Python 6: NumPy

금융공학 프로그래밍 I

Intro to NumPy

❖ What is NumPy?

- Fast array processing
- Iteration via loops in interpreted languages (including Python) is relatively slow
- Operations are sent in batches to optimized C and Fortran code

❖ NumPy Arrays

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

In [2]: a = np.zeros(3)

In [3]: a
Out[3]: array([ 0.,  0.,  0.])

In [4]: type(a)
Out[4]: numpy.ndarray
```

NumPy dtype

- NumPy arrays vs. Python lists
  - Data must be homogeneous (all elements of the same type)
  - These types (dtypes) are provided by NumPy

- dtypes
  - float64 (default type)
  - float32
  - int64
  - int32
  - bool

```
In [7]: a = np.zeros(3)

In [8]: type(a[0])
Out[8]: numpy.float64
```

```
In [9]: a = np.zeros(3, dtype=int)

In [10]: type(a[0])
Out[10]: numpy.int32
```

Shape and Dimension

❖ “flat” array

```
In [11]: z = np.zeros(10)

In [12]: z.shape
Out[12]: (10,)  # Note syntax for tuple with one element
```

❖ Changing shape

```
In [13]: z.shape = (10, 1)

In [14]: z
Out[14]:
array([[ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.]])
```

```
In [15]: z = np.zeros(4)

In [16]: z.shape = (2, 2)

In [17]: z
Out[17]:
array([[ 0.,  0.],
       [ 0.,  0.]])
```

Creating Arrays

❖ Empty array

```
In [18]: z = np.empty(3)

In [19]: z
Out[19]: array([  8.90030222e-307,   4.94944794e+173,   4.04144187e-262])
```

❖ Grid of evenly spaced numbers

```
In [20]: z = np.linspace(2, 4, 5)  # From 2 to 4, with 5 elements
```

❖ Identity

```
In [21]: z = np.identity(2)

In [22]: z
Out[22]:
array([[ 1.,  0.],
       [ 0.,  1.]])
```

Creating Array

❖ From Python lists, tuples, etc.

```
In [23]: z = np.array([10, 20])

In [24]: z
Out[24]: array([10, 20])

In [25]: type(z)
Out[25]: numpy.ndarray

In [26]: z = np.array((10, 20), dtype=float)

In [27]: z
Out[27]: array([ 10.,  20.])

In [28]: z = np.array([[1, 2], [3, 4]])

In [29]: z
Out[29]:
array([[1, 2],
       [3, 4]])
```

Indexing

```
In [30]: z = np.linspace(1, 2, 5)

In [31]: z
Out[31]: array([ 1.  ,  1.25,  1.5 ,  1.75,  2.  ])

In [32]: z[0]
Out[32]: 1.0

In [33]: z[0:2]  # Slice numbering is left closed, right open
Out[33]: array([ 1.  ,  1.25])

In [34]: z[-1]
Out[34]: 2.0
```

2D

```
In [35]: z = np.array([[1, 2], [3, 4]])

In [36]: z
Out[36]:
array([[1, 2],
       [3, 4]])

In [37]: z[0, 0]
Out[37]: 1

In [38]: z[0, 1]
Out[38]: 2

In [39]: z[0,:]
Out[39]: array([1, 2])

In [40]: z[:,1]
Out[40]: array([2, 4])
```

```
In [42]: z
Out[42]: array([ 2. ,  2.5,  3. ,  3.5,  4. ])

In [43]: indices = np.array((0, 2, 3))

In [44]: z[indices]
Out[44]: array([ 2. ,  3. ,  3.5])

In [46]: d = np.array([0, 1, 1, 0, 0], dtype=bool)

In [47]: d
Out[47]: array([False,  True,  True, False, False], dtype=bool)

In [48]: z[d]
Out[48]: array([ 2.5,  3. ])
```

```
In [49]: z = np.empty(3)

In [50]: z
Out[50]: array([ -1.25236750e-041,   0.00000000e+000,   5.45693855e-313])

In [51]: z[:] = 42

In [52]: z
Out[52]: array([ 42.,  42.,  42.])
```

Methods

Attribute		
	dtype	ndim
	shape	size
	T	
Method		
	all	any
	argmax	argmin
	max	min
	mean	
	std	var
	prod	sum
	cumprod	cumsum
	dot	transpose
	diagonal	trace
	sort	copy

```
In [53]: A = np.array((4, 3, 2, 1))

In [54]: A
Out[54]: array([4, 3, 2, 1])

In [55]: A.sort()            # Sorts A in place

In [56]: A
Out[56]: array([1, 2, 3, 4])

In [57]: A.sum()             # Sum
Out[57]: 10

In [58]: A.mean()            # Mean
Out[58]: 2.5

In [64]: A.std()
Out[64]: 1.1180339887498949

In [65]: A.shape = (2, 2)

In [66]: A.T
Out[66]:
array([[1, 3],
       [2, 4]])
```

Algebraic Operation

```
In [75]: a = np.array([1, 2, 3, 4])

In [76]: b = np.array([5, 6, 7, 8])

In [77]: a + b
Out[77]: array([ 6,  8, 10, 12])

In [78]: a * b
Out[78]: array([ 5, 12, 21, 32])

In [79]: a + 10
Out[79]: array([11, 12, 13, 14])

In [82]: a * 10
Out[82]: array([10, 20, 30, 40])
```

```
In [86]: A = np.ones((2, 2))

In [87]: B = np.ones((2, 2))

In [88]: A + B
Out[88]:
array([[ 2.,  2.],
       [ 2.,  2.]])

In [89]: A + 10
Out[89]:
array([[ 11.,  11.],
       [ 11.,  11.]])

In [90]: A * B
Out[90]:
array([[ 1.,  1.],
       [ 1.,  1.]])
```

In particular, `A * B` is *not* the matrix product, it is an elementwise product

Matrix Multiplication

```
In [137]: A = np.ones((2, 2))

In [138]: B = np.ones((2, 2))

In [139]: np.dot(A, B)
Out[139]:
array([[ 2.,  2.],
       [ 2.,  2.]])
```

```
In [91]: A = np.array([1, 2])

In [92]: B = np.array([10, 20])

In [93]: np.dot(A, B)    # Returns a scalar in this case
Out[93]: 50
```

Comparisons

❖ Elementwise comparisons

```
In [97]: z = np.array([2, 3])

In [98]: y = np.array([2, 3])

In [99]: z == y
Out[99]: array([ True,  True], dtype=bool)

In [100]: y[0] = 5

In [101]: z == y
Out[101]: array([False,  True], dtype=bool)

In [102]: z != y
Out[102]: array([ True, False], dtype=bool)

In [103]: z = np.linspace(0, 10, 5)

In [104]: z
Out[104]: array([  0. ,   2.5,   5. ,   7.5,  10. ])

In [105]: z > 3
Out[105]: array([False, False,  True,  True,  True], dtype=bool)
```

```
In [109]: z[z > 3]
Out[109]: array([  5. ,   7.5,  10. ])
```

Vectorized Functions

```
for i in range(n):
    y[i] = np.sin(z[i])
```

```
In [110]: z = np.array([1, 2, 3])

In [111]: np.sin(z)
Out[111]: array([ 0.84147098,  0.90929743,  0.14112001])

In [113]: (1 / np.sqrt(2 * np.pi)) * np.exp(- 0.5 * z**2)
Out[113]: array([ 0.24197072,  0.05399097,  0.00443185])
```

np.where

```
In [114]: import numpy as np

In [115]: x = np.random.randn(4)

In [116]: x
Out[116]: array([-0.25521782,  0.38285891, -0.98037787, -0.083662  ])

In [117]: np.where(x > 0, 1, 0)  # Insert 1 if x > 0 true, otherwise 0
Out[117]: array([0, 1, 0, 0])
```

❖ Vectorized function

```
In [118]: def f(x): return 1 if x > 0 else 0

In [119]: f = np.vectorize(f)

In [120]: f(x)                    # Passing same vector x as previous example
Out[120]: array([0, 1, 0, 0])
```

Other NumPy Functions

```
In [131]: A = np.array([[1, 2], [3, 4]])

In [132]: np.linalg.det(A)             # Compute the determinant
Out[132]: -2.0000000000000004

In [133]: np.linalg.inv(A)             # Compute the inverse
Out[133]:
array([[-2. ,  1. ],
       [ 1.5, -0.5]])

In [134]: Z = np.random.randn(10000)  # Generate standard normals

In [135]: y = np.random.binomial(10, 0.5, size=1000)
```

Q & A

khhwang78@gmail.com