Numpy:

NumPy is one of the fundamental packages for scientific computing in Python. n. It contains functionality for multidimensional arrays, high-level mathematical func- tions such as linear algebra operations and the Fourier transform, and pseudorandom number generators. In scikit-learn, the NumPy array is the fundamental data structure. scikit-learn takes in data in the form of NumPy arrays. Any data you're using will have to be converted to a NumPy array. The core functionality of NumPy is the ndarray class, a multidimensional (n-dimensional) array. All elements of the array must be of the same type.

Pre-Requisites:

Basic Knowledge about Python

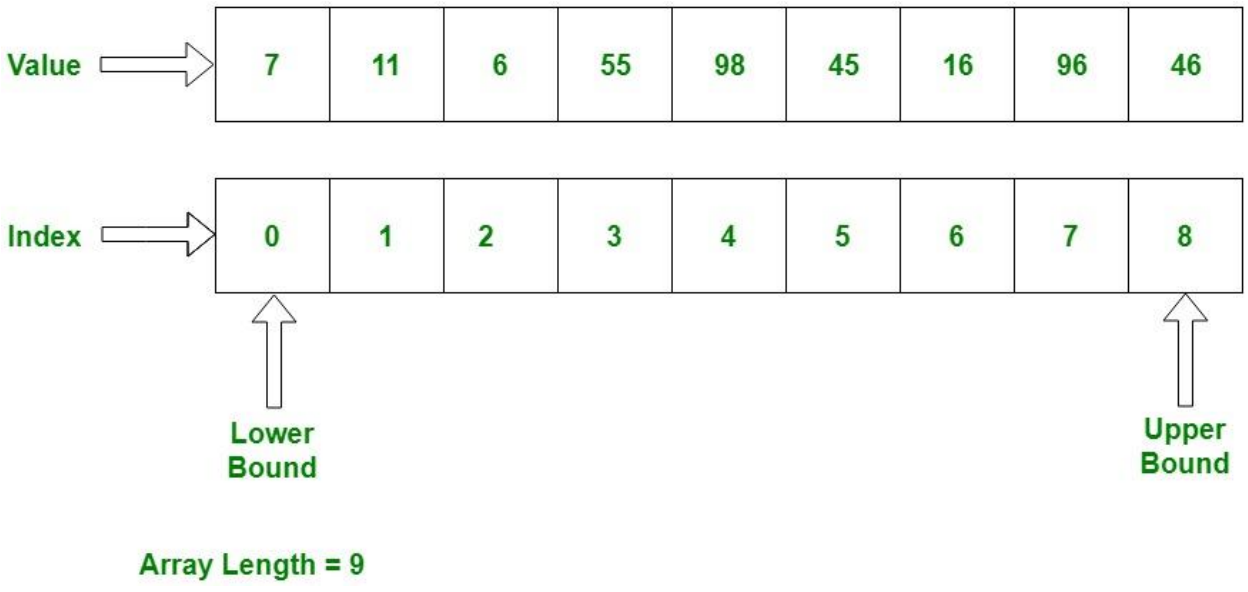
Basic things to know before we start with the topic:

What is array?

Array:

Array is a data structure that contains a group of elements. Typically these elements are all of the same data type, such as an integer or string. Arrays are commonly used in computer programs to organize data so that a related set of values can be easily sorted or search.

In the mathematical concept of a matrix can be represented as a two-dimensional grid, two-dimensional arrays are also sometimes called matrices. In some cases the term “vector” is used in computing to refer to an array, although tuples rather than vectors are the more mathematically correct equivalent. Tables are often implemented in the form of arrays, especially lookup tables; the word *table* is sometimes used as a synonym of *array*.



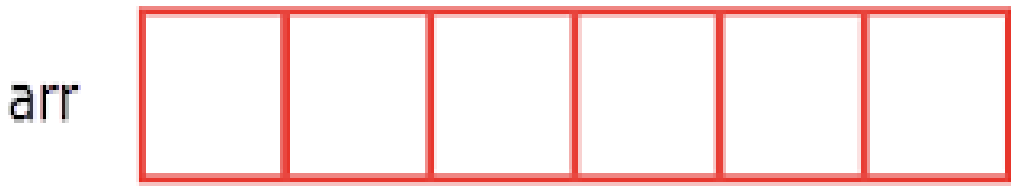
Types of Array

One- dimensional array

Multi-dimensional array

One Dimensional Array:

One dimensional array contains elements only in one dimension. In other words, the shape of the numpy array should contain only one value in the tuple. To create a one dimensional array in Numpy, you can use either of the array(), arange() or linspace() numpy functions.



an array of 6 elements

```
# A character array in C/C++/Java
 char arr1[] = {'D','e','v','I','n','c','e','p','t'};

 # An Integer array in C/C++/Java
 int arr2[] = {10, 20, 30, 40, 50};
```

To access the elements of a **single-dimensional** Array you can use the **indexes** and most of the Arrays are **zero-indexed.**

```
arr1[0]; # gives us D

arr1[2]; # gives us v

arr2[1]; # gives us 20

arr2[4]; # gives us 50
```

Multi-Dimensional Array:

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Multidimensional Array concept can be explained as a technique of defining and storing the data on a format with more than two dimensions (2D). In Python, Multidimensional Array can be implemented by fitting in a list function inside another list function, which is basically a nesting operation for the list function.

Few simple Examples:

```
list=['DevIncept',10,'Sam',100,20]
 print(list[0])
 print(list[2])
 print(list[3])
```

A Python list may contain different types! Indeed, you can store a number, a string, and even another list within a single list. Result:

```
DevIncept
 Sam
 100
```

Why Numpy is used instead List

Numpy data structures perform better in: Size - Numpy data structures take up less space. Performance - they have a need for speed and are faster than lists. Functionality - SciPy and NumPy have optimized functions such as linear algebra operations built in.

Numpy array is densely packed in memory due to its homogeneous type, it also frees the memory faster. So overall a task executed in Numpy is around 5 to 100 times faster than the standard python list, which is a significant leap in terms of speed.

Installing NumPy

If you already have Python, you can install NumPy with:

```
pip install numpy
```

Or

```
conda install numpy
```

How to Import Numpy

Any time you want to use a package or library in your code, you first need to make it accessible. In order to start using NumPy and all of the functions available in NumPy, you'll need to import it. This can be easily done with this import statement:

```
import numpy as np
```

NumPy Basics:

Creating a basic Array

To create a basic NumPy Array you can use the function ***np.array()*** The array object in NumPy is called **ndarray**

```
import numpy as np
 arr1=np.array([1,2,3])
```

```
   arr2=np.array(['a','b','c','d','e'])
   arr3= np.array((1, 2, 3, 4, 5)) # You can also pass
a tuple to creat an Array
   print(arr1)
   print(arr2)
   print(arr3)
   print(type(arr1))
```

Result:

```
  [1 2 3]
   ['a' 'b' 'c' 'd' 'e']
   [1 2 3 4 5]
   <class 'numpy.ndarray'>
```

- To create an Array of Zeroes use ***np.zeros()***
- To create an Array of Ones use ***np.ones()***

To create an Array with a range of elements use ***np.arange()***

```
 import numpy as np
 a=np.zeros(3)
 b=np.ones(5)
 c=np.arange(10)
 d=np.arange(2,9)   # To specify start and stop
 e=np.arange(1,10,2) # To specify start and stop with
step size
 print(a)
 print(b)
 print(c)
 print(d)


print(e)
```

Result:

```
[ 0.  0.  0.]
 [ 1.  1.  1.  1.  1.]
 [0 1 2 3 4 5 6 7 8 9]
 [2 3 4 5 6 7 8]
 [1 3 5 7 9]
```

NdArray(N-Dimensional Array)

1. An array class in Numpy is called as **ndarray**.
2. Number of dimensions of the array is called rank of the array.

Lets Create a 2-D(Dimensional) Array

They are like a matrix or you can say a table

```
 import numpy as np
 arr1=np.array([[1,2,3],[4,5,6]])
 arr2=np.array([[1,2,3],[4,5,6],[7,8,9]])
 arr3=np.array([['a','b','c'],['d','e','f']])

 #creates a 3X3 array with all zeros
 zeros=np.zeros((3,3))

 #creates a 2X2 array with all ones
 ones=np.ones((2,2),dtype='int64')  #specify the type
with (dtype) parameter

 print(arr1)
 print(arr2)
```

```
print(arr3)
print(zeros)
print(ones)
```

Result:

```
[[1 2 3]
 [4 5 6]]

[[1 2 3]
 [4 5 6]
 [7 8 9]]

[['a' 'b' 'c']
 ['d' 'e' 'f']]

# 3X3 Matrix
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]

# 2X2 Matrix
[[1 1]
[1 1]]
```

Basic Operations with Arrays

You can also do basic arithmetic operations with arrays.

```
import numpy as np

arr1=np.array([5,5,5,5])

arr2=np.array([3,3,3,3])

c=arr1+arr2

d=arr1-arr2
```

```
e=arr1/arr2

print(arr1)

print(arr2)

print(c) # addition

print(d) # subtraction

print(e) # division

print("Adding one to all elements:",c+1)

print("Subtracting one from all elements:",d-1)
```

Result:

```
[5 5 5 5]

[3 3 3 3]

[8 8 8 8]

[2 2 2 2]

[1.66666667  1.66666667  1.66666667  1.66666667]

Adding one to all elements: [9 9 9 9]

Subtracting one from all elements: [1 1 1 1]
```

Array Indexing and Slicing

Array Indexing:

- Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

```
import numpy as np

arr1= np.array([3,2,3,4])

arr2= np.array([2,1,1,3])

arr3= np.array(['D','e','V','I','n','c','e','p','t'])

arr4= np.array([[1,2,3,4,5], [6,7,8,9,10]])

print(arr1[0]) # prints first element

print(arr1[-1]) # prints last element

print(arr2[2]) # prints third element

print(arr3[0]) # prints first element

print(arr4[1][3]) #prints 9

print(arr4[1,3]) # Or you can use this type
```

- **Result:**
- 3
- 4
- 1
- D
- 9
- 9

- Slicing in python means taking elements from one given index to another given index.
- We pass slice instead of index like this: `[start:end]`.
- You can also provide Step size as: `[start:end:step]`.

```
import numpy as np arr1=np.array(['D','e','V','I','n','c','e','p','t'])
arr2=np.array([1,2,3,4,5,6,7,8,9])
#2-D Slicing arr3=np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr1[:3]) # default is taken as 0 print(arr1[0:3]) print(arr1[3:]) #
taken till the end print(arr1[::2])
print(arr2[-4:-1]) print(arr2[::2]) # step size as 2
# 2-D Slicing print(arr3[1,1:4]) print(arr3[0:2, 2]) print(arr3[0:2, 1:4])
```

Result:

```
['D' 'e' 'V']

['D' 'e' 'V']

['I' 'n' 'c' 'e' 'p' 't']

['D' 'V' 'n' 'e' 't']



[6 7 8]

[1 3 5 7 9]



# Go over these results carefully they are bit
tricky..

[7 8 9]

[3 8]
```

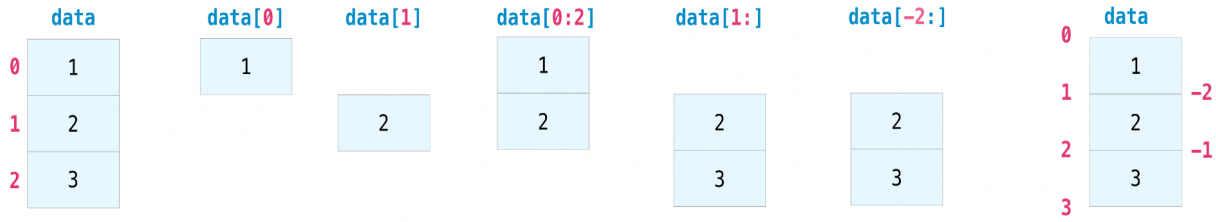
```
[[2 3 4]
 [7 8 9]]
```

All in one example

```
import numpy as np
data = np.array([1, 2, 3])
print(data[1])
print(data[0:2])
print(data[1:])
print(data[-2:])
```

Result:

```
2
[1 2]
[2 3]
[2 3]
```



NumPy Functions

Numpy Array Shape:

`ndarray.shape` will display a tuple of integers that indicate the number of elements stored along each dimension of the array. If, for example, you have a 2-D array with 2 rows and 3 columns, the shape of your array is (2, 3).

```
import numpy as np
 arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
 arr1= np.array([1,2,3,4])

 print(arr.shape)
 print(arr1.shape)
```

Result:

```
(2, 4)
 (4,)
```

All In One Example

```
  array_example = np.array([[[0, 1, 2, 3],
                             [4, 5, 6, 7]],

                            [[0, 1, 2, 3],
                             [4, 5, 6, 7]],

                            [[0 ,1 ,2, 3],
                             [4, 5, 6, 7]]])

   print('No of dimensions of the
array:',array_example.ndim)     # .ndim for dimensions
   print('Total no of elements of the
array:',array_example.size) # .size for size
   print('Shape of the array:',array_example.shape)
# .shape for shape
```

Result:

```
   No of dimensions of the array: 3
    Total no of elements of the array: 24
    Shape of the array: (3, 2, 4)
```

NumPy Array Reshaping

Using `arr.reshape()` will give a new shape to an array without changing the data.

Just remember that when you use the reshape method, the array you want to produce needs to have the same number of elements as the original array. **Reshape from 1-D to 2-D**

```
  import numpy as np

   arr1 = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12])

   newarr1 = arr1.reshape(4, 3)

   arr2 =
np.array(['D','e','V','I','n','c','e','p','t'])

   newarr2 = arr2.reshape(3,3)

   print(newarr1)

   print(newarr2)
```

Result:

```
   [[ 1  2  3]
     [ 4  5  6]
     [ 7  8  9]
     [10 11 12]]


    [['D' 'e' 'V']
     ['I' 'n' 'c']
```

```
     ['e' 'p' 't']]
```

Reshape from 1-D to 3-D

```
   import numpy as np

    arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12])

    newarr = arr.reshape(2, 3, 2)

    print(newarr)
```

Result:

```
   [[[ 1  2]
      [ 3  4]
      [ 5  6]]


     [[ 7  8]
      [ 9 10]
      [11 12]]]
```

Can we Reshape into any Shape??

Yes, as long as the elements required for reshaping are equal in both shapes.

We can reshape an 8 elements 1D array into 4 elements in 2 rows 2D array but we cannot reshape it into a 3 elements 3 rows 2D array as that would require 3x3 = 9 elements.

We can use `reshape(-1)` to convert mult-dimensional array into 1-D array.

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

newarr = arr.reshape(-1)

print(newarr)
```

Result:

```
[1 2 3 4 5 6]
```

NumPy Sorting and Searching

NumPy Sorting

- Sorting means putting elements in an ***ordered sequence.***

The NumPy ndarray object has a function called `sort()`, that will sort a specified array.

```
import numpy as np

 arr1 = np.array([3, 2, 0, 1])

 arr2 = np.array(['red','blue','green'])



 # 2-D array



arr3 = np.array([[3, 2, 4], [5, 0, 1]])



 print(np.sort(arr1))

 print(np.sort(arr2))

 print(np.sort(arr3))
```

Result:

```
[0 1 2 3]

 ['blue' 'green' 'red']

 [[2 3 4]

  [0 1 5]]
```

- **NumPy Searching**
  Searching is an operation or a technique that helps finds the place of a given element or value in the list. Any search is said to be successful or unsuccessful depending upon whether the element that is being searched is found or not.

1. where() :

Returns the indices of the searched value.

```
import numpy as np

arr1 = np.array([1, 2, 3, 4, 5, 4, 4])

x = np.where(arr1 == 4)

y = np.where(arr1%2 == 0)

z = np.where(arr1%2 == 1)

print(x)

print(y)

print(z)
```

Result:

```
(array([3, 5, 6]),) # Which means that the value 4 is present at index 3, 5, and 6.

(array([1, 3, 5, 6]),)
```

```
(array([0, 2, 4]),)
```

2. searchsorted() :

Performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order.

> The **searchsorted()** method is assumed to be used on sorted arrays. ```python import numpy as np arr1 = np.array([6, 7, 8, 9]) x = np.searchsorted(arr1, 7) print(x)

Multiple values
arr2 = np.array([1, 3, 5, 7]) y = np.searchsorted(arr2, [2, 4, 6]) print(y)

Result:

```python
1
```

```
[1 2 3]
```

Median

- **Median** - The mid point value

The median value is the value in the middle, after you have sorted all the values.

```
import numpy

# For one value in the middle
```

```
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]

x = numpy.median(speed)

print(x)



# For two values in the middle

speed1 = [99,86,87,88,86,103,87,94,78,77,85,86]

y = numpy.median(speed1)

print(y)
```

Result:

```
87.0



86.5
```

-

Mode

- **Mode** - The most common value
- The Mode value is the value that appears the most number of times.

This function comes under **SciPy**

```
from scipy import stats

speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

```
x = stats.mode(speed)

print(x)
```

Result: ```python ModeResult(mode=array([86]), count=array([3]))

##The mode() method returns a ModeResult object that contains the mode number (86), and count (how many times the mode number appeared (3)).

```
## Standard Deviation (σ)

* Standard deviation is a number that describes how spread out the values are.

* A low standard deviation means that most of the numbers are close to the mean (average) value.

* A high standard deviation means that the values are spread out over a wider range.

> Use the NumPy **std()** method to find the standard deviation:
```

```
```python

import numpy

speed = [86,87,88,86,87,85,86]

x = numpy.std(speed)
```
```

```
print(x)
```

```
speed1 = [32,111,138,28,59,77,97]
y = numpy.std(speed1)
print(y)
```

Result:

```
0.9035079029052513
```

```
37.84501153334721
```