NumPy

In [35]:
```
import numpy

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

print(arr)
```
```
[1 2 3 4 5]
```

In [2]:
```
#NumPy package can be referred to as np instead of numpy.
import numpy as np

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

print(arr)
```
```
[1 2 3 4 5]
```

In [3]:
```
import numpy as np

print(np.__version__)
```
```
1.21.5
```

Create a NumPy ndarray Object

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

We can create a NumPy ndarray object by using the array() function.

In [4]:
```
import numpy as np

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

print(arr)

print(type(arr))
```
```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

In [6]:
```
#To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:
# Use a tuple to create a NumPy array:

import numpy as np

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

print(arr)
```
```
[1 2 3 4 5]
```

In [7]:
```
# 0-D Arrays

#0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.import numpy as np

arr = np.array(42)

print(arr)
```
```
42
```

In [8]:
```
#1-D Arrays
import numpy as np

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

print(arr)
```
```
[1 2 3 4 5]
```

In [9]:
```
#2-D Arrays
import numpy as np

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

print(arr)
```
```
[[1 2 3]
 [4 5 6]]
```

In [10]:
```
#3-D arrays
import numpy as np

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

print(arr)
```
```
[[[1 2 3]
  [4 5 6]]

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

In [11]:
```
#Check dimensions the arrays
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```
```
0
1
2
3
```

In [12]:
```
#Create an array with 5 dimensions and verify that it has 5 dimensions:

import numpy as np

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

print(arr)
print('number of dimensions :', arr.ndim)
```
```
[[[[[1 2 3 4]]]]]
number of dimensions : 5
```

Array Indexing

In [14]:
```
import numpy as np

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

print(arr[0])
```
```
1
```

In [15]:
```
import numpy as np

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

print(arr[1])
```
```
2
```

In [16]:
```
import numpy as np

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

print(arr[2] + arr[3])
```
```
7
```

In [45]:
```
#2 Dimensional
import numpy as np

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

print( arr[1,3])
```
```
9
```

In [46]:
```
#To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.
import numpy as np

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

print(arr[0, 1, 2])
print(arr[1, 1, 1])
```
```
6
11
```

In [48]:
```
#Negative Indexing
import numpy as np

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

print('Last element from 2nd dim: ', arr[1, -3])
```
```
Last element from 2nd dim:  8
```

In [22]:
```
#Slicing arrays
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5])
```
```
[2 3 4 5]
```

In [23]:
```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[4:])
```
```
[5 6 7]
```

In [24]:
```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[:4])
```
```
[1 2 3 4]
```

In [49]:
```
#Negative Slicing
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[-4:-2])
```
```
[4 5]
```

In [52]:
```
#Step
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:6:2])
```
```
[2 4 6]
```

In [40]:
```
# Python program to demonstrate basic array characteristics
import numpy as np
# Creating array object
arr = np.array( [[ "a","v"],[ "w","e"]] )  # Printing type of arr object
print("Array is of type: ", type(arr))  # Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)  # Printing shape of array
print("Shape of array: ", arr.shape)
# Printing size (total number of elements) of array
print("Size of array: ", arr.size)
# Printing type of elements in array
print("Array stores elements of type: ", arr.dtype)
```
```
Array is of type:  <class 'numpy.ndarray'>
No. of dimensions:  2
Shape of array:  (2, 2)
Size of array:  4
Array stores elements of type:  <U1
```

In [54]:
```
import numpy as np
a=np.array([1,2,3])
print(a)
a[0]
a[1]
```
```
[1 2 3]
```
Out[54]:
```
2
```

In [53]:
```
a=[1,2,3]
print(a)
a[0]
a[1]
```
```
[1, 2, 3]
```
Out[53]:
```
2
```

DATA TYPE

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

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

print(arr.dtype)
```
```
int32
```

In [4]:
```
import numpy as np

arr = np.array(['apple', 'bananas', 'cherry'])

print(arr.dtype)
```
```
<U7
```

In [5]:
```
#Create an array with data type string: dtype that allows us to define the expected data type of the array elements:

import numpy as np

arr = np.array([1, 2, 3, 4], dtype='S')

print(arr)
print(arr.dtype)
```
```
[b'1' b'2' b'3' b'4']
|S1
```

In [50]:
```
import numpy as np

arr = np.array([1, 2, 3, 4], dtype='i8')

print(arr)
print(arr.dtype)
```
```
[1 2 3 4]
int64
```

In [15]:
```
#Change data type from float to integer by using 'i' as parameter value:

import numpy as np

arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype('i')

print(newarr)
print(newarr.dtype)
```
```
[1 2 3]
int32
```

In [16]:
```
#Make a copy, change the original array, and display both arrays:

import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)
```
```
[42  2  3  4  5]
[1 2 3 4 5]
```

In [17]:
```
#Print the shape of a 2-D array:

import numpy as np

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

print(arr.shape)
```
```
(2, 4)
```

In [18]:
```
#Reshape From 1-D to 2-D
import numpy as np

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

newarr = arr.reshape(4, 3)

print(newarr)
```
```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

In [19]:
```
#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)
```
```
[[[ 1  2]
  [ 3  4]
  [ 5  6]]

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

Array Iterating

In [20]:
```
#Iterate on the elements of the following 1-D array:

import numpy as np

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

for x in arr:
  print(x)
```
```
1
2
3
```

In [21]:
```
#Iterate on the elements of the following 2-D array:

import numpy as np

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

for x in arr:
  print(x)
```
```
[1 2 3]
[4 5 6]
```

In [22]:
```
#Iterate on each scalar element of the 2-D array:

import numpy as np

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

for x in arr:
  for y in x:
    print(y)
```
```
1
2
3
4
5
6
```

In [24]:
```
#Iterate on the elements of the following 3-D array:

import numpy as np

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

for x in arr:
  print(x)
```
```
[[1 2 3]
 [4 5 6]]
[[ 7  8  9]
 [10 11 12]]
```

In [25]:
```
#Iterate down to the scalars:

import numpy as np

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

for x in arr:
  for y in x:
    for z in y:
      print(z)
```
```
1
2
3
4
5
6
7
8
9
10
11
12
```

In [23]:
```
#Iterating Arrays Using nditer()
import numpy as np

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

for x in np.nditer(arr):
  print(x)
```
```
1
2
3
4
5
6
7
8
```

Joining Array

In [26]:
```
#Join two arrays

import numpy as np

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

arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))

print(arr)
```
```
[1 2 3 4 5 6]
```

In [51]:
```
#Join two 2-D arrays along rows (axis=1):

import numpy as np

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

arr2 = np.array([[5, 6], [7, 8]])

arr = np.concatenate((arr1, arr2), axis=0)

print(arr)
```
```
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
```

Joining Arrays Using Stack Functions

In [52]:
```
import numpy as np

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

arr2 = np.array([4, 5, 6])

arr = np.stack((arr1, arr2), axis=0)

print(arr)
```
```
[[1 2 3]
 [4 5 6]]
```

In [29]:
```
#hstack() to stack along rows.


import numpy as np

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

arr2 = np.array([4, 5, 6])

arr = np.hstack((arr1, arr2))

print(arr)
```
```
[1 2 3 4 5 6]
```

In [30]:
```
#vstack()  to stack along columns.


import numpy as np

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

arr2 = np.array([4, 5, 6])

arr = np.vstack((arr1, arr2))

print(arr)
```
```
[[1 2 3]
 [4 5 6]]
```

In [31]:
```
#dstack() to stack along height, which is the same as depth.


import numpy as np

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

arr2 = np.array([4, 5, 6])

arr = np.dstack((arr1, arr2))

print(arr)
```
```
[[[1 4]
  [2 5]
  [3 6]]]
```

Splitting Array

In [32]:
```
#Split the array in 3 parts:

import numpy as np

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

newarr = np.array_split(arr, 3)

print(newarr)
```
```
[array([1, 2]), array([3, 4]), array([5, 6])]
```

In [57]:
```
#Split the array in 4 parts:

import numpy as np

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

newarr = np.array_split(arr,4)

print(newarr)
```
```
[array([1]), array([2]), array([3]), array([], dtype=int32)]
```

In [34]:
```
#Access the splitted arrays:

import numpy as np

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

newarr = np.array_split(arr, 3)

print(newarr[0])
print(newarr[1])
print(newarr[2])
```
```
[1 2]
[3 4]
[5 6]
```

In [35]:
```
#Split the 2-D array into three 2-D arrays.

import numpy as np

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

newarr = np.array_split(arr, 3)

print(newarr)
```
```
[array([[1, 2],
       [3, 4]]), array([[5, 6],
       [7, 8]]), array([[ 9, 10],
       [11, 12]])]
```

In [36]:
```
#Split the 2-D array into three 2-D arrays.

import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])

newarr = np.array_split(arr, 3)

print(newarr)
```
```
[array([[1, 2, 3],
       [4, 5, 6]]), array([[ 7,  8,  9],
       [10, 11, 12]]), array([[13, 14, 15],
       [16, 17, 18]])]
```

In [55]:
```
#Split the 2-D array into three 2-D arrays along rows.

import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])

newarr = np.array_split(arr, 3, axis=1)

print(newarr)
```
```
[array([[ 1],
       [ 4],
       [ 7],
       [10],
       [13],
       [16]]), array([[ 2],
       [ 5],
       [ 8],
       [11],
       [14],
       [17]]), array([[ 3],
       [ 6],
       [ 9],
       [12],
       [15],
       [18]])]
```

In [38]:
```
#Use the hsplit() method to split the 2-D array into three 2-D arrays along rows.

import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])

newarr = np.hsplit(arr, 3)

print(newarr)
```
```
[array([[ 1],
       [ 4],
       [ 7],
       [10],
       [13],
       [16]]), array([[ 2],
       [ 5],
       [ 8],
       [11],
       [14],
       [17]]), array([[ 3],
       [ 6],
       [ 9],
       [12],
       [15],
       [18]])]
```

Searching Arrays

In [39]:
```
#Find the indexes where the value is 4:

import numpy as np

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

x = np.where(arr == 4)

print(x)
```
```
(array([3, 5, 6], dtype=int64),)
```

In [40]:
```
#Find the indexes where the values are even:

import numpy as np

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

x = np.where(arr%2 == 0)

print(x)
```
```
(array([1, 3, 5, 7], dtype=int64),)
```

In [41]:
```
import numpy as np

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

x = np.where(arr%2 == 1)

print(x)
```
```
(array([0, 2, 4, 6], dtype=int64),)
```

In [42]:
```
#searchsorted() which 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.

import numpy as np

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7)

print(x)
```
```
1
```

Sorting Arrays

In [43]:
```
#Sort the array:

import numpy as np

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

print(np.sort(arr))
```
```
[0 1 2 3]
```

In [44]:
```
#Sort the array alphabetically:

import numpy as np

arr = np.array(['banana', 'cherry', 'apple'])

print(np.sort(arr))
```
```
['apple' 'banana' 'cherry']
```

In [45]:
```
#Sort a 2-D array:

import numpy as np

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

print(np.sort(arr))
```
```
[[2 3 4]
 [0 1 5]]
```

Filter Array

In [46]:
```
#filter an array using a boolean index list.

#A boolean index list is a list of booleans corresponding to indexes in the array.
import numpy as np

arr = np.array([41, 42, 43, 44])

x = [True, False, True, False]

newarr = arr[x]

print(newarr)
```
```
[41 43]
```

In [47]:
```
import numpy as np

arr = np.array([41, 42, 43, 44])

# Create an empty list
filter_arr = []

# go through each element in arr
for element in arr:
  # if the element is higher than 42, set the value to True, otherwise False:
  if element > 42:
    filter_arr.append(True)
  else:
    filter_arr.append(False)

newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```
```
[False, False, True, True]
[43 44]
```

In [ ]: