chap2 - part2

June 16, 2022

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

[5]:
```
# create numpy array using arange() function
var1 = np.arange(1, 11, dtype='f')
```

[6]:
```
print(var1)
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
```

[7]:
```
print(np.arange(1, 6, dtype='D'))
```

```
[1.+0.j 2.+0.j 3.+0.j 4.+0.j 5.+0.j]
```

[8]:
```
# dtype constructors
print(np.dtype(float))
```

```
float64
```

[9]:
```
print(np.dtype('f'))
```

```
float32
```

[10]:
```
print(np.dtype('d'))
```

```
float64
```

[11]:
```
print(np.dtype('f8'))
```

```
float64
```

[12]:
```
# dtype attributes
# create numpy array
var2 = np.array([1, 2, 3], dtype = 'float64')
```

[13]:
```
print(var2.dtype.char)
```

```
d
```

[14]:
```
print(var2.dtype.type)
```

```
<class 'numpy.float64'>
```

[15]:

```python
# manipulating array shapes

# func. reshape()

# create an array
arr = np.arange(12)
```

[16]:

```python
print(arr)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

[17]:

```python
# reshape the array dimension
new_arr = arr.reshape(4,3)
```

[18]:

```python
print(new_arr)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

[19]:

```python
# reshape the array dimension
new_arr2 = arr.reshape(3,4)
```

[20]:

```python
print(new_arr2)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

[21]:

```python
# func. flattern - transform n-dimensional array into a
# one-dimentional array

# create an array
arr = np.arange(1,10).reshape(3,3)
```

[22]:

```python
print(arr)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

[23]:

```python
# flattern an array
print(arr.flatten())
```

```
[1 2 3 4 5 6 7 8 9]
```

[24]:

```
# func. transpose() - converting rows into columns
# and columns into rows in the matrix
print(arr.transpose())
```

```
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

[25]:

```
# the stacking of numPy arrays
# stacking - joining the same dimensional arrays along with
# a new axis
arr1 = np.arange(1,10).reshape(3,3)
```

[26]:

```
print(arr1)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

[27]:

```
arr2 = 2*arr1
```

[28]:

```
print(arr2)
```

```
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
```

[29]:

```
# horizontal stacking (axis x)
arr3 = np.hstack((arr1, arr2))
```

[30]:

```
print(arr3)
```

```
[[ 1  2  3  2  4  6]
 [ 4  5  6  8 10 12]
 [ 7  8  9 14 16 18]]
```

[31]:

```
# vertical stacking - joing along the same dimensional
# arrays vertically (axis y)
arr5 = np.vstack((arr1, arr2))
```

[32]:

```
print(arr5)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
```

[33]:

```
# depth stacking - the same dimensional arrays
# are joined along with a third axis (depth) using
# the dstack() function.
arr7 = np.dstack((arr1, arr2))
```

[34]:

```
print(arr7)
```

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

 [[ 4  8]
  [ 5 10]
  [ 6 12]]

 [[ 7 14]
  [ 8 16]
  [ 9 18]]]
```

[35]:

```
# column stacking - stack multiple sequence one-dimensional
# arrays as columns into a single two-dimensional array.

# create 1-D array
arr1 = np.arange(4,7)
```

[36]:

```
print(arr1)
```

```
[4 5 6]
```

[37]:

```
arr2 = arr1 * 2
```

[38]:

```
print(arr2)
```

```
[ 8 10 12]
```

[40]:

```
# create column stack
arr_col_stack = np.column_stack((arr1, arr2))
```

[41]:

```
print(arr_col_stack)
```

```
[[ 4  8]
 [ 5 10]
 [ 6 12]]
```

[42]:

```
# row stacking - stacks multiple sequence one-dimensional
# arrays as rows into a single two-dimensional arrays
# create row stack
arr_row_stack = np.row_stack((arr1, arr2))
```

[43]:

```
print(arr_row_stack)
```

```
[[ 4  5  6]
 [ 8 10 12]]
```

[44]:

```
# partitioning numPy arrays
# horizontal splitting - in horizontal split, the given array is divided
# into N equal sub-arrays along the horizonatal axis using the hsplit()

# create an array
arr = np.arange(1,10).reshape(3,3)
```

[45]:

```
print(arr)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

[46]:

```
arr_hor_split = np.hsplit(arr,3)
```

[47]:

```
print(arr_hor_split)
```

```
[array([[1],
       [4],
       [7]]), array([[2],
       [5],
       [8]]), array([[3],
       [6],
       [9]])]
```

[48]:

```
# vertical split, divide into N equal subarrays along the vertical axis
# usign vsplit()

# vertical split
arr_ver_split = np.vsplit(arr,3)
```

[49]:

```
print(arr_ver_split)
```

```
[array([[1, 2, 3]]), array([[4, 5, 6]]), array([[7, 8, 9]])]
```

[50]:

```
# changing the data type of NumPy arrays
# create an array
arr = np.arange(1,10).reshape(3,3)
```

[51]:

```
print("Integer Array: ", arr)
```

```
Integer Array:  [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

[52]:

```
# change datatype of array
arr = arr.astype(float)
```

[53]:

```
# print array
print("Float Array: ", arr)
```

```
Float Array:  [[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```

[54]:

```
# check new data type of array
print("Changed Datatype: ", arr.dtype)
```

```
Changed Datatype:  float64
```

[55]:

```
# tolist() - function converts a NumPy array into a Python list

# create an array
arr = np.arange(1, 10)
```

[56]:

```
# convert NumPy array to Python list
list1 = arr.tolist()
```

[57]:

```
print(list1)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[58]:

```
# creating NumPy view and copies
# create an array
arr = np.arange(1, 5).reshape(2,2)
```

[59]:

```
print(arr)
```

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

[60]:

```
# create no copy only assignment
arr_no_copy = arr
```

[61]:

```
# create deep copy
arr_copy = arr.copy()
```

[62]:

```
# create shallow copy using View
arr_view = arr.view()
```

[65]:
```
print("Orginal Array     : ", id(arr))
print("Assignment        : ", id(arr_no_copy))
print("Deep Copy         : ", id(arr_copy))
print("Shallow Copy(View): ", id(arr_view))
```

```
Orginal Array     :  1941841173168
Assignment        :  1941841173168
Deep Copy         :  1941841172976
Shallow Copy(View):  1941850013296
```

[66]:
```
# update the value of original array
arr[1] = [98,99]
```

[67]:
```
# check values of array view
print("View Array:\n", arr_view)
```

```
View Array:
 [[ 1  2]
 [98 99]]
```

[68]:
```
# check values of array copy
print("Copied Array:\n", arr_copy)
```

```
Copied Array:
 [[1 2]
 [3 4]]
```

[69]:
```
# slicing arrays
arr = np.arange(0, 10)
```

[70]:
```
print(arr)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

[71]:
```
print(arr[3:6])
```

```
[3 4 5]
```

[72]:
```
print(arr[3:])
```

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

[73]:
```
print(arr[-3:])
```

```
[7 8 9]
```

[74]:
```
print(arr[2:7:2])
```

```
[2 4 6]
```

[75]:

```
# boolean and fancy indexing
arr = np.arange(21,41,2)
print("Orginal Array:\n", arr)

# boolean indexing
print("After boolean condition: ", arr[arr>30])
```

```
Orginal Array:
 [21 23 25 27 29 31 33 35 37 39]
After boolean condition:  [31 33 35 37 39]
```

[76]:

```
arr = np.arange(1,21).reshape(5,4)
print("Orginal Array:\n", arr)

# selecting 2nd and 3rd row
indices = [1,2]
print("Selected 1st and 2nd row:\n", arr[indices])

# selecting 3rd and 4th row
indices = [2,3]
print("Selected 2nd and 3rd row:\n", arr[indices])
```

```
Orginal Array:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]]
Selected 1st and 2nd row:
 [[ 5  6  7  8]
 [ 9 10 11 12]]
Selected 2nd and 3rd row:
 [[ 9 10 11 12]
 [13 14 15 16]]
```

[80]:

```
# broadcasting arrays
arr1 = np.arange(1,5).reshape(2,2)
print("arr1 =\n", arr1)

arr2 = np.arange(5,9).reshape(2,2)
print("arr2 = \n", arr2)

# add two matrices
print("arr1 + arr2 = \n", arr1+arr2)

# multiply two matrices
print("arr1 + arr2 = \n", arr1*arr2)
```

```python
# add a scalar value
print("arr1 + 3 = \n", arr1+3)

#multiply with a scalar value
print("arr1 * 3 = \n", arr1*3)
```

```
arr1 =
 [[1 2]
 [3 4]]
arr2 =
 [[5 6]
 [7 8]]
arr1 + arr2 =
 [[ 6  8]
 [10 12]]
arr1 + arr2 =
 [[ 5 12]
 [21 32]]
arr1 + 3 =
 [[4 5]
 [6 7]]
arr1 * 3 =
 [[ 3  6]
 [ 9 12]]
```

[ ]: