Python Fundamentals For DataScience

In [ ]:
```
1
```

1. Variables

In [ ]:
```
1
```

In [3]:
```
#Integer
a=10
#float
r = 4.5
#String
city = "Bangalore"
#Boolean
s= True
#Complex number
num=6+7j

```

In [4]:
```
#check the type of variable
type(num)
```

Out[4]:
```
complex
```

2.Keywords

In [ ]:
```
1
```

3. Conditional Statements

In [38]:
```
#simple if
n=24
if n%2==0:
    print("Even number")
```

```
Even number
```

In [39]:
```
# if else
age=34
if age>=18:
    print("Eligible to vote")
else:
    print("Not eligible")
```

```
Eligible to vote
```

In [40]:
```
#if else elif
marks=23
if marks>20:
    print("Just pass")
elif marks>90:
    print("Excellent")
else:
    print("You are fail")
```

```
Just pass
```

4.Looping Statements

In [41]:
```
#while loop
c=1
while c<=10:
    print(c)
    c=c+1

```

```
1
2
3
4
5
6
7
8
9
10
```

In [50]:
```
#for loop
for i in range(25):
    if i%2==0:
        print(i,"it is even number")
```

```
0 it is even number
2 it is even number
4 it is even number
6 it is even number
8 it is even number
10 it is even number
12 it is even number
14 it is even number
16 it is even number
18 it is even number
20 it is even number
22 it is even number
24 it is even number
```

In [ ]:
```

```

5. Functions

In [ ]:
```
#function
def calculate_bill(price, quantity):
    total = price * quantity
    return total

bill = calculate_bill(150, 3)
print("Total bill amount:", bill)

```

What is a lambda function in Python?

A lambda function is a small, one line function. It is used when you need a function for a short time and the logic is simple.

```
lambda arguments : expression
```

- arguments → inputs
- expression → logic (must be one line)
- result is returned automatically

In [68]:
```
square = lambda x: x * x
square(6)
```

Out[68]: 36

6. DataStructures in python

6. 1 Lists

In [11]:
```
#creating list
num = [10, 20, 30, 40, 50]
```

In [6]:
```
# 1. Access element
print(num[0])

```

```
10
```

In [12]:
```
# 2. Add element
num.append(60)
num

```

Out[12]: [10, 20, 30, 40, 50, 60]

In [13]:
```
# 3. Insert element
num.insert(2, 25)

num
```

Out[13]: [10, 20, 25, 30, 40, 50, 60]

In [14]:
```
# 4. Remove element
num.remove(40)

num
```

Out[14]: [10, 20, 25, 30, 50, 60]

In [15]:
```
# 5. Length of list
print(len(numbers))

print(numbers)
```

```
6
[10, 20, 25, 30, 50, 60]
```

In [67]:
```
#List comprehension
squares = [i * i for i in range(1, 6)]
print(squares)
```

```
[1, 4, 9, 16, 25]
```

6.2 Tuple

In [22]:
```
days = ("Mon","Mon", "Tue", "Wed", "Thu", "Fri")

```

In [23]:
```
# 1. Access element
print(days[1])

# 2. Length
print(len(days))

# 3. Count element
print(days.count("Mon"))

# 4. Index of element
print(days.index("Wed"))

# 5. Loop through tuple
for day in days:
    print(day)

```

```
Mon
6
2
3
Mon
Mon
Tue
Wed
Thu
Fri
```

6.3 Sets

In [19]:
```
cities = {"Bangalore", "Delhi", "Mumbai", "Delhi"}
print(cities)  # Duplicate removed
```

```
{'Mumbai', 'Bangalore', 'Delhi'}
```

In [ ]:
```
# 1. Add element
cities.add("Chennai")

# 2. Remove element
cities.remove("Delhi")

# 3. Union
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1.union(set2))

# 4. Intersection
print(set1.intersection(set2))

# 5. Difference
print(set1.difference(set2))

```

6.4 Dictionary

In [20]:
```
#create dictionary
student = {
    "name": "Rahul",
    "age": 22,
    "course": "Data Science"
}

```

In [21]:
```
# 1. Access value
print(student["name"])

# 2. Add new key-value
student["college"] = "ABC University"

# 3. Update value
student["age"] = 23

# 4. Get all keys
print(student.keys())

# 5. Get all values
print(student.values())

print(student)

