Python 2: Data Types

금융공학 프로그래밍 I

Data Types

❖ Boolean (True or False)

```
In [1]: x = True

In [2]: y = 100 < 10

In [3]: y
Out[3]: False

In [4]: type(y)
Out[4]: bool
```

```
In [5]: x + y
Out[5]: 1

In [6]: x * y
Out[6]: 0

In [7]: True + True
Out[7]: 2

In [8]: bools = [True, True, False, True]

In [9]: sum(bools)
Out[9]: 3
```

Data Types

❖ Integer & Floats

```
In [1]: a, b = 1, 2

In [2]: c, d = 2.5, 10.0

In [3]: type(a)
Out[3]: int

In [4]: type(c)
Out[4]: float
```

```
In [5]: 1 / 2       # Integer division in Python 2.x
Out[5]: 0

In [6]: 1.0 / 2.0   # Floating point division
Out[6]: 0.5

In [7]: 1.0 / 2     # Floating point division
Out[7]: 0.5
```

Data Types

❖ bit_length (Integer)

```
In [3]: a = 100000
        a.bit_length()

Out[3]: 17

In [4]: googol = 10 ** 100
        googol

Out[4]: 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
        0000000000000000000000000000L

In [5]: googol.bit_length()

Out[5]: 333
```

❖ Floating Point

```
In [11]: b = 0.35
         type(b)

Out[11]: float

In [12]: b + 0.1

Out[12]: 0.4499999999999996
```

Data Types

❖ String

```
In [23]: t = 'this is a string object'

In [24]: t.capitalize()
Out[24]: 'This is a string object'

In [25]: t.split()
Out[25]: ['this', 'is', 'a', 'string', 'object']

In [26]: t.find('string')
Out[26]: 10

In [27]: t.find('Python')
Out[27]: -1

In [28]: t.replace(' ', '|')
Out[28]: 'this|is|a|string|object'
```

Data Types

```
In [293]: a = 5.6

In [294]: s = str(a)

In [295]: s
Out[295]: '5.6'

In [296]: s = 'python'

In [297]: list(s)
Out[297]: ['p', 'y', 't', 'h', 'o', 'n']

In [298]: s[:3]
Out[298]: 'pyt'

In [299]: s = '12\\34'

In [300]: print s
12\34

In [301]: s = r'this\has\no\special\characters'

In [302]: s
Out[302]: 'this\\has\\no\\special\\characters'

In [303]: a = 'this is the first half '

In [304]: b = 'and this is the second half'

In [305]: a + b
Out[305]: 'this is the first half and this is the second half'
```

String Formatting

```
a = 42
b = 13.142783
c = "hello"
d = {'x':13, 'y':1.54321, 'z':'world'}
e = 5628398123741234

r = "a is %d" % a              # r = "a is 42"
r = "%10d %f" % (a,b)          # r = "        42 13.142783"
r = "%+010d %E" % (a,b)        # r = "+000000042 1.314278E+01"
r = "%(x)-10d  %(y)0.3g" % d   # r = "13         1.54"
r = "%0.4s %s" % (c, d['z'])   # r = "hell world"
r = "%*.*f" % (5,3,b)          # r = "13.143"
r = "e = %d" % e               # r = "e = 5628398123741234"

stock = {
    'name' : 'GOOG',
    'shares' : 100,
    'price' : 490.10 }

r = "%(shares)d of %(name)s at %(price)0.2f" % stock
# r = "100 shares of GOOG at 490.10"
```

Advanced String Formatting

```
r = "{0} {1} {2}".format('GOOG',100,490.10)
r = "{name} {shares} {price}".format(name='GOOG',shares=100,price=490.10)
r = "Hello {0}, your age is {age}".format("Elwood",age=47)
r = "Use {{ and }} to output single curly braces".format()


name = "Elwood"
r = "{0:<10}".format(name)      # r = 'Elwood    '
r = "{0:>10}".format(name)      # r = '    Elwood'
r = "{0:^10}".format(name)      # r = '  Elwood  '
r = "{0:=^10}".format(name)     # r = '==Elwood=='


x = 42
r = '{0:10d}'.format(x)         # r = '        42'
r = '{0:10x}'.format(x)         # r = '        2a'
r = '{0:10b}'.format(x)         # r = '    101010'
r = '{0:010b}'.format(x)        # r = '0000101010'

y = 3.1415926
r = '{0:10.2f}'.format(y)       # r = '      3.14'
r = '{0:10.2e}'.format(y)       # r = '  3.14e+00'
r = '{0:+10.2f}'.format(y)      # r = '     +3.14'
r = '{0:+010.2f}'.format(y)     # r = '+000003.14'
r = '{0:+10.2%}'.format(y)      # r = '  +314.16%'
```

Type Casting

- Type Casting

```
In [315]: s = '3.14159'

In [316]: fval = float(s)
In [317]: type(fval)
Out[317]: float

In [318]: int(fval)
Out[318]: 3

In [319]: bool(fval)
Out[319]: True

In [320]: bool(0)
Out[320]: False
```

- None
  - Python null value type

```
In [321]: a = None
In [322]: a is None
Out[322]: True

In [323]: b = 5
In [324]: b is not None
Out[324]: True
```

Dynamic Typing

- Dynamic Typing

```
In [245]: a = 5
```

```
In [246]: type(a)
Out[246]: int
```

```
In [247]: a = 'foo'
```

```
In [248]: type(a)
Out[248]: str
```

- Strongly-typed
  - Implicit conversions will occur only in certain obvious circumstances

```
In [249]: '5' + 5
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-249-f9dbf5f0b234> in <module>()
----> 1 '5' + 5
TypeError: cannot concatenate 'str' and 'int' objects
```

Basic Data Structures

❖ Tuples

```
In [37]: t = (1, 2.5, 'data')
         type(t)
Out[37]: tuple

In [39]: t[2]
Out[39]: 'data'
In [40]: type(t[2])
Out[40]: str

In [41]: t.count('data')
Out[41]: 1
In [42]: t.index(1)
Out[42]: 0
```

Tuples (and lists) can be "unpacked" as follows

```
In [21]: integers = (10, 20, 30)

In [22]: x, y, z = integers
```

Basic Data Structures

❖ Lists

```
In [43]: l = [1, 2.5, 'data']
         l[2]
Out[43]: 'data'

In [44]: l = list(t)
         l
Out[44]: [1, 2.5, 'data']
In [45]: type(l)
Out[45]: list
```

Insert & Remove

```
In [46]: l.append([4, 3])  # append list at the end
         l
Out[46]: [1, 2.5, 'data', [4, 3]]
In [47]: l.extend([1.0, 1.5, 2.0])  # append elements of list
         l
Out[47]: [1, 2.5, 'data', [4, 3], 1.0, 1.5, 2.0]
In [48]: l.insert(1, 'insert')  # insert object before index position
         l
Out[48]: [1, 'insert', 2.5, 'data', [4, 3], 1.0, 1.5, 2.0]
In [49]: l.remove('data')  # remove first occurrence of object
         l
Out[49]: [1, 'insert', 2.5, [4, 3], 1.0, 1.5, 2.0]
In [50]: p = l.pop(3)  # removes and returns object at index
         print l, p
Out[50]: [1, 'insert', 2.5, 1.0, 1.5, 2.0] [4, 3]
```

