Detailed Notes on Use of NumPy in Python

Introduction to NumPy

What is NumPy?

- **NumPy (Numerical Python)** is a powerful **open-source Python library** used for numerical and scientific computing.
- It provides:
  - **N-dimensional array objects** (ndarray)
  - **Fast mathematical operations**
  - **Linear algebra, random number generation, Fourier transforms**
  - **Integration with C/C++ and other Python libraries** like pandas, scikit-learn, and TensorFlow.

Why Use NumPy?

- Pure Python lists are **slow** for numerical computation.
- NumPy uses **vectorization** and **optimized C code**, making it much **faster** and **memory efficient**.

1. Basic Level (60%)

1.1 Installing and Importing NumPy

```
# Install NumPy (only once)
# pip install numpy

# Importing NumPy
import numpy as np
```

1.2 Creating NumPy Arrays

From Python Lists

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
print("Array:", arr)
print("Type:", type(arr))
```

2D Array

```
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print(arr2)
```

1.3 Basic Array Attributes

```
print("Shape:", arr2.shape)   # (2, 3)
print("Dimensions:", arr2.ndim)
print("Data Type:", arr2.dtype)
print("Size:", arr2.size)
print("Item Size (bytes):", arr2.itemsize)
```

1.4 Array Creation Methods

```
# Arrays filled with zeros or ones
a = np.zeros((3,3))
b = np.ones((2,4))
c = np.full((2,2), 7)
d = np.eye(3)  # Identity matrix
e = np.arange(0, 10, 2)  # start, stop, step
f = np.linspace(0, 1, 5)  # equally spaced
```

1.5 Indexing and Slicing

```
arr = np.array([10, 20, 30, 40, 50])
print(arr[1])      # 20
print(arr[1:4])    # [20 30 40]
print(arr[-1])     # 50

# 2D Example
matrix = np.array([[10, 20, 30],
                   [40, 50, 60],
                   [70, 80, 90]])
print(matrix[1, 2])     # 60
print(matrix[:, 1])     # 2nd column
print(matrix[0:2, 1:3]) # subarray
```

1.6 Basic Operations

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

print(arr1 + arr2)
print(arr1 * arr2)
print(arr1 ** 2)
print(np.sqrt(arr2))
print(np.sum(arr2))
```

1.7 Broadcasting

- Broadcasting allows NumPy to perform arithmetic operations between arrays of **different shapes**.

```
arr = np.array([1, 2, 3])
matrix = np.array([[10], [20], [30]])
result = matrix + arr
print(result)
```

1.8 Boolean Indexing

```
arr = np.array([5, 10, 15, 20, 25])
print(arr[arr > 15])   # [20 25]
```

1.9 Common Mathematical Functions

```
arr = np.array([1, 2, 3, 4, 5])

print(np.mean(arr))   # average
print(np.std(arr))    # standard deviation
print(np.min(arr))    # minimum
print(np.max(arr))    # maximum
print(np.sum(arr))    # sum
print(np.prod(arr))   # product
```

2. Intermediate Level (20%)

2.1 Reshaping and Flattening Arrays

```
arr = np.arange(12)
print(arr.reshape(3, 4))    # reshape
print(arr.reshape(2, -1))   # auto-dimension
print(arr.flatten())        # convert to 1D
```

2.2 Stacking and Splitting Arrays

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])

# Stacking
print(np.hstack((a, b)))
print(np.vstack((a, b)))

# Splitting
x = np.arange(1, 10)
print(np.split(x, 3))
```

2.3 Random Module in NumPy

```
np.random.seed(42)  # reproducible results

print(np.random.rand(3))         # random floats (0-1)
print(np.random.randint(1, 10, 5)) # random integers
print(np.random.randn(2, 3))     # normal distribution
print(np.random.choice([1, 2, 3, 4], 5))
```

2.4 Linear Algebra Operations

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

print(np.dot(A, B))         # matrix multiplication
print(np.transpose(A))      # transpose
print(np.linalg.inv(A))     # inverse
print(np.linalg.det(A))     # determinant
```

2.5 Saving and Loading Data

```
arr = np.array([1, 2, 3, 4, 5])
np.save('array_data.npy', arr)
loaded = np.load('array_data.npy')
print(loaded)
```

3. Advanced Level (20%)

3.1 Vectorization and Performance

Instead of using loops, NumPy performs operations on entire arrays at once.

```
import time
x = np.arange(1_000_000)
y = np.arange(1_000_000)

start = time.time()
result = x + y   # Vectorized operation
end = time.time()
print("Time (NumPy):", end - start)
```

Compare with Python list loop:

```
a = list(range(1_000_000))
b = list(range(1_000_000))
start = time.time()
result = [a[i] + b[i] for i in range(len(a))]
end = time.time()
print("Time (List):", end - start)
```

NumPy is much faster!

3.2 Fancy Indexing and Advanced Slicing

```
arr = np.arange(10, 100, 10)
indices = [1, 3, 5, 7]
print(arr[indices])  # [20 40 60 80]

# Using conditions
print(arr[(arr > 30) & (arr < 80)])
```

3.3 Structured Arrays

Useful for handling mixed data types.

```
data = np.array([
    ('Ali', 23, 72.5),
    ('Sara', 21, 60.3)
], dtype=[('name', 'U10'), ('age', 'i4'), ('weight', 'f4')])

print(data['name'])
print(data['age'])
```

3.4 Advanced Broadcasting Example

```
x = np.arange(3)
y = np.arange(3).reshape(3, 1)
result = x + y
print(result)
```

Explanation:
NumPy automatically “expands” the smaller array dimensions to match larger ones.

3.5 Masked Arrays (Handling Missing Data)

```
from numpy import ma

arr = np.array([1, 2, 3, -999, 5])
masked = ma.masked_values(arr, -999)
print(masked)
print(masked.mean())
```

3.6 Integration with Other Libraries

NumPy forms the **foundation** for many libraries:

- **pandas** (data manipulation)
- **matplotlib** (visualization)
- **scikit-learn** (machine learning)
- **TensorFlow / PyTorch** (deep learning)

Example with pandas:

```
import pandas as pd

data = np.random.randint(1, 100, (3, 3))
df = pd.DataFrame(data, columns=['A', 'B', 'C'])
print(df)
```

Summary Table

Level	Concept	Description
Basic	Array creation, indexing, slicing	Foundation for numerical work
Basic	Broadcasting, math functions	Perform vectorized math
Intermediate	Reshape, stack, random	Moderate data manipulation
Intermediate	Linear algebra	Solve matrix equations
Advanced	Vectorization, fancy indexing	Performance optimization
Advanced	Structured/masked arrays	Handle complex/missing data

Conclusion

- NumPy provides **speed, simplicity, and flexibility** for numerical and matrix operations.
- It is the **backbone of data science and machine learning in Python**.
- Mastering array operations, broadcasting, and vectorization unlocks efficient computation.