```

```
Rahul
dict_keys(['name', 'age', 'course', 'college'])
dict_values(['Rahul', 23, 'Data Science', 'ABC University'])
{'name': 'Rahul', 'age': 23, 'course': 'Data Science', 'college': 'ABC Univer
sity'}
```

NumPy

NumPy is a fundamental library for numerical computing in Python. It provides a powerful N-dimensional array object and functions for manipulating arrays efficiently. NumPy is the foundation for many other libraries in the Python scientific ecosystem and is widely used for data manipulation and pre-processing in machine learning.
 numpy

In [ ]:
```
!pip install numpy # installing numpy

```

In [ ]:
```
import numpy as np
```

7. Arrays

In [2]:
```
#creting arrays From a list or tuple
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
arr1 = np.array((1, 2, 3, 4, 5))
print("Array from list:", arr)
print("Array from tuple:", arr1)
arr.shape
```

```
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\n
umpy\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\n
umpy\.libs\libopenblas.EL2C6PLE4ZYW3ECEVIV3OXXGRN2NRFM2.gfortran-win_amd64.dl
l
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\n
umpy\.libs\libopenblas64__v0.3.21-gcc_10_3_0.dll
  warnings.warn("loaded more than 1 DLL from .libs:"

Array from list: [1 2 3 4 5]
Array from tuple: [1 2 3 4 5]
```

Out[2]:
```
(5,)
```

In [3]:
```
# Multi-dimensional array
arr2D = np.array([[1, 2, 3], [4, 5, 6]])
arr2D.shape
```

Out[3]:
```
(2, 3)
```

In [27]:
```
# Zeros Array
zeros = np.zeros((3, 3))  # 3x3 matrix filled with zeros
zeros
```

Out[27]:
```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

In [28]:
```
# Ones Array
ones = np.ones((2, 4))  # 2x4 matrix filled with ones
ones
```

Out[28]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [29]:
```
# Identity Matrix
identity_matrix = np.eye(3)  # 3x3 identity matrix
identity_matrix
```

Out[29]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [51]:
```
#array reshape
arr = np.arange(1, 10)  # 1D array
print("Array is \n", arr)
reshaped_arr = arr.reshape(3, 3)  # Reshape into 3x3 matrix
print("\nReshaped Array:\n", reshaped_arr)
```

```
Array is
 [1 2 3 4 5 6 7 8 9]

Reshaped Array:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [57]:
```
#Basic mathematical opeartions
arr1 = np.array([[1, 2, 3],[0,9,8]])
arr2 = np.array([[4, 5, 6],[8,7,3]])
print("\nAddition:\n", arr1 + arr2)
print("Subtraction:\n", arr1 - arr2)
print("Multiplication:\n", arr1 * arr2)
print("Division:", arr1 / arr2)
```

```
Addition:
 [[ 5  7  9]
 [ 8 16 11]]
Subtraction:
 [[-3 -3 -3]
 [-8  2  5]]
Multiplication:
 [[ 4 10 18]
 [ 0 63 24]]
Division: [[0.25       0.4        0.5       ]
 [0.         1.28571429 2.66666667]]
```

In [58]:
```
#Aggregate function
arr = np.array([1, 2, 3, 4, 5])
print("\nSum:", np.sum(arr))
print("Mean:", np.mean(arr))
print("Max:", np.max(arr))
print("Min:", np.min(arr))
print("Standard Deviation:", np.std(arr))
print("Product:", np.prod(arr))
```

```
Sum: 15
Mean: 3.0
Max: 5
Min: 1
Standard Deviation: 1.4142135623730951
Product: 120
```

In [59]:
```
#array concatination
import numpy as np
arr1=np.array([[2,3,5],[1,4,7]])
arr2=np.array([[6,7,8],[10,20,30]])
concat_arr = np.concatenate((arr1, arr2))
print("\nConcatenated Array:\n", concat_arr)
```

```
Concatenated Array:
 [[ 2  3  5]
 [ 1  4  7]
 [ 6  7  8]
 [10 20 30]]
```

In [60]:
```
#transpose of matrix
arr2D = np.array([[1, 2, 3], [4, 5, 6]])
print("\nTransposed Matrix:\n", arr2D.T)
```

```
Transposed Matrix:
 [[1 4]
 [2 5]
 [3 6]]
```

In [61]:
```
#sum of elements in array
import numpy as np

# Creating a 3x4 array
matrix = np.array([
    [10, 20, 30, 40],
    [5, 15, 25, 35],
    [2, 4, 6, 8]
])

# Default sum (across all elements)
print("Total sum:", matrix.sum())

# Sum along axis 0 (column-wise sum)
print("Sum along axis 0:", matrix.sum(axis=0))

# Sum along axis 1 (row-wise sum)
print("Sum along axis 1:", matrix.sum(axis=1))
```

```
Total sum: 200
Sum along axis 0: [17 39 61 83]
Sum along axis 1: [100  80  20]
```

In [62]:
```
#Sorting array
data = np.array([
    [7, 1, 4],
    [8, 6, 5],
    [1, 2, 3]
])

print(np.sort(data))  # Sorts each row individually
print(np.sort(data, axis=0))  # Sorts each column individually
print(np.sort(data, axis=None))  # Flattens and sorts entire array
print(np.sort(data, axis=None)[::-1])  # sorts in descending order
```

```
[[1 4 7]
 [5 6 8]
 [1 2 3]]
[[1 1 3]
 [7 2 4]
 [8 6 5]]
[1 1 2 3 4 5 6 7 8]
[8 7 6 5 4 3 2 1 1]
```

In [63]:
```
#Determenant
x=[[2,3,4],
   [5,6,7],
   [8,9,0]]
np.linalg.det(x)
```

Out[63]: 29.99999999999999

In [64]:
```
#Dot product
import numpy as np

a = np.array([[3, 4, 5],
              [9, 0, 3]])

b = np.array([[2, 6, 1],
              [8, 5, 3],
              [8, 5, 3]])

print(np.dot(a,b))
```

```
[[78 63 30]
 [42 69 18]]
```

In [65]:
```
#Diagonal and trace of Matrix
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

d = np.diagonal(matrix)

print("\n The diagonal elements are:",d)
print("\n The trace of a matrix is:",np.trace(matrix))

```

```
 The diagonal elements are: [1 5 9]

 The trace of a matrix is: 15
```

8. DataVisualization

Matplotlib and Seaborn Visualization

What is Matplotlib?

Matplotlib is a powerful Python library for creating a wide range of data visualizations, from simple line charts to complex 3D plots.

How does it work?

8.1 Line plot

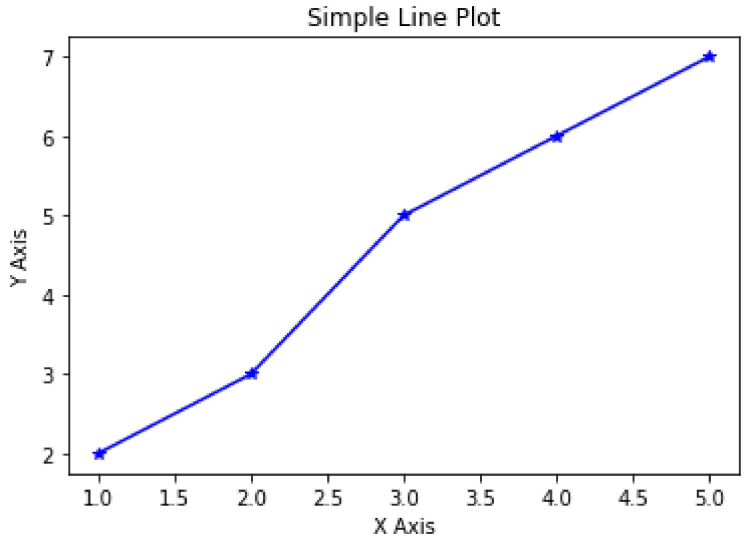
Used to show trends over time or continuous data

In [4]:
```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 6, 7]

plt.plot(x, y, color='blue', marker='*', linestyle='-')
plt.title("Simple Line Plot")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.show()

plt.show()
```



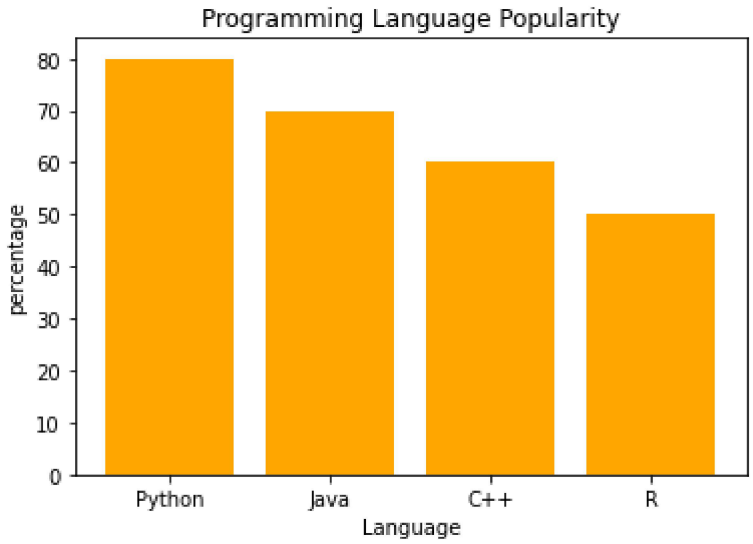
8.2 BarPlot

Bar Chart: Used to compare categories or groups

In [5]:
```
x = ['Python', 'Java', 'C++', 'R']
y = [80, 70, 60, 50]

plt.bar(x, y,color='orange')
plt.title("Programming Language Popularity")
plt.xlabel("Language")
plt.ylabel("percentage")
plt.show()

```



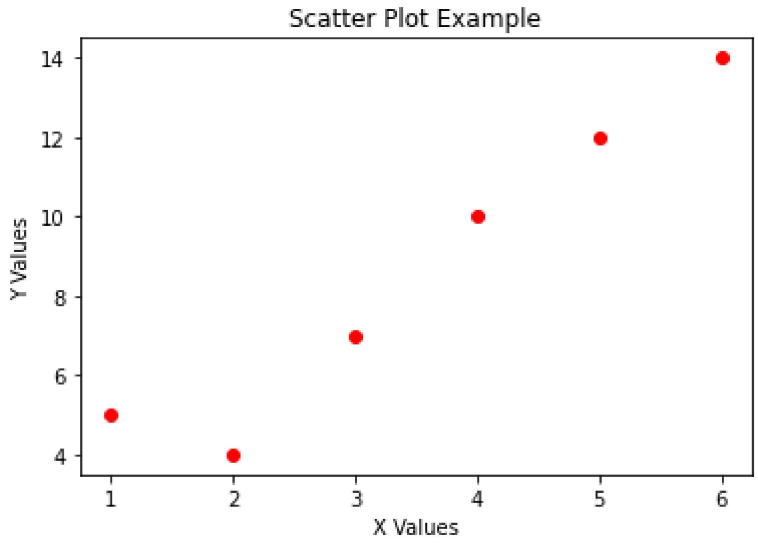
8.3 Scatterplot

Scatter Plot: Used to show relationship (correlation) between two variables. Example:

In [33]:
```
x = [1, 2, 3, 4, 5, 6]
y = [5, 4, 7, 10, 12, 14]

plt.scatter(x, y, color='red')
plt.title("Scatter Plot Example")
plt.xlabel("X Values")
plt.ylabel("Y Values")
plt.show()

```



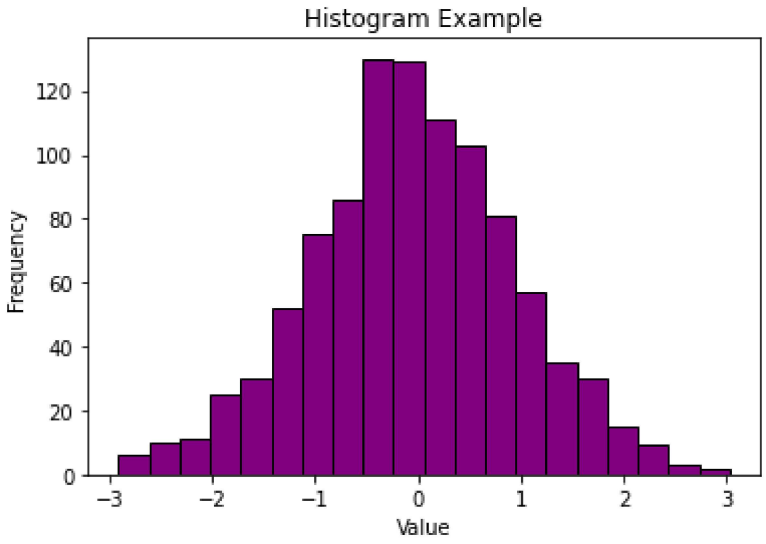
8.4 Histogram

Used to show distribution of numeric data how values are spread out.

In [32]:
```
import numpy as np
data = np.random.randn(1000)

plt.hist(data, bins=20, color='purple', edgecolor='black')
plt.title("Histogram Example")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()

```



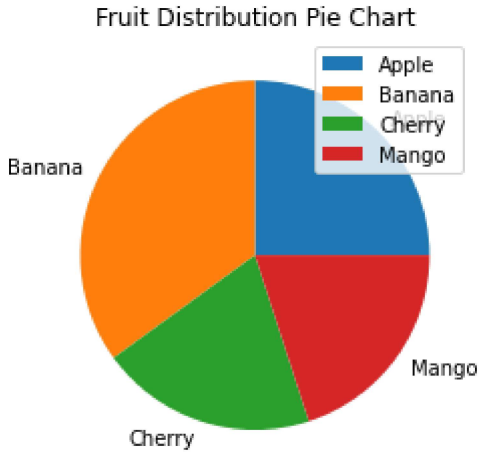
8.5 Pie chart

Shows parts of a whole percentage contribution of each category.

In [7]:
```
labels = ['Apple', 'Banana', 'Cherry', 'Mango']
sizes = [25, 35, 20, 20]

plt.pie(sizes, labels=labels)
plt.title("Fruit Distribution Pie Chart")
plt.legend()
plt.show()

```

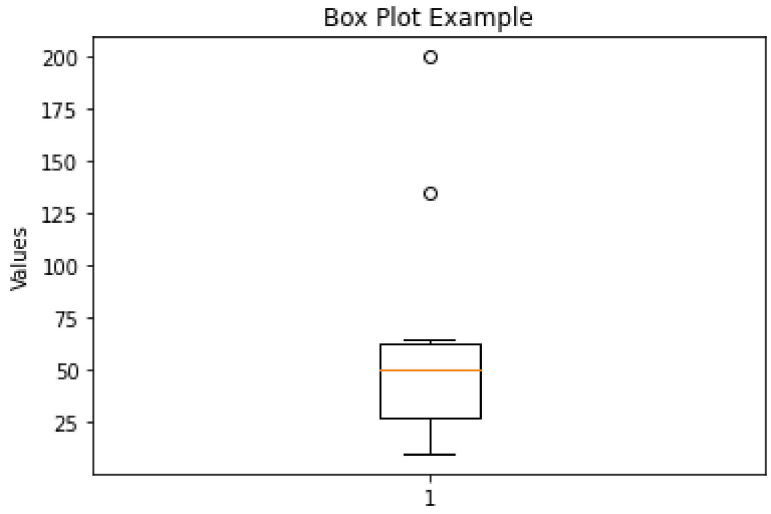


8.6 BoxPlot

Displays data spread, outliers, median, and quartiles.

In [9]:
```
data = [10, 20, 30, 25, 40, 135, 50, 55, 60, 65,200]
plt.boxplot(data)
plt.title("Box Plot Example")
plt.ylabel("Values")
plt.show()

```



Seaborn is a package,which is also used for Data visualization

List of important Seaborn functions and plots

- **sns.scatterplot():** Creates a scatter plot for two continuous variables.
- **sns.lineplot():** Creates a line plot to show trends over time or a sequential variable.
- **sns.histplot():** Plots a histogram to show the distribution of a single variable.
- **sns.barplot():** Plots a bar plot to show the relationship between a categorical variable and a continuous variable.
- **sns.boxplot()**: Creates a box plot to show the distribution of a continuous variable and detect outliers.
- **sns.violinplot():** Combines a box plot and a kernel density plot to show the distribution of a continuous variable.
- **sns.pairplot():** Creates a matrix of scatter plots to show relationships between multiple variables.
- **sns.heatmap():** Plots a heatmap to visualize matrix-like data, such as a correlation matrix.
- **sns.distplot():** (Deprecated in favor of sns.histplot and sns.kdeplot) Combines a histogram and kernel density estimate plot.
- **sns.kdeplot():** Plots a kernel density estimate to show the distribution of a single variable.
- **sns.jointplot():** Plots a scatter plot with histograms for the x and y axes to show the relationship between two variables.
- **sns.lmplot():** Creates a scatter plot with a linear regression model fit.
- **sns.catplot():** Creates a categorical plot, combining several types like strip, swarm, and box plots.

In [ ]:
```
1
```