A Beginner's Guide to NumPy- Part 1

NumPy, short for Numerical Python, is a fundamental library in Python for numerical and scientific computing. It provides efficient data storage with multi-dimensional arrays (ndarray) and a wide range of mathematical functions for array operations. NumPy's significance lies in its efficiency, numerical stability, and interoperability with other scientific libraries, making it essential for tasks like data analysis, machine learning, and engineering simulations in Python.

NumPy Arrays

The fundamental data structure of **NumPy** is the ndarray, short for **N-dimensional array**. It represents a grid of values, all of the same type, and is *indexed by a tuple of non-negative integers*. With NumPy arrays, you can perform efficient mathematical operations on entire data arrays, rather than iterating through individual elements.

Creation of Array:

Lets see various methods to create array

In [3]:
```
import numpy as np
# Creating a 1D array from a Python list
arr_1d = np.array([1, 2, 3, 4, 5])

# Creating a 2D array from a list of lists
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print (type(arr_1d))
print("1-D array: \n", arr_1d)
print (type(arr_2d))
print("2-D array: \n",arr_2d)
```

```
<class 'numpy.ndarray'>
1-D array:
 [1 2 3 4 5]
<class 'numpy.ndarray'>
2-D array:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

NumPy provides several built-in functions that we can use to create arrays with specific initial values. I am dividing this section as,

- Identical values
- Random values
- Evenly spaced

Creating Numpy Arrays with Identical values:

```
In [7]: # Creating an array of zeros - Shape 3*3
zeros_array = np.zeros((3, 3))

print("Zeros array: \n", zeros_array)

# Creating an array of ones shape 2*4
ones_array = np.ones((2, 4))
print("ones array: \n", ones_array)

# Creating an array with fill value. Shape 2*2 , fill value=10
arr_full = np.full((2, 2), 10 )
print("Filled array: \n", arr_full)
```

```
Zeros array: 
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
ones array: 
 [[1. 1. 1. 1.]
 [1. 1. 1. 1.]]
Filled array: 
 [[10 10]
 [10 10]]
```

Creating Numpy Arrays with Random values:

- np.random.rand(shape) - This function generates random numbers from a uniform distribution between 0 and 1.
- np.random.randn(shape) - This function generates random numbers from a standard normal distribution (mean 0 and variance 1).
- np.random.randint(low,high,size/shape)- This function generates random integers between a specified low (inclusive) and high (exclusive) value.

```
In [10]: #np.random.rand(shape)
rand_array = np.random.rand(2, 3)
print("Rand array: \n",rand_array)

#np.random.randn(shape)
randn_array = np.random.randn(2, 3)
print("Randn array: \n",randn_array)

#np.random.randint(low,high,size/shape)
randint_array= np.random.randint(1, 10, (2, 3))
print("Randint array: \n",randint_array)

randint_1Darray = np.random.randint(1, 10, 5)
print("Randint_1Darray: \n",randint_1Darray)
```

```
Rand array:
 [[0.724757   0.67219801 0.67651388]
 [0.43748284 0.1241466  0.12269913]]
Randn array:
 [[-0.33363644  0.94272159 -0.33546413]
 [-0.94222396  1.06473796  0.28713394]]
Randint array:
 [[2 5 1]
 [2 8 7]]
Randint_1Darray:
 [1 3 8 5 8]
```

Creating Numpy Arrays with Evenly spaced values:

- np.arange(start, stop) - Creates an array with evenly spaced values in a given range. Step-optional parameter given to give the required spacing
- np.linspace(start, stop, num) - create an array of evenly spaced numbers over a specified range.

In [18]:
```
arr_arange=np.arange(1, 6)
print("Arange array: \n",arr_arange)
arr_arange_step=np.arange(1, 6, 3)
print("Arange array(Step): \n",arr_arange_step)

arr_lin= np.linspace(1,6, num=3)
print("Arange array(Linspace): \n",arr_lin)
```

```
Arange array:
 [1 2 3 4 5]
Arange array(Step):
 [1 4]
Arange array(Linspace):
 [1.  3.5 6. ]
```

Numpy Broadcasting - Facilitating Array Operations

NumPy broadcasting is a powerful feature that allows arrays of different shapes to be combined in arithmetic operations. It simplifies the process of performing element-wise operations on arrays with different shapes by automatically aligning them along their dimensions.

- Equal Dimensions: If two arrays have the same shape or dimensions, element-wise operations are performed directly on them.
- Dimensions of Size 1: If one of the dimensions of an array has size 1, NumPy automatically "broadcasts" the array with smaller shape to match the shape of the larger array along that dimension.
- Expansion (Stretch): NumPy replicates the elements of the smaller array along the dimensions where it needs to match the shape of the larger array.
- Braodcasting starts with the trailing (right most) dimension and works its way left.

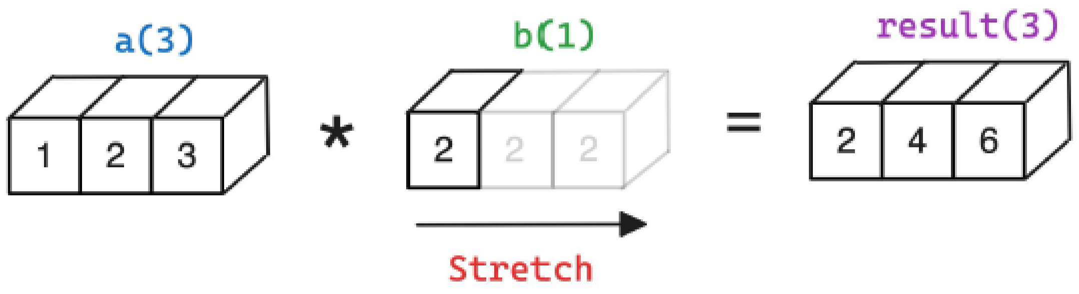
Let's start with a scalar operation with an array.

In [30]:
```python
a= np.arange(1,4) # (1*3 ) array
b=2 #scalar value 2
print(arr_scalar)
print("-"*50)
print(arr_scalar+b)
```

```
[1 2 3]
--------------------------------------------------
[3 4 5]
```

How did this happen? Scalar variable 'b' got stretched in to an array with the same shape as 'a'.

Review the picture below to understand.



In [35]:
```python
a1= np.random.randint(1, 10, (2, 3)) # (2*3 array)
print ("Input Array: \n",array1)
print("-"*50)
print ("Result Array: \n",array1+b)

# Scalar b will stretch itself to match 2*3 array and will get added to every value in
```

```
Input Array:
 [[7 1 9]
 [3 9 8]]
--------------------------------------------------
Result Array:
 [[ 9  3 11]
 [ 5 11 10]]
```

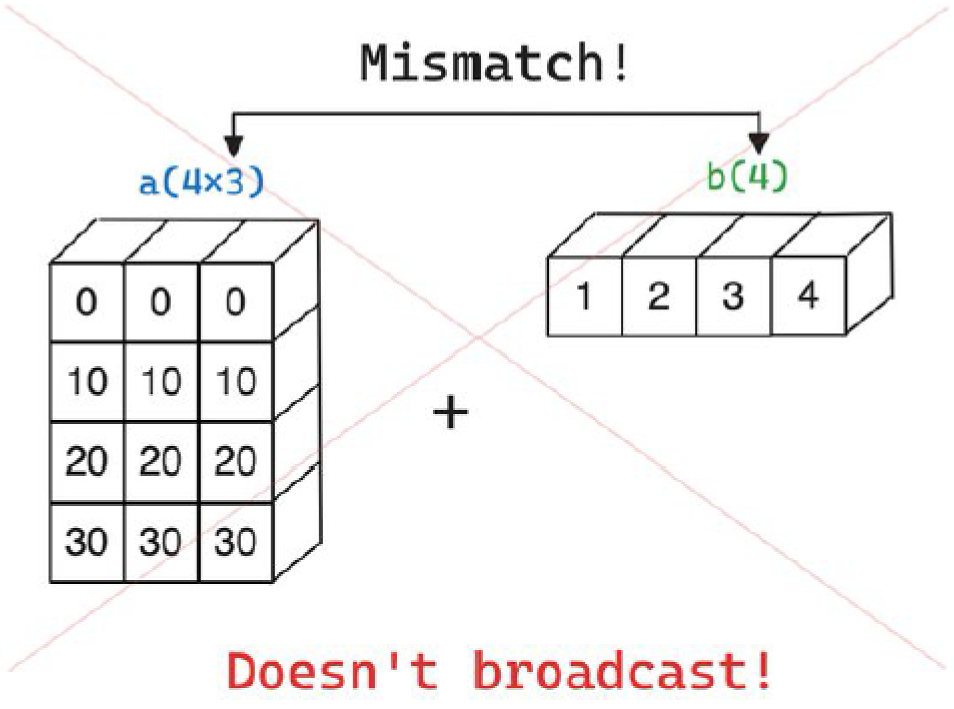
What doesnt work?

In [38]:
```python
arr1= np.array([[0, 0, 0],
                [10, 10, 10],
                [20, 20, 20],
                [30, 30, 30]])

arr2= np.arange(1,5)
print ("Array1:\n",arr1)
print ("Array2:\n",arr2)
```

```
Array1:
 [[ 0  0  0]
 [10 10 10]
 [20 20 20]
 [30 30 30]]
Array2:
 [1 2 3 4]
```

In [39]:
```
print ("Result:\n",arr1+arr2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[39], line 1
----> 1 print ("Result:\n",arr1+arr2)

ValueError: operands could not be broadcast together with shapes (4,3) (4,)
```

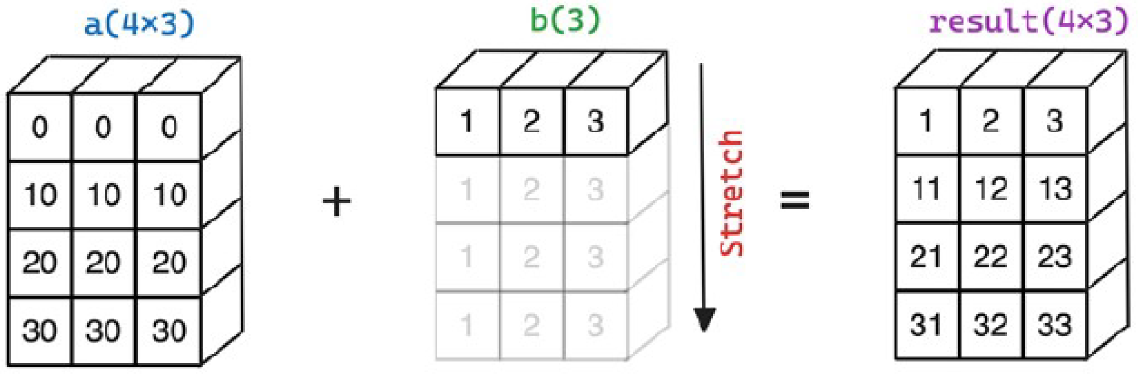


Broadcasting starts from trailing dimension. **(4x3) and (1x4) are not compatible.**

Let's make it compatible by redefining arr2 as a (1x3) array.

In [41]:
```
arr2= np.arange(1,4)
print ("Array1:\n",arr1)
print ("Array2:\n",arr2)
print ("Result:\n",arr1+arr2)
```

```
Array1:
 [[ 0  0  0]
 [10 10 10]
 [20 20 20]
 [30 30 30]]
Array2:
 [1 2 3]
Result:
 [[ 1  2  3]
 [11 12 13]
 [21 22 23]
 [31 32 33]]
```



In this notebook i have explored about the various methods of creating NumPy arrays, arithmetic operations and the role of broadcasting to assist arithmetic operation for arrays of different shape.

In the next notebook, I will explore Scientific operations , Array Manipulations and Slicing

to be continued...