Numpy

# Overview: -

- Why python: -
    - Easy to learn
    - Easy to read
    - Dynamic structure
    - Larger libraries
    - Suitable
    - Large community support
- Why numpy: -
    - Very large and effective library
    - We can find answer to any question
    - Easy to read
    - Coding is easy
    - We can **make data more flexible** and **customize very easily**
    - We can do **linear algebra operations** very easily
    - Very **popular in data science as well as machine learning**
- Data science in python uses algorithms to understand raw data. Python data science seeks to find patterns in data and use those patterns to predict future data.
- Data science using python includes preparing, analyzing, and processing data.
- Data Scientists use machine learning to discover hidden patterns in large amounts of raw data to shed light on real problems. The steps taken by the data scientists to find the hidden patterns are: -
    - Understand the problem
    - Get the data
    - Clean the data
    - Analyze data using ML models
    - Draw insights from the data
- Data scientists deal with large amounts of data, and **they store a lot of that data in relational databases**. SQL is a query language designed for relational databases.
- Python is the most popular programming language for data science. It is a universal language that has a lot of libraries available.

- In object-oriented programming, a developer completes a programming project by creating objects in code that represent objects in the actual world. These objects can contain both the data and functionality of the real-world object.

- Limitations of Python: -
    - **Slow** as compared to compiled and statically typed language like C.
    - **Consumes more memory** as compared to others because of its dynamic type system.
    - Python virtual engine that runs python code single-threaded, makes **concurrency another limitation**.
    - **Higher memory consumption** and CPU usage limits it from developing high quality 3D games.
- Usage of Python: -
    - Scripting
    - Automating tasks
    - Machine learning, data analytics, data science, data visualization.
    - Create desktop and mobile applications.
    - Web and app development using the frameworks such as Django and flask.
- Anaconda distribution is the distribution that includes python and many tools that are used with python, python interpreter etc.
- **IDE is integrated development environment**; it consolidates basic tools required to write and test software. Development tools often include text editors, code libraries, compilers and test platforms.

# Fundamentals of python: -

#Data types in python: -

- Variables are just to **reserve the memory location** to store the values. When we create a variable, **we reserve some space in memory**.
- Based on the data type of the variable the interpreter allocates memory and decides what can be stored inside the memory.
- While programming we may need to store some numbers, names or some data that we use so we can do this through creating variables.
- Naming of a variable: -
    - Starts with a letter or "_".
    - Cannot start with a number
    - Contain only alpha-numeric character and "_"
    - Variable names are case sensitive
    - Variable name cannot be a keyword defined in python.
- With the help of **type("variable name")** function we can find the type of the variable.
- **print() and input()** are the built-in function in python and we can use them to print the values on the screen and to input a value from the user respectively.
- We can also do the type conversion among the data types.

#Operators in python: -

- Operators are the constructs which can manipulate the value of operands. Python is a language supports few types of operators such as: -
    - Arithmatic operators → +, -, /, *, %, **
    - Comparison operators → ==, !=, >, <, >=, <=
    - Assignment operators → =, +=, -=, *=, /=
    - Logical operators → they are used to combine conditional statements. **and, or, not**.

# Conditionals in python: -

- Decision making is anticipation of conditions occurring while executing the program and specifying actions taken according to those conditions.
- Decision structures evaluates multiple expressions which produce true or false as their outcomes.
- Control statements are: -
    - **If statements** → it contains a logical expression using data to be compared and the decision is made based on the result of comparison. If the Boolean expression inside if statement is true then the block of code inside if statement is executed.
    - **Else statement** → an else statement can be combined with an if statement and else statement contains a block of code that executes if the conditional expression in the if statement resolves to 0 or false value.
    - **Elif statements** → it allows us to check multiple expressions for true and then execute a block of code as soon as one of the conditions evaluates to be true.

#Loops in python: -

- **Loops are the statements that make code run repeatedly**. There may be a situation when we want to execute a block of code many times, now at that point, a loop statement will allow us to execute the statement or group of statement multiple times.
- There are two type of loop statements: -
    - **For loops** → it allows us to repeat an action several times for each item in the sequence of items.
    - **While loops** → it allows us to repeat one or more actions while a condition remains true.
- Loop control statement change execution from its normal sequence. So, when execution leaves the scope, all the automatic object that were created in that scope are destroyed.
- Python support following **control statements**: -
    - **Break**
    - **Continue**
    - **Pass** → we use pass when a statement is required syntactically, but we don’t want any command or code to execute. It is a null operation so nothing happens when it executes. Generally, **we use it as a placeholder.**

- **Range(start, stop(not include), step)** is a predefined function in python, it creates a structure based on the values that we get it. The structure creates a sequence of numbers that is similar to lists starting, ending and incrementing(optionally). So, we use the range function to represent numbers within a certain range.

#List, tuples, sets and dictionaries in python: -

- The **most basic data structure in python is sequences**, each element of a sequence is assigned a number, its position or index and indexes in python starts from 0.
- Python has six built-in types of sequences: -
  - Strings
  - List
  - Set
  - Tuple
  - Dictionaries
- Lists: -
  - Most versatile data type
  - Comma-separated, square brackets
  - Items don't need to be of same type in list
  - [,] this is how we create a list
  - It typically stores homogenous data
  - **Elements of lists are mutable**
- Tuples: -
  - Similar to list
  - **Immutable**
  - Used where we want immutable data in our programs.
  - (,) this is how we create a tuple
  - Comma-separated
- Dictionaries: -
  - Each element is kept as a **key-value**
  - It is different from all the other data types
  - {key:value} this is how we create dictionaries.
  - **Keys should be unique** while the values should not be
  - When duplicate keys encountered during the assignment, the last assignment went.
  - Values can be of any type but the **keys must be of an immutable data type**.

