Y.C.C.E

DATE :

PRACTICAL - 1

Aim :- Manipulating single and multidimensional arrays using NumPy.

Theory :- NumPy

NumPy, short for 'Numerical Python' provides an efficient interface to store and operate on dense data buffers. NumPy arrays are like Python's built-in list type, but NumPy arrays provide much more efficient storage and data operations as the arrays grow bigger in size.

NumPy arrays form the core of nearly the entire ecosystem of datascience tools in Python. Therefore it is very valuable to learn & use NumPy.

Installation of NumPy - Syntax :- pip install numpy

Importing NumPy - Syntax :- import numpy

* Advantages of NumPy

(i) Speed & Performance
(ii) Memory Efficiency
(iii) Convenience & Easy to use
(iv) Scalability & Cross-Platform Compatibility.

Y.C.C.E

DATE :

★ Creating Arrays using NumPy

1) 1-D Array :- It is the simplest form of array where elements are stored linearly and can be accessed individually by specifying index value as they have the same data type.

Syntax :-
```
import numpy as np
arr = np.array([1,2,3,4,5])
print(arr)
```

2) 2-D Array :- These are multidimensial arrays. They majorly represent matrix

Syntax :-
```
arr = np.array([[1,2,3],[4,5,6]])
print(arr)
```

* Functions used in manipulating 1-D & 2-D Arrays

1) np.zeros :- Used to create a new array filled with zeros.
Syntax :-
```
z = np.zeros(10, dtype = int)
print(z)
```

2) np.ones :- used to create a new array filled with ones
Syntax :-
```
variable = np.ones(array_type, dtype = datatype)
print(variable)
```

Y.C.C.E
DATE :

3) np.full :- used to create an array filled with a given value
Syntax :- numpy. np.full (shape , value)

4) np.arange :- used to create an array with equally spaced values within a specified range.

Syntax :- numpy.arange (start, stop, space)

5) np.linspace :- used to create an array of evenly spaced numbers over a specified interval.

Syntax :- np.linspace (start, stop, divisions)

* Use of 'Random' Function

1) np.random.random() :- used to create arrays with random values for testing, generating or initializing parameters

Syntax :- np.random.random (shape)

2) np.random.normal() :- used to create arrays from a Gaussian normal distribution. It is defined by mean and standard deviation.

Syntax :- np.random.normal (default , scale , shape)
(mean) (σ)

Y.C.C.E

DATE :

* Array Indexing & Slicing

I) Indexing in Numpy refers to accessing specific elements from an array. It depends on size of the array.

For 1-D Array

```
a1 = np.array([10,20,30,40])
print(a1[2])
>>> 30
```

For 2-D Array

```
a2 = np.array([[10,20,30],[40,50,60]])
print(a2[0][2])
>>> 60
```

* More Types of Indexing

1) Boolean :- uses a boolean array to filter elements.
   Syntax :-
   ```
   condition = a1 > 20
   print(a1[condition])
   ```

2) Fancy :- uses array of indices to select specific elements.
   Syntax :-
   ```
   indices = [0,2,4]
   print(a1[indices])
   ```

II) Slicing in Numpy allows you to access a subset of an array using a range of indices.

Syntax :-
```
a1 = np.array([10,20,30,40,50])
print(a1[start:stop:step])
```

Y.C.C.E
DATE :

Operation on Arrays using Numpy

1) Array Broadcasting

- It is a powerful feature that allows you to perform element-wise operations on arrays of different shapes.

Example

```
a1 = np.array ([1,2,3])
a2 = np.array ([[10,20,30],[40,50,60]])
a3 = a1 + a2
print (a3)
```

>>> [[11. 22 33][41.51.61]

2) Reshaping Arrays

- It is a common operation that allows you to change shape of array.

Syntax :-

```
array_reshape = np.reshape (size)
print (array_reshape)
```

3) Flattening Arrays

- It is used to convert multidimensional array into a one-dimensional array.

Syntax :-

```
array_flattened = np.flatten ()
print (array_flattened)
```

Y.C.C.E

DATE :

4) Combining Arrays via Stacking

Syntax :- s1 = np.vstack ((a1, a2)) // vertical stack
print (s1)
s2 = np.hstack ((a1, a2)) // horizontal stack
print (s2)

★ Mathematical Operation on Arrays

1) Dot Product :- np.dot (arr1, arr2)

2) Statistical Operations
(a) mean :- np.mean (arr1)
(b) median :- np.median (arr1)
(c) minimum value :- np.min (arr1)
(d) maximum value :- np.max (arr1)
(e) sum :- np.sum (arr1)

3) Logical Operations
(a) AND :- np.logical_and (conditions)
(b) OR :- np.logical_or (conditions)