

Python Fundamentals For DataScience

In []:

1

1. Variables

In []:

1

In [3]:

```
1 #Integer
2 a=10
3 #Float
4 r = 4.5
5 #String
6 city = "Bangalore"
7 #Boolean
8 s= True
9 #Complex number
10 num=6+7j
11
```

In [4]:

```
1 #check the type of variable
2 type(num)
```

Out[4]: complex

2.Keywords

In []:

1

3. Conditional Statements

In [38]:

```
1 #simple if
2 n=24
3 if n%2==0:
4     print("Even number")
```

Even number

```
In [39]: 1 # if else
2 age=34
3 if age>=18:
4     print("Eligible to vote")
5 else:
6     print("Not eligible")
```

Eligible to vote

```
In [40]: 1 #if else elif
2 marks=23
3 if marks>20:
4     print("Just pass")
5 elif marks>90:
6     print("Excellent")
7 else:
8     print("You are fail")
```

Just pass

4. Looping Statements

```
In [41]: 1 #while Loop
2 c=1
3 while c<=10:
4     print(c)
5     c=c+1
6
```

1
2
3
4
5
6
7
8
9
10

In [50]:

```

1 #for Loop
2 for i in range(25):
3     if i%2==0:
4         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 []:

1

5. Functions

In []:

```

1 #function
2 def calculate_bill(price, quantity):
3     total = price * quantity
4     return total
5
6 bill = calculate_bill(150, 3)
7 print("Total bill amount:", bill)
8

```

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]:

```

1 square = lambda x: x * x
2 square(6)

```

Out[68]: 36

6. DataStructures in python

6. 1 Lists

```
In [11]: 1 #creating list  
2 num = [10, 20, 30, 40, 50]
```

```
In [6]: 1 # 1. Access element  
2 print(num[0])  
3
```

10

```
In [12]: 1 # 2. Add element  
2 num.append(60)  
3 num  
4
```

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

```
In [13]: 1 # 3. Insert element  
2 num.insert(2, 25)  
3  
4 num
```

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

```
In [14]: 1 # 4. Remove element  
2 num.remove(40)  
3  
4 num
```

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

```
In [15]: 1 # 5. Length of List  
2 print(len(numbers))  
3  
4 print(numbers)
```

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

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

[1, 4, 9, 16, 25]

6.2 Tuple

```
In [22]: 1 days = ("Mon", "Mon", "Tue", "Wed", "Thu", "Fri")
          2
```

```
In [23]: 1 # 1. Access element
          2 print(days[1])
          3
          4 # 2. Length
          5 print(len(days))
          6
          7 # 3. Count element
          8 print(days.count("Mon"))
          9
          10 # 4. Index of element
          11 print(days.index("Wed"))
          12
          13 # 5. Loop through tuple
          14 for day in days:
          15     print(day)
          16
```

Mon

6

2

3

Mon

Mon

Tue

Wed

Thu

Fri

6.3 Sets

```
In [19]: 1 cities = {"Bangalore", "Delhi", "Mumbai", "Delhi"}
          2 print(cities) # Duplicate removed
```

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

```
In [ ]: 1 # 1. Add element
2 cities.add("Chennai")
3
4 # 2. Remove element
5 cities.remove("Delhi")
6
7 # 3. Union
8 set1 = {1, 2, 3}
9 set2 = {3, 4, 5}
10 print(set1.union(set2))
11
12 # 4. Intersection
13 print(set1.intersection(set2))
14
15 # 5. Difference
16 print(set1.difference(set2))
17
```

6.4 Dictionary

```
In [20]: 1 #create dictionary
2 student = {
3     "name": "Rahul",
4     "age": 22,
5     "course": "Data Science"
6 }
7
```

```
In [21]: 1 # 1. Access value
2 print(student["name"])
3
4 # 2. Add new key-value
5 student["college"] = "ABC University"
6
7 # 3. Update value
8 student["age"] = 23
9
10 # 4. Get all keys
11 print(student.keys())
12
13 # 5. Get all values
14 print(student.values())
15
16 print(student)
17
```

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

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.



```
In [ ]: 1 !pip install numpy # installing numpy
2
```

```
In [ ]: 1 import numpy as np
```

7. Arrays

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

```
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\numpy\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\numpy\libs\libopenblas.EL2C6PLE4ZYW3ECEVIV30XXGRN2NRFM2.gfortran-win_amd64.dll
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\numpy\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]: 1 # Multi-dimensional array
2 arr2D = np.array([[1, 2, 3], [4, 5, 6]])
3 arr2D.shape
```

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

```
In [27]: 1 # Zeros Array
2 zeros = np.zeros((3, 3)) # 3x3 matrix filled with zeros
3 zeros
```

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

In [28]:

```
1 # Ones Array
2 ones = np.ones((2, 4)) # 2x4 matrix filled with ones
3 ones
```

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

In [29]:

```
1 # Identity Matrix
2 identity_matrix = np.eye(3) # 3x3 identity matrix
3 identity_matrix
```

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

In [51]:

```
1 #array reshape
2 arr = np.arange(1, 10) # 1D array
3 print("Array is \n", arr)
4 reshaped_arr = arr.reshape(3, 3) # Reshape into 3x3 matrix
5 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]:

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

```

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

```

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

In [59]:

```

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

```

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

In [60]:

```

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

```

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

In [61]:

```

1 #sum of elements in array
2 import numpy as np
3
4 # Creating a 3x4 array
5 matrix = np.array([
6     [10, 20, 30, 40],
7     [5, 15, 25, 35],
8     [2, 4, 6, 8]
9 ])
10
11 # Default sum (across all elements)
12 print("Total sum:", matrix.sum())
13
14 # Sum along axis 0 (column-wise sum)
15 print("Sum along axis 0:", matrix.sum(axis=0))
16
17 # Sum along axis 1 (row-wise sum)
18 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]:

```

1 #Sorting array
2 data = np.array([
3     [7, 1, 4],
4     [8, 6, 5],
5     [1, 2, 3]
6 ])
7
8 print(np.sort(data)) # Sorts each row individually
9 print(np.sort(data, axis=0)) # Sorts each column individually
10 print(np.sort(data, axis=None)) # Flattens and sorts entire array
11 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]:

```

1 #Determinant
2 x=[[2,3,4],
3   [5,6,7],
4   [8,9,0]]
5 np.linalg.det(x)

```

Out[63]: 29.99999999999999

In [64]:

```

1 #Dot product
2 import numpy as np
3
4 a = np.array([[3, 4, 5],
5             [9, 0, 3]])
6
7 b = np.array([[2, 6, 1],
8             [8, 5, 3],
9             [8, 5, 3]])
10
11 print(np.dot(a,b))

```

```

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

```

In [65]:

```

1 #Diagonal and trace of Matrix
2 matrix = np.array([[1, 2, 3],
3                     [4, 5, 6],
4                     [7, 8, 9]])
5
6 d = np.diagonal(matrix)
7
8 print("\n The diagonal elements are:",d)
9 print("\n The trace of a matrix is:",np.trace(matrix))
10

```

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?

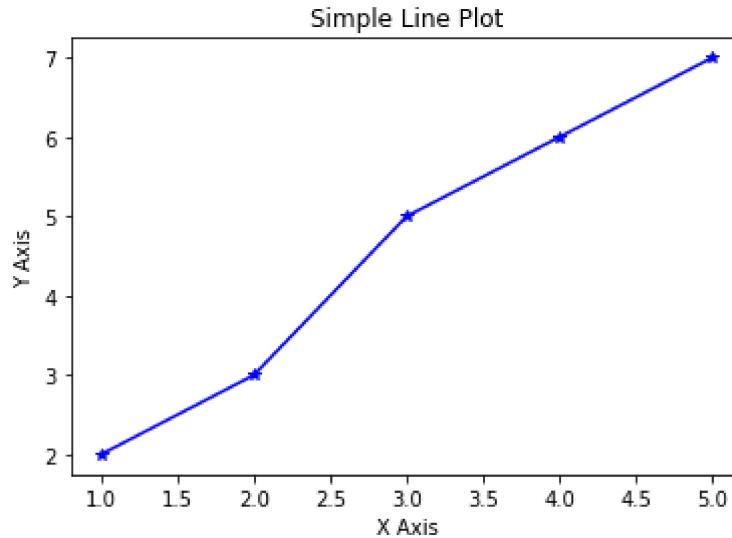
We import matplotlib.pyplot and use simple functions like plot(), bar(), and scatter() to visualize your data with full customization

8.1 Line plot

Used to show trends over time or continuous data

In [4]:

```
1 import matplotlib.pyplot as plt
2
3 x = [1, 2, 3, 4, 5]
4 y = [2, 3, 5, 6, 7]
5
6 plt.plot(x, y, color='blue', marker='*', linestyle='--')
7 plt.title("Simple Line Plot")
8 plt.xlabel("X Axis")
9 plt.ylabel("Y Axis")
10 plt.show()
11
12 plt.show()
```

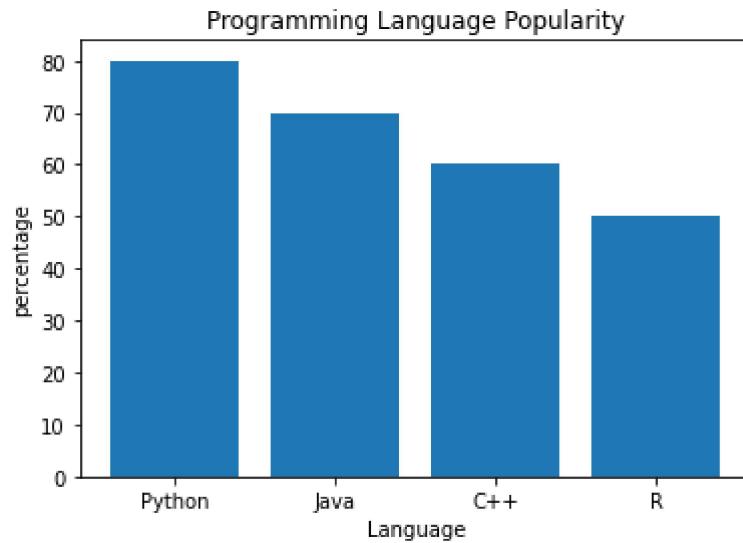


8.2 BarPlot

Bar Chart: Used to compare categories or groups

In [5]:

```
1 x = ['Python', 'Java', 'C++', 'R']
2 y = [80, 70, 60, 50]
3
4 plt.bar(x, y)
5 plt.title("Programming Language Popularity")
6 plt.xlabel("Language")
7 plt.ylabel("percentage")
8 plt.show()
9
```

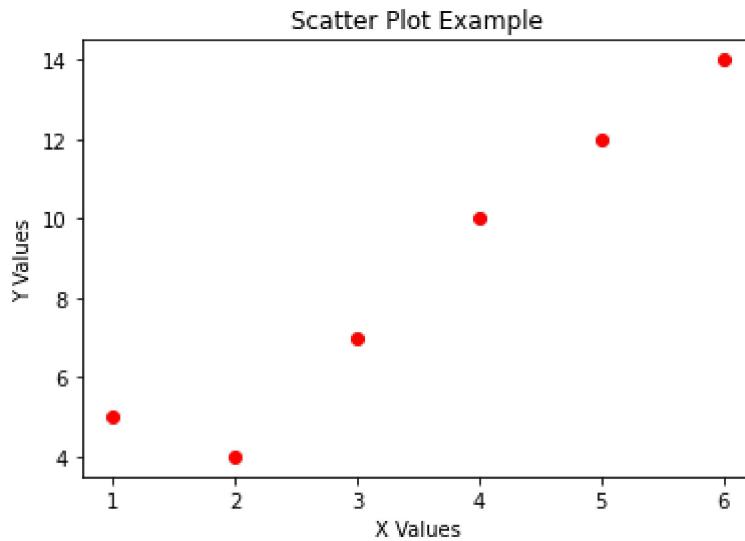


8.3 Scatterplot

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

In [33]:

```
1 x = [1, 2, 3, 4, 5, 6]
2 y = [5, 4, 7, 10, 12, 14]
3
4 plt.scatter(x, y, color='red')
5 plt.title("Scatter Plot Example")
6 plt.xlabel("X Values")
7 plt.ylabel("Y Values")
8 plt.show()
9
```

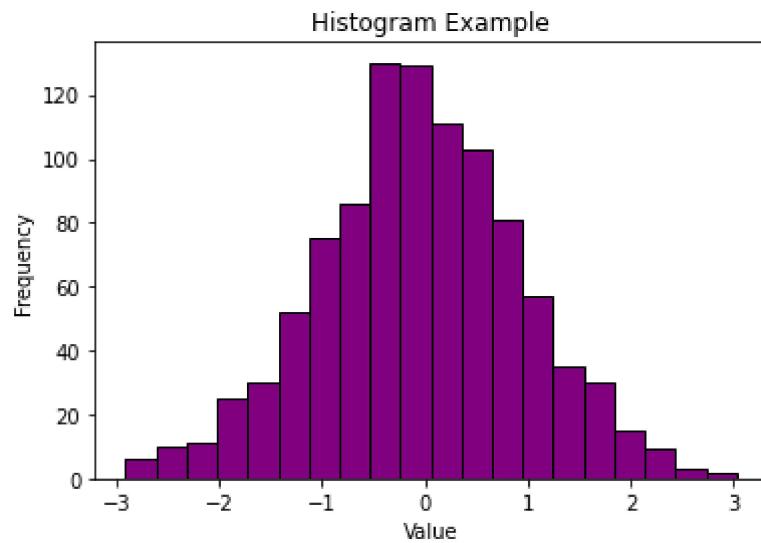


8.4 Histogram

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

In [32]:

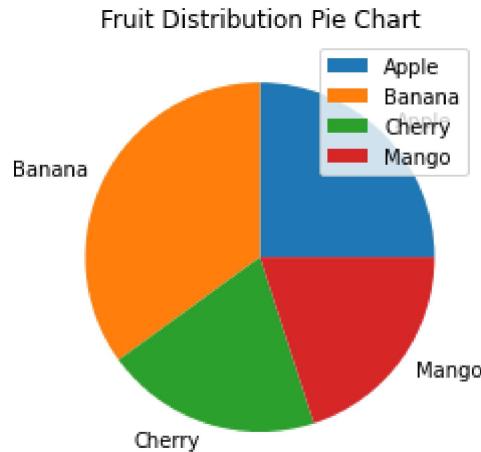
```
1 import numpy as np
2 data = np.random.randn(1000)
3
4 plt.hist(data, bins=20, color='purple', edgecolor='black')
5 plt.title("Histogram Example")
6 plt.xlabel("Value")
7 plt.ylabel("Frequency")
8 plt.show()
9
```



8.5 Pie chart

Shows parts of a whole percentage contribution of each category.

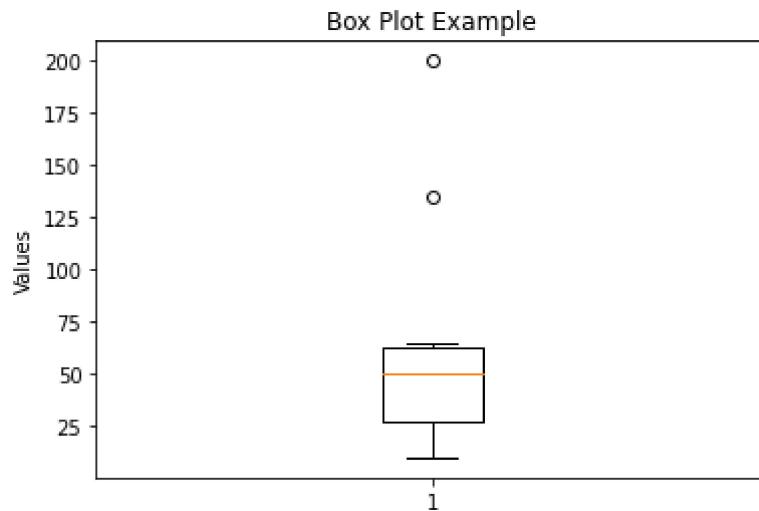
```
In [7]: 1 labels = ['Apple', 'Banana', 'Cherry', 'Mango']
2 sizes = [25, 35, 20, 20]
3
4 plt.pie(sizes, labels=labels)
5 plt.title("Fruit Distribution Pie Chart")
6 plt.legend()
7 plt.show()
8
```



8.6 BoxPlot

Displays data spread, outliers, median, and quartiles.

```
In [9]: 1 data = [10, 20, 30, 25, 40, 135, 50, 55, 60, 65, 200]
2 plt.boxplot(data)
3 plt.title("Box Plot Example")
4 plt.ylabel("Values")
5 plt.show()
6
```



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