- Sets: -
    - Comma-separated
    - { , } this is how we create sets
    - When the set is created then the duplicate element is automatically eliminated.

# Data type operators and methods: -

- Indexing means we are referring to an element of an iterable by its position within the iterable. For indexing we use **variable_name[ ] square brackets**. Indexing in python starts from 0.
- We can also find **reverse indexing** in python using -ve numbers like **variable_name[-2]**.
- We can also find a substring like **variable_name[start:stop:step]** this is called as **slicing**.
- With the help of **len(variable_name)** we can find the length of the data type.
- We can add the elements in the variable using the functions like **.append(), .add()** etc.
- We **can't add the elements in tuple** because they are immutable.
- We can also remove an element from our variables like using the method **.pop()** and it will remove the last element.
- With the help of **variable_name.count(value)** method we can count how many times a value occurs in the variable.
- With the help of **variable_name1.update(variable_name2)** method, we can combine two different dictionaries or sets.

#Modules in python: -

- **A module is a file consisting of python code**. A module can define **functions, classes and variables**. Modules can also include runnable code.
- By integrating a python module in our program, we can use the functions and classes written inside the modules and write our programs more effectively.
- Without the concept of module, we would have to write each function and class in our programs ourselves.
- The **import** statement is going to allow us to **use any python source file as a module** in another source file.

- When an interpreter encounters an **import statement**, it imports the module if the module is present in the search path and the search path is the list of directories that the interpreter searches before importing a module.
- Module is imported only once regardless the number of times that it is imported.
- Python has numbers of readymade libraries written by python developers.
- One of the modules in python is math module and we can import it like **import math**. And now we can use all the math functions in our code.
- With the **dir(module_name)** method we can see the inside of a module.
- **help(module_name)** tells us what a particular function will do in a module.
- Or we can import the module along with the function inside it like **from module_name import function_name**.

#Functions in python: -

- Functions are structures that have **certain functions in programming that we use repeatedly**.
- The task of the function is to bring together the complex operations and it'll enable us to perform these operations in one step like the **print()** function we've used since beginning.
- Function only runs when it's called only.
- Python has many built-in functions in this library. But we can also create our own functions.
- To define a function, we use the **def keyword** followed by a function name and then the optional parameters/arguments.
- Doc string is written inside the function in "”” "”” and **it specifies what the function does**.
- The **return statement** in the function means that the function **returns a value to the place** where it was called after the operation. Or we need to use it because we can store the value we receive in a function in a variable and use the variable elsewhere in the program we follow.
- The **lambda expression is also called as an anonymous function** and we can define it with **lambda expressions**. With the lambda expression we can define the function in one line, it is used when we want to write a short function with no complex code in it.
- To write the lambda expression, we use the **lambda keyword**.

# OOP: -

#Logic of OOP: -

- Once written, the code can be used in very efficient way using functions.
- Objects in python are elements that **contain some methods and some values**. E.g. list, sets, tuples, dictionaries are indeed objects.
- Classes are the type of data that allow us to produce objects. **We can think of class as a prototype of objects.** It contains data members, methods, instances etc.
- We define the class using the **keyword class**.

#Constructor in OOP: -

- **__init__()** is also called as a constructor function. **Using this we can initialize the object using different values**. This method is the first function that's automatically called when creating our objects.
- The **keyword self** is a reference to the object when we created it and it is a parameter that should be present in our methods in the very first place so we can use all the properties and methods of an object through this reference.
- We can give the default values in the arguments also in order to not get errors.

#Methods in OOP: -

- Just as we define the **__init__()** method, we can define as many methods as we want in a class.

#Inheritance in OOP: -

- Inheritance is when a class inherits properties and methods from another class.
- While defining some hierarchy, some properties are common in every class, instead of defining these common properties and methods in these classes over and over again, **we can define one main class and get them to give the methods and attributes of this class to the derived class**.
- When using inheritance, we can only use the features in the parent class and cannot add another method.

#Overriding and Overloading in OOP: -

- If we decide to redefine our methods that we inherit in our class with the same name then the method we created will work not the inherited method this is **called override methods**.
- We can always override the parent class methods. Also, we can define methods and function to our specific subclass at any time.
- **Overloaded methods** are the methods that are **not specifically called by us, but they do belong to every class**. Most of them define by python by default, even if we don’t define them.
- Some of the overloaded methods are **__init__(), __str__(), __len__(), __del__()**.

# Numpy Library: -

# Introduction to numpy library: -

- It is one of the **most important and useful python libraries**. Python libraries offer modules to use while coding. We can achieve the goal we're trying to accomplish through these modules.
- Libraries in general are **collection of information resources**.
- **Numpy** is a library that allows use to **perform large-scale mathematical-scientific calculations** efficiently, quickly and flexibly. Numpy is a **fundamental package for scientific computing** in python.
- Numpy is short form for numerical python.
- Numpy is a python library that **provides a multidimensional array object**, **various derived objects**.
- **An array is a grid of values** and it contains **information about the raw data**, **how to locate an element, and how to interpret an element.** We can have 1D, 2D, 3D arrays and multi-dimensional arrays.
- The arrays in numpy works similar to the matrix in mathematics.
- Advantages of numpy arrays over python lists: -
    - Consumes less memory
    - Fast as compared to lists
    - Convenient to use
- **Lists** looks like: - [1,2,3,4] and the **Arrays** looks like: - [1 2 3 4].
- We can **install numpy** using the command **pip install numpy** in the command prompt.
- We can **import the numpy** library/package in our notebook using the command **import numpy** we can also give it an alias name as **import numpy as np**.
- Importance of numpy in python: -
    - Wide variety of **mathematical operation**s on array.
    - Provides enormous library for **high-level mathematical functions** that operate on the arrays and matrices.
    - Mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulations and much more.

#Python lists V/S numpy arrays: -

- Difference between lists and arrays: -
    - **Data types storage →** lists can store different type of data directly but in arrays we can work on only one of data at a time.
    - **Importing modules →** we do not need to import any packages or libraries to use lists. We can use them directly. But to use arrays we have to install it in our system and then type import it.
    - **Numerical operations →** arrays do the numerical operation fast and more efficient than lists. We can do all the matrix operations on numpy arrays. Arrays can be converted into matrix while the lists cannot be.
    - **Modification capabilities →** lists has many functions to do modifications as compared to arrays.
    - **Consumes less memory →** Arrays consumes very less memory as compared to lists.
    - Fast as compared to lists.
    - **Convenient to use →** arrays are more convenient to use because they are faster as compared to lists.
- **%timeit** function is used to calculate the time take to execute a block a code.
- **.arange(start, end) →** this function is used to create array within a given range. It is like the .range() function.

#Numpy array creation: -

- To create numpy array, we use the function **np.array()** like: -
    - **import numpy as np**
      a = np.array([1,2,3])
      print(a)
- We pass a list inside the. array() function and it will create a numpy array of that corresponding list.
- Dimensions in array: -
    - **1-D arrays →** 1D array looks like [1 2 3 4]
    - **2-D arrays →** 2D array looks like [[1 2 3 4]]
    - **3-D arrays →** 3D array looks like [[[1 2 3 4]]]
    - **Higher dimension array**
- To find the dimension of the array count the number of square brackets from left to right and the number of brackets will be the dimension of the array.
- To find the dimension of the array we use the **array_name.ndim** function to find the array is of which dimension.

- To create 2 or more-dimension array, then number of elements in the list should be same.
- **ndmin** is used to create multi dimension array without actually defining the brackets. We just have to pass the number, of how many dimensions array we want and it will create that dimension array. **var1 = np.array([1,2,3,4], ndmin=4)** this is how we create multi dimension array using the **ndmin** function.

#Special types of arrays: -

- We've seen that how to create numpy arrays using the numpy function. Now we will look into some special types of numpy arrays and how to create them using the numpy function.
- Special types of arrays are: -
    - **Array fill with 0's** → with the help of **np.zeros()** function we can create the array filled with 0. We can also pass the dimension inside that function also like **np.zeros((3,4))** and it will create an array with 3 rows and 4 columns.
    - **Array filled with 1's** → with the help of **np.ones()** function we can create the array filled with 1. We can also pass the dimension inside that function also like **np.ones((3,4))** and it will create an array with 3 rows and 4 columns.
    - **Create an empty array** → with the help of **np.empty()** function we can create the empty array. **But it will hold the data of the previous memory** and display that data to us.
    - **An array with a range of elements** → with the help of **np.arange(start,stop)** function we can create the array with the give range. It is like a range function in the list. It will give the value of the given range excluding the stop point.
    - **Array diagonal elements filled with 1's** → with the help of **np.eye()** function we can create the array with all the diagonal elements as 1 and the rest will be 0. Just like **identity matrix in mathematics**. **np.eye(3)** means it will create an array of 3*3 with all the diagonal elements as 1.
    - **Array with values that are spaced linearly in a specified interval** → with the help of **np.linspace()** function we can create the array with particular specified interval. **np.linspace(1,10,num=5)** means it will create an array from any number between 1 to 10 and give us 5 numbers only with equal interval.

#Creating random valued arrays: -

- Here we will see how to create numpy arrays using the random numbers and what are the functions available to this. Different function for creating random valued arrays is: -
    - **random.rand(n)** → the function is used to generate a random value between 0 to 1.
    - **random.randn(n)** → the function is used to generate the random value close to 0. This may return +ve and -ve numbers as well.
    - **random.ranf(n)** → the function for doing **random sampling in numpy**. It returns an array of specified shape and fills it with random floats in the half open interval [0.0, 1.0) including 0 but excluding 1.
    - **random.randint(min, max, total_numbers)** → the function is used to generate a random number between a given range.

#Data type in numpy arrays: -

- Suppose we have to store some data in a variable and that data which we want to store should have some data type. Like whether we are storing list, tuple, dictionary, string etc.
- Similarly, we store data in the numpy array and that data also have a data type.
- Some of the data types in numpy arrays are: -
    - bool_ → Boolean (true of false) stored as byte
    - int_ → default integer type
    - Intc → identical to c integer
    - Intp → integer used for indexing
    - int8 → byte
    - int16
    - int32
    - int64
    - unit8 → unsigned integer (0 to 255)
    - unit16
    - unit32
    - unit64
    - float_ → shorthand for float64
    - float16 → half precision float
    - float32 → single precision float
    - float64 → double precision float