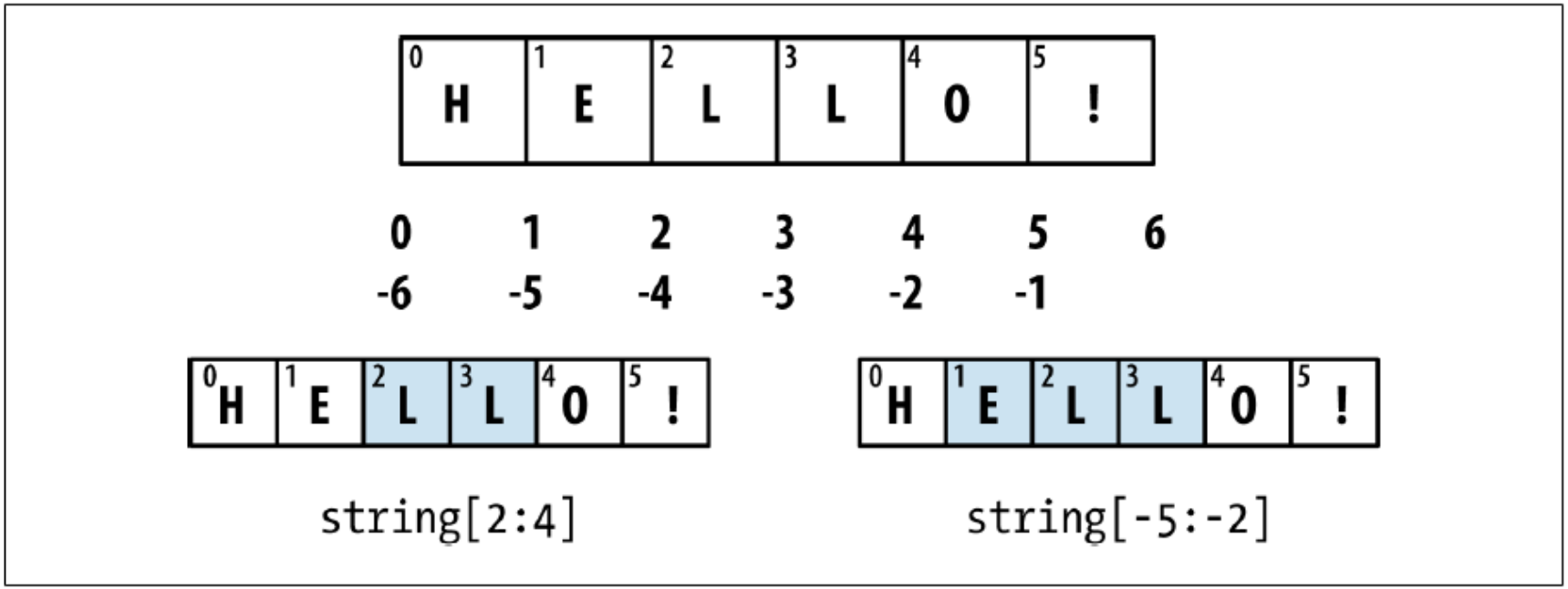
Indexing & Slicing

```
In [14]: a = [2, 4, 6, 8]

In [15]: a[1:]
Out[15]: [4, 6, 8]

In [16]: a[1:3]
Out[16]: [4, 6]
```

```
In [17]: a[-2:]  # Last two elements of the list
Out[17]: [6, 8]
```

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

b = a[::2]          # b = [0, 2, 4, 6, 8 ]
c = a[::-2]         # c = [9, 7, 5, 3, 1 ]
d = a[0:4:2]        # d = [0,2]
e = a[5:0:-2]       # e = [5,3,1]
f = a[:5:1]         # f = [0,1,2,3,4]
g = a[:5:-1]        # g = [9,8,7,6]
h = a[5::1]         # h = [5,6,7,8,9]
i = a[5::-1]        # i = [5,4,3,2,1,0]
j = a[5:0:-1]       # j = [5,4,3,2,1]
```

Slicing



Basic Data Structures

- **Dicts**
  - **Key-Value**

```
In [66]: d = {
             'Name' : 'Angela Merkel',
             'Country' : 'Germany',
             'Profession' : 'Chancelor',
             'Age' : 60
             }
         type(d)
Out[66]: dict
In [67]: print d['Name'], d['Age']
Out[67]: Angela Merkel 60
```

```
In [68]: d.keys()
Out[68]: ['Country', 'Age', 'Profession', 'Name']
In [69]: d.values()
Out[69]: ['Germany', 60, 'Chancelor', 'Angela Merkel']
In [70]: d.items()
Out[70]: [('Country', 'Germany'),
          ('Age', 60),
          ('Profession', 'Chancelor'),
          ('Name', 'Angela Merkel')]
In [71]: birthday = True
         if birthday is True:
             d['Age'] += 1
         print d['Age']
Out[71]: 61
```

```
In [439]: d1
Out[439]: {'a': 'some value', 'b': [1, 2, 3, 4]}

In [440]: d1[7] = 'an integer'

