern ( '-', '+', 'x', '\\', '\*', 'o', 'O', '.'). By including hatch in `df.plot()` function.

# Importing relevant modules

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

```
import pandas as pd
```

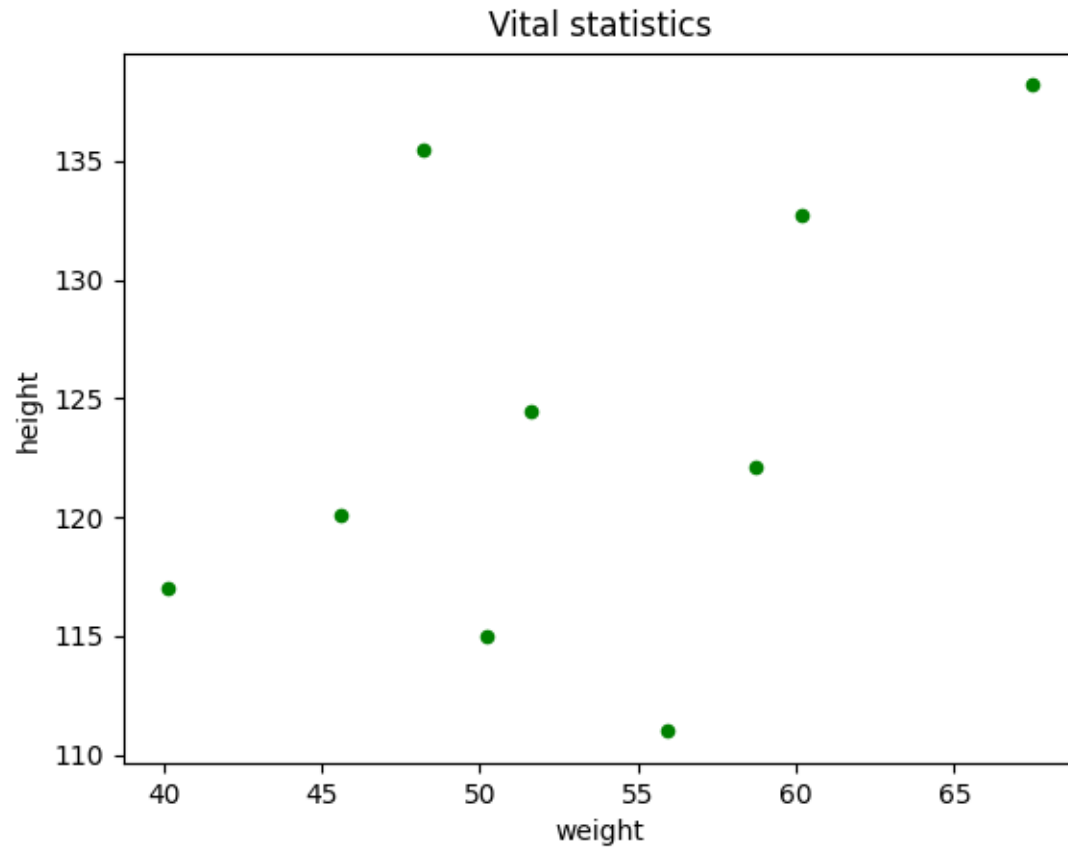
```
Data={'height':[115,120.1,135.5,111,138.2,117,122.1,124.5,132.7],
```

```
'weight':[50.2,45.6,48.2,55.9,67.5,40.1,58.7,51.6,60.2]}
```

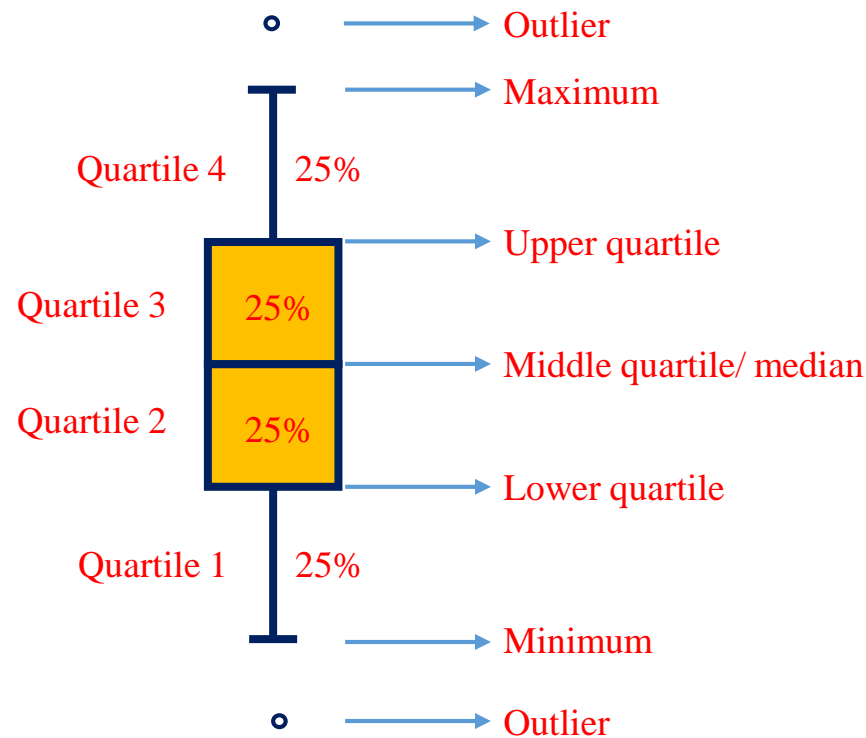
```
df=pd.DataFrame(Data)
```

```
df.plot(kind='scatter', x= 'weight', y='height', title='Vital statistics', color='green')
```

```
plt.show()
```



- A Box Plot is the visual representation of the statistical summary of a given data set.
- The Box plot includes Minimum value, Quartile 1, Quartile 2, Median, Quartile 3, Quartile 4, Maximum value and two outliers.
- An outlier is an observation that is numerically distant from the rest of the data.



# Plotting Data using data frame

## Box plot

# Importing relevant modules

```
import pandas as pd
```

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

```
data= pd.read_csv('Box_data.csv')
```

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

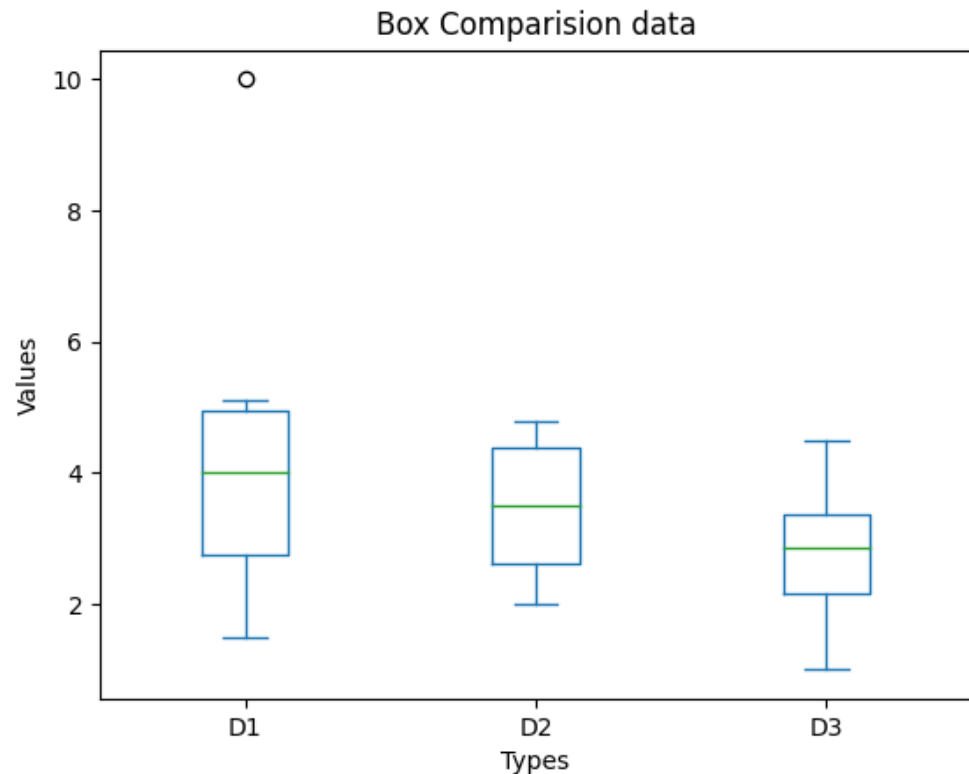
```
df.plot(kind='box',title='Box Comparision data')
```

```
plt.xlabel('Types')
```

```
plt.ylabel('Values')
```

```
plt.show()
```

For having box in horizontal direction specify **vert=False** in df.plot() function.



# Importing relevant modules

```
import pandas as pd
```

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

```
df = pd.DataFrame({'values': [1, 5.4 , 5, 7, 10, 20]} ,
```

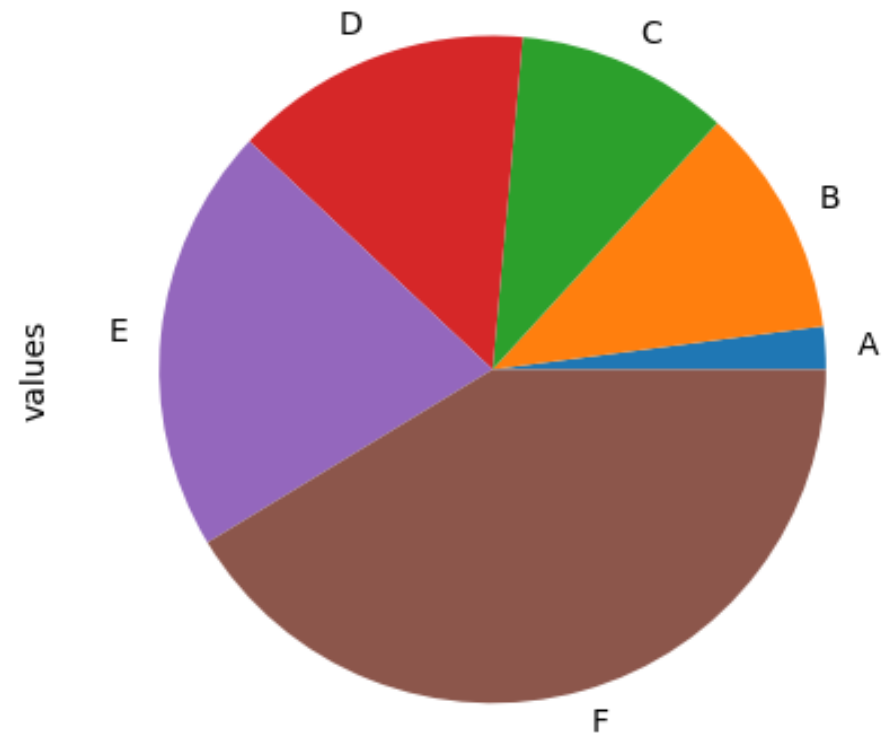
```
index=['A', 'B', 'C', 'D', 'E', 'F'])
```

```
df.plot(kind='pie',y='values', legend=False)
```

```
plt.show()
```

Use **explode** in `df.plot()` function to separate particular sector of the circle. For this define `explode` of each sector before `df.plot ()` function.

Use `autopct="%.2f"` to define % of that part as label





# Problem

---

- Write the following code using for loop-

```
count = 1
```

```
while count <= 10 :
```

```
    print(count)
```

```
    count = count + 1
```

# Problem

---

- Write a program to print first 25 odd numbers using range ( ).

# Problem

---

- Write a program using loop to find the value of one number raised to the power of another by entering two numbers through keyboard.

# Solution

---

```
x = int(input("Enter a number: "))
y = int(input("Enter the power it is raised to: "))
z=1
for i in range(1,y+1):
    z= z * x
print("x raised to power y")
```

***THANK YOU***