Numpy Part 2

```
# ============================================================
# == create array from the built-in functions on the Numpy ==
# ============================================================

# zeros((roes, cols))
# create an array with 0's of elements
# default data type is: float64
x = np.zeros((3,4))
print(x)


'''
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
'''
# get the type of array
print(x.dtype)      # float64

# ones((rows, cols))
# create an array of 1's elements
# default data type is: float64
x = np.ones((3, 4), dtype="int")
print(x)
'''
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
'''
# get the type of the array
print(x.dtype)      # int64


# full((rows, cols), fill_value)
x = np.full((3, 3), 10)
print(x)
'''
[10 10 10]
 [10 10 10]
 [10 10 10]]
'''

# eye(shape)
# create identity matrix
# create a 2-D array with ones on the diagonal and zeros elsewhere.
# this is very important matrix in linear algebra
x = np.eye(3)
print(x)
'''
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
'''
```

```python
# diag()
# creates a new ndarray with the given 1D array as its diagonal elements
# diagonal matrix is always squar matrix
x = np.diag([1, 2, 3])
print(x)
'''
[[1 0 0]
 [0 2 0]
 [0 0 3]]
'''

# arange(start, stop, step)
# create an array of sequence of elements that take start and stop numbers
# the start element can be discard, stop is not included
x = np.arange(0, 11)
print(x)        # [ 0  1  2  3  4  5  6  7  8  9 10]

# start can be discard
# end is not included
x = np.arange(5)
print(x)    # [0 1 2 3 4]

x = np.arange(0, 11, 2)
print(x)        # [ 0  2  4  6  8 10]


# linspace(start, stop, length, float_numbers)
# length: refers to the specified interval
# means linear space, used to make
# ● default end is included
x = np.linspace(0, 10, 5, dtype=int)
print(x)        # [ 0  2  5  7 10]

# determine the number of elements
x = np.linspace(0, 100, num=5, dtype=int)
print(x)        # [  0  25  50  75 100]

# print the step number in the linspace() method
x = np.linspace(0, 100, 6, retstep=True)
print(x)    # (array([  0.,  20.,  40.,  60.,  80., 100.]), np.float64(20.0))

# can make the stop element not included
x = np.linspace(0, 25, 5, endpoint=False)
print(x)        # [ 0.  5. 10. 15. 20.]

# can make the stop element included
x = np.linspace(0, 25, 5, endpoint=True)
print(x)        # [ 0.    6.25 12.5  18.75 25.  ]

# also can determine the number of floating point numbers
# bust must make the "retstep = True"
x = np.linspace(0, 10, num=4, retstep=True)
print(x)    # (array([ 0. ,  3.33333333,  6.66666667, 10.]),
np.float64(3.3333333333333335))
```

```
# linspace can take negative number
x = np.linspace(-2, 2, 5)
print(x)        # [-2. -1.  0.  1.  2.]

# generating random floats from 0 to 1 from a "uniform dirtibution"
# array elements are interchangeable
# syntax rand(rows, cols)   --> this is the shape of array
x = np.random.rand(3, 3)
print(x)


'''
[[0.42785013 0.34672848 0.30430415]
 [0.05043191 0.74574375 0.1959387 ]
 [0.40477965 0.21277481 0.78565254]]
'''

# array elements are interchangeable
x = np.random.rand(2, 2)
print(x)

'''
[[0.41239869 0.01741946]
 [0.43183886 0.0638892 ]]
'''

# generate random floats from 0 to 5 from a uniform distribution
# syntx: uniform(start, stop, size=(rows, cols))
x = np.random.uniform(0, 5, size=(3, 3))
print(x)
'''
[[3.17474891 1.19079224 3.11683978]
 [4.93504505 0.24832204 3.90031708]
 [3.87192876 1.19182176 1.17820186]]
'''

# generate random floats from 0 to 1
# random(rows, cols)
x = np.random.random((2, 2))
print(x)
'''
[[0.20973168 0.00460969]
 [0.44444258 0.01245372]]
'''


# generate random integer numbers
# syntax: randint(start, stop, (rows, cols))
x = np.random.randint(0, 5, (3, 3))
print(x)
'''
[[3 2 1]
 [3 1 1]
 [4 0 4]]
'''
```

```python
# generate only one integer number
# syntax: randint(start, stop)
x = np.random.randint(1, 3)
print(x)        # 1

# generate 1D array
# syntax: randint((),)
x = np.random.randint((1, 3))
print(x)        # [0 1]
print(f'number of dimensions: {x.ndim}')       # number of dimensions: 1


# generate only 1 random integer number
# syntax: randint(end)
x = np.random.randint(10)
print(x)        # 5

# generate random integer number
# syntax: randint(start, stop)
x = np.random.randint(5, 10)
print(x)        # 5


# generate 1D random integer numbers
# this contains 2 elements
# syntax: randint
x = np.random.randint((5, 10))
print(x)        # [0 5]

# generate 2D array with random integer numbers
# syntax: randint(start, stop, size=(rows, cols))
# size: refers to the shape of the array s
x = np.random.randint(0, 5, size=(3, 3))
print(x)
''''
[[1 2 2]
 [3 1 3]
 [0 0 2]]
'''

# generate random integer numbers with a range
# syntax: randint(start, stop, size)
# stop is not included
# size: this is the shape of the array
x = np.random.randint(0, 5, 10)
print(x)        # [0 2 0 1 0 1 3 3 2 4]


# get the normal data distribution
# syntax: normal(Mean, Standard_Deviation)
x = np.random.normal(0, 5, size=(3, 3))
print(x)
'''
[[-0.27628177  1.76609725  2.73169248]
 [-1.21602161  4.21737154  4.80921188]
 [-3.752037   -2.34669928 -3.27788727]]
'''
```

```python
# get the standard normal distribution
# syntax: randn(rows, cols) --> this is the shape
x = np.random.randn(3, 3)
print(x)


# select random element from array
# syntax: choice(array, size)
# size: refers to the shape
array = np.arange(10)
x = np.random.choice(array, size=3)
print(x)        # [9 0 9]

# select only one element if not determine
# syntax: choice(array)
x = np.random.choice(array)
print(x)        # 5


array = np.arange(10)
print(array)        # [0 1 2 3 4 5 6 7 8 9]

# creating a random permutation of an array
# the numbers can be interchangable
permutation_array = np.random.permutation(array)
print(permutation_array)        # [2 9 7 6 5 1 4 3 8 0]

# ========================
# == Array Manipulation ==
# ========================

x = np.linspace(0, 25, 20, endpoint=False)
print(x)
'''
[ 0.    1.25  2.5   3.75  5.    6.25  7.5   8.75 10.   11.25 12.5  13.75
 15.   16.25 17.5  18.75 20.   21.25 22.5  23.75]
'''

# ===============
# == Note That ==
# ===============
# the number of elements in old array must equal the number of elements in new array


# reshape the array
x = np.reshape(x, (4, 5))
print(x)
'''
[[ 0.    1.25  2.5   3.75  5.  ]
 [ 6.25  7.5   8.75 10.   11.25]
 [12.5  13.75 15.   16.25 17.5 ]
 [18.75 20.   21.25 22.5  23.75]]
'''

x = np.reshape(x, (10, 2))
print(x)
```

```
'''
[[ 0.    1.25]
 [ 2.5   3.75]
 [ 5.    6.25]
 [ 7.5   8.75]
 [10.   11.25]
 [12.5  13.75]
 [15.   16.25]
 [17.5  18.75]
 [20.   21.25]
 [22.5  23.75]]
'''


y = np.arange(10).reshape(5, 2)
print(y)
'''
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
'''
```