In [ ]:

```
########################################################
#             Basics of Python Programming              #
########################################################

# Python is a high-level, interpreted programming language known for its simplicity and readability.
# It emphasizes code readability with its notable use of significant whitespace. Python supports multiple programming paradigms,
# including procedural, object-oriented, and functional programming styles.

# 1. Syntax: Python uses indentation to define code blocks, typically with four spaces per level.

# 2. Variables and Data Types:
#    - Variables: Created by assigning a value using =.
#    - Data Types: Include integers (int), floating-point numbers (float), strings (str), booleans (bool), lists (list), tuples (tuple), dictionaries (dict), etc.

# 3. Control Structures:
#    - Conditional Statements: if, elif, else.
#    - Loops: for loop, while loop.

# 4. Functions:
#    - Defined using def keyword.
#    - Can accept parameters and return values.

# 5. Lists, Tuples, and Dictionaries:
#    - Lists: Ordered, mutable collection of items.
#    - Tuples: Ordered, immutable collection of items.
#    - Dictionaries: Unordered collection of key-value pairs.

# 6. String Manipulation:
#    - Strings can be manipulated using various methods like concatenation, slicing, formatting, etc.

# 7. Input and Output:
#    - input() function to take user input.
#    - print() function to display output.

# 8. Modules and Packages:
#    - Modules: Python files containing reusable code.
#    - Packages: Collection of related modules.

# 9. Exceptions Handling:
#    - try, except, finally blocks to handle exceptions.

# 10. File Handling:
#     - Opening, reading, writing, and closing files using open() function.

# 11. Classes and Objects:
#     - Classes: Blueprint for creating objects.
#     - Objects: Instances of classes.

# 12. Inheritance and Polymorphism:
#     - Inheritance: Subclass inheriting properties and behaviors from a superclass.
#     - Polymorphism: Ability of objects to take on multiple forms.

# 13. Modules for Specific Tasks:
#     - math for mathematical operations.
#     - random for generating random numbers.
#     - datetime for date and time operations.
#     - pandas for DataFrame and Series.
```

In [1]:

```
# Addition (Sum)
```

```
a = 5
b = 3
sum_result = a + b
print("Sum:", sum_result)

# Multiplication
a = 5
b = 3
multiply_result = a * b
print("Multiplication:", multiply_result)

# Concatenation
str1 = "Hello"
str2 = "World"
concat_result = str1 + str2
print("Concatenation:", concat_result)

# Division
a = 10
b = 3
division_result = a / b
print("Division:", division_result)

# Integer Division
a = 10
b = 3
integer_division_result = a // b
print("Integer Division:", integer_division_result)

# Subtraction
a = 10
b = 3
subtraction_result = a - b
print("Subtraction:", subtraction_result)

# Exponentiation
a = 2
b = 3
exponentiation_result = a ** b
print("Exponentiation:", exponentiation_result)
```

```
Sum: 8
Multiplication: 15
Concatenation: HelloWorld
Division: 3.3333333333333335
Integer Division: 3
Subtraction: 7
Exponentiation: 8
```

In [2]:

```
# Addition (Sum)
a = int(input("Enter the first number for addition: "))
b = int(input("Enter the second number for addition: "))
sum_result = a + b
print("Sum:", sum_result)

# Multiplication
a = int(input("Enter the first number for multiplication: "))
b = int(input("Enter the second number for multiplication: "))
multiply_result = a * b
print("Multiplication:", multiply_result)

# Concatenation
str1 = input("Enter the first string for concatenation: ")
str2 = input("Enter the second string for concatenation: ")
concat_result = str1 + str2
print("Concatenation:", concat_result)

# Division
a = int(input("Enter the numerator for division: "))
```

```
b = int(input("Enter the denominator for division: "))
division_result = a / b
print("Division:", division_result)

# Integer Division
a = int(input("Enter the numerator for integer division: "))
b = int(input("Enter the denominator for integer division: "))
integer_division_result = a // b
print("Integer Division:", integer_division_result)

# Subtraction
a = int(input("Enter the first number for subtraction: "))
b = int(input("Enter the second number for subtraction: "))
subtraction_result = a - b
print("Subtraction:", subtraction_result)

# Exponentiation
a = int(input("Enter the base number for exponentiation: "))
b = int(input("Enter the exponent for exponentiation: "))
exponentiation_result = a ** b
print("Exponentiation:", exponentiation_result)
```

```
Enter the first number for addition: 1
Enter the second number for addition: 2
Sum: 3
Enter the first number for multiplication: 3
Enter the second number for multiplication: 4
Multiplication: 12
Enter the first string for concatenation: 5
Enter the second string for concatenation: 6
Concatenation: 56
Enter the numerator for division: 7
Enter the denominator for division: 8
Division: 0.875
Enter the numerator for integer division: 9
Enter the denominator for integer division: 1
Integer Division: 9
Enter the first number for subtraction: 2
Enter the second number for subtraction: 3
Subtraction: -1
Enter the base number for exponentiation: 4
Enter the exponent for exponentiation: 5
Exponentiation: 1024
```

In [3]:

```
# Function to check voting eligibility
def check_voting_eligibility(age):
    if age >= 18:
        print("You are eligible to vote.")
    else:
        print("You are not eligible to vote yet.")


# Voting Eligibility Check
age = int(input("Enter your age to check voting eligibility: "))
check_voting_eligibility(age)
```

```
Enter your age to check voting eligibility: 18
You are eligible to vote.
```

In [ ]:

```
#python is basics of computer and computer is basic need of human
```

In [ ]:

```
#1-D,contain int, float, list, string, etc
#index = The data label associated with a particular value is called its index
#Column = The vertical label associated with a particular value is called its column
```

In [1]:

```
#main importing all libraries for this stuff
import pandas as pd #pandas full form = panel data
import numpy as np
```

In [2]:

```
print("----------------------------------------------")
print("----------#simple value wali sereies----------")
print("----------------------------------------------")

# Creating a pandas Series from a list of integers
List = [11, 12, 13, 14, 15]
S1 = pd.Series(List)
print(S1)

print()
print("----------------------------------------------")
print("-------#simple character wali series----------")
print("----------------------------------------------")

# Creating a pandas Series from a string
List2 = "Op"
S2 = pd.Series(List2)
print(S2)

print()
print("----------------------------------------------")
print("-------#simple index wali series--------------")
print("----------------------------------------------")

# Creating a pandas Series with custom indices
S3 = pd.Series(["Arnav", "XO", "XO"], index=[1, 2, 3])
print(S3)

print()
print("----------------------------------------------")
print("-------#simple numpy wali series--------------")
print("----------------------------------------------")

# Creating a pandas Series from a numpy array
NP = np.array([11, 12, 13, 14, 15])
S4 = pd.Series(NP)
print(S4)

print()
print("----------------------------------------------")
print("-------#simple dictionary wali series---------")
print("----------------------------------------------")

# Creating a pandas Series from a dictionary
dict = {'India': 'NewDelhi', 'UK': 'London', 'Japan': 'Tokyo'}
S5 = pd.Series(dict)
print(S5)
```

```
----------------------------------------------
----------#simple value wali sereies----------
----------------------------------------------
0    11
1    12
2    13
3    14
```

```
3    14
4    15
dtype: int64

----------------------------------------------
-------#simple character wali series----------
----------------------------------------------
0    Op
dtype: object

----------------------------------------------
-------#simple index wali series--------------
----------------------------------------------
1    Arnav
2       XO
3       XO
dtype: object

----------------------------------------------
-------#simple numpy wali series--------------
----------------------------------------------
0    11
1    12
2    13
3    14
4    15
dtype: int32

----------------------------------------------
-------#simple dictionary wali series---------
----------------------------------------------
India    NewDelhi
UK         London
Japan       Tokyo
dtype: object
```

In [3]:

```
# Series for indexing question
print("-------------------------------------------------------")
print("-------#Series for indexing question-------------------")
print("-------------------------------------------------------")

# Creating a pandas Series with numerical values
seriesNum = pd.Series([10, 20, 30])
print(seriesNum)

# Accessing the element at index 2, which is 30
print("\nAccessing the element at index 2, which is 30:")
print(seriesNum[2])

# Series with indexes
print()
print("------------------------------------------------------")
print("------#Series with indexes----------------------------")
print("------------------------------------------------------")

# Creating a pandas Series with numerical values and custom index labels
seriesMnths = pd.Series([2, 3, 4], index=["Feb", "Mar", "Apr"])
print(seriesMnths)

# Accessing the element with index label "Mar", which is 3
print("\nAccessing the element with index label 'Mar', which is 3:") #/n symbolize baxodi
khatam
print(seriesMnths["Mar"])

# Series for splicing question
print("\n------------------------------------------------------")
print("-------------#Series for splicing question------------")
print("------------------------------------------------------")
```

```python
# Creating a pandas Series with string values and custom index labels
seriesCapCntry = pd.Series(['NewDelhi', 'WashingtonDC', 'London', 'Paris'],
                            index=['India', 'USA', 'UK', 'France'])
print(seriesCapCntry)

# Accessing the element with index label 'India' that is 'NewDelhi'
print("\nAccessing the element with index label 'India' that is 'NewDelhi':")
print(seriesCapCntry['India'])

# Accessing the element at index 1, which is 'WashingtonDC'
print("\nAccessing the element at index 1, which is 'WashingtonDC':")
print(seriesCapCntry[1])

# Accessing elements at indexes 3 and 2, which are 'Paris' and 'London' respectively
print("\nAccessing elements at indexes 3 and 2, which are 'Paris' and 'London' respectively:")
print(seriesCapCntry[[3, 2]])

# Accessing elements with index labels "UK" and "USA", which are "London" and "WashingtonDC" respectively
print("\nAccessing elements with index labels 'UK' and 'USA', which are 'London' and 'WashingtonDC' respectively:")
print(seriesCapCntry[['UK', 'USA']])

# Modifying the index labels of the seriesCapCntry Series
seriesCapCntry.index = [10, 20, 30, 40]
print("\nModified Series with updated index labels:")
print(seriesCapCntry)  # Outputting the modified seriesCapCntry Series with the updated index labels

# Series for splicing question too
print()
print("------------------------------------------------------")
print("-------------#Series for splicing question too--------")
print("------------------------------------------------------")

# Re-creating the seriesCapCntry Series
seriesCapCntry = pd.Series(['NewDelhi', 'WashingtonDC', 'London', 'Paris'],
                            index=['India', 'USA', 'UK', 'France'])
print(seriesCapCntry)

# Slicing using index positions, excluding the value at index position 3
print("\nSlicing using index positions (excluding the value at index position 3):")
print(seriesCapCntry[1:3])

# Slicing using index labels from 'USA' to 'France'
print("\nSlicing using index labels from 'USA' to 'France':")
print(seriesCapCntry['USA': 'France'])

# Reversing the order of elements in the Series
print("\nReversing the order of elements in the Series:")
print(seriesCapCntry[::-1])

# Creating a pandas Series with numerical values and custom index labels
seriesAlph = pd.Series(np.arange(10, 16, 1), index=['a', 'b', 'c', 'd', 'e', 'f'])

# Modifying values using index positions
seriesAlph[1:3] = 50

# Modifying values using index labels
seriesAlph['c':'e'] = 500

print("\nModified Series after value modifications:")
print(seriesAlph)
```

```
-------------------------------------------------------
-------#Series for indexing question-------------------
-------------------------------------------------------
0    10
1    20
2    30
dtype: int64
```

```
  --
Accessing the element at index 2, which is 30:
30

------------------------------------------------------
------#Series with indexes----------------------------
------------------------------------------------------
Feb    2
Mar    3
Apr    4
dtype: int64

Accessing the element with index label 'Mar', which is 3:
3

------------------------------------------------------
-------------#Series for splicing question------------
------------------------------------------------------
India         NewDelhi
USA       WashingtonDC
UK              London
France           Paris
dtype: object

Accessing the element with index label 'India' that is 'NewDelhi':
NewDelhi

Accessing the element at index 1, which is 'WashingtonDC':
WashingtonDC

Accessing elements at indexes 3 and 2, which are 'Paris' and 'London' respectively:
France     Paris
UK        London
dtype: object

Accessing elements with index labels 'UK' and 'USA', which are 'London' and 'WashingtonDC
' respectively:
UK           London
USA    WashingtonDC
dtype: object

Modified Series with updated index labels:
10        NewDelhi
20    WashingtonDC
30          London
40           Paris
dtype: object

------------------------------------------------------
-------------#Series for splicing question too--------
------------------------------------------------------
India         NewDelhi
USA       WashingtonDC
UK              London
France           Paris
dtype: object

Slicing using index positions (excluding the value at index position 3):
USA    WashingtonDC
UK           London
dtype: object

Slicing using index labels from 'USA' to 'France':
USA       WashingtonDC
UK              London
France           Paris
dtype: object

Reversing the order of elements in the Series:
France           Paris
UK              London
USA       WashingtonDC
USA        WashingtonDC
India          NewDelhi
dtype: object

Modified Series after value modifications:
a     10
b     50
c    500
d    500
e    500
f     15
dtype: int32
```

```
C:\Users\PCL\AppData\Local\Temp\ipykernel_6384\1583100706.py:44: FutureWarning: Series.__
getitem__ treating keys as positions is deprecated. In a future version, integer keys wil
l always be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  print(seriesCapCntry[1])
C:\Users\PCL\AppData\Local\Temp\ipykernel_6384\1583100706.py:48: FutureWarning: Series.__
getitem__ treating keys as positions is deprecated. In a future version, integer keys wil
l always be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  print(seriesCapCntry[[3, 2]])
```

In [4]:

```python
print()
print("------------------------------------------------------")
print("-------------#Series for attribute question-----------")
print("------------------------------------------------------")
print(seriesCapCntry)

# Assigning a name to the seriesCapCntry Series
seriesCapCntry.name = 'Capitals'

# Assigning a name to the index of the seriesCapCntry Series
seriesCapCntry.index.name = 'Countries'

# Outputting the values of the seriesCapCntry Series
print("\nValues of seriesCapCntry Series:")
print(seriesCapCntry.values)

# Outputting the size (number of elements) of the seriesCapCntry Series
print("\nSize of seriesCapCntry Series:")
print(seriesCapCntry.size)

# Checking if the seriesCapCntry Series is empty
print("\nIs seriesCapCntry Series empty")
print(seriesCapCntry.empty)

# Creating an empty pandas Series
seriesEmpt = pd.Series()

# Checking if the empty series (seriesEmpt) is empty
print("\nIs seriesEmpt empty")
print(seriesEmpt.empty)

# Accessing series attributes
print("\nSeries Attributes:")
print("Name of the seriesCapCntry Series:", seriesCapCntry.name)
print("Name of the index of seriesCapCntry Series:", seriesCapCntry.index.name)
```

```
------------------------------------------------------
-------------#Series for attribute question-----------
------------------------------------------------------
India          NewDelhi
USA        WashingtonDC
UK               London
France            Paris
dtype: object

Values of seriesCapCntry Series:
['NewDelhi' 'WashingtonDC' 'London' 'Paris']
```

```
['NewDelhi' 'WashingtonDC' 'London' 'Paris']

Size of seriesCapCntry Series:
4

Is seriesCapCntry Series empty
False

Is seriesEmpt empty
True

Series Attributes:
Name of the seriesCapCntry Series: Capitals
Name of the index of seriesCapCntry Series: Countries
```

In [5]:

```python
print()
print("-------------------------------------------------------")
print("---------#Series for method of finding question--------")
print("-------------------------------------------------------")
seriesTenTwenty = pd.Series(np.arange(10, 20, 1))
print(seriesTenTwenty)

# Getting the first 2 elements of the seriesTenTwenty Series
print("\nFirst 2 elements:")
print(seriesTenTwenty.head(2))

# Getting the first 5 elements of the seriesTenTwenty Series (default behavior)
print("\nFirst 5 elements:")
print(seriesTenTwenty.head())

# Counting the number of non-NaN (non-missing) values in the seriesTenTwenty Series
print("\nCount of non-NaN values:")
print(seriesTenTwenty.count())

# Getting the last 2 elements of the seriesTenTwenty Series
print("\nLast 2 elements:")
print(seriesTenTwenty.tail(2))

# Getting the last 5 elements of the seriesTenTwenty Series (default behavior)
print("\nLast 5 elements:")
print(seriesTenTwenty.tail())
```

```
-------------------------------------------------------
---------#Series for method of finding question--------
-------------------------------------------------------
0    10
1    11
2    12
3    13
4    14
5    15
6    16
7    17
8    18
9    19
dtype: int32

First 2 elements:
0    10
1    11
dtype: int32

First 5 elements:
0    10
1    11
2    12
3    13
4    14
dtype: int32
```

```
Count of non-NaN values:
10

Last 2 elements:
8    18
9    19
dtype: int32

Last 5 elements:
5    15
6    16
7    17
8    18
9    19
dtype: int32
```

In [6]:

```python
print()
print("------------------------------------------------------")
print("------#Series 1 for calculations question-------------")
print("------------------------------------------------------")

seriesA = pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e'])
print(seriesA)

print()
print("------------------------------------------------------")
print("------#Series 2 for calculations question-------------")
print("------------------------------------------------------")

seriesB = pd.Series([10, 20, -10, -50, 100], index=['z', 'y', 'a', 'c', 'e'])
print(seriesB)

# Adding Series A and Series B
print("\nSeries A + Series B:")
print(seriesA + seriesB)

# Adding Series A and Series B with fill_value to handle missing values
print("\nSeries A + Series B with fill_value:")
print(seriesA.add(seriesB, fill_value=0))  # when we don't want to have NaN values in the resulting Series

# Subtracting Series B from Series A using the subtraction operator
print("\nSeries A - Series B:")
print(seriesA - seriesB)

# Subtracting Series B from Series A using the sub method with fill_value
print("\nSeries A - Series B with fill_value:")
print(seriesA.sub(seriesB, fill_value=1000))  # using fill value 1000 while making an explicit call of the method

# Multiplying Series A and Series B using the multiplication operator
print("\nSeries A * Series B:")
print(seriesA * seriesB)

# Multiplying Series A and Series B using the mul method with fill_value
print("\nSeries A * Series B with fill_value:")
print(seriesA.mul(seriesB, fill_value=0))

# Dividing Series A by Series B
print("\nSeries A / Series B:")
print(seriesA / seriesB)

# Dividing Series A by Series B using the div method with fill_value
print("\nSeries A / Series B with fill_value:")
print(seriesA.div(seriesB, fill_value=0))
```

```
------------------------------------------------------
------#Series 1 for calculations question-------------
------------------------------------------------------
a    1
```

```
b    2
c    3
d    4
e    5
dtype: int64

------------------------------------------------------
------#Series 2 for calculations question-------------
------------------------------------------------------
z     10
y     20
a    -10
c    -50
e    100
dtype: int64

Series A + Series B:
a     -9.0
b      NaN
c    -47.0
d      NaN
e    105.0
y      NaN
z      NaN
dtype: float64

Series A + Series B with fill_value:
a     -9.0
b      2.0
c    -47.0
d      4.0
e    105.0
y     20.0
z     10.0
dtype: float64

Series A - Series B:
a    11.0
b     NaN
c    53.0
d     NaN
e   -95.0
y     NaN
z     NaN
dtype: float64

Series A - Series B with fill_value:
a     11.0
b   -998.0
c     53.0
d   -996.0
e    -95.0
y    980.0
z    990.0
dtype: float64

Series A * Series B:
a    -10.0
b      NaN
c   -150.0
d      NaN
e    500.0
y      NaN
z      NaN
dtype: float64

Series A * Series B with fill_value:
a    -10.0
b      0.0
c   -150.0
d      0.0
e    500.0
```

```
e    500.0
y      0.0
z      0.0
dtype: float64

Series A / Series B:
a   -0.10
b     NaN
c   -0.06
d     NaN
e    0.05
y     NaN
z     NaN
dtype: float64

Series A / Series B with fill_value:
a   -0.10
b     inf
c   -0.06
d     inf
e    0.05
y    0.00
z    0.00
dtype: float64
```

In [ ]:

```
#some MAIN things like head and tail has 5 as default and fill value help to remove shit NaN error aur vector process sirf
#same index value mei hoga until and unless you uses fill value and any other (later discuss.. error handling type leave)
#some common full form panda = panel data aur numpy = number python all are libraries extracted imported from other
```

In [ ]:

```
#2-D,Contain rows and column (both) looks like mysql table
#Rows = The horizontal line associated with a particular value is called its column
#Column = The vertical label associated with a particular value is called its column
```

In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
print("----------------------------------------------")
print("----------#blank dataframe /empty df----------")
print("----------------------------------------------")

dFrameEmt = pd.DataFrame()
print(dFrameEmt)

print()
print("----------------------------------------------")
print("----------#DataFrame from shhitty np----------")
print("----------------------------------------------")

array1 = np.array([10,20,30])
array2 = np.array([100,200,300])
array3 = np.array([-10,-20,-30, -40])
dFrameNp = pd.DataFrame([array1, array3,array2], columns=[ 'A', 'B', 'C', 'D'])
print(dFrameNp)

print()
print("----------------------------------------------")
print("----------#DataFrame from shhit list----------")
print("----------------------------------------------")

LIST1=[[1,2,3],[4,5,6],[7,8,9]]
dFrameLIST=pd.DataFrame(LIST1)
print(dFrameLIST)

print()
print("----------------------------------------------")
print("----#DataFrame with index and column----------")
print("----------------------------------------------")

dFrameLIST2=pd.DataFrame(LIST1,index=["A","B","C"],
               columns=["Col1","Col2","Col3"])
print(dFrameLIST2)

print()
print("----------------------------------------------")
print("-------#DataFrame from listdict---------------")
print("----------------------------------------------")

listDict = [{'a':10, 'b':20}, {'a':5,'b':10, 'c':20}]
dFrameListDict = pd.DataFrame(listDict)
print(dFrameListDict)

print()
print("----------------------------------------------")
print("------#DataFrame from multilist---------------")
print("----------------------------------------------")

dictForest = {'State': ['Assam', 'Delhi','Kerala'],
             'GArea': [78438, 1483, 38852] ,
             'VDF' : [2797, 6.72,1663]}
dFrameForest= pd.DataFrame(dictForest)
print(dFrameForest)
print()
print("-----------------------------------------------")
print("------#DataFrame from MultiSeries-------------")
print("-----------------------------------------------")
seriesA = pd.Series([1,2,3,4,5],
                     index = ['a', 'b', 'c', 'd', 'e'])
seriesB = pd.Series ([1000,2000,-1000,-5000,1000],
                     index = ['a', 'b', 'c', 'd', 'e'])
seriesC = pd.Series([10,20,-10,-50,100],
                     index = ['z', 'y', 'a', 'c', 'e'])

dFrameMS = pd.DataFrame([seriesA, seriesC])
print(dFrameMS)

print()
print("-----------------------------------------------")
print("------#DataFrame from SeriesDict--------------")
print("-----------------------------------------------")

ResultSheet={'Arnav': pd.Series([90, 91, 97], index=['Maths','Science','Hindi']),
            'Ramit': pd.Series([92, 81, 96], index=['Maths','Science','Hindi']),
            'Samridhi': pd.Series([89, 91, 88],index=['Maths','Science','Hindi']),
            'Riya': pd.Series([81, 71, 67],index=['Maths','Science','Hindi']),
            'Mallika': pd.Series([94, 95, 99],index=['Maths','Science','Hindi'])}
ResultDF = pd.DataFrame(ResultSheet)
print(ResultDF)

type(ResultDF) #to identify wheter it is a series or DataFrame
type(ResultDF.Arnav) #to identify wheter it is a series or DataFrame

print()
print("-----------------------------------------------")
print("------#DataFrame from union of series---------")
print("-----------------------------------------------")
dictForUnion = { 'Series1' :pd.Series([1,2,3,4,5],index = ['a', 'b', 'c', 'd', 'e']) ,
                 'Series2' :pd.Series([10,20,-10,-50,100],index = ['z', 'y', 'a', 'c', '
e']),
                 'Series3' :pd.Series([10,20,-10,-50,100],index = ['z', 'y', 'a', 'c', '
e']) }
dFrameUnion = pd.DataFrame(dictForUnion)
print(dFrameUnion)
```

```
---------------------------------------------
----------#blank dataframe /empty df----------
---------------------------------------------
Empty DataFrame
Columns: []
Index: []

---------------------------------------------
----------#DataFrame from shhitty np----------
---------------------------------------------
     A    B    C     D
0   10   20   30   NaN
1  -10  -20  -30 -40.0
2  100  200  300   NaN

---------------------------------------------
----------#DataFrame from shhit list----------
---------------------------------------------
   0  1  2
0  1  2  3
1  4  5  6
2  7  8  9

---------------------------------------------
----#DataFrame with index and column----------
---------------------------------------------
   Col1  Col2  Col3
A     1     2     3
B     4     5     6
```

```
C     7     8     9

----------------------------------------------
-------#DataFrame from listdict---------------
----------------------------------------------
    a   b     c
0  10  20   NaN
1   5  10  20.0

----------------------------------------------
------#DataFrame from multilist---------------
----------------------------------------------
    State  GArea      VDF
0   Assam  78438  2797.00
1   Delhi   1483     6.72
2  Kerala  38852  1663.00

----------------------------------------------
------#DataFrame from MultiSeries-------------
----------------------------------------------
      a    b     c    d      e     z     y
0   1.0  2.0   3.0  4.0    5.0   NaN   NaN
1 -10.0  NaN -50.0  NaN  100.0  10.0  20.0

----------------------------------------------
------#DataFrame from SeriesDict--------------
----------------------------------------------
         Arnav  Ramit  Samridhi  Riya  Mallika
Maths       90     92        89    81       94
Science     91     81        91    71       95
Hindi       97     96        88    67       99

----------------------------------------------
------#DataFrame from union of series---------
----------------------------------------------
   Series1  Series2  Series3
a      1.0    -10.0    -10.0
b      2.0      NaN      NaN
c      3.0    -50.0    -50.0
d      4.0      NaN      NaN
e      5.0    100.0    100.0
y      NaN     20.0     20.0
z      NaN     10.0     10.0
```

In [3]:

```python
# Printing section separator
print()
print("---------------------------------------------------")
print("----#DataFrame for operation of rows and column----")
print("---------------------------------------------------")

# Printing the DataFrame ResultDF
print(ResultDF)

# Adding new columns 'Preeti' and 'Ramit' to ResultDF
print("\nAdding new columns 'Preeti' and 'Ramit' to ResultDF:")
ResultDF['Preeti'] = [89, 78, 76]  # Correcting the length of values to match the index
ResultDF['Ramit'] = [99, 98, 78]  # Correcting the length of values to match the index
ResultDF['Arnav'] = 90  # Changing entire column values
print(ResultDF)
```

```
---------------------------------------------------
----#DataFrame for operation of rows and column----
---------------------------------------------------
         Arnav  Ramit  Samridhi  Riya  Mallika
Maths       90     92        89    81       94
Science     91     81        91    71       95
Hindi       97     96        88    67       99

Adding new columns 'Preeti' and 'Ramit' to ResultDF:
         Arnav  Ramit  Samridhi  Riya  Mallika  Preeti
         Arnav  Ramit  Samridhi  Riya  Mallika  Preeti
Maths       90     99        89    81       94      89
Science     90     98        91    71       95      78
Hindi       90     78        88    67       99      76
```

In [4]:

```
# Adding a new row 'English' to ResultDF
print("\nAdding a new row 'English' to ResultDF:")
# Ensure that the length of the list matches the number of columns
ResultDF.loc['English'] = [95, 86, 95, 80, 90, 99]  # Removing the extra value
print(ResultDF)
```

```
Adding a new row 'English' to ResultDF:
         Arnav  Ramit  Samridhi  Riya  Mallika  Preeti
Maths       90     99        89    81       94      89
Science     90     98        91    71       95      78
Hindi       90     78        88    67       99      76
English     95     86        95    80       90      99
```

In [5]:

```
# Changing entire value of the 'Maths' row to 0
print("\nChanging entire value of the 'Maths' row to 0:")
ResultDF.loc['Maths'] = 0
print(ResultDF)
```

```
Changing entire value of the 'Maths' row to 0:
         Arnav  Ramit  Samridhi  Riya  Mallika  Preeti
Maths        0      0         0     0        0       0
Science     90     98        91    71       95      78
Hindi       90     78        88    67       99      76
English     95     86        95    80       90      99
```

In [6]:

```
# Setting all values in ResultDF to 0
print("\nSetting all values in ResultDF to 0:")
ResultDF[:] = 0
print(ResultDF)
```

```
Setting all values in ResultDF to 0:
         Arnav  Ramit  Samridhi  Riya  Mallika  Preeti
Maths        0      0         0     0        0       0
Science      0      0         0     0        0       0
Hindi        0      0         0     0        0       0
English      0      0         0     0        0       0
```

In [7]:

```
# Printing ResultDF after modifying it
print("\nResultDF after modifying:")
print(ResultDF)
```

```
ResultDF after modifying:
         Arnav  Ramit  Samridhi  Riya  Mallika  Preeti
Maths        0      0         0     0        0       0
Science      0      0         0     0        0       0
Hindi        0      0         0     0        0       0
English      0      0         0     0        0       0
```

In [8]:

```
# Removing the row labeled 'Science' from ResultDF
print("\nRemoving the row labeled 'Science' from ResultDF:")
ResultDF = ResultDF.drop('Science')

print(ResultDF)
```

```
Removing the row labeled 'Science' from ResultDF:
         Arnav  Ramit  Samridhi  Riya  Mallika  Preeti
Maths        0      0         0     0        0       0
```

```
Hindi        0      0         0     0        0       0
English      0      0         0     0        0       0
```

In [9]:

```
# Renaming rows and columns in ResultDF
print("\nRenaming rows and columns in ResultDF:")
ResultDF = ResultDF.rename({'Maths': 'Sub1', 'Science': 'Sub2', 'English': 'Sub3', 'Hindi': 'Sub4'}, axis='index')
# Renaming column labels
ResultDF = ResultDF.rename({'Arnab': 'Student1', 'Ramit': 'Student2', 'Samridhi': 'Student3', 'Mallika': 'Student4'}, axis='columns')

# Printing ResultDF after renaming
print("\nResultDF after renaming:")
print(ResultDF)
```

```
Renaming rows and columns in ResultDF:

ResultDF after renaming:
      Arnav  Student2  Student3  Riya  Student4  Preeti
Sub1      0         0         0     0         0       0
Sub4      0         0         0     0         0       0
Sub3      0         0         0     0         0       0
```

In [10]:

```
# Dropping columns with labels 'Samridhi', 'Ramit', and 'Riya' from ResultDF
print("\nDropping columns with labels 'Samridhi', 'Ramit', and 'Riya' from ResultDF:")
ResultDF = ResultDF.drop(['Student2', 'Student3', 'Riya'], axis=1)  # Ensure to specify the axis as 1 for columns
print(ResultDF)

#axis=0: Drop rows , axis=1: Drop columns.
```

```
Dropping columns with labels 'Samridhi', 'Ramit', and 'Riya' from ResultDF:
      Arnav  Student4  Preeti
Sub1      0         0       0
Sub4      0         0       0
Sub3      0         0       0
```

In [11]:

```
print()
print("----------------------------------------------")
print("------#DataFrame For Indexing-----------------")
print("----------------------------------------------")

ResultSheet = {
    'Arnav': pd.Series([90, 91, 97], index=['Maths', 'Science', 'Hindi']),
    'Ramit': pd.Series([92, 81, 96], index=['Maths', 'Science', 'Hindi']),
    'Samridhi': pd.Series([89, 91, 88], index=['Maths', 'Science', 'Hindi']),
    'Riya': pd.Series([81, 71, 67], index=['Maths', 'Science', 'Hindi']),
    'Mallika': pd.Series([94, 95, 99], index=['Maths', 'Science', 'Hindi'])
}
ResultDF2 = pd.DataFrame(ResultSheet)
print(ResultDF2)

print()
print("Result of loc['Science']:")
print(ResultDF2.loc['Science'])   # A SINGLE ROW WILL BE SHOWN

print()
print("Result of loc[:, 'Arnav']:")
print(ResultDF2.loc[:, 'Arnav'])   # A Row starting from . to Arnav

print()
print("Result of ['Arnav']:")
print(ResultDF2['Arnav'])   # will give result of Arnav

print()
```

```python
print("Result of loc['Maths'] > 90:")
print(ResultDF2.loc['Maths'] > 90)  # will show those who have marks more than 90 in Maths

print()
print("Result of loc['Maths': 'Science']:")
print(ResultDF2.loc['Maths': 'Science'])  # accessing DataFrame elements through slicing

print()
print("Result of loc['Maths': 'Science', 'Arnav']:")
print(ResultDF2.loc['Maths': 'Science', 'Arnav'])  # give Maths to Science result of Arnav

print()
print("Result of loc['Maths': 'Science', 'Arnav':'Samridhi']:")
print(ResultDF2.loc['Maths': 'Science', 'Arnav':'Samridhi'])  # give Maths to Science result of Arnav to Samridhi

print()
print("Result of loc['Maths': 'Science', ['Arnav','Samridhi']]:")
print(ResultDF2.loc['Maths': 'Science', ['Arnav', 'Samridhi']])  # give Maths to Science result of Arnav and Samridhi

print()
print("Result of loc[[True, False, True]]:")
print(ResultDF2.loc[[True, False, True]])

print()
print("----------------------------------------------")
print("------#NumericalDataFrame For Indexing--------")
print("----------------------------------------------")

dFrame10Multiples = pd.DataFrame([10, 20, 30, 40, 50])
print(dFrame10Multiples)

print()
print("Result of loc[2]:")
print(dFrame10Multiples.loc[2])  # it will interpret as label of integer
```

```
----------------------------------------------
------#DataFrame For Indexing-----------------
----------------------------------------------
         Arnav  Ramit  Samridhi  Riya  Mallika
Maths       90     92        89    81       94
Science     91     81        91    71       95
Hindi       97     96        88    67       99

Result of loc['Science']:
Arnav       91
Ramit       81
Samridhi    91
Riya        71
Mallika     95
Name: Science, dtype: int64

Result of loc[:, 'Arnav']:
Maths      90
Science    91
Hindi      97
Name: Arnav, dtype: int64

Result of ['Arnav']:
Maths      90
Science    91
Hindi      97
Name: Arnav, dtype: int64

Result of loc['Maths'] > 90:
Arnav       False
Ramit        True
Samridhi    False
Riya        False
```

```
Mallika      True
Name: Maths, dtype: bool

Result of loc['Maths': 'Science']:
         Arnav  Ramit  Samridhi  Riya  Mallika
Maths       90     92        89    81       94
Science     91     81        91    71       95

Result of loc['Maths': 'Science', 'Arnav']:
Maths      90
Science    91
Name: Arnav, dtype: int64

Result of loc['Maths': 'Science', 'Arnav':'Samridhi']:
         Arnav  Ramit  Samridhi
Maths       90     92        89
Science     91     81        91

Result of loc['Maths': 'Science', ['Arnav','Samridhi']]:
         Arnav  Samridhi
Maths       90        89
Science     91        91

Result of loc[[True, False, True]]:
       Arnav  Ramit  Samridhi  Riya  Mallika
Maths     90     92        89    81       94
Hindi     97     96        88    67       99

----------------------------------------------
------#NumericalDataFrame For Indexing--------
----------------------------------------------
    0
0  10
1  20
2  30
3  40
4  50

Result of loc[2]:
0    30
Name: 2, dtype: int64
```

In [12]:

```
print("----------------------------------------------")
print("------#DataFrame For Atrribute function-------")
print("----------------------------------------------")

ForestArea = {'Assam' :pd.Series([78438, 2797,10192, 15116], index = ['GeoArea', 'VeryDense','ModeratelyDense', 'OpenForest']),
              'Kerala' :pd.Series([ 38852, 1663,9407, 9251], index = ['GeoArea' ,'VeryDense','ModeratelyDense', 'OpenForest']),
              'Delhi' :pd.Series([1483, 6.72, 56.24,129.45], index = ['GeoArea', 'VeryDense','ModeratelyDense', 'OpenForest'])}

ForestAreaDF = pd.DataFrame(ForestArea)
print(ForestAreaDF)

print()
print(ForestAreaDF.columns) #to display column label

print()
print(ForestAreaDF.index) #to display row labels

print()
print(ForestAreaDF.dtypes) #to display data type of each column

print()
print(ForestAreaDF.values) #to display ndarray of dataframe without index

print()
print(ForestAreaDF.shape) #to display how many rows and columns (rows,column)

print()
print(ForestAreaDF.size) #to determine size of whole dataset

print()
print(ForestAreaDF.head(2)) #by default it shows 5 result

print()
print(ForestAreaDF.tail(2)) #by default it shows 5 result

print()
print(ForestAreaDF.empty) #shows if there is any NaN value or nulled value in DataFrame
```

```
-----------------------------------------------
------#DataFrame For Atrribute function-------
-----------------------------------------------
                 Assam  Kerala    Delhi
GeoArea          78438   38852  1483.00
VeryDense         2797    1663     6.72
ModeratelyDense  10192    9407    56.24
OpenForest       15116    9251   129.45

Index(['Assam', 'Kerala', 'Delhi'], dtype='object')

Index(['GeoArea', 'VeryDense', 'ModeratelyDense', 'OpenForest'], dtype='object')

Assam       int64
Kerala      int64
Delhi     float64
dtype: object

[[7.8438e+04 3.8852e+04 1.4830e+03]
 [2.7970e+03 1.6630e+03 6.7200e+00]
 [1.0192e+04 9.4070e+03 5.6240e+01]
 [1.5116e+04 9.2510e+03 1.2945e+02]]

(4, 3)

12

           Assam  Kerala    Delhi
GeoArea    78438   38852  1483.00
VeryDense   2797    1663     6.72

                 Assam  Kerala   Delhi
ModeratelyDense  10192    9407   56.24
OpenForest       15116    9251  129.45

False
```

In [13]:

```
print("-----------------------------------------------")
print("------#DataFrame For Atrribute function-------")
print("-----------------------------------------------")

ForestArea = {'Assam' :pd.Series([78438, 2797,10192, 15116], index = ['GeoArea', 'VeryDe
nse','ModeratelyDense', 'OpenForest']),
              'Kerala' :pd.Series([ 38852, 1663,9407, 9251], index = ['GeoArea' ,'VeryDe
nse','ModeratelyDense', 'OpenForest']),
              'Delhi' :pd.Series([1483, 6.72, 56.24,129.45], index = ['GeoArea', 'VeryDe
nse','ModeratelyDense', 'OpenForest'])}

ForestAreaDF = pd.DataFrame(ForestArea)
print(ForestAreaDF)

print()
print(ForestAreaDF.columns) #to display column label

print()
print(ForestAreaDF.index) #to display row labels

print()
print(ForestAreaDF.dtypes) #to display data type of each column

print()
print(ForestAreaDF.values) #to display ndarray of dataframe without index

print()
print(ForestAreaDF.shape) #to display how many rows and columns (rows,column)

print()
print(ForestAreaDF.size) #to determine size of whole dataset

print()
print(ForestAreaDF.head(2)) #by default it shows 5 result

print()
print(ForestAreaDF.tail(2)) #by default it shows 5 result

print()
print(ForestAreaDF.empty) #shows if there is any NaN value or nulled value in DataFrame
```

```
----------------------------------------------
------#DataFrame For Atrribute function-------
----------------------------------------------
                 Assam  Kerala    Delhi
GeoArea          78438   38852  1483.00
VeryDense         2797    1663     6.72
ModeratelyDense  10192    9407    56.24
OpenForest       15116    9251   129.45

Index(['Assam', 'Kerala', 'Delhi'], dtype='object')

Index(['GeoArea', 'VeryDense', 'ModeratelyDense', 'OpenForest'], dtype='object')

Assam       int64
Kerala      int64
Delhi     float64
dtype: object

[[7.8438e+04 3.8852e+04 1.4830e+03]
 [2.7970e+03 1.6630e+03 6.7200e+00]
 [1.0192e+04 9.4070e+03 5.6240e+01]
 [1.5116e+04 9.2510e+03 1.2945e+02]]

(4, 3)

12

           Assam  Kerala    Delhi
GeoArea    78438   38852  1483.00
VeryDense   2797    1663     6.72

                 Assam  Kerala   Delhi
ModeratelyDense  10192    9407   56.24
OpenForest       15116    9251  129.45

False
```

In [14]:

```
ResultDF2

#export file to files and file.csv
ResultDF2.to_csv(path_or_buf='file.csv', sep=',') #sep is used to separate value like , o
r arrow etc
ResultDF2.to_csv('files.csv',sep = '@', header = False, index= False) #save wthout heade
r and footer
```

In [15]:

```
#import DataFrame from file.csv

marks = pd.read_csv("file.csv",sep =",", header=0)
print(marks)

print()
marks1 = pd.read_csv("file.csv",sep=",",names=['RNo','StudentName', 'Sub1','Sub2']) #giv
e with names
print(marks1)
```

```
  Unnamed: 0  Arnav  Ramit  Samridhi  Riya  Mallika
0      Maths     90     92        89    81       94
1    Science     91     81        91    71       95
2      Hindi     97     96        88    67       99

                 RNo StudentName  Sub1     Sub2
NaN     Arnav  Ramit    Samridhi  Riya  Mallika
Maths   90        92          89    81       94
Science 91        81          91    71       95
Hindi   97        96          88    67       99
```

In [16]:

```
print("----------------------------------------------")
print("------#DataFrame1 For appending---------------")
print("----------------------------------------------")

# Creating the first DataFrame, dFrame1
dFrame1 = pd.DataFrame([[1, 2, 3], [4, 5, None], [6, None, None]],
                       columns=['C1', 'C2', 'C3'],
                       index=['R1', 'R2', 'R3'])

# Displaying the first DataFrame
print("dFrame1:")
print(dFrame1)
print("\n")


print()
print("----------------------------------------------")
print("------#DataFrame2 For appending---------------")
print("----------------------------------------------")
dFrame2 = pd.DataFrame([[10, 20], [30, None], [40, 50]],
                       columns=['C2', 'C5'],
                       index=['R4', 'R2', 'R5'])

# Displaying the second DataFrame
print("dFrame2:")
print(dFrame2)
print("\n")

# Appending dFrame2 to dFrame1
dFrame1 = dFrame1.append(dFrame2)

# Displaying the appended DataFrame
print("Appended dFrame1:")
print(dFrame1)
print("\n")

# Appending dFrame1 to dFrame2 with column labels sorted
dFrame2 = dFrame2.append(dFrame1, sort=True)

# Displaying the appended DataFrame with sorted column labels
print("Appended dFrame2 with sorted columns:")
print(dFrame2)
print("\n")

# Appending dFrame1 to dFrame2 with column labels unsorted
dFrame2 = dFrame2.append(dFrame1, sort=False)

# Displaying the appended DataFrame with unsorted column labels
```

```
print("Appended dFrame2 with unsorted columns:")
print(dFrame2)
print("\n")

# Appending dFrame1 to dFrame2 with ignoring index labels
dFrame1 = dFrame1.append(dFrame2, ignore_index=True)

# Displaying the appended DataFrame with ignoring index labels
print("Appended dFrame1 with ignoring index labels:")
print(dFrame1)

#all coded is good but just in new pandas there is no append toh kat gya yeh not necessar
y to read
```

```
----------------------------------------------
------#DataFrame1 For appending---------------
----------------------------------------------
dFrame1:
    C1   C2   C3
R1   1  2.0  3.0
R2   4  5.0  NaN
R3   6  NaN  NaN



----------------------------------------------
------#DataFrame2 For appending---------------
----------------------------------------------
dFrame2:
    C2    C5
R4  10  20.0
R2  30   NaN
R5  40  50.0



---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [16], in <cell line: 30>()
     27 print("\n")
     29 # Appending dFrame2 to dFrame1
---> 30 dFrame1 = dFrame1.append(dFrame2)
     32 # Displaying the appended DataFrame
     33 print("Appended dFrame1:")

File ~\miniconda3\lib\site-packages\pandas\core\generic.py:6202, in NDFrame.__getattr__(s
elf, name)
   6195 if (
   6196     name not in self._internal_names_set
   6197     and name not in self._metadata
   6198     and name not in self._accessors
   6199     and self._info_axis._can_hold_identifiers_and_holds_name(name)
   6200 ):
   6201     return self[name]
-> 6202 return object.__getattribute__(self, name)

AttributeError: 'DataFrame' object has no attribute 'append'
```

In [ ]:

```
#Pandas Series are labeled one-dimensional arrays supporting various data types and missi
ng values, ideal for data analysis.
#NumPy ndarray are multi-dimensional arrays suited for numerical operations, offering hom
ogeneous data types and high efficiency
#some common full form csv = comma separated variable
```

In [ ]:

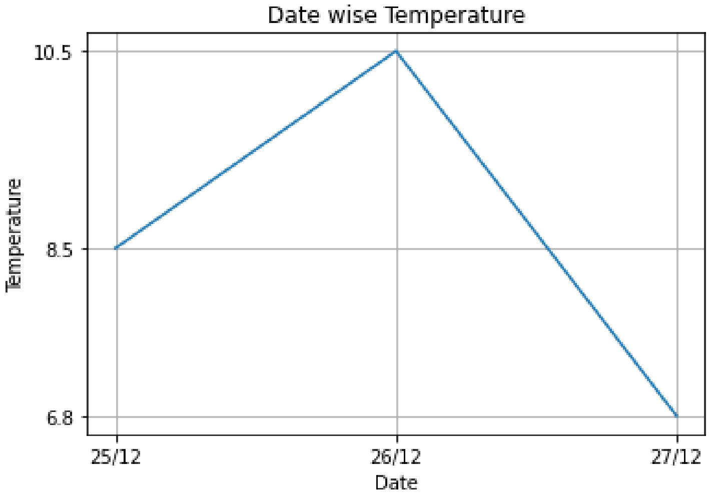
```
#Matplot what is matplot a library which help us to visualisze things effectively whiotu
any issues
#can create 2 d plot graph
```

In [1]:

```
#sabsey important
import matplotlib.pyplot as plt #pyplot means module of matplot which is collection of fu
nction that can be use to create graph
from matplotlib import style
import pandas as pd
```

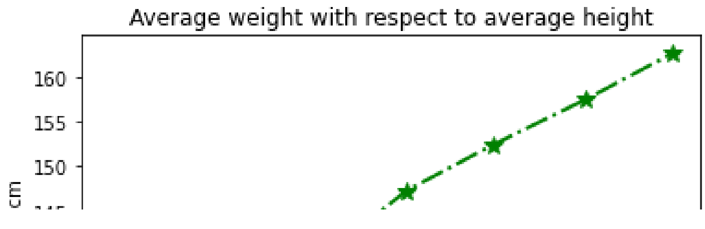
In [2]:

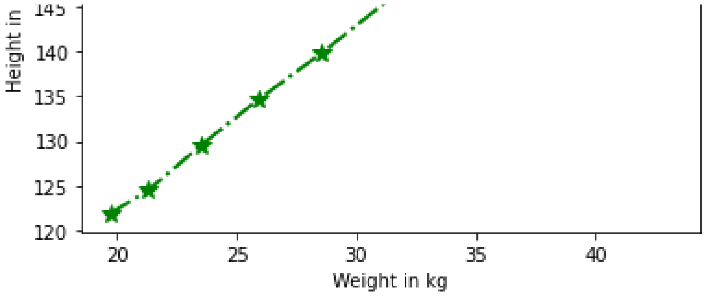
```
#simple line graph from list
date=["25/12","26/12","27/12"] #data for creating plot
temp=[8.5,10.5,6.8]
plt.plot(date, temp) #here (x axis ,y axis)
plt.xlabel("Date") #add the Label on x-axis
plt.ylabel("Temperature") #add the Label on y-axis
plt.title("Date wise Temperature") #add the title to the chart
plt.grid(True) #add gridlines to the background
plt.grid(True) #add gridlines to the background
plt.yticks(temp)
plt.show()
```



In [3]:

```
#line graph by data frame
height=[121.9,124.5,129.5,134.6,139.7,147.3,152.4,157.5,162.6]
weight=[19.7,21.3,23.5,25.9,28.5,32.1,35.7,39.6,43.2]
df=pd.DataFrame({"height":height,"weight":weight})
plt.xlabel('Weight in kg')
plt.ylabel('Height in cm')
plt.title('Average weight with respect to average height')

#attribute with respective name marker mark lagana color color karna width size style dot
type etcc..
plt.plot(df.weight,df.height,marker='*',markersize=10,color='green',linewidth=2, linesty
le='dashdot')
plt.show() #plt.show()important for showing graph
```

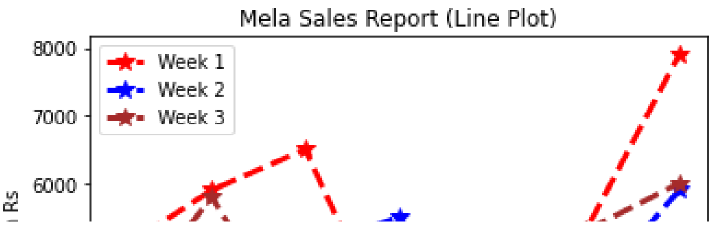


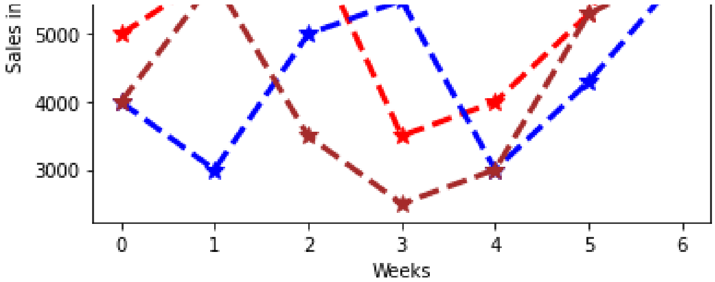


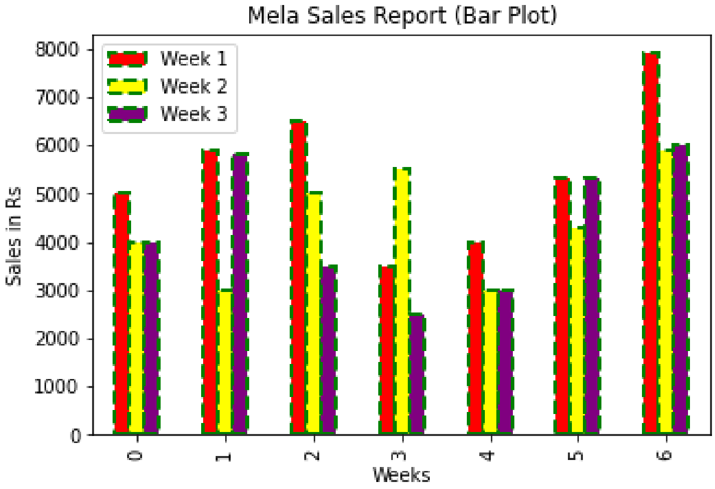
In [4]:

```
# Data for the dataframe
data = {
    'Week 1': [5000, 5900, 6500, 3500, 4000, 5300, 7900],
    'Week 2': [4000, 3000, 5000, 5500, 3000, 4300, 5900],
    'Week 3': [4000, 5800, 3500, 2500, 3000, 5300, 6000]
}

# Creating the dataframe
df = pd.DataFrame(data)

# Print the dataframe
print("-----------------------------------------------")
print("---------#dataframe for plot function----------")
print("-----------------------------------------------")
print(df)

# Plotting the line plot
df.plot(kind='line', color=['red', 'blue', 'brown'], marker="*", markersize=10, linewidt
h=3, linestyle="--")

# Setting title and labels for the line plot
plt.title('Mela Sales Report (Line Plot)')
plt.xlabel('Weeks')
plt.ylabel('Sales in Rs')

# Display the line plot
plt.show()

# Plotting the bar plot
df.plot(kind='bar', color=['red', 'yellow', 'purple'], edgecolor='green', linewidth=2, l
inestyle='--')

# Setting title and labels for the bar plot
plt.title('Mela Sales Report (Bar Plot)')
plt.xlabel('Weeks')
plt.ylabel('Sales in Rs')

# Display the bar plot
plt.show()
```

```
-----------------------------------------------
---------#dataframe for plot function----------
-----------------------------------------------
   Week 1  Week 2  Week 3
0    5000    4000    4000
1    5900    3000    5800
2    6500    5000    3500
3    3500    5500    2500
4    4000    3000    3000
5    5300    4300    5300
6    7900    5900    6000
```



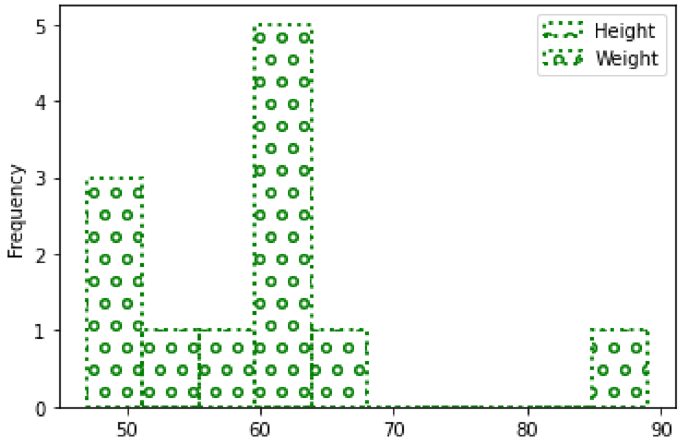




In [5]:

```
import pandas as pd
import matplotlib.pyplot as plt

data = {'Name': ['Arnav', 'Sheela', 'Azhar', 'Bincy', 'Yash', 'Nazar'],
        'Height': [60, 61, 63, 65, 61, 60],
        'Weight': [47, 89, 52, 58, 50, 47]}
df = pd.DataFrame(data)
print("---------------------------------------------")
print("---#dataframe for histogram function----------")
print("---------------------------------------------")
print(df)

df.plot(kind='hist',edgecolor='Green',linewidth=2,linestyle=':',fill=False,hatch='o')
plt.show()
```

```
---------------------------------------------
---#dataframe for histogram function----------
---------------------------------------------
     Name  Height  Weight
0   Arnav      60      47
1  Sheela      61      89
2   Azhar      63      52
3   Bincy      65      58
4    Yash      61      50
5   Nazar      60      47
```



In [ ]:

```
#kind include line bar hist and many other
#we can use pd.read_csv instead of creating full DataFrame
#(x, y label) come wrt plot function
#some common full form hist - histogram x label y labe
```