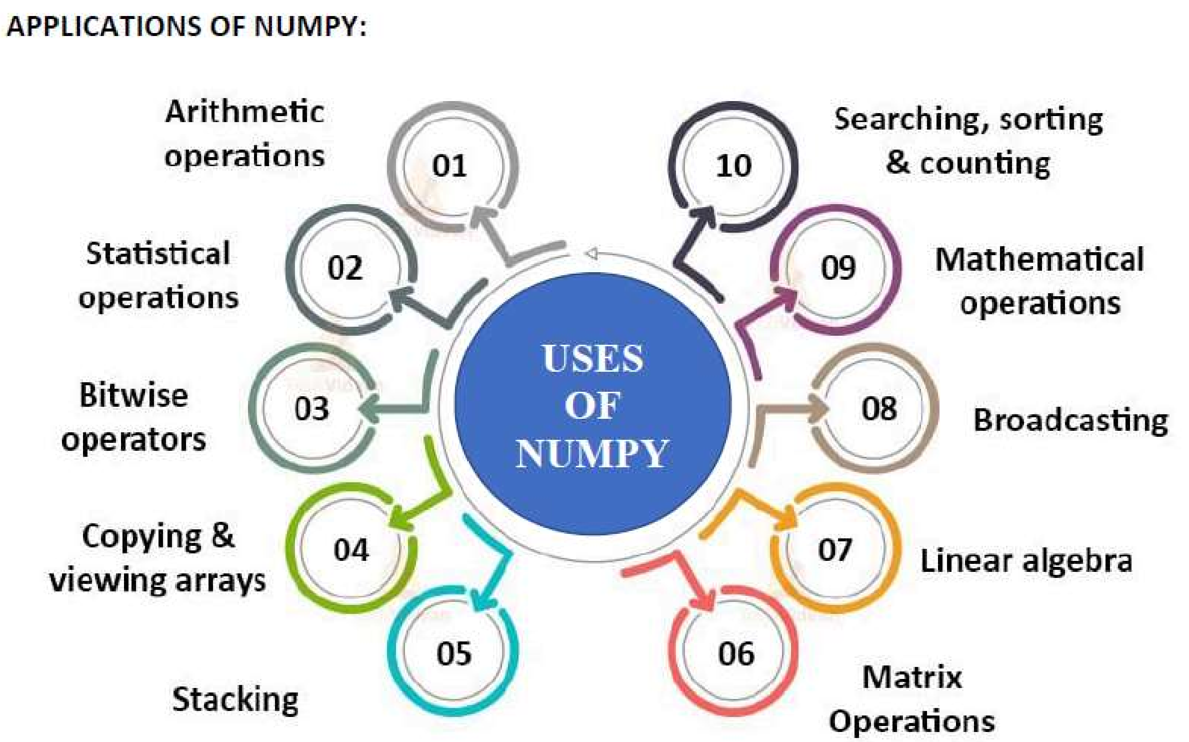
NumPy

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]:
```python
import numpy as np
```

```
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\.libs\libopenblas.EL2C6PLE4ZYW3ECEVIV3OXXGRN2NRFM2.gfortran-win_amd64.dll
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\.libs\libopenblas.XWYDX2IKJW2NMTWSFYNGFUWKQU3LYTCZ.gfortran-win_amd64.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

1. Creating Arrays

In [2]:
```python
# From a list or tuple
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)
```

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

In [3]:
```
# Multi-dimensional array
arr2D = np.array([[1, 2, 3], [4, 5, 6]])
print("\n2D Array:\n", arr2D)
```

```
2D Array:
 [[1 2 3]
 [4 5 6]]
```

In [4]:
```
# Zeros Array
zeros = np.zeros((3, 3))  # 3x3 matrix filled with zeros
print("\nZeros Array:\n", zeros)
```

```
Zeros Array:
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

In [5]:
```
# Ones Array
ones = np.ones((2, 4))  # 2x4 matrix filled with ones
print("\nOnes Array:\n", ones)
```

```
Ones Array:
 [[1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

In [6]:
```
# Empty Array (random values)
empty_arr = np.empty((2, 3))  # 2x3 matrix with random values
print("\nEmpty Array:\n", empty_arr)
```

```
Empty Array:
 [[6.23042070e-307 4.67296746e-307 1.69121096e-306]
 [3.22646744e-307 2.67015654e-306 2.42092166e-322]]
```

In [9]:
```
# Array with a Range of Values
arr_range = np.arange(0, 10 ,4)  # start=0, stop=10, step=2
print("\nArray with range:\n", arr_range)
```

```
Array with range:
 [0 4 8]
```

In [20]:
```
# Array with Linearly Spaced Values
lin_space = np.linspace(0, 10, 5)  # 5 values between 0 and 10
print("\nLinearly spaced array:\n", lin_space)
```

```
Linearly spaced array:
 [ 0.   2.5  5.   7.5 10. ]
```

In [21]:
```python
# Identity Matrix
identity_matrix = np.eye(3)  # 3x3 identity matrix
print("\nIdentity Matrix:\n", identity_matrix)
```

```
Identity Matrix:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

2. Accessing Elements

In [26]:
```python
arr=[2,4,6,8,9,0]
print("\nFirst element:", arr[0])  # First element
print("Last element:", arr[-1]) # Last element
print("Slice [1:4]:", arr[1:4])  # Elements from index 1 to 3
```

```
First element: 2
Last element: 0
Slice [1:4]: [4, 6, 8]
```

3. Reshaping Arrays

In [27]:
```python
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]]
```

4. Basic Mathematical Operations

In [22]:
```
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]]
```

5. Aggregate Functions

In [11]:
```
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
```

6. Array Concatenation

In [12]:
```
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]]
```

7. Transpose of a Matrix

In [37]:
```
arr2D = np.array([[1, 2, 3], [4, 5, 6]])
print("\nTransposed Matrix:\n", arr2D.T)
```

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

8. Random Number Generation

In [38]:
```
rand_arr = np.random.rand(3, 3)  # 3x3 matrix with random values
print("\nRandom Array:\n", rand_arr)

rand_ints = np.random.randint(1, 10, (3, 3))  # 3x3 matrix with random integ
print("\nRandom Integers Matrix:\n", rand_ints)
```

```
Random Array:
 [[0.63486049 0.38724691 0.36542308]
 [0.97422643 0.90003958 0.99953383]
 [0.09503771 0.25216084 0.51989326]]

Random Integers Matrix:
 [[1 9 5]
 [8 7 5]
 [2 9 3]]
```

9. Boolean Masking & Filtering

In [31]:
```
arr = np.array([1, 2, 3, 4, 5, 6])
print("\nElements greater than 3:", arr[arr > 3])
```

```
Elements greater than 3: [4 5 6]
```

In [2]:
```
import numpy as np

# Creating a structured array
numbers = np.linspace(5, 50, 24, dtype=int).reshape(4,-1)
#-1 is to automatically calculate the number of columns
#based on the total elements
print(numbers)
# Creating a mask where numbers are divisible by 4
mask = numbers % 4 == 0

# Using the mask to filter values
filtered_values = numbers[mask]

print(filtered_values)  # Extracted numbers
```

```
[[ 5  6  8 10 12 14]
 [16 18 20 22 24 26]
 [28 30 32 34 36 38]
 [40 42 44 46 48 50]]
[ 8 12 16 20 24 28 32 36 40 44 48]
```

10. Sum of elements in array

In [13]:
```
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]
```

11. Sorting of an array

In [44]:
```
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]
```

12. Standardizing the data

In [14]:
```python
import numpy as np

# Sample dataset (rows = samples, columns = features)
data = np.array([
    [50, 2000, 3.5],
    [20, 1500, 2.1],
    [30, 1800, 4.3],
    [40, 2100, 3.9]
])

# Compute mean and standard deviation
mean = np.mean(data, axis=0)
std_dev = np.std(data, axis=0)

# Standardization formula: (X - mean) / std_dev
standardized_data = (data - mean) / std_dev

print("Original Data:\n", data)
print("\nStandardized Data:\n", standardized_data)
data.shape
print(mean)
print(std_dev)
```

```
Original Data:
 [[  50.  2000.     3.5]
 [  20.  1500.     2.1]
 [  30.  1800.     4.3]
 [  40.  2100.     3.9]]

Standardized Data:
 [[ 1.34164079  0.65465367  0.06030227]
 [-1.34164079 -1.52752523 -1.62816126]
 [-0.4472136  -0.21821789  1.02513857]
 [ 0.4472136   1.09108945  0.54272042]]
[  35.    1850.      3.45]
[ 11.18033989 229.12878475   0.8291562 ]
```

In [15]:
```python
a=50-35
a=a/11.18
a=140/4
a
```

Out[15]:
```
35.0
```

13.Determinant

In [23]:
```
x=[[2,3,4],
   [5,6,7],
   [8,9,0]]
np.linalg.det(x)
```

Out[23]: 29.99999999999999

14. Dot Product

In [4]:
```
import numpy as np

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

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

print(np.dot(a,b))

```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [4], in <cell line: 9>()
      3 a = np.array([[3, 4, 5],
      4               [9, 0, 3]])
      6 b = np.array([[2, 6, 1],
      7               [8, 5, 3]])
----> 9 print(np.dot(a,b))

File <__array_function__ internals>:180, in dot(*args, **kwargs)

ValueError: shapes (2,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)
```

In [2]:
```
# Transpose b to make it (3,2)
b_T = b.T

# Now perform matrix multiplication
product = np.dot(a, b_T)  # OR a @ b_T

print(product)

```

```
[[35 59]
 [21 81]]
```

14. Diagonal &Trace of Matrix

In [19]:

```python
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
```

15. Matrix Inversion (np.linalg.inv())

The inverse of a square matrix A is such that `A * A_inv = I` , where `I` is the identity matrix.

The inverse of a square matrix A is denoted as A^{-1}, and it satisfies:

$$A \cdot A^{-1} = A^{-1} \cdot A = I$$

where I is the identity matrix.

💡 A matrix is invertible only if its determinant is nonzero $(\det(A) \neq 0)$.

Example:

$$A = \begin{bmatrix} 4 & 7 \\ 2 & 6 \end{bmatrix}$$

Inverse Formula for a 2×2 Matrix:

$$A^{-1} = \frac{1}{\det(A)} \cdot \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

where $\det(A) = (4)(6) - (7)(2) = 10$.

$$A^{-1} = \frac{1}{10} \begin{bmatrix} 6 & -7 \\ -2 & 4 \end{bmatrix}$$

In [35]:
```
import numpy as np

A = np.array([[4, 7], [2, 6]])
A_inv = np.linalg.inv(A)
print("The Inverse of Matrix is\n",A_inv)
print(A_inv @ A)
print(A @ A_inv)
```

```
The Inverse of Matrix is
 [[ 0.6 -0.7]
 [-0.2  0.4]]
[[ 1.00000000e+00 -2.22044605e-16]
 [ 0.00000000e+00  1.00000000e+00]]
[[ 1.00000000e+00 -1.11022302e-16]
 [-1.11022302e-16  1.00000000e+00]]
```

16. Eigenvalues and Eigenvectors

For a square matrix A, eigenvalues and eigenvectors satisfy Av = λv.

$$Av = \lambda v$$

To find eigenvalues, solve the characteristic equation:

$$\det(A - \lambda I) = 0$$

Example:

$$A = \begin{bmatrix} 4 & -2 \\ 1 & 1 \end{bmatrix}$$

Solving $\det(A - \lambda I) = 0$:

$$\begin{vmatrix} 4-\lambda & -2 \\ 1 & 1-\lambda \end{vmatrix} = 0$$

$$(4-\lambda)(1-\lambda) - (-2)(1) = 0$$

Solving for λ, we get the eigenvalues.

In [36]:
```
A = np.array([[4, 7], [2, 6]])
eig_values, eig_vectors = np.linalg.eig(A)
print('Eigenvalues:', eig_values)
print('Eigenvectors:\n', eig_vectors)
```

```
Eigenvalues: [1.12701665 8.87298335]
Eigenvectors:
 [[-0.92511345 -0.82071729]
 [ 0.37969079 -0.57133452]]
```

17. Solving Linear Equations (np.linalg.solve())

Solving a system `Ax = B` for `x`.

A system of equations:

$$Ax = B$$

For example:

$$2x + 3y = 8$$

$$5x + 7y = 19$$

can be written as:

$$A = \begin{bmatrix} 2 & 3 \\ 5 & 7 \end{bmatrix}, \quad B = \begin{bmatrix} 8 \\ 19 \end{bmatrix}$$

$$x = A^{-1}B$$

In [37]:

```
A = np.array([[2, 3], [5, 7]])
B = np.array([8, 19])
solution = np.linalg.solve(A, B)
print('Solution:', solution)
```

```
Solution: [1. 2.]
```

18. Matrix Norm (np.linalg.norm())

Computes the Euclidean norm (magnitude) of a vector.

In [38]:

```
v = np.array([3, 4])
norm_v = np.linalg.norm(v)
print(norm_v)
```

```
5.0
```

@@@@@@@@@@@@@@@@@@@@@@@@@@@

In [ ]:

```

```