

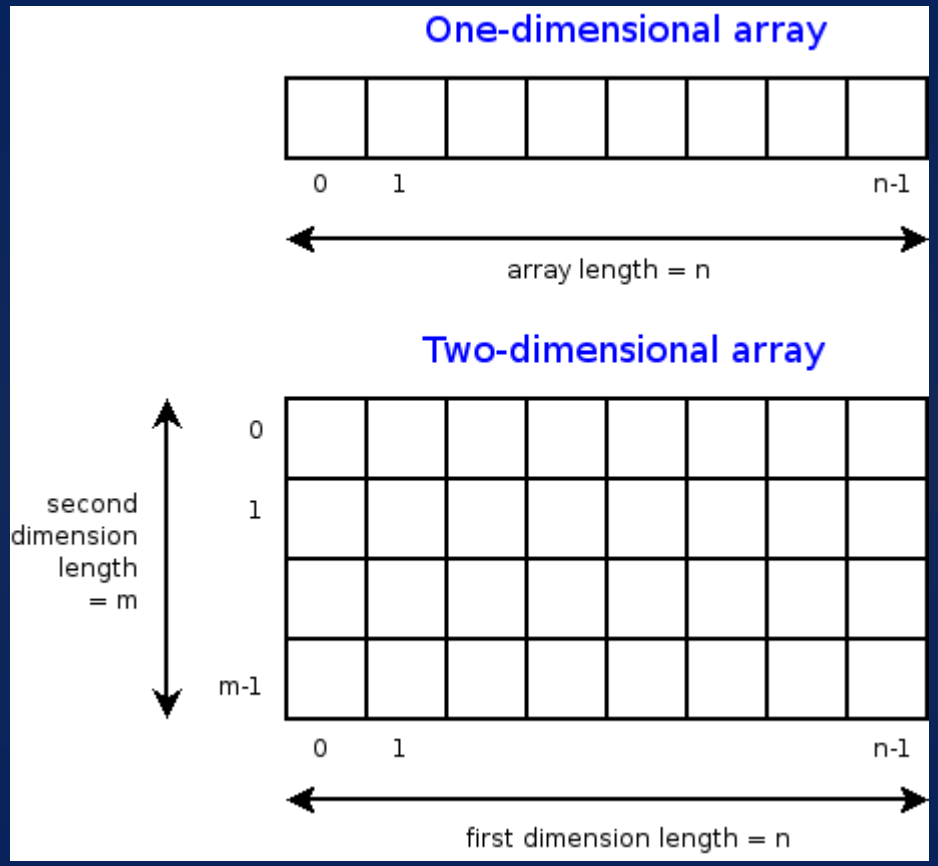
Functions in Python

Introduction to NumPy

- NumPy stands for ‘Numerical Python’ or ‘Numeric Python’. It is an open source module of Python which provides fast mathematical computation on arrays and matrices.
- Since, arrays and matrices are an essential part of the Machine Learning ecosystem, NumPy along with Machine Learning modules like Scikit-learn, Pandas, Matplotlib, TensorFlow, etc. complete the Python Machine Learning Ecosystem.
- NumPy is also useful in linear algebra, random number capability etc.
- In order to use NumPy, you must import in your module by using a statement like:
  - import numpy as np

Array in NumPy

- A NumPy array is a simply a grid that contain values of same/homogeneous type.
- NumPy Array come in two forms:
  - 1 D (One Dimensional) array known as vectors (having single row/columns only).
  - Multidimensional array known as Matrices (can have multiple rows and columns).



Creating Array from exiting List

```
import numpy as np
x=[4,5,7,8,9]
arr=np.array(x)
print(arr)
#[4 5 7 8 9]
```

```
import numpy as np
x=[[4,5,5],[5,8,9],[4,7,8]]
arr=np.array(x)
print(arr)

[[4 5 5]
 [5 8 9]
 [4 7 8]]
```

Data Types in NumPy

- In NumPy, data types (or dtypes) specify the type of data elements in an array.
- NumPy supports a wide range of data types, which can be categorized into the following major groups:
  - Numeric Types (integer, floating point number, complex number)
  - Boolean Type (True/False)
  - String Types (Text)
  - Object Type
  - Structured Arrays (1 D array, 2 D Array)

Array Attribute in NumPy

- NumPy array have a set of attributes that you can access.
- ndim: Display the dimension of the array
- Shape: Returns a tuple of integers indicating the size of the array.
- size : Return the total number of elements in the NumPy array
- dtype : Return the type of elements in the array
- Itemsize : Returns the size of each elements of an array in bytes.
- Reshape : Reshapes the NumPy array.

Example

```
import numpy as np
x=[1,2,3,4]
y=[[4,5,5],[5,8.2,9]]
arr1=np.array(x)
arr2=np.array(y)
print("Array 1 = ",arr1)
print("Array 2 = ",arr2)
print("Dimension of Array 1 = ",arr1.ndim)
print("Dimension of Array 2 = ",arr2.ndim)
print("Shape of Array 1 = ",arr1.shape)
print("Shape of Array 2 = ",arr2.shape)
print("Size of Array 1 = ",arr1.size)
print("Size of Array 2 = ",arr2.size)
print("Data Type of Array 1 = ",arr1.dtype)
print("Data Type  of Array 2 = ",arr2.dtype)
print("Item Size of Array 1 = ",arr1.dtype)
print("Item Size   of Array 2 = ",arr2.dtype)
arr1=arr1.reshape(2,2)
arr2=arr2.reshape(3,2)
print("Array 1 = ",arr1)
print("Array 2 = ",arr2)
print("Shape of Array 1 = ",arr1.shape)
print("Shape of Array 2 = ",arr2.shape)
```

```
Array 1 =  [1 2 3 4]
Array 2 =  [[4.  5.  5. ]
 [5.  8.2 9. ]]
Dimension of Array 1 =  1
Dimension of Array 2 =  2
Shape of Array 1 =  (4,)
Shape of Array 2 =  (2, 3)
Size of Array 1 =  4
Size of Array 2 =  6
Data Type of Array 1 =  int32
Data Type  of Array 2 =  float64
Item Size of Array 1 =  int32
Item Size   of Array 2 =  float64
Array 1 =  [[1 2]
 [3 4]]
Array 2 =  [[4.  5. ]
 [5.  5. ]
 [8.2 9. ]]
Shape of Array 1 =  (2, 2)
Shape of Array 2 =  (3, 2)
```

Methods/Function in NumPy

- empty()
- zeros()
- ones()
- copy()
- arange()

empty()

- Empty function is used to create empty arrays or an uninitialized array of specified shape and dtype, in which you can store actual data as and when required.
- Note that empty() created array with random garbage values.

```
import numpy as np
arr1=np.empty([3,2],dtype=int)
print(arr1)

--------------
[[1 0]
 [8 0]
 [0 0]]
```

Zeros()

- The function zeros() take same attribute as empty(), and creates an array with specifies size and type but filled with zeros.
- Default dtype is float.

```
import numpy as np
arr1=np.zeros([3,2])
print(arr1)
print(arr1.dtype)

------------
[[0. 0.]
 [0. 0.]
 [0. 0.]]
float64
```

ones()

- The function ones() takes same attributes as empty(), and creates an array with specified size and type but filled with ones.

```
import numpy as np
arr1=np.ones([3,2],dtype=float)
print(arr1)

----------------
[[1. 1.]
 [1. 1.]
 [1. 1.]]
```

copy()

- To copy one array to another.

```
import numpy as np
arr1=np.array([4,5,6,8])
arr2=np.copy(arr1) # create a duplicate copy on array 1
arr3=arr1 # aliasing
arr1[0]=50
print(arr1)
print(arr2)
print(arr3)
```

```
[50  5  6  8]
[4 5 6 8]
[50  5  6  8]
>>>
```

arange()

- arange() function take the start index of the array, the end index, and the step size (which is optional) and returns an ndarray object containing evenly spaced values within a given range.
- Syntax:
  - numpy.arange(start,stop,step,dtype)

```
import numpy as np
arr1=np.arange(2,9,2,float)
print(arr1)

------------

[2. 4. 6. 8.]
```

```
import numpy as np
arr1=np.arange(start=5,stop=12,step=3)
print(arr1)

------------------

[ 5  8 11]
```

asarray()

- asarray() is used for converting the python sequence into ndarray.

```
import numpy as np
ls1=[4,5,6,7]
tp1=(4,6,7,8)
arr1=np.asarray(ls1)
arr2=np.asarray(tp1)
print(arr1)
print(arr2)
--------------------

[4 5 6 7]
[4 6 7 8]
```

linspace()

- Linspace() is used to generate evenly spaced elements between two given limits.
- Syntax:

  numpy.linspace(start,stop,num,endpoint,retstep,dtype)

  - **start & end** : start and end value of the sequence.
  - **num**: number of evenly spaced samples to be generated. (default: 50)
  - **endpoint** : True by default , hence the stop value included in the sequence.
  - **retstep**: if true return samples and step between the consecutive number.
  - **dtype**: data type for output ndarray.

Example

```
import numpy as np
x=np.linspace(10,20,5,endpoint=False)
print(x)
---------
[10. 12. 14. 16. 18.]
```

```
import numpy as np
x=np.linspace(10,15,3,endpoint=False,retstep=False)
print(x)
x=np.linspace(10,15,3,endpoint=False,retstep=True,dtype=int)
print(x)

---------------
[10.         11.66666667 13.33333333]

(array([10, 11, 13]), 1.6666666666666667)
```

Join Array : Concatenate()

- The np.concatenate() function joins arrays.

```
import numpy as np
ar1=np.array([1,3,4])
ar2=np.array([2,5,7])
ar3=np.concatenate([ar1,ar2])
print(ar3)
```

```
[1 3 4 2 5 7]
```

Join 2d Array : hstack() & vstack()

- The np.hstack function stacks arrays in sequence horizontally (column-wise), and np.vstack stacks arrays in sequence vertically (row-wise).

```
import numpy as np
ar1=np.array([[1,3,4],[5,3,4]])
ar2=np.array([[2,5,7],[3,4,5]])
print("Horizontal Stack\n",np.hstack((ar1,ar2)))
print("Verical Stack\n",np.vstack((ar1,ar2)))
```

```
Horizontal Stack
 [[1 3 4 2 5 7]
 [5 3 4 3 4 5]]
Verical Stack
 [[1 3 4]
 [5 3 4]
 [2 5 7]
 [3 4 5]]
```

Split Array

- The np.split() function splits an array into multiple sub-arrays.

```
import numpy as np
ar1=np.array([5,6,7,8,8,9,9,5,4,3,5])
x,y,z=np.split(ar1,[4,6])
print(x)# from index 0 to 3
print(y) # from index 4 to 5
print(z) # from index 6 to end
```

```
[5 6 7 8]
[8 9]
[9 5 4 3 5]
```

Arithmetic Operation in 1D array

```
import numpy as np
ar1=np.array([1,2,4,5])
ar2=np.array([2,3,5,6])
print("Array 1 ",ar1)
print("Array 2 ",ar2)
print("SUM ",ar1+ar2)
print("SUB ",ar1-ar2)
print("MUL ",ar1*ar2)
print("DIV ",ar1/ar2)
```

```
Array 1  [1 2 4 5]
Array 2  [2 3 5 6]
SUM  [ 3  5  9 11]
SUB  [-1 -1 -1 -1]
MUL  [ 2  6 20 30]
DIV  [0.5        0.66666667 0.8        0.83333333]
```

Arithmetic Operation in 2D array

```
import numpy as np
ar1=np.array([[1,2,4],[3,4,6]])
ar2=np.array([[2,3,5],[4,3,5]])
print("Array 1 \n",ar1)
print("Array 2 \n",ar2)
print("SUM \n",ar1+ar2)
print("SUB \n",ar1-ar2)
print("MUL \n",ar1*ar2)
print("DIV \n",ar1/ar2)
```

```
Array 1
 [[1 2 4]
 [3 4 6]]
Array 2
 [[2 3 5]
 [4 3 5]]
SUM
 [[ 3  5  9]
 [ 7  7 11]]
SUB
 [[-1 -1 -1]
 [-1  1  1]]
MUL
 [[ 2  6 20]
 [12 12 30]]
DIV
 [[0.5        0.66666667 0.8       ]
 [0.75       1.33333333 1.2       ]]
```

Thank You

Address: G. R. Complex Preetam Nagar, Dhoomanganj
Prayagraj Uttar Pradesh 211011

Mobile No.: 9598948810,8874588766,9532878816
Website : www.infomaxacademy.com

