NUMPY

INTRODUCTION TO NUMPY

**A python List**can be used to store a group of elements together in a sequence. It can contain heterogeneous elements.

Following are some examples of List:

1. item\_list = ['Bread', 'Milk', 'Eggs', 'Butter', 'Cocoa']
2. student\_marks = [78, 47, 96, 55, 34]
3. hetero\_list = [ 1,2,3.0, ‘text’, True, 3+2j ]

To perform operations on the List elements, one needs to iterate through the List. For example, if five extra marks need to be awarded to all the entries in the student marks list. The following approach can be used to achieve the same:

1. student\_marks = [78, 47, 96, 55, 34]
2. for i in range(len(student\_marks)):
3. student\_marks[i]+=5
4. print(student\_marks)

It can be observed that, there is use of a loop. The code is lengthy and becomes computationally expensive with increase in the size of the List.

Data Science is a field that utilizes scientific methods and algorithms to generate insights from the data. These insights can then be made actionable and applied across a broad range of application domains. Data Science deals with large datatsets. Operating on such data with lists and loops is time consuming and computationally expensive.

Let us understand why Python Lists can become a bottleneck if they are used for large data.

Consider that 1 million numbers must be added from two different lists.

1. %%time
2. *#Used to calculate total operation time*
3. list1 = list(range(1,1000000))
4. list2 = list(range(2,1000001))
5. list3 = []
6. for i in range(len(list1)):
7. list3.append(list1[i]+list2[i])

C:\Users\CS-L2-PC02\AppData\Local\Microsoft\Windows\INetCache\Content.Word\x331627544433921.png

**Note:** Time taken will be different in different systems.

Let us understand, how Numpy can solve the same in minimal time.

**Note: Ignore the syntax and focus on only the output.**

1. %%time
2. *#Used to calculate total operation time*
3. *#Importing Numpy*
4. import numpy as np
5. *#Creating a numpy array of 1 million numbers*
6. a = np.arange(1,1000000)
7. b = np.arange(2,1000001)
8. c = a+b



It can be observed that the same operation has been completed in 12 milliseconds when compared to 395 milliseconds taken by Python List. As the data size and the complexity of operations increases, the difference between the performance of Numpy and Python Lists broadens.

