Assignment For Numpy

Difficulty Level **Beginner**

1. Import the numpy package under the name np

In [1]:
```
import numpy as np
```

1. Create a null vector of size 10

In [2]:
```
arr1 = np.zeros(10)
arr1
```

Out[2]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

1. Create a vector with values ranging from 10 to 49

In [3]:
```
arr2 = np.arange(10, 50)
arr2
```

Out[3]:
```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
       27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43,
       44, 45, 46, 47, 48, 49])
```

1. Find the shape of previous array in question 3

In [4]:
```
arr2.shape
```

Out[4]:
```
(40,)
```

1. Print the type of the previous array in question 3

In [5]:
```
arr2.dtype
```

Out[5]:
```
dtype('int32')
```

1. Print the numpy version and the configuration

In [6]:
```
print('Numpy Version:', np.__version__, end='\n\n')
np.show_config()
```

```
Numpy Version: 1.19.5

blas_mkl_info:
  NOT AVAILABLE
blis_info:
  NOT AVAILABLE
openblas_info:
    library_dirs = ['D:\\a\\1\\s\\numpy\\build\\openblas_info']
    libraries = ['openblas_info']
    language = f77
```

```
    define_macros = [('HAVE_CBLAS', None)]
blas_opt_info:
    library_dirs = ['D:\\a\\1\\s\\numpy\\build\\openblas_info']
    libraries = ['openblas_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None)]
lapack_mkl_info:
  NOT AVAILABLE
openblas_lapack_info:
    library_dirs = ['D:\\a\\1\\s\\numpy\\build\\openblas_lapack_info']
    libraries = ['openblas_lapack_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None)]
lapack_opt_info:
    library_dirs = ['D:\\a\\1\\s\\numpy\\build\\openblas_lapack_info']
    libraries = ['openblas_lapack_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None)]
```

1. Print the dimension of the array in question 3

In [7]:
```
arr2.ndim
```

Out[7]:
```
1
```

1. Create a boolean array with all the True values

In [8]:
```
arr3 = np.full(10, True, dtype='bool')
arr3
```

Out[8]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True])
```

1. Create a two dimensional array

In [9]:
```
arr4 = np.arange(10).reshape(2, 5)
arr4
```

Out[9]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

1. Create a three dimensional array

In [10]:
```
arr5 = np.arange(1, 10).reshape(3, 3)
arr5
```

Out[10]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Difficulty Level **Easy**

1. Reverse a vector (first element becomes last)

In [11]:
```
arr6 = np.flip(arr2)
arr6
```

Out[11]:
```
array([49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33,
```

```
       32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16,
       15, 14, 13, 12, 11, 10])
```

1. Create a null vector of size 10 but the fifth value which is 1

In [12]:
```
arr7 = np.where(np.arange(10) == 4, 1, 0)
arr7
```

Out[12]:
```
array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0])
```

1. Create a 3x3 identity matrix

In [13]:
```
arr8 = np.eye(3, 3)
arr8
```

Out[13]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

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

Convert the data type of the given array from int to float

In [14]:
```
arr9 = np.array([1, 2, 3, 4, 5])
print(arr9, 'type =', arr9.dtype)
arr10 = np.asfarray(arr9)
print(arr10, 'type =', arr10.dtype)
```

```
[1 2 3 4 5] type = int32
[1. 2. 3. 4. 5.] type = float64
```

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

                    [4., 5., 6.]])

   arr2 = np.array([[0., 4., 1.],

                    [7., 2., 12.]])

Multiply arr1 with arr2

In [15]:
```
arr11 = np.array([[1, 2, 3], [4, 5, 6]])
arr12 = np.array([[0, 4, 1], [7, 2, 12]])
arr13 = arr11 * arr12
arr13
```

Out[15]:
```
array([[ 0,  8,  3],
       [28, 10, 72]])
```

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

                    [4., 5., 6.]])

   arr2 = np.array([[0., 4., 1.],

```
       [7., 2., 12.]])
```

Make an array by comparing both the arrays provided above

In [16]:
```
arr14 = np.array([1, 2, 3, 4, 5, 6])
arr15 = np.array([0, 4, 1, 7, 2, 12])
arr16 = arr14 > arr15
arr16
```

Out[16]:
```
array([ True, False,  True, False,  True, False])
```

1. Extract all odd numbers from arr with values(0-9)

In [17]:
```
arr17 = np.arange(10)
print(arr17)
arr18 = arr17[arr17 % 2 == 1]
print(arr18)
```

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

1. Replace all odd numbers to -1 from previous array

In [18]:
```
arr19 = np.where(arr17 % 2 == 1, -1, arr17)
arr19
```

Out[18]:
```
array([ 0, -1,  2, -1,  4, -1,  6, -1,  8, -1])
```

1. arr = np.arange(10)

Replace the values of indexes 5,6,7 and 8 to **12**

In [19]:
```
arr20 = np.arange(10)
print(arr20)
arr20[5:9] = 12
print(arr20)
```

```
[0 1 2 3 4 5 6 7 8 9]
[ 0  1  2  3  4 12 12 12 12  9]
```

1. Create a 2d array with 1 on the border and 0 inside

In [20]:
```
arr21 = np.ones((5, 5))
print('Before', arr21, sep='\n')
arr21[1:-1, 1:-1] = 0
print('After', arr21, sep='\n')
```

```
Before
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
After
[[1. 1. 1. 1. 1.]
```

```
[1. 0. 0. 0. 1.]
[1. 0. 0. 0. 1.]
[1. 0. 0. 0. 1.]
[1. 1. 1. 1. 1.]]
```

Difficulty Level **Medium**

1. arr2d = np.array([[1, 2, 3],

                    [4, 5, 6],

                    [7, 8, 9]])

Replace the value 5 to 12

In [21]:
```
arr22 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print('Before', arr22, sep='\n')
arr22[1, 1] = 12
print('After', arr22, sep='\n')
```

```
Before
[[1 2 3]
 [4 5 6]
 [7 8 9]]
After
[[ 1  2  3]
 [ 4 12  6]
 [ 7  8  9]]
```

1. arr3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

Convert all the values of 1st array to 64

In [22]:
```
arr23 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print('Before', arr23, sep='\n')
arr23[0, 0] = 64
print('After', arr23, sep='\n')
```

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

 [[ 7  8  9]
  [10 11 12]]]
After
[[[64 64 64]
  [ 4  5  6]]

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

1. Make a 2-Dimensional array with values 0-9 and slice out the first 1st 1-D array from it

In [23]:
```
arr24 = np.arange(10).reshape(2, 5)
print(arr24)
print('First 1D Array', arr24[0])
```

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

1. Make a 2-Dimensional array with values 0-9 and slice out the 2nd value from 2nd 1-D array from it

In [24]:
```
arr25 = np.arange(10).reshape(2, 5)
print(arr25)
print('2nd value from 2nd 1D array:', arr25[1, 1])
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
2nd value from 2nd 1D array: 6
```

1. Make a 2-Dimensional array with values 0-9 and slice out the third column but only the first two rows

In [25]:
```
arr26 = np.arange(9).reshape(3, 3)
print(arr26)
print('Slice:', arr26[:2, 2])
```

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

1. Create a 10x10 array with random values and find the minimum and maximum values

In [26]:
```
arr27 = np.random.randn(10,10)
print(arr27)
print('Minimum Value:', arr27.min())
print('Maximum Value:', arr27.max())
```

```
[[-0.95263734  1.6007148  -0.06988234 -0.92275101  0.7993354  -0.50957048
   2.54546706 -0.24716168  0.59483232 -0.56808041]
 [-0.45437304 -0.78480446  1.74024439 -0.53006016 -2.89346524  0.35097159
   1.14800729  0.60975057 -0.78536731  1.80892948]
 [ 0.41072401  0.56138686 -0.34020573 -1.90209198  0.51137102  0.12473482
  -0.01789415  0.33554477 -0.90156792 -1.12294063]
 [ 0.43014471 -2.66360032  0.88013567 -1.65297448  1.14035119  0.52256125
  -1.20649228  0.16914957 -0.13499559  0.88657965]
 [ 1.39816974  0.6916734  -1.41593505  0.44470149  1.3358983  -1.54820316
   0.51518744  0.02828644  0.85862676  1.67384272]
 [-1.43023309  0.68510316  1.37507145 -0.74859669 -0.14836743  0.60114566
  -0.23287043  0.08963687 -0.12554224  0.25929805]
 [ 1.41290639  1.49220089  1.78039451  0.10904111 -1.23210962  1.38457151
  -0.76894164 -0.08070125  1.19504872 -0.7316895 ]
 [-0.30102586 -0.96517628 -1.56384139 -1.11895237  0.05561409  1.15828558
   1.21528655 -0.66906386 -0.23896497 -0.46147789]
 [-2.00376385  0.20504316 -0.8409214  -1.6492532  -0.5309332  -2.34973897
  -0.18166433 -0.13221587  0.96257307 -1.77182949]
 [ 0.34049743  0.15651489  0.59856371  0.17024237 -1.56432713  0.08015083
  -0.47450681 -1.1160994  -2.78960486  1.36118751]]
Minimum Value: -2.8934652397715355
Maximum Value: 2.545467055784386
```

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

Find the common items between a and b

In [27]:
```
arr28 = np.array([1, 2, 3, 2, 3, 4, 3, 4, 5, 6])
arr29 = np.array([7, 2, 10, 2, 7, 4, 9, 4, 9, 8])
```

```
np.intersect1d(arr28, arr29)
```

Out[27]:
```
array([2, 4])
```

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

Find the positions where elements of a and b match

In [28]:
```
arr30 = np.subtract(arr28, arr29)
np.arange(arr30.size)[arr30 == 0]
```

Out[28]:
```
array([1, 3, 5, 7])
```

1. names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe']) data = np.random.randn(7, 4)

Find all the values from array **data** where the values from array **names** are not equal to **Will**

In [29]:
```
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
data = np.random.randn(7, 4)
data[names != 'Will']
```

Out[29]:
```
array([[ 1.27151051,  0.7043036 , -0.0433803 , -0.2215327 ],
       [-1.51223184,  0.50555664,  0.64047469, -0.23664498],
       [ 0.0425699 ,  2.52939071,  1.08149419, -1.41954942],
       [-1.72337487, -0.09923194,  0.62096248,  2.38558495],
       [-0.15380026, -0.94030792,  0.63313374,  0.54840199]])
```

1. names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe']) data = np.random.randn(7, 4)

Find all the values from array **data** where the values from array **names** are not equal to **Will** and **Joe**

In [30]:
```
data[(names != 'Will') & (names != 'Joe')]
```

Out[30]:
```
array([[ 1.27151051,  0.7043036 , -0.0433803 , -0.2215327 ],
       [ 0.0425699 ,  2.52939071,  1.08149419, -1.41954942]])
```

Difficulty Level **Hard**

1. Create a 2D array of shape 5x3 to contain decimal numbers between 1 and 15.

In [31]:
```
np.random.uniform(1, 15, size=(5,3))
```

Out[31]:
```
array([[11.94161708, 14.65742707,  3.36635493],
       [12.31073668,  9.02648262, 14.77470225],
       [ 7.58904901, 10.60889899,  2.48938723],
       [14.02858329, 14.65921699,  5.61266061],
       [ 3.47871612,  7.97521308, 14.56219294]])
```

1. Create an array of shape (2, 2, 4) with decimal numbers between 1 to 16.

In [32]:
```
arr32 = np.random.uniform(1, 16, (2, 2, 4))
```

```
arr32
```

Out[32]:
```
array([[[ 6.60180683,  8.42736298,  1.18214799,  1.6590384 ],
        [14.64924421, 12.40768037,  1.88266384, 11.70697176]],

       [[ 4.61706622, 15.09181781,  5.98416138, 12.62314883],
        [ 1.58964554,  9.2542423 ,  6.88882232, 11.8366818 ]]])
```

1. Swap axes of the array you created in Question 32

In [33]:
```
arr32.swapaxes(0,2)
```

Out[33]:
```
array([[[ 6.60180683,  4.61706622],
        [14.64924421,  1.58964554]],

       [[ 8.42736298, 15.09181781],
        [12.40768037,  9.2542423 ]],

       [[ 1.18214799,  5.98416138],
        [ 1.88266384,  6.88882232]],

       [[ 1.6590384 , 12.62314883],
        [11.70697176, 11.8366818 ]]])
```

1. Create an array of size 10, and find the square root of every element in the array, if the values less than 0.5, replace them with 0

In [34]:
```
arr33 = np.random.uniform(0, 5, 10)
print('Array:', arr33, sep='\n')
arr34 = arr33 ** 0.5
print('After Square Root:', arr34, sep='\n')
arr35 = np.where(arr34 < 0.5, 0, arr34)
print('After Replacing:', arr35, sep='\n')
```

```
Array:
[4.04349002 4.09612912 4.20622544 0.07784413 0.65352659 4.46423709
 3.39946965 4.50541728 2.2894518  1.64745529]
After Square Root:
[2.01084311 2.0238896  2.05090844 0.2790056  0.80840992 2.11287413
 1.84376508 2.12259683 1.51309345 1.28353235]
After Replacing:
[2.01084311 2.0238896  2.05090844 0.         0.80840992 2.11287413
 1.84376508 2.12259683 1.51309345 1.28353235]
```

1. Create two random arrays of range 12 and make an array with the maximum values between each element of the two arrays

In [35]:
```
arr36 = np.random.randint(10, size=12)
print(arr36)
arr37 = np.random.randint(10, size=12)
print(arr37)
np.maximum(arr36, arr37)
```

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

Out[35]:
```
array([9, 9, 5, 6, 6, 8, 7, 1, 8, 7, 4, 5])
```

1. names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])

Find the unique names and sort them out!

In [36]:
```
np.unique(names)
```

Out[36]:
```
array(['Bob', 'Joe', 'Will'], dtype='<U4')
```

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

From array a remove all items present in array b

In [37]:
```
a = np.array([1, 2, 3, 4, 5])
b = np.array([5, 6, 7, 8, 9])
np.setdiff1d(a, b)
```

Out[37]:
```
array([1, 2, 3, 4])
```

1. Following is the input NumPy array delete column two and insert following new column in its place.

---

sampleArray = numpy.array([[34,43,73],[82,22,12],[53,94,66]])

---

newColumn = numpy.array([[10,10,10]])

In [38]:
```
sampleArray = np.array([[34,43,73],[82,22,12],[53,94,66]])
newColumn = np.array([10,10,10])
sampleArray[:, 1] = newColumn
sampleArray
```

Out[38]:
```
array([[34, 10, 73],
       [82, 10, 12],
       [53, 10, 66]])
```

1. x = np.array([[1., 2., 3.], [4., 5., 6.]]) y = np.array([[6., 23.], [-1, 7], [8, 9]])

Find the dot product of the above two matrix

In [39]:
```
x = np.array([[1., 2., 3.], [4., 5., 6.]])
y = np.array([[6., 23.], [-1, 7], [8, 9]])
np.dot(x, y)
```

Out[39]:
```
array([[ 28.,  64.],
       [ 67., 181.]])
```

1. Generate a matrix of 20 random values and find its cumulative sum

In [40]:
```
arr40 = np.arange(20)
print(arr40)
np.cumsum(arr40)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

Out[40]:
```
array([  0,   1,   3,   6,  10,  15,  21,  28,  36,  45,  55,  66,  78,
```

```
91, 105, 120, 136, 153, 171, 190], dtype=int32)
```