    - complex_ → shorthand for complex128
    - complex64
    - complex128
- **array_name. dtype** function in numpy is used to find the type of data used in the numpy array.
- We can also convert the type of data inside numpy array from one data type to another. List of characters that are used to represent **dtype** in numpy array are: -
    - **i** → integer
    - **b** → Boolean
    - **u** → unsigned integer
    - **f** → float
    - **c** → complex float
    - **m** → timedelta
    - **M** → datetime
    - **O** → object
    - **S** → string
    - **U** → Unicode string
    - **V** → the fixed chunk of memory for other types (void)
- **x = np.array([1,2,3,4], dtype = np.int8)** → this is how we convert one data type into other in numpy arrays **using the dtype function**.
- **x1 = np.array([1,2,3,4], dtype = "f")** → this is also a way if we want to convert our data from one data type to another by **passing the special characters as string in dtype function**.
- **new = np.float32(x2)** → this is how we create a function also to convert one data type to another of numpy array elements.
- **new1 = x3.astype(float)** → With the help of **.astype()** function we can change the type of data from one data type to another also.

#Shape and reshape in numpy: -

- We will study how to shape the numpy array or what is the shape of the numpy array and what are the functions used to know the shape of the numpy array.
- Also, we will study how to convert the 1D array in multi dimension array.
- **array_name.shape** → this function will give the shape of the array that what is your dimension of the array.
- We can also **Reshape the array** means we can convert 1D array to multi-dimension array and from multi-dimension array to 1D array.

- **array_name.reshape(rows,column) →** this function is used to reshape the array from one dimension to another.
- **array_name.reshape(2,3,2) →** this is how we reshape an array to 3D array by giving 3 arguments.
- If the number of elements in the rows are out of the range, then we get an error that the data is insufficient here and we cannot resize the array. In numpy we don't have a blank row ever.
- **array_name.reshape(-1) →**this will convert any dimension array to 1D array.

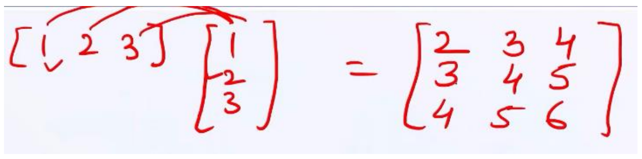
#Numpy-Arithmatic operations: -

- Here we will study, how to apply Arithmatic operations on arrays and how to change arrays while applying Arithmatic operations.
- Arithmatic operations in numpy array are: -
  - **np.add(a,b) →** a+b
  - **np.subtract(a,b) →** a-b
  - **np.multiply(a,b) →** a * b
  - **np.divide(a,b) →** a/b
  - **np.mod(a,b) →** a%b
  - **np.power(a,b) →** a**b
  - **np.reciprocal(a) →** 1/a
- We can use the Arithmatic operations directly or we can use the functions related to them.
- Some more basic Arithmatic functions which we can apply on numpy array are: -
  - np.min(x)
  - np.max(x)
  - np.argmin(x) → gives the position of minimum element
  - np.argmax(x) → gives the position of maximum element
  - np.sqrt(x)
  - np.sin(x)
  - np.cos(x)
  - np.cumsum(x) → it will add each number with the previous number and then give the array. It is used to find median in statistics.

- we can also apply these arithmatic operations on 2D array. Suppose we have to find the maximum element in the 2D array then using the same **np.max(array, axis)** function we can find the maximum element. But here we have to pass 2 arguments in the function i.e. 1st is the array and 2nd is the axis.
- There are 2 axes i.e. axis = 0, axis = 1. **axis = 0 works along rows** and **axis = 1 works along columns**.

#Broadcasting with numpy arrays: -

- Broadcast method says that while performing Arithmatic operations between two arrays we have to keep some important things in mind like: -
  - Both the arrays are of same dimension
  - If the dimensions are different then the operations will perform correctly if one of the dimensions in both the array will contain atleast one 1.

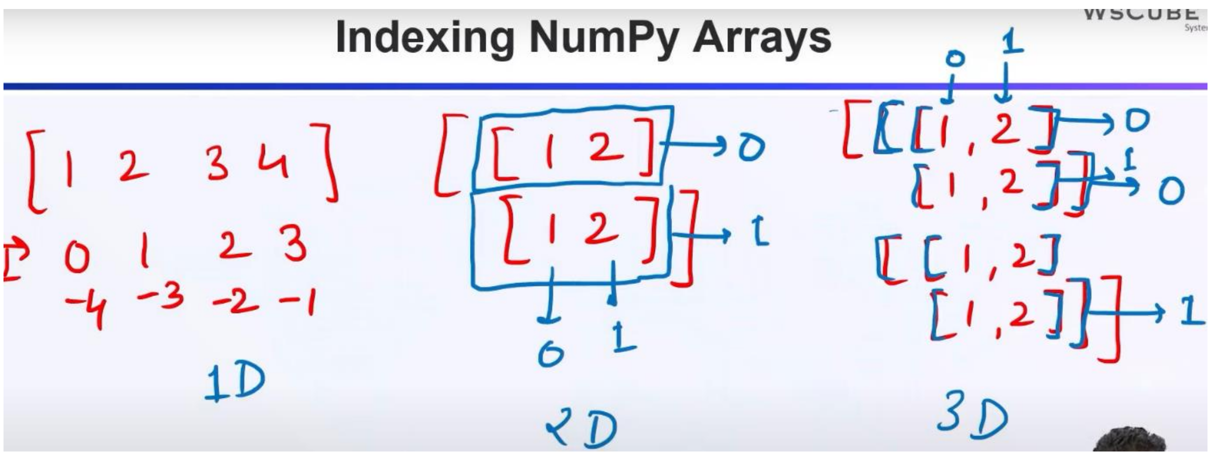


$$[1\ 2\ 3]\ \begin{bmatrix}1\\2\\3\end{bmatrix} = \begin{bmatrix}2 & 3 & 4\\3 & 4 & 5\\4 & 5 & 6\end{bmatrix}$$

  - It will perform the addition of different dimension array like shown in above image and gave the result array in the maximum dimension of both the array like 3 * 3 array in above case.

#Indexing and slicing with numpy arrays: -

- Here we will study how to do indexing and slicing in numpy arrays.
- Indexing in the **numpy arrays starts from 0**. We can also give -ve indexing in the numpy arrays and that will help us to access the elements from reverse.
- For 1D array the indexing will be like: -
  - [1 2 3 4] → 0 1 2 3
- For 2D array the indexing will be like: -
  - [[1 2] [1 2]] → 00 01 10 11



- To get the element on particular index we use the syntax **array_name[index].**
- **Slicing** is to take a particular range of data set from a numpy array.
- To do slicing in arrays we have the syntax like: -
  - **array_name[start:stop(not included):step)**
- **var1[1,1:]** this is the syntax used to do the slicing in 2D array.

# Numpy array iterating: -

- Here we will study how to do iteration in numpy arrays and what are the functions available for it.
- We can iterate numpy arrays through **for-loop**. Iterating numpy arrays means going through each element of the array and do the operations on them.
- For 1D array we apply 1 for-loop, for 2D array we use 2-for loop and for 3D array we use 3-for loop.

- If we want to iterate the array in one time without applying the **for-loop again and again** then we use the function **np.nditer(array_name)** in place of the for-loop.
- Suppose we are doing iteration of very large data set, then the indexing of the data in not known to us, then **to get the data with indexing** we have a function known as **np.ndenumerate(array_name)**.

# The difference between copy and view in numpy arrays: -

- Both the functions **copy()** and **view()** are used to copy the data, but the difference between them is: -
    - **Copy** owns the data but the **View** does not own the data.
    - **Copy** of an array is the new array and the **View** is of the original array,
    - The changes made in the **Copy** of the data does not reflect in the original array and any changes made to the **View** will affect the original array, and any changes made to the original array will affect the **View**.

# Join() and split() functions in numpy arrays: -

- We will study how to join two or more arrays and create a new array and how to split an array.
- **Joining** means putting contents of two or more arrays in a single array. One thing to keep in mind while joining the array is that the number of elements in the arrays should be same.
- **We can also merge the arrays along their axis**.
- **np.concatenate((array1, array2))** function is used to join/merge two arrays to create a new single array.
- To do concatenation in 2D arrays we can merge the arrays **along their axis**.
- **ar_new = np.concatenate((vr,vr1), axis=1)** → this is how we concatenate 2D arrays together by passing the axis along which we want to do the merging, **axis=1 (column) axis=0 (row)**.
- **np.stack((array1, array2),axis=)** is the function which is also used to merge the arrays but it works along the axis to perform merging of the array so we have to pass axis as the argument in the function.
- **np.hstack((array1, array2))** → it is used to merge the array in horizontal direction and works along the row.

- **np.vstack((array1, array2)) →** it is used to merge the array in vertical direction and works along column.
- **np.dstack((array1, array2)) →** it is used to merge the array and works along height.
- **Splitting** breaks one array into multiple arrays.
- **np.array_split(array, no. of arrays after split) →** this is the function used to split the array in the given numbers of the array and **it will give us the list of the arrays**. We can also access individual array from the list by passing the index of an array.
- **np.array_split(spl1,3, axis=1) →** this is how we split the array along the axis also.

# Search, sort, search sorted, filter functions: -

- **Search →** an array for a certain value, and return the indexes that get a match. We can search the array from 1D, 2D or 3D array.
- For searching the element from the array we use the **where keyword**.
- **np.where(var==2) →** this is how we search the element in the array and it gives us the positions of the element where it is present in the array.
- We can apply different operations while searching for the element in the array.
- **Search sorted array →** performs a **binary search in the array**, and returns the index where the specified value would be inserted to maintain the search order.
- **np.searchsorted(array_name,element to insert) →** this is the function used to perform search sorted in an array and it gives us the position where we can **insert the element in the sorted array.**
- The above function will search the array in left to right direction to get the available position of the element to be inserted, but we can also make it **search the array in right to left direction** by passing a **parameter side** in the **.searchsorted() function**.
- **Sort array →** ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.
- **np.sort(array_name) →** is the function used to sort the array in ascending order.
- We can also sort characters and strings. We can also sort a 2D array also.

- **Filter array →** getting some elements out of an existing array and creating a new array out of them.
  - fil_list = [True, False, True, False]
    new_fil = fil[fil_list]
- This will filter the list and give us the elements that has corresponding true value in it.

# Numpy array functions: -

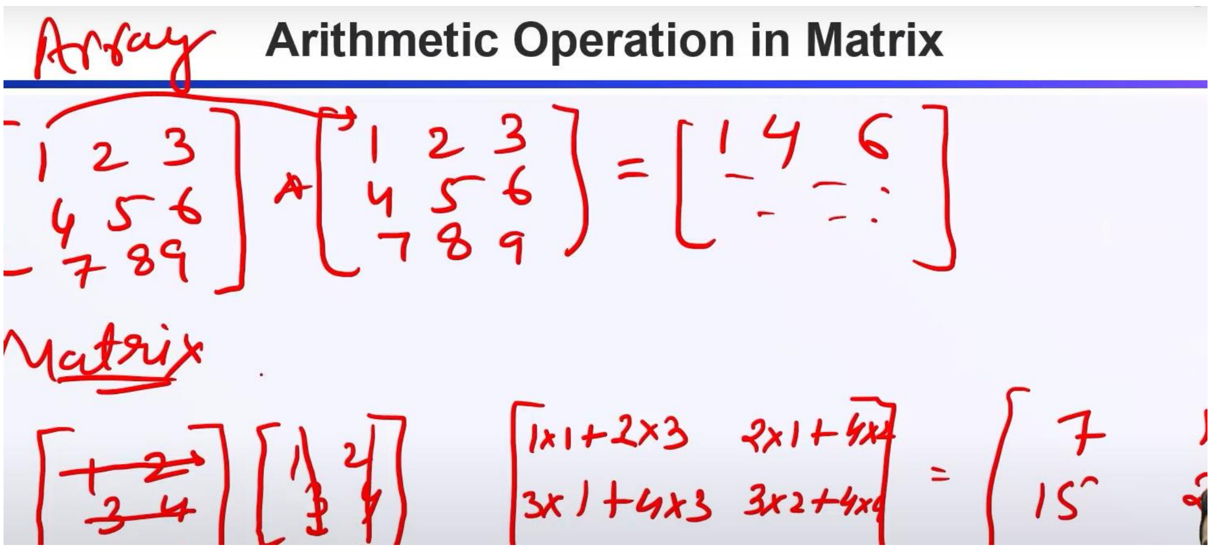
- Some of the numpy Arithmatic functions are **shuffle, unique, resize, flatten, ravel.**
- **Shuffle** is used to shuffle the data in the numpy array. **np.random.shuffle(array_name)** is the function used to shuffle an numpy array.
- **Unique** function is used to give the unique data from a numpy array if there are duplicate values present in it. It will also give us the information about the index number and counting of an element. **np.unique(var1, return_index="true", return_counts="true")** is the function which gives the unique data from an array it also give us the index of the elements and their count.
- **Resize** function is used to resize a numpy array. **np.resize(var2, (2,3))** is the function used to resize an array and we have to pass the dimensions of the new array in it.
- **Flatten and Ravel** both the functions are used to convert the 2D array into 1D array.
- **Array_name.flatten(order=" ")** this is how we convert an 2D array into 1D array. We can also pass the order in it to represent the data in 1D differently.
- **np.ravel(array_name, order=" ")** this is how we use Ravel function to convert the 2D array into 1D array. we can also pass order as an argument in it. The order can be **c, f, k, a**.

# Insert and Delete functions: -

- Here we will study how to insert and delete data from a numpy array.
- We can insert the data in the numpy array with the help of **np.insert(array_name, position, element)** function. We can also insert multiple data in it by giving multiple positions as an argument.
- Whenever we insert the data then the **insert() function** will only insert integer data and it will not insert the float data and if we give float data for insertion then it will convert that data into an integer data.
- **np.insert(var1, 2,[22,23], axis=1)** this is how we insert the data in 2D array.
- **np.append(array_name, value)** we can also use the append() function to insert the value in the numpy 1d or 2d array at the end of the array.
- We can also delete the data from the numpy array using **np.delete(array_name, index)** function.

Numpy Matrix: -

- Here we will study what is matrix, how to use the matrix and how do we create matrix in the numpy array.
- **Matrix is used to perform operations** on large amount of data. We use matrix when we are working on large amount of data.
- Both the numpy arrays and matrix are almost similar, elements in both of them are present in row or column wise.
- The difference in the numpy arrays and matrix is in **multiplications only**.
- When we multiply the data of a numpy array to another numpy array then the data gets multiplied one by one, while multiplication in the matrix has some different rules. In matrix multiplication is done as a **dot product** (row * column).
- We can create matrix using numpy with the help of **np.matrix()** function.



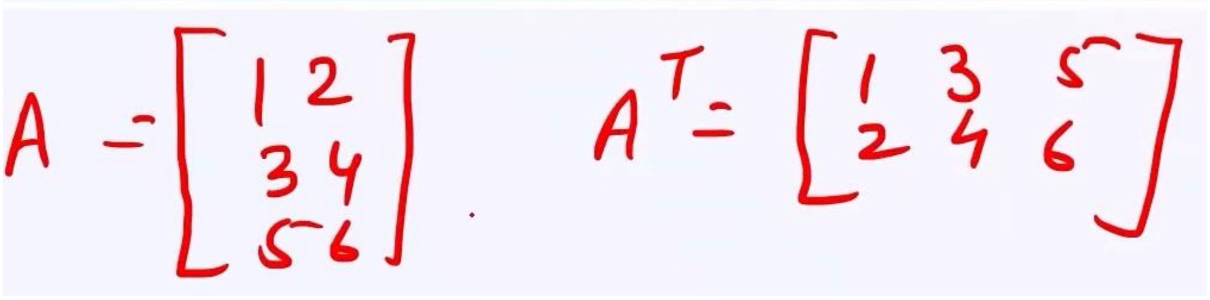
$$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} * \begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} = \begin{bmatrix}1&4&6\\-&-&\cdot\\-&-&\cdot\end{bmatrix}$$

$$\begin{bmatrix}1&2\\3&4\end{bmatrix}\begin{bmatrix}1&2\\3&4\end{bmatrix} \quad \begin{bmatrix}1\times1+2\times3 & 2\times1+4\times4\\3\times1+4\times3 & 3\times2+4\times4\end{bmatrix} = \begin{bmatrix}7\\15\end{bmatrix}$$

- **array1.dot(array2)** is the function used to perform dot product in the numpy matrix.
- To do matrix multiplication, the elements of the matrix should be in broadcast to perform the multiplication otherwise it will give the dimension error.

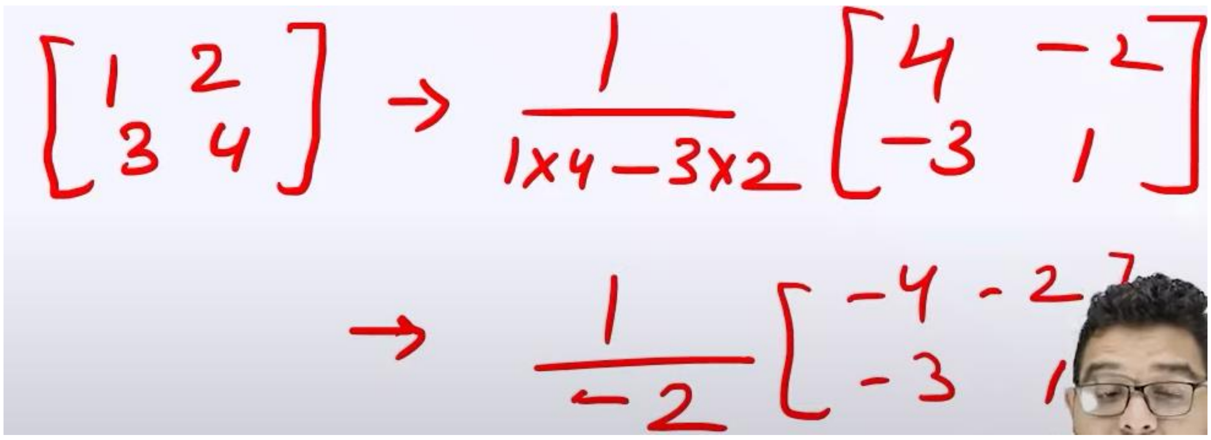
# Matrix Functions: -

- Some of the basic functions that we can apply on matrix are **transpose, swapaxes, inverse, power, determinate**.
- **Transpose** of matrix is used to inverse the axis of the matrix. **np.transpose(matrix_name)** this is how we use the transpose function. **matrix_name.T** this is also used to perform transpose of matrix function.



$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

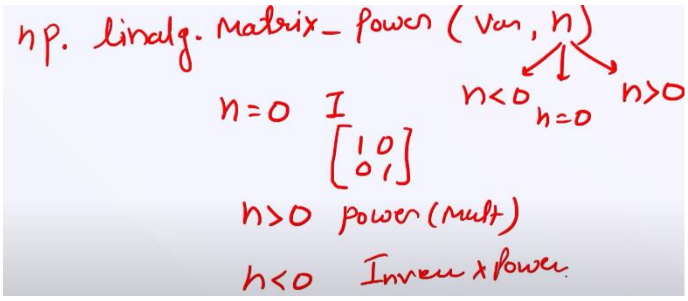
- **Swapaxes** is also used to convert the axis into one another (row→column, column → row). **np.swapaxes(matrix_name, axis1, axis2)** this is how we use the **swapaxes()** function to convert the axis.
- **Inverse** matrix looks like this. **np.linalg.inv(matrix_name)** this is the function used to create the inverse matrix.



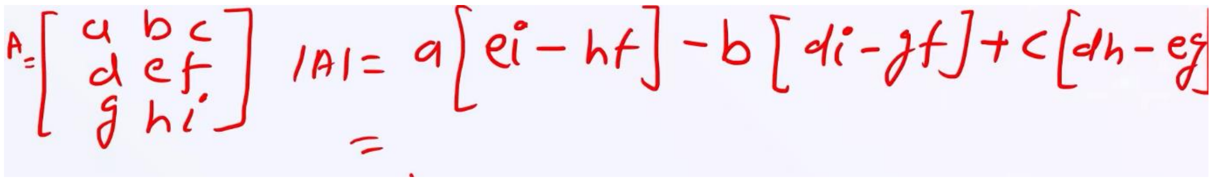
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow \frac{1}{1 \times 4 - 3 \times 2} \begin{bmatrix} 4 & -2 \\ -3 & 1 \end{bmatrix}$$

$$\rightarrow \frac{1}{-2} \begin{bmatrix} -4 & -2 \\ -3 & 1 \end{bmatrix}$$

- **Power** is used to calculate different power of matrix. **np.linalg.matrix_power(matrix_name,n)** this is how we can find the power of matrix.



- **Determinate** of a matrix looks like this. **np.linalg.det(var5)** this is how we can find the determinate of a matrix.



$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \quad |A| = a[ei - hf] - b[di - gf] + c[dh - eg$$

$$=$$

# Numpy Ufuncs: -

# Intro: -

- Ufuncs stands for **universal functions** and these are numpy functions that operate on **ndarray** object.
- Ufuncs are used to implement vectorization in numpy which is faster than iterating over elements.
- Ufuncs also takes additional arguments like: -
    - where → Boolean array or condition defining where the operation takes place
    - dtype → defining the return type of the element
    - out → output array where the return value should be copied.
- Converting iterative statement in a **vector-based operation** is called **vectorization**.

# Create your own Ufunc: -

- To create our own Ufunc, we have to define a function, like we define normal function in python and then add that function to the numpy ufunc library with the **frompyfunc() method**.
- The **frompyfunc()** method takes the following arguments: -
    - function → name of the function
    - inputs → the number of input arguments
    - outputs → the number of output arrays.
- To create a numpy ufunc: -

    -
      ```
      def myadd(x,y):
              return x+y
      myadd = np.frompyfunc(myadd, 2, 1)
      print(myadd([1,2,3,4],[5,6,7,8])
      ```
- To check if a function is a ufunc or not we use the **type(np.ufunc)** and it will give us the type as **<class 'numpy.ufunc'>.**

# Simple Arithmatic: -

- In this section we will discuss the functions that can take an array-like objects e.g. list, tuple etc. and perform **Arithmatic conditionally** means we can define conditions where the Arithmatic operation should happen.

- The functions are: -
  - **np.add()** → sums the content of two array, return the result in the new array.
  - **np.subtract()** → subtract the value from one array with the value from another array.
  - **np.multiply()** → multiplies the values from one array with the values from another array.
  - **np.divide()** → divides the values from one array with the values from another array.
  - **np.power()** → rises the values from the first array to the power of the values of the second array.
  - **np.mod()/np.remainder()** → return the remainder of the values in the first array corresponding to the values in the second array.
  - **np.divmod()** → return value is two arrays, the **first array contains** the **quotient** and **second array contains the mod**.
  - **np.abs()** → functions do the same absolute operation element-wise.

# Rounding decimals: -

- There are primarily 5 ways of rounding decimals: -
  - **np.trunc()/np.fix()** → remove the decimals, and return the float number closest to zero.
  - **np.around()** → increments the preceding digit or decimal by 1 if >=5 else do nothing. E.g. **np.around(3.1666, 2) outputs to 3.17**.
  - **np.floor()** → rounds off decimal to nearest lower integer.
  - **np.ceil()** → rounds off decimal to nearest upper integer.

# Numpy Logs: -

- Numpy provides functions to perform log at the base 2, e, 10. And all the functions will place -inf or inf in the elements if the log cannot be computed.
- The different functions are: -
  - **np.log2()** → performs log at the base 2.
  - **np.log10()** → performs log at the base 10.
  - **np.log()** → performs log at the base e.

  - NumPy does not provide any function to take log at any base, so we can use the **np.frompyfunc()** function along with inbuilt function **math.log()** with two input parameters and one output parameter.

# Numpy Summations: -

- The difference between **addition** and **summation** is that addition is done between 2 arguments whereas summation happens between n arguments.
- **np.sum()** is the function used to do summations. For e.g.
  - **arr1 = np.array([1, 2, 3])**
    arr2 = np.array([1, 2, 3])
    newarr = np.sum([arr1, arr2], axis = 0) → it will give us the output as 12
- If we can specify the **axis = 1** then the summation will do column wise and the output will be in that case is **6,6.**
- Cumulative sum means partially adding the elements in the array. we can do cumulative sum using the function **np.cumsum()**.
- E.g. The partial sum of [1, 2, 3, 4] would be [1, 1+2, 1+2+3, 1+2+3+4] = [1, 3, 6, 10].

# Numpy products: -

- **np.prod()** → to find the products of the elements in the array.
  - **arr1 = np.array([1, 2, 3, 4])**
    arr2 = np.array([5, 6, 7, 8])
    x = np.prod([arr1, arr2]) → output will be 1*2*3*4*5*7*8*6 = 40320
- If we specify **axis = 1** then numpy will return the product of each axis. After that we will get the output 1*2*3*4 = 24 and 5*6*7*8 = 1680.
- Cumulative product means partially taking multiplication of the elements in the array. we can do cumulative product using the function **np.cumprod()**.
- E.g. The partial product of [1, 2, 3, 4] is [1, 1*2, 1*2*3, 1*2*3*4] = [1, 2, 6, 24].

# Numpy Differences: -

- Discrete difference means **subtracting two successive elements**. E.g. for [1, 2, 3, 4], the discrete difference would be [2-1, 3-2, 4-3] = [1, 1, 1]
- **np.diff()** is used to find the discrete difference.
- We can perform this operation repeatedly by giving **parameter n**. E.g. for [1, 2, 3, 4], the discrete difference with n = 2 would be [2-1, 3-2, 4-3] = [1, 1, 1] , then, since n=2, we will do it once more, with the new result: [1-1, 1-1] = [0, 0].

# Numpy LCM: -

- The lowest common multiple is the smallest number that is a common multiple of two numbers.
- **np.lcm()** function is used to find the lowest common multiple of two numbers.
- To find the LCM of all the values in the array we use the **np.lcm.reduce()** method. This method reduces the array by 1 dimension.

# Numpy GCD: -

- The greatest common divisor / highest common factor is the largest number that is a common factor of both the number.
- **np.gcd(num1, num2)** is the method used to find the GCD of two numbers.
- To find the GCD of all the values in the array we use the method **np.gcd.reduce()**.

# Numpy Set operations: -

- A set in mathematics is a collection of unique elements.
- We can use the numpy **np.unique()** method to find the unique elements in the array. Set arrays should only be 1-D arrays.
- To find the unique values of two arrays, use the **np.union1d() method**.
- To find the values that are present in both the arrays, use the **np.intersect1d() method**. This method takes an optional argument **assume_unique** that should be set to True if we want to speed up the computation.
- To find only the values in the first set that is NOT present in the second set, use the **np.setdiff1d() method**.
- To find only the values that are NOT present in both the sets, we use **np.setxor1d() method**.