In [441]: d1
Out[441]: {7: 'an integer', 'a': 'some value', 'b': [1, 2, 3, 4]}

In [442]: d1['b']
Out[442]: [1, 2, 3, 4]

In [443]: 'b' in d1
Out[443]: True

In [444]: d1[5] = 'some value'

In [445]: d1['dummy'] = 'another value'

In [446]: del d1[5]

In [447]: ret = d1.pop('dummy')

In [448]: ret
Out[448]: 'another value'

In [451]: d1.update({'b' : 'foo', 'c' : 12})

In [452]: d1
Out[452]: {7: 'an integer', 'a': 'some value', 'b': 'foo', 'c': 12}
```

Basic Data Structures

❖ Sets

```
In [74]: s = set(['u', 'd', 'ud', 'du', 'd', 'du'])
         s
Out[74]: {'d', 'du', 'u', 'ud'}
In [75]: t = set(['d', 'dd', 'uu', 'u'])
In [76]: s.union(t)  # all of s and t
Out[76]: {'d', 'dd', 'du', 'u', 'ud', 'uu'}
In [77]: s.intersection(t)  # both in s and t
Out[77]: {'d', 'u'}
In [78]: s.difference(t)  # in s but not t
Out[78]: {'du', 'ud'}
In [79]: t.difference(s)  # in t but not s
Out[79]: {'dd', 'uu'}
In [80]: s.symmetric_difference(t)  # in either one but not both
Out[80]: {'dd', 'du', 'ud', 'uu'}
```

set operations

Function	Alternate Syntax	Description
`a.add(x)`	N/A	Add element x to the set a
`a.remove(x)`	N/A	Remove element x from the set a
`a.union(b)`	`a | b`	All of the unique elements in a and b.
`a.intersection(b)`	`a & b`	All of the elements in *both* a and b.
`a.difference(b)`	`a - b`	The elements in a that are not in b.
`a.symmetric_difference(b)`	`a ^ b`	All of the elements in a or b but *not both*.
`a.issubset(b)`	N/A	`True` if the elements of a are all contained in b.
`a.issuperset(b)`	N/A	`True` if the elements of b are all contained in a.
`a.isdisjoint(b)`	N/A	`True` if a and b have no elements in common.

Mutable & Immutable

- Mutable: lists, dicts

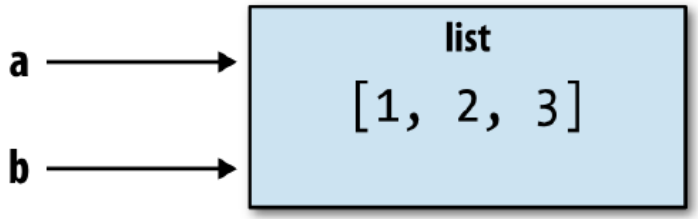
```
In [274]: a_list = ['foo', 2, [4, 5]]

In [275]: a_list[2] = (3, 4)

In [276]: a_list
Out[276]: ['foo', 2, (3, 4)]
```

- Immutable: tuples, strings

```
In [277]: a_tuple = (3, 5, (4, 5))

In [278]: a_tuple[1] = 'four'
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-278-b7966a9ae0f1> in <module>()
----> 1 a_tuple[1] = 'four'
TypeError: 'tuple' object does not support item assignment
```

Pass-by-reference

```
In [241]: a = [1, 2, 3]

In [242]: b = a

In [243]: a.append(4)

In [244]: b
Out[244]: [1, 2, 3, 4]
```



deep copy

```
>>> import copy
>>> a = [1, 2, [3, 4]]
>>> b = copy.deepcopy(a)
>>> b[2][0] = -100
>>> b
[1, 2, [-100, 4]]
>>> a                          # Notice that a is unchanged
[1, 2, [3, 4]]
```

Imports

From the start, Python has been designed around the twin principles of

- a small core language
- extra functionality in separate *libraries* or *modules*

```
In [1]: import math

In [2]: math.sqrt(4)
Out[2]: 2.0
```

Or

```
In [3]: from math import *

In [4]: sqrt(4)
Out[4]: 2.0
```

Dates and Times

- Built-in Python *datetime* module

```
In [325]: from datetime import datetime, date, time

In [326]: dt = datetime(2011, 10, 29, 20, 30, 21)

In [327]: dt.day
Out[327]: 29

In [328]: dt.minute
Out[328]: 30

In [329]: dt.date()
Out[329]: datetime.date(2011, 10, 29)

In [330]: dt.time()
Out[330]: datetime.time(20, 30, 21)

In [331]: dt.strftime('%m/%d/%Y %H:%M')
Out[331]: '10/29/2011 20:30'

In [332]: datetime.strptime('20091031', '%Y%m%d')
Out[332]: datetime.datetime(2009, 10, 31, 0, 0)

In [334]: dt2 = datetime(2011, 11, 15, 22, 30)

In [335]: delta = dt2 - dt

In [336]: delta
Out[336]: datetime.timedelta(17, 7179)

In [337]: type(delta)
Out[337]: datetime.timedelta
```

Operators

Operation	Description
`a + b`	Add a and b
`a - b`	Subtract b from a
`a * b`	Multiply a by b
`a / b`	Divide a by b
`a // b`	Floor-divide a by b, dropping any fractional remainder
`a ** b`	Raise a to the b power
`a & b`	`True` if both a and b are `True`. For integers, take the bitwise AND.
`a | b`	`True` if either a or b is `True`. For integers, take the bitwise OR.
`a ^ b`	For booleans, `True` if a or b is `True`, but not both. For integers, take the bitwise EXCLUSIVE-OR.
`a == b`	`True` if a equals b
`a != b`	`True` if a is not equal to b
`a <= b, a < b`	`True` if a is less than (less than or equal) to b
`a > b, a >= b`	`True` if a is greater than (greater than or equal) to b
`a is b`	`True` if a and b reference same Python object
`a is not b`	`True` if a and b reference different Python objects

Operations on Sequence

Operation	Description
`s + r`	Concatenation
`s * n, n * s`	Makes `n` copies of `s`, where `n` is an integer
`v1, v2..., vn = s`	Variable unpacking
`s[i]`	Indexing
`s[i:j]`	Slicing
`s[i:j:stride]`	Extended slicing
`x in s, x not in s`	Membership
`for x in s:`	Iteration
`all(s)`	Returns `True` if all items in `s` are true.
`any(s)`	Returns `True` if any item in `s` is true.
`len(s)`	Length
`min(s)`	Minimum item in `s`
`max(s)`	Maximum item in `s`
`sum(s [, initial])`	Sum of items with an optional initial value

Copies of a sequence

```
>>> a = [3,4,5]
>>> b = [a]
>>> c = 4*b
>>> c
[[3, 4, 5], [3, 4, 5], [3, 4, 5], [3, 4, 5]]
>>> a[0] = -7
>>> c
[[-7, 4, 5], [-7, 4, 5], [-7, 4, 5], [-7, 4, 5]]
```

Input & Output

❖ File output & input

```
In [35]: f = open('newfile.txt', 'w')    # Open 'newfile.txt' for writing

In [36]: f.write('Testing\n')            # Here '\n' means new line

In [37]: f.write('Testing again')

In [38]: f.close()
```

```
In [39]: f = open('newfile.txt', 'r')

In [40]: out = f.read()

In [41]: out
Out[41]: 'Testing\nTesting again'

In [42]: print(out)
Out[42]:
Testing
Testing again
```

File mode

Mode	Description
`r`	Read-only mode
`w`	Write-only mode. Creates a new file (deleting any file with the same name)
`a`	Append to existing file (create it if it does not exist)
`r+`	Read and write
`b`	Add to mode for binary files, that is `'rb'` or `'wb'`
`U`	Use universal newline mode. Pass by itself `'U'` or appended to one of the read modes like `'rU'`

Change working directory

- Current working directory (cwd)

```
import os
print(os.getcwd())
```

(In the IPython notebook, `pwd` should also work)

- Specifing the full path to the file

```
In [43]: f = open('insert_full_path_to_file/newfile.txt', 'r')
```

- Changing the current working directory

```
os.chdir('path_to_file')
```

Q & A