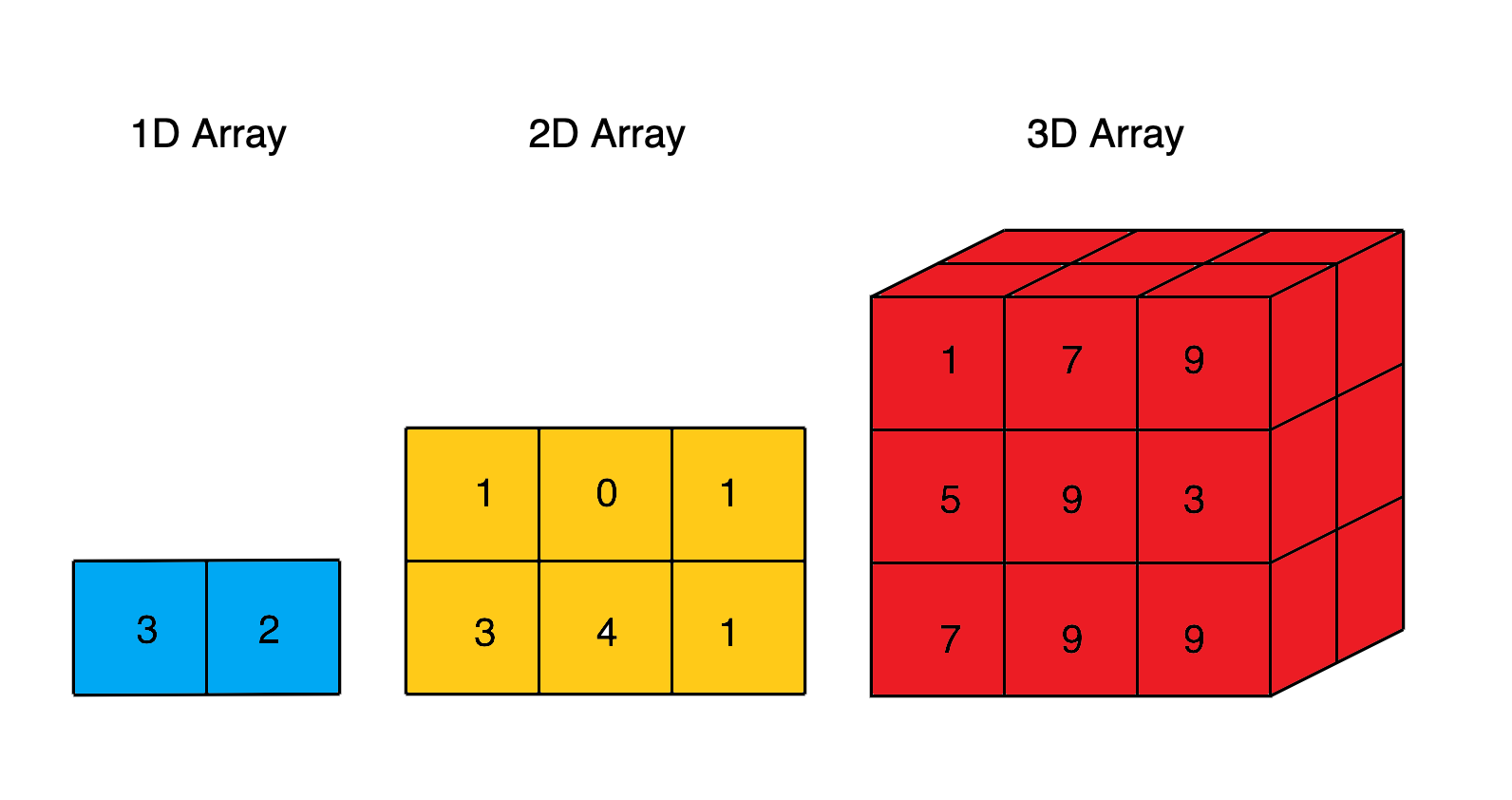
In Data Science, there are millions of records to be dealt with. The performance limitations faced by using Python List  can be managed by usage of advanced Python libraries like Numpy.

**Num**eric-**Py**thon (Numpy), is a Python library that is used for numeric and scientific operations. It serves as a building block for many libraries available in Python.

**Data structures in Numpy**

The main data structure of NumPy is the ndarray or n-dimensional array.

The ndarray is a multidimensional container of elements of the **same type** as depicted below. It can easily deal with **matrix and vector operations.**



Following are the benefits of using Numpy:

* As the array size increases, Numpy can execute more parallel operations, thereby making computation faster. When the array size gets close to 5,000,000, NumPy gets around 120 times faster than Python List.
* NumPy has many optimized built-in mathematical functions. These functions helps in performing variety of complex mathematical computations faster and with very minimal code.
* Another great feature of NumPy is that it has multidimensional array data structures that can represent vectors and matrices. This can be useful as lot of machine learning algorithms rely on matrix operations.

**Importing Numpy**

Numpy library needs to be imported in the environment before it can be used as shown below. 'np' is the standard alias used for Numpy.

1. import numpy as np

**Numpy object creation**

Numpy array can be created by using array() function. The array() function in Numpy returns an array object named ndarray.

**Syntax:  np.array(object, dtype)**

   object – A python object(for example, a list)

   dtype – data type of object (for example, integer)

Example: Consider the following marks scored by students:

|  |  |
| --- | --- |
| Student ID | Marks |
| 1 | 78 |
| 2 | 92 |
| 3 | 36 |
| 4 | 64 |
| 5 | 89 |

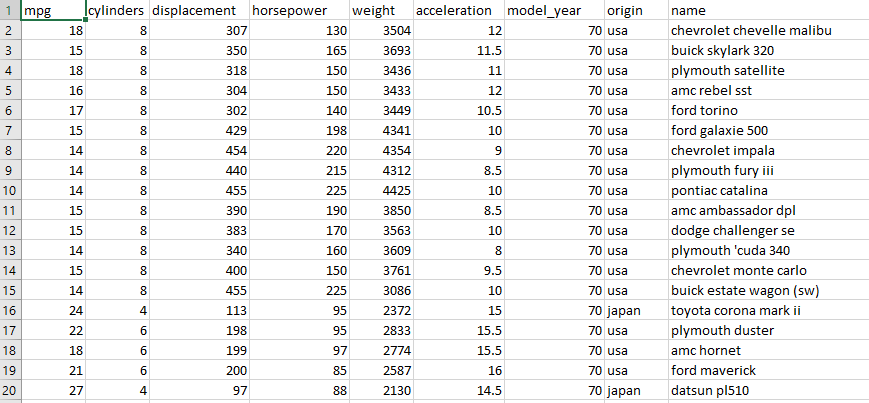
These marks can be represented in a one-dimensional Numpy array as shown below:

1. import numpy as np
2. student\_marks\_arr = np.array([78, 92, 36, 64, 89])
3. student\_marks\_arr

C:\Users\CS-L2-PC02\AppData\Local\Microsoft\Windows\INetCache\Content.Word\11627302163627.png

This is one way to create a simple one-dimensional array.

The following dataset has been provided by XYZ Custom Cars. This data comes in a csv file format.



There are various columns in this dataset. Each column contains multiple values. These values can be represented as lists of items. Since each column contains homogenous values, Numpy arrays can be used to represent them.

Let us understand , how to represent the car ‘horsepower’ values in a Numpy array.

1. *#creating a list of 5 horsepower values*
2. horsepower = [130, 165, 150, 150, 140]
3. *#creating a numpy array from horsepower list*
4. horsepower\_arr = np.array(horsepower)
5. horsepower\_arr

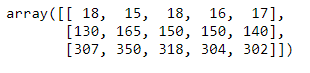


### How can multiple columns be represented together?

This can be achieved by creating the Numpy array from List of Lists.

Let us understand , how to represent the car 'mpg', ‘horsepower’, and 'acceleration' values in a Numpy array.

1. *#creating a list of lists of 5 mpg, horsepower and acceleration values*
2. car\_attributes = [[18, 15, 18, 16, 17],[130, 165, 150, 150, 140],[307, 350, 318, 304, 302]]
3. *#creating a numpy array from car\_attributes list*
4. car\_attributes\_arr = np.array(car\_attributes)
5. car\_attributes\_arr



The example demonstrates that the Numpy array created using the List of Lists results in a two-dimensional array.

The **numpy.ndarray.shape** returns a tuple that describes the shape of the array.

For example:

* a one-dimensional array having 10 elements will have a shape as (10,)
* a two-dimensional array having 10 elements distributed evenly in two rows will have a shape as (2,5)

Let us comprehend, how to find out the shape of car attributes array.

1. *#creating a list of lists of mpg, horsepower and acceleration values*
2. car\_attributes = [[18, 15, 18, 16, 17],[130, 165, 150, 150, 140],[307, 350, 318, 304, 302]]
3. *#creating a numpy array from attributes list*
4. car\_attributes\_arr = np.array(car\_attributes)
5. car\_attributes\_arr.shape



Here, 3 represents the number of rows and 5 represents the number of elements in each row.

**'dtype'** refers to the data type of the data contained by the array. Numpy supports multiple datatypes like integer, float, string, boolean etc.

Below is an example of using dtype property to identify the data type of elements in an array.

1. *#creating a list of lists of 5 mpg, horsepower and acceleration values*
2. car\_attributes = [[18, 15, 18, 16, 17],[130, 165, 150, 150, 140],[307, 350, 318, 304, 302]]
3. *#creating a numpy array from attributes list*
4. car\_attributes\_arr = np.array(car\_attributes)
5. car\_attributes\_arr.dtype

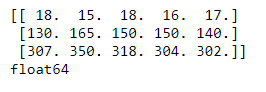


### Changing dtype

Numpy dtype can be changed as per requirements. For example, an array of integers can be converted to float.

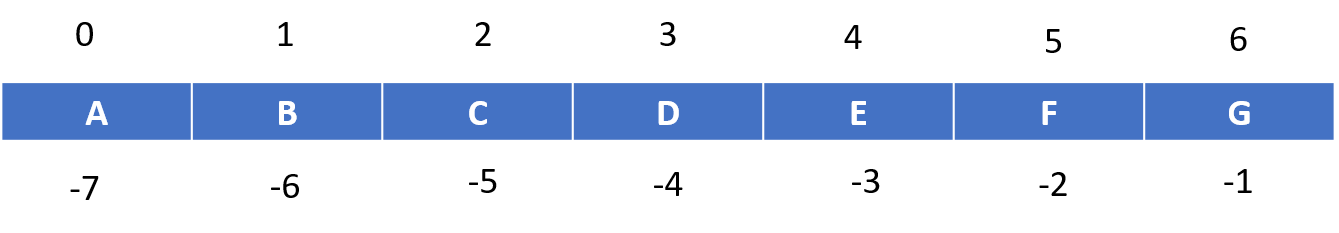
Below is an example of using dtype as an argument of np.array() function to convert the data type of elements from integer to float.

1. *#creating a list of lists of 5 mpg, horsepower and acceleration values*
2. car\_attributes = [[18, 15, 18, 16, 17],[130, 165, 150, 150, 140],[307, 350, 318, 304, 302]]
3. *#converting dtype*
4. car\_attributes\_arr = np.array(car\_attributes, dtype = 'float')
5. print(car\_attributes\_arr)
6. print(car\_attributes\_arr.dtype)



OPERATION ON NUMPY

The elements in the ndarray are accessed using index within the square brackets [ ]. In Numpy, both positive and negative indices can be used to access elements in the ndarray. Positive indices start from the beginning of the array, while negative indices start from the end of the array. Array indexing starts from 0 in positive indexing and from -1 in negative indexing.



Below are some examples of accessing data from numpy arrays:

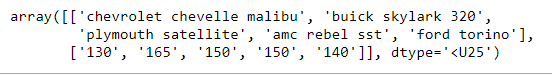
### 1. Accessing element from 1D array.

1. *#creating an array of cars*
2. cars = np.array(['chevrolet chevelle malibu', 'buick skylark 320', 'plymouth satellite', 'amc rebel sst', 'ford torino'])
3. *#accessing the second car from the array*
4. cars[1

C:\Users\CS-L2-PC02\AppData\Local\Microsoft\Windows\INetCache\Content.Word\91627368529308.png

### 2. Accessing elements from a 2D array

1. *#Creating a 2D array consisting car names and horsepower*
2. car\_names = ['chevrolet chevelle malibu', 'buick skylark 320', 'plymouth satellite', 'amc rebel sst', 'ford torino']
3. horsepower = [130, 165, 150, 150, 140]
4. car\_hp\_arr = np.array([car\_names, horsepower])
5. car\_hp\_arr



1. *#Creating a 2D array consisting car names and horsepower*
2. car\_names = ['chevrolet chevelle malibu', 'buick skylark 320', 'plymouth satellite', 'amc rebel sst', 'ford torino']
3. horsepower = [130, 165, 150, 150, 140]
4. car\_hp\_arr = np.array([car\_names, horsepower])
5. *#Accessing car names*
6. car\_hp\_arr[0]



1. *#Creating a 2D array consisting car names and horsepower*
2. car\_names = ['chevrolet chevelle malibu', 'buick skylark 320', 'plymouth satellite', 'amc rebel sst', 'ford torino']
3. horsepower = [130, 165, 150, 150, 140]
4. car\_hp\_arr = np.array([car\_names, horsepower])
5. *#Accessing horsepower*
6. car\_hp\_arr[1]



1. *#Creating a 2D array consisting car names and horsepower*
2. car\_names = ['chevrolet chevelle malibu', 'buick skylark 320', 'plymouth satellite', 'amc rebel sst', 'ford torino']
3. horsepower = [130, 165, 150, 150, 140]
4. car\_hp\_arr = np.array([car\_names, horsepower])
5. *#Accessing second car - 0 represents 1st row and 1 represents 2nd element of the row*
6. car\_hp\_arr[0,1]



1. *#Creating a 2D array consisting car names and horsepower*
2. car\_names = ['chevrolet chevelle malibu', 'buick skylark 320', 'plymouth satellite', 'amc rebel sst', 'ford torino']
3. horsepower = [130, 165, 150, 150, 140]
4. car\_hp\_arr = np.array([car\_names, horsepower])
5. *#Accessing name of last car using negative indexing*
6. ar\_hp\_arr[0,-1]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

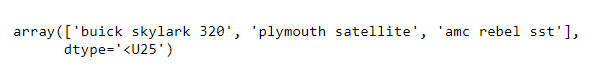


Slicing is a way to access and obtain subsets of ndarray in Numpy.

**Syntax:   array\_name[start : end]** – index starts at ‘start’ and ends at ‘end - 1’.

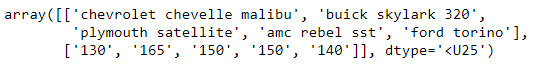
### 1.Slicing from 1D array

1. *#creating an array of cars*
2. cars = np.array(['chevrolet chevelle malibu', 'buick skylark 320', 'plymouth satellite', 'amc rebel sst', 'ford torino'])
3. *#accessing a subset of cars from the array*
4. cars[1:4]



### 2. Slicing from a 2D array

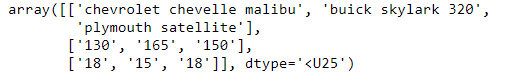
1. *#Creating a 2D array consisting car names, horsepower and acceleration*
2. car\_names = ['chevrolet chevelle malibu', 'buick skylark 320', 'plymouth satellite', 'amc rebel sst', 'ford torino']
3. horsepower = [130, 165, 150, 150, 140]
4. acceleration = [18, 15, 18, 16, 17]
5. car\_hp\_acc\_arr = np.array([car\_names, horsepower, acceleration])
6. car\_hp\_acc\_arr
8. *#Creating a 2D array consisting car names, horsepower and acceleration*
9. car\_names = ['chevrolet chevelle malibu', 'buick skylark 320', 'plymouth satellite', 'amc rebel sst', 'ford torino']
10. horsepower = [130, 165, 150, 150, 140]
11. acceleration = [18, 15, 18, 16, 17]
12. car\_hp\_acc\_arr = np.array([car\_names, horsepower, acceleration])
13. *#Accessing name and horsepower*
14. car\_hp\_acc\_arr[0:2]



1. *#Creating a 2D array consisting car names, horsepower and acceleration*
2. car\_names = ['chevrolet chevelle malibu', 'buick skylark 320', 'plymouth satellite', 'amc rebel sst', 'ford torino']
3. horsepower = [130, 165, 150, 150, 140]
4. acceleration = [18, 15, 18, 16, 17]
5. car\_hp\_acc\_arr = np.array([car\_names, horsepower, acceleration])
6. *#Accessing name and horsepower of last two cars*
7. car\_hp\_acc\_arr[0:2, 3:5]



1. *#Creating a 2D array consisting car names, horsepower and acceleration*
2. car\_names = ['chevrolet chevelle malibu', 'buick skylark 320', 'plymouth satellite', 'amc rebel sst', 'ford torino']
3. horsepower = [130, 165, 150, 150, 140]
4. acceleration = [18, 15, 18, 16, 17]
5. car\_hp\_acc\_arr = np.array([car\_names, horsepower, acceleration])
6. *#Accessing name, horsepower and acceleration of first three cars*
7. car\_hp\_acc\_arr[0:3, 0:3]



### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Problem Statement:

The engineers at XYZ Custom Cars want to know about the mean and median of horsepower.

### Solution:

The mean and median can be calculated with the help of following code:

1. *#creating a list of 5 horsepower values*
2. horsepower = [130, 165, 150, 150, 140]
3. *#creating a numpy array from horsepower list*
4. horsepower\_arr = np.array(horsepower)
5. *#mean horsepower*
6. print("Mean horsepower = ",np.mean(horsepower\_arr))



1. *#creating a list of 5 horsepower values*
2. horsepower = [130, 165, 150, 150, 140]
3. *#creating a numpy array from horsepower list*
4. horsepower\_arr = np.array(horsepower)
5. *#median horsepower*
6. print("Median horsepower = ",np.median(horsepower\_arr))



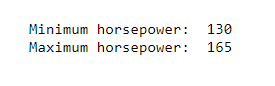
### Problem Statement:

The engineers at XYZ Custom Cars want to know about the minimum and maximum horsepower.

### Solution:

The min and max can be calculated with the help of following code:

1. *#creating a list of 5 horsepower values*
2. horsepower = [130, 165, 150, 150, 140]
3. *#creating a numpy array from horsepower list*
4. horsepower\_arr = np.array(horsepower)
5. print("Minimum horsepower: ", np.min(horsepower\_arr))
6. print("Maximum horsepower: ", np.max(horsepower\_arr))



### Finding the index of minimum and maximum values:

'argmin()' and 'argmax()' return the index of minimum and maximum values in an array respectively.

1. *#creating a list of 5 horsepower values*
2. horsepower = [130, 165, 150, 150, 140]
3. *#creating a numpy array from horsepower list*
4. horsepower\_arr = np.array(horsepower)
5. print("Index of Minimum horsepower: ", np.argmin(horsepower\_arr))
6. print("Index of Maximum horsepower: ", np.argmax(horsepower\_arr))



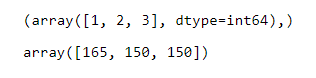
Problem Statement:

The engineers at XYZ Custom Cars want to know the horsepower of cars that are greater than or equal to 150.

Solution:

The 'where' function can be used for this requirement. Given a condition, 'where' function returns the indexes of the array where the condition satisfies. Using these indexes, the respective values from the array can be obtained.

1. *#creating a list of 5 horsepower values*
2. horsepower = [130, 165, 150, 150, 140]
3. *#creating a numpy array from horsepower list*
4. horsepower\_arr = np.array(horsepower)
5. x = np.where(horsepower\_arr >= 150)
6. print(x) *# gives the indices*
7. *# With the indices , we can find those values*
8. horsepower\_arr[x]



Problem Statement:

The Engineers at XYZ Custom Cars want to create a separate array consisting of filtered values of horsepower greater than 135.

### Solution:

Getting some elements out of an existing array based on certain conditions and creating a new array out of them is called filtering.

The following code can be used to accomplish this:

1. *#creating a list of 5 horsepower values*
2. horsepower = [130, 165, 150, 150, 140]
3. *#creating a numpy array from horsepower list*
4. horsepower\_arr = np.array(horsepower)
5. *#creating filter array*
6. filter\_arr = horsepower\_arr > 135
7. newarr = horsepower\_arr[filter\_arr]
8. print(filter\_arr)
9. print(newarr)



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Problem Statement:

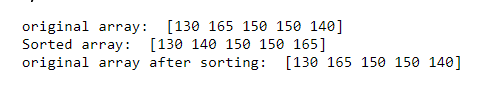
The engineers at XYZ Custom Cars want the horsepower in sorted order.

### Solution:

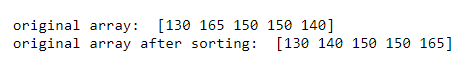
The  numpy array can be sorted by passing the array to the function *sort(array)*or by *array.sort.*

So, what is the difference between these two functions though they are used for the same functionality?

1. *#creating a list of 5 horsepower values*
2. horsepower = [130, 165, 150, 150, 140]
3. *#creating a numpy array from horsepower list*
4. horsepower\_arr = np.array(horsepower)
5. *#using sort(array)*
6. print('original array: ', horsepower\_arr)
7. print('Sorted array: ', np.sort(horsepower\_arr))
8. print('original array after sorting: ', horsepower\_arr)



1. *#creating a list of 5 horsepower values*
2. horsepower = [130, 165, 150, 150, 140]
3. *#creating a numpy array from horsepower list*
4. horsepower\_arr = np.array(horsepower)
5. *#using sort(array)*
6. print('original array: ', horsepower\_arr)
7. horsepower\_arr.sort()
8. print('original array after sorting: ', horsepower\_arr)



The difference is that the array.sort() function modifies the original array by default, whereas the sort(array) function does not.

The mathematical operations can be performed on Numpy arrays. Numpy makes use of optimized, pre-compiled code to perform mathematical operations on each array element. This eliminates the need of using loops, thereby enhancing the performance. This process is called vectorization.

Numpy provides various mathematical functions such as sum(), add(), sub(), log(), sin() etc. which uses vectorization.

Consider an example of marks scored by a student:

|  |  |
| --- | --- |
| Subject | Marks |
| English | 78 |
| Mathematics | 92 |
| Physics | 36 |
| Chemistry | 64 |
| Biology | 89 |

### Problem Statement:

Calculate the sum of all the marks.

### Solution:

The sum() function can be used which internally uses vectorizaton .

1. student\_marks\_arr = np.array([78, 92, 36, 64, 89])
2. print(np.sum(student\_marks\_arr))

### Problem Statement:

Award extra marks in subjects as follows:

English: +2

Mathematics: +2

Physics: +5

Chemistry: +10

Biology: +2

### Solution:

Below is the solution to the problem:

1. additional\_marks = [2, 2, 5, 10, 1]
2. student\_marks\_arr += additional\_marks
3. student\_marks\_arr



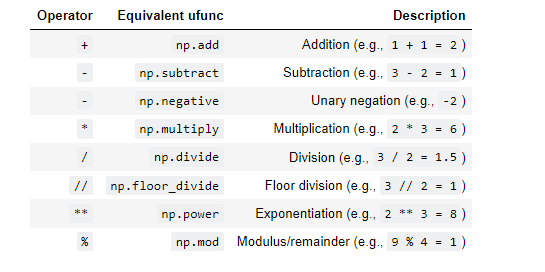
Also, the same operation can be performed as shown below:

1. student\_marks\_arr = np.array([78, 92, 36, 64, 89])
2. student\_marks\_arr = np.add(student\_marks\_arr, additional\_marks)
3. student\_marks\_arr



Both the above methods use vectorization internallly eliminating the need of loops.

Other arithmetic operations can also be performed in a similar manner.



In addition to arithmetic operations, several other mathematical operations like exponents, logarithms and trigonometric functions are also available in Numpy. This makes Numpy a very useful tool for scientific computing.

"Broadcasting" refers to the term on how Numpy handles arrays with different shapes during arithmetic operations. Array of smaller size is stretched or copied across the larger array.

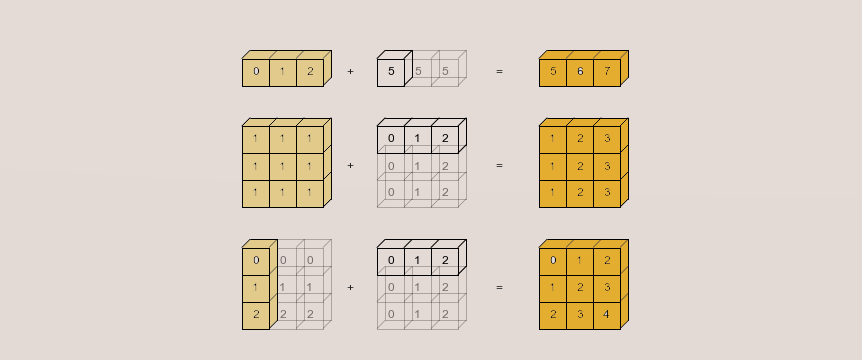
For example, considering the following arithmetic operations across 2 arrays:

1. import numpy as np
2. *# Array 1*
3. array1=np.array([5, 10, 15])
4. *# Array 2*
5. array2=np.array([5])
6. array3= array1 \* array2
7. array3



In this example, the array2 is being stretchedor copied to match array1 during the arithmetic operation resulting in new array array3 with the same shape as array1.

The following diagram explains broadcasting:



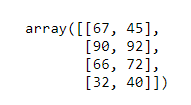
* In the first operation, the shape of first array is 1x3 and the shape of second array is 1x1. Hence, according to broadcasting rules, the second array gets stretched to match the shape of first array and the shape of the resulting array is 1x3.
* In the second operation, the shape of first array is 3x3 and the shape of second array is 1x3. Hence, according to broadcasting rules, the second array gets stretched to match the shape of first array and the shape of the resulting array is 3x3.
* In the third operation, the shape of first array is 3x1 and the shape of second array is 1x3. Hence, according to broadcasting rules, both first and second arrays get stretched and the shape of the resulting array is 3x3.

### Consider the following table consisting marks scored by four student in two different subjects:

|  |  |  |
| --- | --- | --- |
| Students | Chemistry | Physics |
| Subodh | 67 | 45 |
| Ram | 90 | 92 |
| Abdul | 66 | 72 |
| John | 32 | 40 |

The teacher of these students wants to represent their marks in an array. To do so, the marks can be stored using a 4x2 array as follows:

1. *#Students marks in 4 subjects*
2. students\_marks = np.array([[67, 45],[90, 92],[66, 72],[32, 40]])
3. students\_marks

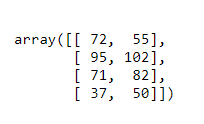


Problem Statement:

Now the teacher wants to award extra five marks in Chemistry and extra ten marks in Physics.

### Solution:

1. *#Students marks in 4 subjects*
2. students\_marks = np.array([[67, 45],[90, 92],[66, 72],[32, 40]])
3. *#Broadcasting*
4. students\_marks += [5,10]
5. students\_marks



The student's marks array is a 2D array of shape 4x2. The marks to be added are in the form of a 1D array of size 1x2. According to the broadcasting rules, the marks to be added get stretched to match the shape of student marks array and the shape of the resulting array is 4x2.

EXERCISE

### Problem Statement:

Lee decides to walk 10000 steps every day to combat the effect that lockdown has had on his body’s agility, mobility, flexibility and strength. Consider the following data from fitness tracker over a period of 10 days

|  |  |
| --- | --- |
| Day number | Steps walked |
| 1 | 6012 |
| 2 | 7079 |
| 3 | 6886 |
| 4 | 7230 |
| 5 | 4598 |
| 6 | 5564 |
| 7 | 6971 |
| 8 | 7763 |
| 9 | 8032 |
| 10 | 9569 |

1. Represent the above data in a **10x2** array. In each row, the first element should contain day number and second element should contain steps walked.
2. Lee notices that the tracker’s battery dies every day at 7 pm. Lee discovers that on an average, he walks 2000 steps every day after 7 pm. Perform an appropriate operation on your array to add 2000 steps to all the observations.
3. Write a program that returns the steps walked if the steps walked are more than 9000.
4. Print an array containing steps walked in sorted order.