array buildin functions

1.Array Creation Functions

In [1]:

```python
import numpy as np
```

In [2]:

```python
# create an array from a list
a = np.array([10,20,30,40,50])
print('Array a:',a)
```

```
Array a: [10 20 30 40 50]
```

In [3]:

```python
# create an array with evenly spaced values
b = np.arange(0,10,2)
print("Array b:",b)
```

```
Array b: [0 2 4 6 8]
```

In [4]:

```python
# create an array with linearly spaced values
c = np.linspace(1,2,6)
print("Array c:",c)
```

```
Array c: [1.  1.2 1.4 1.6 1.8 2. ]
```

In [5]:

```python
c1 = np.linspace(5,9,5)
c1
```

Out[5]:

```
array([5., 6., 7., 8., 9.])
```

In [6]:

```python
c3 = np.linspace(0,1)
c3
```

Out[6]:

```
array([0.        , 0.02040816, 0.04081633, 0.06122449, 0.08163265,
       0.10204082, 0.12244898, 0.14285714, 0.16326531, 0.18367347,
       0.20408163, 0.2244898 , 0.24489796, 0.26530612, 0.28571429,
       0.30612245, 0.32653061, 0.34693878, 0.36734694, 0.3877551 ,
       0.40816327, 0.42857143, 0.44897959, 0.46938776, 0.48979592,
       0.51020408, 0.53061224, 0.55102041, 0.57142857, 0.59183673,
       0.6122449 , 0.63265306, 0.65306122, 0.67346939, 0.69387755,
       0.71428571, 0.73469388, 0.75510204, 0.7755102 , 0.79591837,
       0.81632653, 0.83673469, 0.85714286, 0.87755102, 0.89795918,
       0.91836735, 0.93877551, 0.95918367, 0.97959184, 1.        ])
```

In [7]:

```python
# create an array filled with zeros
d = np.zeros((2,5))
print("Array d:\n",d)
```

```
Array d:
 [[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

In [8]:

```
# Array filled wih ones
e = np.ones((5,3))
print("Array e:\n",e)
```

```
Array e:
 [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

In [9]:

```
e1 = np.ones((5,3),dtype=int)
print("Array e1:\n",e1)
```

```
Array e1:
 [[1 1 1]
 [1 1 1]
 [1 1 1]
 [1 1 1]
 [1 1 1]]
```

In [10]:

```
# Identity matrix
f = np.eye(3) # 3x3 matrix
print("Identity matrix : f\n",f)
```

```
Identity matrix : f
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

In [11]:

```
f1 = np.eye(5)
print("Identity matrix :f1\n",f1)
```

```
Identity matrix :f1
 [[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

2.Array Manipulation Functions

In [12]:

```
g = np.array([2,4,5,7])
reshaped = np.reshape(g,(2,2))
print("reshaped array :\n",reshaped)
```

```
reshaped array :
 [[2 4]
 [5 7]]
```

In [13]:

```
g1 = np.array([1,2,3,4,5,6,4,5,3]).reshape(3,3)
print("reshaped array : g1\n",g1)
```

```
reshaped array : g1
 [[1 2 3]
 [4 5 6]
 [4 5 3]]
```

In [14]:

```
# flatten an array
# converting a multi-dimensional structure into a one-dimensional list
h = np.array([[1,2],[3,5]])
flattened = np.ravel(h)
print("flattened array :",flattened)
```

```
flattened array : [1 2 3 5]
```

In [15]:

```
h1 = np.array([[4,2],[1,8]]).ravel()
h1
```

Out[15]:

```
array([4, 2, 1, 8])
```

In [16]:

```
# transpose an array ( which swaps the rows and columns)
i = np.array([[2,3],[5,7]])
transposed = np.transpose(i)
print("transposed:\n",transposed)
```

```
transposed:
 [[2 5]
 [3 7]]
```

In [17]:

```
# stack arrays vertically
a1 = np.array([1,3])
b1 = np.array([2,4])
stacked = np.vstack([a1,b1])
print("stacked arrays :\n",stacked)
```

```
stacked arrays :
 [[1 3]
 [2 4]]
```

3.mathematical Functions

In [34]:

```
# Add two arrays
# element wise addition
j = np.array([1,2,3,4,5])
added = np.add(j,3) # add 3 to each element
print("added 2 to j:",added)
```

```
added 2 to j: [4 5 6 7 8]
```

In [35]:

```
j1 = np.array([1,2,3,4,5])
added = np.add(j1,1) # add 1 to each element
print("added 1 to j1:",added)
```

```
added 1 to j1: [2 3 4 5 6]
```

In [38]:

```
# square each element
squared = np.power(j,2)
```

```
print("squared : g",squared)
```

```
squared : g [ 1  4  9 16 25]
```

In [40]:

```
# square root of each element
sqrt = np.sqrt(j)
print("square root of j:",sqrt)
```

```
square root of j: [1.         1.41421356 1.73205081 2.         2.23606798]
```

In [42]:

```
print(g)
print(j)
```

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

In [44]:

```
# Dot products of two arrays
dot_product = np.dot(g,j)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[44], line 2
      1 # Dot products of two arrays
----> 2 dot_product = np.dot(g,j)

ValueError: shapes (4,) and (5,) not aligned: 4 (dim 0) != 5 (dim 0)
```

In [46]:

```
A1 = np.array([1,2,3])
B1 = np.array([4,5,6])
```

In [47]:

```
print(A1)
print(B1)
```

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

In [50]:

```
dot_product = np.dot(A1,B1)
print("Dot Product of A1 and B1 : ",dot_product)
```

```
Dot Product of A1 and B1 :  32
```

4.Statistical Functions

In [ ]:

```
# mean
```

In [22]:

```
s = np.array([1,3,5,7])
mean = np.mean(s)
print("mean of s :",mean)
```

```
mean of s : 4.0
```

In [25]:

```
# Standard deviation of an array
std_dev = np.std(s)
print("standard deviation:",std_dev)
```

```
standard deviation: 2.23606797749979
```

In [26]:
```
min(s)
```

Out[26]:
```
1
```

In [33]:
```
# minimum element of an array
minimum = np.min(s)
print("minimum of s :",minimum)
```

```
minimum of s : 1
```

In [32]:
```
# maximum element of an array
maximum = np.max(s)
print("maximum of s :",maximum)
```

```
maximum of s : 7
```

5.Linear Algebra Functions

In [37]:
```
# create a matrix
matrix = np.array([[1,2],[3,4]])
print("matrix:\n",matrix)
```

```
matrix:
 [[1 2]
 [3 4]]
```

In [40]:
```
# Determinant of a matrix
det = np.linalg.det(matrix)
print("determinant of matrix:",det)
```

```
determinant of matrix: -2.0000000000000004
```

In [43]:
```
# Inverse of a matrix
inverse = np.linalg.inv(matrix)
print("inverse of a matrix:\n",inverse)
```

```
inverse of a matrix:
 [[-2.   1. ]
 [ 1.5 -0.5]]
```

6.Random Samping Functions

In [47]:
```
# generate random values between 0 and 1
random_val = np.random.rand(3)
print("random values :",random_val)
```

```
random values : [0.95800574 0.83736579 0.27101469]
```

In [83]:
```
# set seed for reproducibility
np.random.seed(10) # seed() -->to get the same random number each time you run the
np.random.randint(9)
```

Out[83]:
```
4
```

In [92]:

```
np.random.seed(10) # we will get same random numbers
np.random.rand(4)
```

Out[92]:

```
array([0.77132064, 0.02075195, 0.63364823, 0.74880388])
```

In [93]:

```
r = np.random.rand(8)
r
```

Out[93]:

```
array([0.49850701, 0.22479665, 0.19806286, 0.76053071, 0.16911084,
       0.08833981, 0.68535982, 0.95339335])
```

In [98]:

```
np.random.randint(1,50,3)
```

Out[98]:

```
array([26, 30,  7])
```

In [100…

```
np.random.seed(10)
np.random.randint(1,30,5)
```

Out[100…

```
array([10,  5, 16,  1, 18])
```

In [ ]:

In [ ]: