NUMPY TUTORIAL

June 29, 2022

[ ]:
```
#NUMPY TUTORIAL
#it is used to work with arrays
```

[1]:
```
#creating an array
import numpy as np
array = np.array([1,2,3,4,5])
print(array)
```

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

[2]:
```
#using tuple to create an numpy array
import numpy as np
arr=np.array((1,2,3,4,5))
print(arr)
```

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

[3]:
```
#0-D arrays
arr=np.array(42)
print(arr)
```

```
42
```

[4]:
```
#1-D arrays
#an array that has 0-D arrays as its elements is called 1-D array or␣
 ↪unidimenstioanl array
arr=np.array([1,2,3,4,5])
print(arr)
```

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

[5]:
```
#2-D arrays
#An array that has 1-D arrays as its elements is called a 2-D array.
#These are often used to represent matrix or 2nd order tensors.
import numpy as np

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

print(arr)
```

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

[6]:
```
#3-D arrays
#An array that has 2-D arrays (matrices) as its elements is called 3-D array.
#These are often used to represent a 3rd order tensor.
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]]]
```

[7]:
```
#to check the dimenstion of the array
#NumPy Arrays provides the ndim attribute that returns an integer that tells us␣
 ↪how many dimensions the array have.
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
```

[8]:
```
#higher dimension arrays
#An array can have any number of dimensions.
#When the array is created, you can define the number of dimensions by using␣
 ↪the ndmin argument.
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
```

[9]:
```
#Indexing an array
#below are some examples of array indexing
import numpy as np

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

print(arr[0])
```

```
1
```

[10]:
```
import numpy as np

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

print(arr[1])
```

```
2
```

[11]:
```
import numpy as np

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

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

```
7
```

[12]:
```
#accessing 2-D array
import numpy as np

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

print('2nd element on 1st row: ', arr[0, 1])
```

```
2nd element on 1st row:  2
```

[13]:
```
import numpy as np

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

print('5th element on 2nd row: ', arr[1, 4])
```

```
5th element on 2nd row:  10
```

[14]:
```
#accessing 3-D array
import numpy as np

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

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

```
6
```

[15]:
```
#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, -1])
```

```
Last element from 2nd dim:  10
```

[ ]:
```
#ARRAY SLICING
#Slicing in python means taking elements from one given index to another given␣
↪index.
#We pass slice instead of index like this: [start:end].
#We can also define the step, like this: [start:end:step].
#If we don't pass start its considered 0
#If we don't pass end its considered length of array in that dimension
#If we don't pass step its considered 1

#below are some examples
```

[16]:
```
import numpy as np

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

print(arr[4:])
```

```
[5 6 7]
```

[17]:
```
import numpy as np

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

print(arr[:4])
```

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

[18]:
```
#negative slicing
import numpy as np

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

print(arr[-3:-1])
```

```
[5 6]
```

[19]:
```
#Use the step value to determine the step of the slicing:
import numpy as np

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

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

```
[2 4]
```

[20]:
```
import numpy as np

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

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

```
[1 3 5 7]
```

[21]:
```
#slicing 2-D array
import numpy as np

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

print(arr[1, 1:4])
```

```
[7 8 9]
```

[22]:
```
#From both elements, return index 2:
import numpy as np

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

print(arr[0:2, 2])
```

```
[3 8]
```

[23]:
```
#From both elements, slice index 1 to index 4 (not included), this will return␣
↪a 2-D array:
import numpy as np

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

print(arr[0:2, 1:4])
```

```
[[2 3 4]
 [7 8 9]]
```

[ ]:
```
#numpy data types
#strings - used to represent text data, the text is given under quote marks. e.
↪g. "ABCD"
#integer - used to represent integer numbers. e.g. -1, -2, -3
```

```
#float - used to represent real numbers. e.g. 1.2, 42.42
#boolean - used to represent True or False.
#complex - used to represent complex numbers. e.g. 1.0 + 2.0j, 1.5 + 2.5j

#Below is a list of all data types in NumPy and the characters used to␣
 ↪represent them.

#i - integer
#b - boolean
#u - unsigned integer
#f - float
#c - complex float
#m - timedelta
#M - datetime
#O - object
#S - string
#U - unicode string
#V - fixed chunk of memory for other type ( void )
```

[24]:
```
#The NumPy array object has a property called dtype that returns the data type␣
 ↪of the array:
import numpy as np

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

print(arr.dtype)
```

```
int32
```

[25]:
```
import numpy as np

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

print(arr.dtype)
```

```
<U6
```

[26]:
```
#Create an array with data type string:
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
```

[27]:
```
#Create an array with data type 4 bytes integer:
import numpy as np

arr = np.array([1, 2, 3, 4], dtype='i4')

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

```
[1 2 3 4]
int32
```

[ ]:
```
#difference between copy and veiw
#The main difference between a copy and a view of an array is that the copy is␣
 ↪a new array, and the view is just a view of the original array.
#The copy owns the data and any changes made to the copy will not affect␣
 ↪original array, and any changes made to the original array will not affect␣
 ↪the copy.
#The view does not own the data and any changes made to the view will affect␣
 ↪the original array, and any changes made to the original array will affect␣
 ↪the view.
```

[28]:
```
#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]
```

[29]:
```
#Make a view, change the original array, and display both arrays:
import numpy as np

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

print(arr)
print(x)
```

```
[42  2  3  4  5]
[42  2  3  4  5]
```

[30]:
```
#making changes in the veiw
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
x[0] = 31

print(arr)
print(x)
```

```
[31  2  3  4  5]
[31  2  3  4  5]
```

[ ]:
```
#shaping of an array
#below are some few examples
#The shape of an array is the number of elements in each dimension.
```

[31]:
```
import numpy as np

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

print(arr.shape)
```

```
(2, 4)
```

[ ]:
```
#reshaping an array
```

[32]:
```
#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]]
```

[33]:
```
#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
```

[34]:
```
#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
```

[35]:
```
#Iterating 2-D Arrays
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]
```

[36]:
```
#we can iterate the elements on 2-D array by below code
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
```

[37]:
```
#Iterating 3-D Arrays
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]]
```

[38]:
```
# we can also iterate 3-D arrays by below code
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
```

[ ]:
```
#joining arrays
```

[39]:
```
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]
```

[42]:
```
import numpy as np

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

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

arr = np.concatenate((arr1, arr2), axis=1)

print(arr)
```

```
[[1 2 5 6]
 [3 4 7 8]]
```

[43]:
```
import numpy as np

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

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

arr = np.stack((arr1, arr2), axis=1)

print(arr)
```

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

[44]:
```
#stacking along rows
#NumPy provides a helper function: 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]
```

[45]:
```
#stacking along columns
#NumPy provides a helper function: 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]]
```

[46]:
```
#stacking along height(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]]]
```

[47]:
```
#splitting an array
#We use array_split() for splitting arrays, we pass it the array we want to␣
 ↪split and the number of splits.
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])]
```

[48]:
```
import numpy as np

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

newarr = np.array_split(arr, 4)

print(newarr)
```

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

[49]:
```
#splitting 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]])]
```

[50]:
```
#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]])]
```

[51]:
```
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]])]
```

[52]:
```
#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
```

[53]:
```
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),)
```

[54]:
```
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),)
```

[55]:
```
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),)
```

[ ]:
```
#SORTING ARRAYS
```

[ ]:
```
#Sorting means putting elements in an ordered sequence.

#Ordered sequence is any sequence that has an order corresponding to elements,␣
↪like numeric or alphabetical, ascending or descending.

#The NumPy ndarray object has a function called sort(), that will sort a␣
↪specified array.
```

[56]:
```
import numpy as np

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

print(np.sort(arr))
```

```
[0 1 2 3]
```

[57]:
```
import numpy as np

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

print(np.sort(arr))
```

```
['apple' 'banana' 'cherry']
```

[58]:
```
import numpy as np

arr = np.array([True, False, True])

print(np.sort(arr))
```

```
[False  True  True]
```

[59]:
```
#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]]
```

[ ]:
```
#NUMPY RANDOM
#NumPy offers the random module to work with random numbers.
#Generate Random Number
```

[60]:
```
from numpy import random

x = random.randint(100)

print(x)
```

```
87
```

[61]:
```
#The random module's rand() method returns a random float between 0 and 1.
from numpy import random

x = random.rand()

print(x)
```

```
0.9971613026716966
```

[ ]:
```
#Generate Random Array
#In NumPy we work with arrays, and you can use the two methods from the above␣
 ↪examples to make random arrays.
```

[62]:
```
#The randint() method takes a size parameter where you can specify the shape of␣
 ↪an array.
```

[63]:
```
from numpy import random

x=random.randint(100, size=(5))

print(x)
```

```
[33 41 72 19 73]
```

[64]:
```
from numpy import random

x = random.randint(100, size=(3, 5))
```

```
print(x)
```

```
[[23 68 76 10  9]
 [79 78 55 83 94]
 [24 41 12  9 41]]
```

[65]:
```
#The rand() method also allows you to specify the shape of the array.
from numpy import random

x = random.rand(5)

print(x)
```

```
[0.25369794 0.79033196 0.59608854 0.60801162 0.05081223]
```

[66]:
```
from numpy import random

x = random.rand(3, 5)

print(x)
```

```
[[0.40939038 0.20554723 0.5474056  0.49005401 0.31722767]
 [0.92380807 0.26175113 0.25582555 0.09792446 0.82498997]
 [0.00939874 0.84695531 0.34342003 0.81383544 0.92954289]]
```

[ ]:
```
#Generate Random Number From Array
#The choice() method allows you to generate a random value based on an array of␣
 ↪values.

#The choice() method takes an array as a parameter and randomly returns one of␣
 ↪the values.
```

[67]:
```
from numpy import random

x = random.choice([3, 5, 7, 9])

print(x)
```

```
5
```

[68]:
```
from numpy import random

x = random.choice([3, 5, 7, 9], size=(3, 5))

print(x)
```

```
[[7 7 7 3 9]
 [3 3 3 7 9]
 [9 3 9 9 5]]
```

[ ]: