# Numpy (Numerical Python)

## is a Python library used for working with arrays

- **NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.**
- **The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.**
- **Arrays are very frequently used in data science, where speed and resources are very important.**

## Why is NumPy Faster Than Lists?

- **NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.**

## Dimensions in Arrays

- **0-D Arrays**
- **1-D Arrays**
- **2-D Arrays**
- **3-D arrays**
- **Higher Dimensional Arrays**

## The Basics

- **Get Dimension ➔ arr.ndim**
- **Get Shape ➔ arr.shape**
- **Get Type ➔ arr.dtype**
- **Get Size ➔ arr.itemsize**
- **Get total size ➔ arr.nbytes**
- **Get number of elements ➔ arr.size**

# 1. Creating an Array in NumPy

Array can either be vector or matrix. A vector is a one-dimensional array, and a matrix is a two or more-dimensional array.
`np.array([elements])` ➔ Create a NumPy array from a list or tuple.

## 1.1 Generating Array

NumPy offers various options to generate an array depending on need, such as:

- Generating identity array
- Generating zero array of a given size
- Generating one's array with a given size
- Generating an array in a given range
- Generating an array with random values

`np.zeros((shape))` ➔ All 0s matrix

`np.ones((shape))` ➔ All 1s matrix

`np.full((shape), value)` ➔ Array filled with a constant value

`np.empty((shape))` ➔ Uninitialized array

`np.arange(start, stop, step)` ➔ Array with evenly spaced values within a given interval

`np.linspace(start, stop, num)` ➔ Array of num evenly spaced values between start and stop
`np.random.rand(d0, d1, ...)` ➔ Create an array of random values (uniform distribution)
`np.random.randn(d0, d1, ...)` ➔ Create an array of random values (normal distribution)
`np.random.randint(low, high, size)` ➔ Create an array of random integers
`np.eye()` ➔ The identity flexible matrix
`np.identity()` ➔ The identity square matrix

# 2. Data Selection: Indexing and slicing

Indexing: Selecting individual elements from the array
Slicing: Selecting a group of elements from the array.

**arr[index]** ➔ Access a specific element by index

**arr[start:stop]** ➔ Slice an array from index start to stop

**arr[start:stop:step]** ➔ Slice with a step

**arr[condition]** ➔ Return elements where the condition is true

# 3. Basic Array Operations

## 3.1 Quick Arithmetic operation: Addition, Subtraction, Multiplication, Division, Squaring

How addition works:

[0, 1, 2, 3, 4] + [6, 7, 8, 9, 10] = [ 6, 8, 10, 12, 14] same way as other operations.

## 3.2 Universal functions

NumPy universal functions allow to compute math, trigonometric, logical and comparison operations such as sin, cos, tan, exponent(exp), log, square, greater, less, etc...

**np.add(arr1, arr2)** ➔ Add arrays

**np.subtract(arr1, arr2)** ➔ Subtract arrays

**np.multiply(arr1, arr2)** ➔ Multiply arrays

**np.divide(arr1, arr2)** ➔ Divide arrays

**np.dot(arr1, arr2)** ➔ Dot product of two arrays

**np.matmul(arr1, arr2)** ➔ Matrix multiplication

**np.sin(arr)** ➔ sin of array elements

**np.cos(arr)** ➔ cos of array elements

**np.tan(arr)** ➔ tan of array elements

**np.exp(arr)** ➔ Element-wise exponentiation

**np.log(arr)** ➔ Element-wise natural logarithm

**np.sqrt(arr)** ➔ Element-wise square root

**np.abs(arr)** ➔ Absolute values of elements

**np.power(arr1,arr2)** ➔ power of two arrays

**np.greater(arr1,arr2)** ➔ Comparison of two arrays

**np.less(arr1,arr2)** ➔ Comparison of two arrays

**np.sum(arr)** ➔ Sum of array elements

**np.cumsum(arr)** ➔ Cumulative sum

**np.cumprod(arr)** ➔ Cumulative product

# 4. Basic Statistics

With NumPy, we can compute the basic statistics such as the standard deviation (std), variance (var), mean, median, minimum value, maximum value of an array.

**np.mean(arr)** ➔ Mean of array elements

**np.median(arr)** ➔ Median of array elements

**np.std(arr)** ➔ Standard deviation

**np.var(arr)** ➔ Variance

**np.min(arr)** ➔ Minimum value

**np.max(arr)** ➔ Maximum value

# 5. Data Manipulation

In NumPy, array manipulation includes reshaping, combining, splitting, adding dimensions, and more. These operations help manage data more effectively without changing the underlying data, just how it's organized or represented.

**np.reshape(arr, new_shape)** ➔ Change the shape of an array

**np.transpose(arr)** ➔ Transpose an array (swap rows and columns)

**np.swapaxes(arr, axis1, axis2)** ➔ Swap two axes of an array

**np.ravel(arr)** ➔ Flatten an array into 1D

**np.concatenate((arr1, arr2), axis)** ➔ Concatenate arrays along an axis

**np.stack((arr1, arr2), axis)** ➔ Stack arrays along a new axis

**np.hstack((arr1, arr2))** ➔ Horizontally stack arrays

**np.vstack((arr1, arr2))** ➔ Vertically stack arrays

**np.split(arr, sections, axis)** ➔ Split an array into multiple sub-arrays

**np.expand_dims(arr, axis)** ➔ Add a new dimension to an array

**np.squeeze(arr)** ➔ Remove single-dimensional entries from the shape of an array