NUMPY_COMPLETE_Buildin_Function

In [2]:
```
import numpy as np
```

In [3]:
```
# Create an array from a list
a= np.array([1,2,3])
print('Array a:',a)
```

```
Array a: [1 2 3]
```

In [4]:
```
# Create an array with evenly spaced values
b = np.arange(0,10,2)     # Values from 0 to 10 with step 2
print('Array b:',b)
```

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

In [5]:
```
# Create an array with linear spaced values
c = np.linspace(0,1,5)      # 5 values evenly spaced between 0 and 1
print('Array c:',c)
```

```
Array c: [0.   0.25 0.5  0.75 1.  ]
```

In [6]:
```
#Create an array filled with zeros
ex = np.zeros(5)
print('Array ex:',ex)
```

```
Array ex: [0. 0. 0. 0. 0.]
```

In [7]:
```
# Create an array filled with zeros
d = np.zeros((2,3))      # if use single bracket 3 will consider as datatype
print('Array d:\n',d)
```

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

In [8]:
```
 # Create an array filled with zeros
e = np.ones((3,2))      # if use single bracket 2 will consider as datatype
print('Array e:\n',e)
```

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

In [9]:
```
# Create an identity matrix
f = np.eye(4)      # 4*4 identity matrix
print('identity matrix f:\n',f)
```

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

In [10]:

```
# Reshape an array
a1 = np.array([1,2,3])
reshaped = np.reshape(a1,(1,3))   # reshape as 1*3 matrix
print('Reshaped array:',reshaped)
```

```
Reshaped array: [[1 2 3]]
```

In [11]:

```
a= np.arange(0,4)
b= np.reshape(a,(2,2))
b
```

Out[11]:

```
array([[0, 1],
       [2, 3]])
```

In [12]:

```
# Flatten an array
f1 = np.array([[1,2],[3,4]])
flattened = np.ravel(f1)    # flatten to 1 D array
print('Flattedned array f1:', flattened)
```

```
Flattedned array f1: [1 2 3 4]
```

In [13]:

```
# Transpose an array
e1 = np.array([[1,2],[3,4]])
transposed = np.transpose(e1)     # Transpose the array
print('Transposed array:\n',transposed)
# transpose array is new array got by swaping the values of original array
```

```
Transposed array:
 [[1 3]
 [2 4]]
```

In [14]:

```
# Stack arrays vertically
a2 = np.array([1,2])
b2 = np.array([3,4])
stacked = np.vstack([a2,b2])      # Stack a and b vertically
print('Stacked arrays:\n',stacked)
```

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

In [15]:

```
# Add two arrays
g = np.array([1,2,3,4])
added = np.add(g,2)    # Add 2 to each element
print('Added 2 to g:',added)
```

```
Added 2 to g: [3 4 5 6]
```

In [16]:

```
# Square of each element
squared= np.power(g,2) # Square of each element
print('Squared g:',squared)
```

```
Squared g: [ 1  4  9 16]
```

In [17]:

```
# Square root of each element
sqrt_val = np.sqrt(g)   # Square root of each element
print('Square root of g:',sqrt_val)
```

```
Square root of g: [1.         1.41421356 1.73205081 2.        ]
```

In [18]:
```
print (a1)
print(g)
```

```
[1 2 3]
[1 2 3 4]
```

In [19]:
```
# Dot product of two arrays
a2 = np.array([1,2,3])
dot_product = np.dot(a2,g)             # dot product of a and g
print('Dot product of a and g;', dot_product)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[19], line 3
      1 # Dot product of two arrays
      2 a2 = np.array([1,2,3])
----> 3 dot_product = np.dot(a2,g)             # dot product of a and g
      4 print('Dot product of a and g;', dot_product)

ValueError: shapes (3,) and (4,) not aligned: 3 (dim 0) != 4 (dim 0)
```

In [20]:
```
a= np.arange(1,4)
```

In [21]:
```
print(a)
print(a1)
```

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

In [22]:
```
dot_product = np.dot(a1,a)       # Dot product of a and g
print('Dot product of a1 and a:',dot_product)
```

```
Dot product of a1 and a: 14
```

In [23]:
```
# Mean of array
s = np.array([1,2,3,4])
mean = np.mean(s)
print('Mean of s:',mean)
```

```
Mean of s: 2.5
```

In [24]:
```
# Standard deviation of array
std_dev = np.std(s)
print('Standard deviation of s:',std_dev)
```

```
Standard deviation of s: 1.118033988749895
```

In [25]:
```
# Minimum element of array
minimum = np.min(s)
print('Minimum value of s:',minimum)
```

```
Minimum value of s: 1
```

In [26]:
```
# Maximum element of an array
maximum = np.max(s)
print('Maximum of s:',maximum)
```

```
Maximum of s: 4
```

In [27]:

```
# Create a matrix
matrix = np.array([[1,2],[3,4]])
```

In [28]:

```
# Determinant of a matrix
determinant = np.linalg.det(matrix)
print('Determinant of matrix:',determinant)
```

```
Determinant of matrix: -2.0000000000000004
```

In [29]:

```
# Inverse of a metrix
inverse = np.linalg.inv(matrix)
print('Inverse of matrix:\n',inverse)
```

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

In [30]:

```
# Generate random values between 0 and 1
random_vals = np.random.rand(3)    # Array of 3 random values between 0 to 1
print('Rndom valiues:',random_vals)
```

```
Rndom valiues: [0.52523076 0.21826376 0.22879057]
```

In [31]:

```
# Set seed for reproduciability
np.random.seed(0)
# Generate random values between 0 and 1
random_vals = np.random.rand(3)    # Array of 3 random values between 0 to 1
print('Rndom valiues:',random_vals)
```

```
Rndom valiues: [0.5488135  0.71518937 0.60276338]
```

In [32]:

```
# Generate random integers
rand_ints = np.random.randint(0,10,size=5)    #  5 random integers between 0 to 10
print('Randonintegers:',rand_ints)
```

```
Randonintegers: [3 7 9 3 5]
```

In [33]:

```
# Setting seed for reproduciability
np.random.seed(0)
# there can be any value in place of 0; random output will be according to this see
# Generate random integers
rand_ints = np.random.randint(0,10,size=5)    #  5 random integers between 0 to 10
print('Randonintegers:',rand_ints)
```

```
Randonintegers: [5 0 3 3 7]
```

In [34]:

```
# Setting seed for reproduciability
np.random.seed(3)
# there can be any value in place of 0; random output will be according to this see
# Generate random integers
rand_ints = np.random.randint(0,10,size=5)    #  5 random integers between 0 to 10
print('Randonintegers:',rand_ints)
```

```
Randonintegers: [8 9 3 8 8]
```

for that specific seed value it reproduces same random output values

In [35]:

```
# Check if all elements are True
# all
logical_test = np.array([True,False,True])
all_true = np.all(logical_test)  #  Check if all elements are True
print("All elements True:",all_true)
```

```
All elements True: False
```

In [36]:

```
# Check if all elements are True
# all
logical_test = np.array([False,False,False])
all_true = np.all(logical_test)  #  Check if all elements are True
print("All elements True:",all_true)
```

```
All elements True: False
```

In [37]:

```
# Check if any elements are True
# any
logical_test = np.array([True,False,True])
any_true = np.any(logical_test)  #  Check if any are True
print("Any elements True:",any_true)
```

```
Any elements True: True
```

In [38]:

```
# Intersection of two arrays
set_a = np.array([1,2,3,4])
set_b = ([3,4,5,6])
intersection = np.intersect1d(set_a,set_b)
print('Intersection of a and b:',intersection)
```

```
Intersection of a and b: [3 4]
```

In [39]:

```
# Union of two arrays
union = np.union1d(set_a, set_b)
print('Union of a and b:',union)
```

```
Union of a and b: [1 2 3 4 5 6]
```

In [40]:

```
# Array attributes
a = np.array([1,2,3])
shape = a.shape          # shape of the aarray
size = a.size    # Number of element
dimensions = a.ndim  # Number of dimensions
dtype = a.dtype   # Data type of the array
print('Shape of a:',shape)
print('Size of a:',size)
print('Dimensions of a:',dimensions)
print('Data type of a :',dtype)
```

```
Shape of a: (3,)
Size of a: 3
Dimensions of a: 1
Data type of a : int32
```

In [41]:
```
# Create a copy of an array
a = np.array([1,2,3])
copied_array = np.copy(a) # Create a copy of array a
print('Copied array:',copied_array)
```

```
Copied array: [1 2 3]
```

In [42]:
```
# Size in bytes of an array
array_size_in_bytes = a.nbytes     # size in bytes
print('Size of array in bytes:',array_size_in_bytes)
```

```
Size of array in bytes: 12
```

In [43]:
```
# Check if two array share memory
shared = np.shares_memory(a,copied_array)        # check if  a and copied_array sha
print('Do a and copied_array share memory? :',shared)
```

```
Do a and copied_array share memory? : False
```