(Ch 24) Modules for Data Structure

- collections
- array
- queue
- heapq
- bisect

“collections” Module

Object들의 저장소를 보다 더 다양하고 편하게 제공하고자!

- The **collections** module implements specialized container data types
  - providing alternatives to the built-in types **list**, **dict**, **set,** and **tuple**

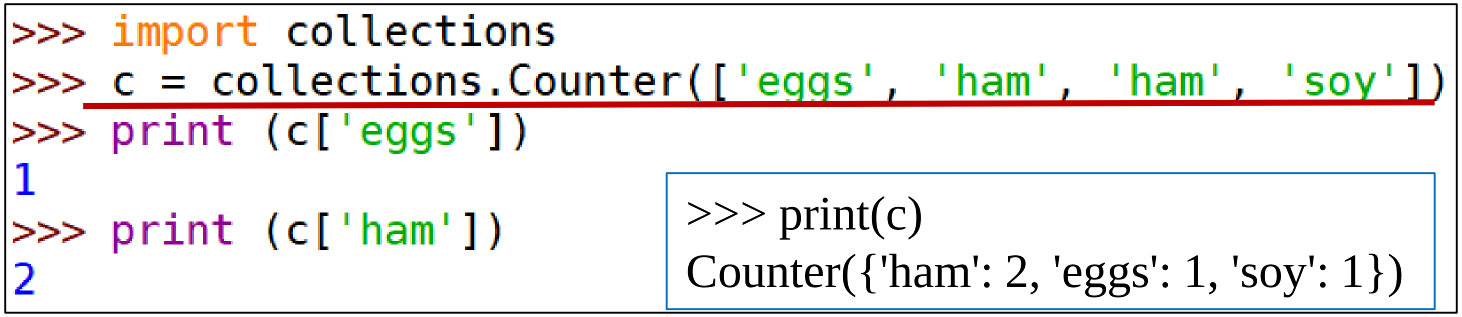
- **Counter class**
  - **Subclass of “dict” class**
  - **Counting hashable objects**

- **deque function**
  - list-like container with fast appends and pops on either end
  - a generalization of stacks and queues (the name is pronounced “deck” and is short for “double-ended queue”)
- **OrderedDict class**
  - just like regular dictionaries but they remember the order that items were inserted.

- ChainMap class,, UserDict class, UserList class etc…

“collections” Module – Counter Class [1/2]

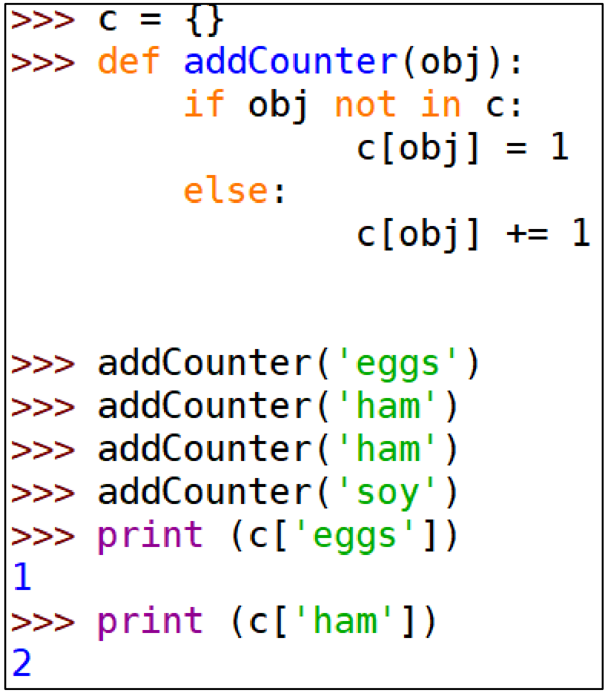
- A **Counter** class is a dictionary-like container that tracks how many times equivalent values are added



```
>>> import collections
>>> c = collections.Counter(['eggs', 'ham', 'ham', 'soy'])
>>> print (c['eggs'])
1
>>> print (c['ham'])
2
```

```
>>> print(c)
Counter({'ham': 2, 'eggs': 1, 'soy': 1})
```

Unless using collections module!

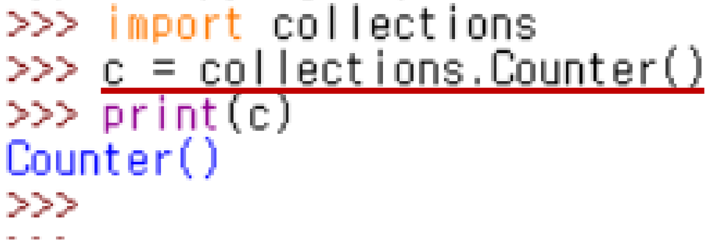


```
>>> c = {}
>>> def addCounter(obj):
        if obj not in c:
                c[obj] = 1
        else:
                c[obj] += 1


>>> addCounter('eggs')
>>> addCounter('ham')
>>> addCounter('ham')
>>> addCounter('soy')
>>> print (c['eggs'])
1
>>> print (c['ham'])
2
```

“collections” Module – Counter Class [2/2]

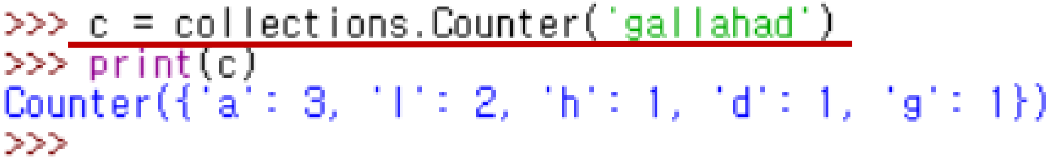
- Counter constructors

```
# a new, empty counter
```



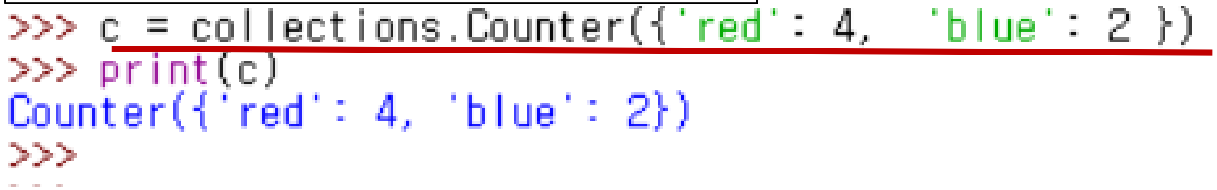
```
>>> import collections
>>> c = collections.Counter()
>>> print(c)
Counter()
>>>
```

```
# a new counter from an iterable
```



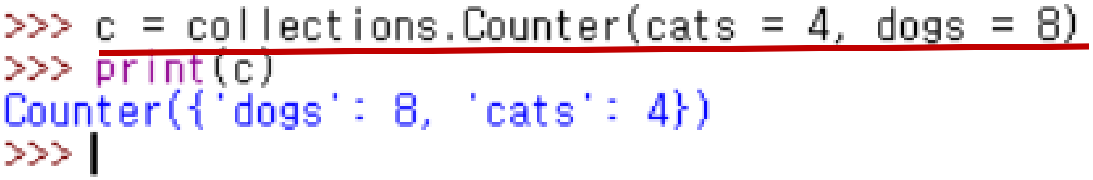
```
>>> c = collections.Counter('gallahad')
>>> print(c)
Counter({'a': 3, 'l': 2, 'h': 1, 'd': 1, 'g': 1})
>>>
```

```
# a new counter from a mapping
```



```
>>> c = collections.Counter({'red': 4,  'blue': 2 })
>>> print(c)
Counter({'red': 4, 'blue': 2})
>>>
```

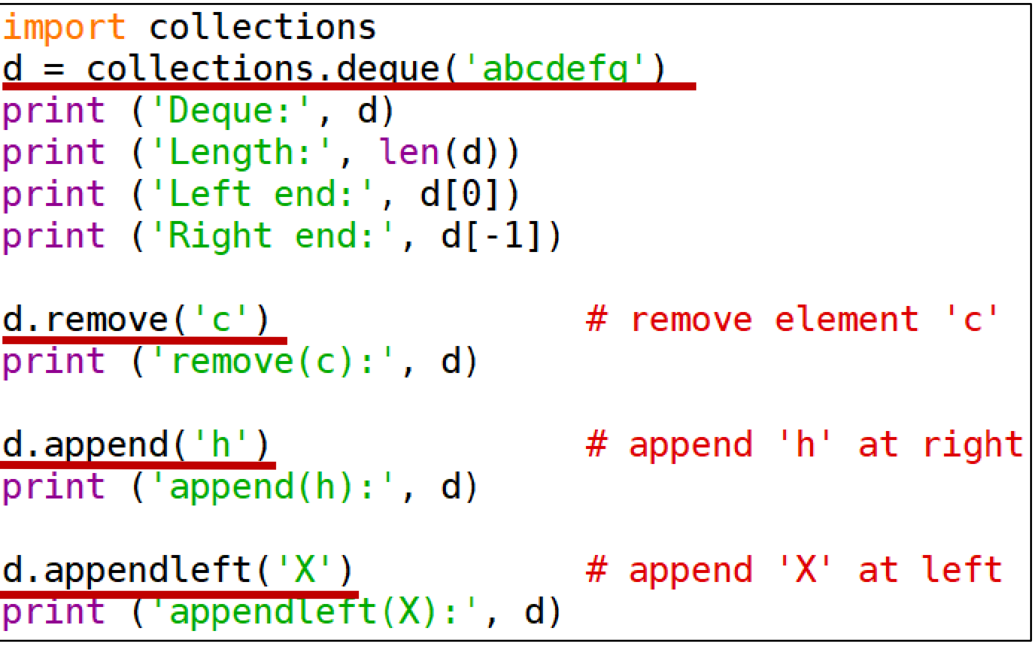
```
# a new counter from keyword args
```



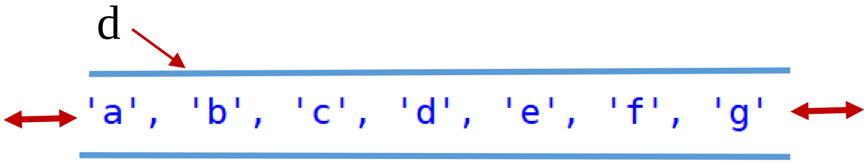
```
>>> c = collections.Counter(cats = 4, dogs = 8)
>>> print(c)
Counter({'dogs': 8, 'cats': 4})
>>>
```

“collections” Module – deque( ) Function [1/3]

- **deque** function returns a list-like object initialized left-to-right
  - supports adding and removing elements from either end
- The more commonly used stacks and queues are forms of **deque**, where the inputs and outputs are restricted to a single end

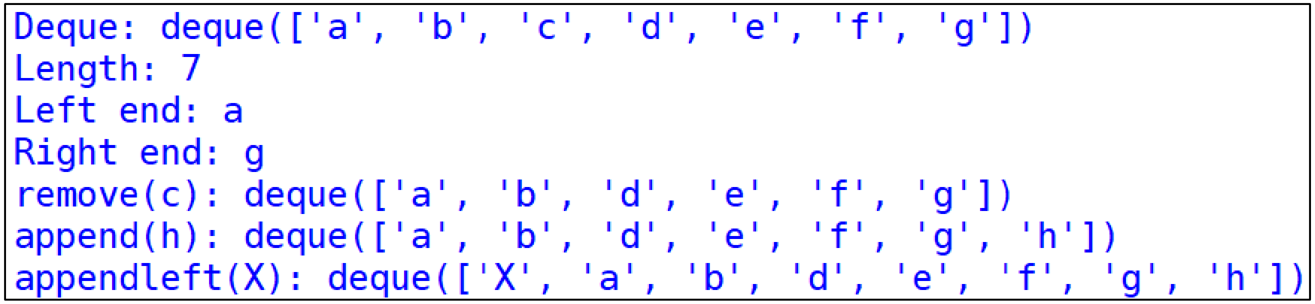


```
import collections
d = collections.deque('abcdefg')
print ('Deque:', d)
print ('Length:', len(d))
print ('Left end:', d[0])
print ('Right end:', d[-1])

d.remove('c')               # remove element 'c'
print ('remove(c):', d)

d.append('h')               # append 'h' at right
print ('append(h):', d)

d.appendleft('X')           # append 'X' at left
print ('appendleft(X):', d)
```



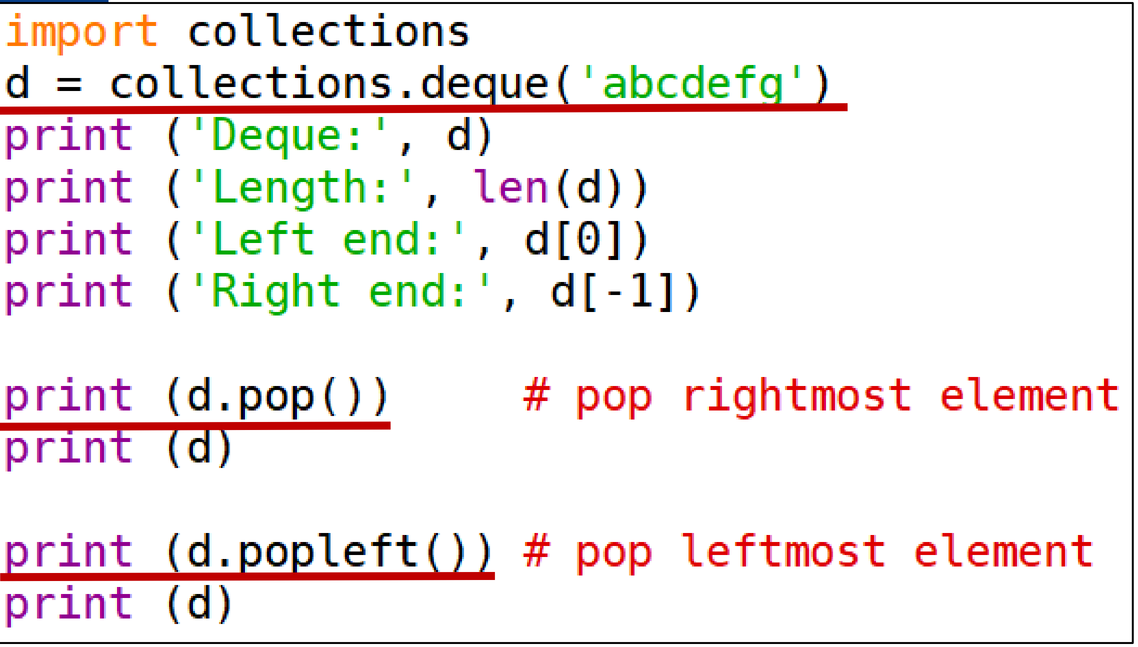
dequeue 양쪽끝에서 element를 insert or delete할수 있고, dequeue의 중간에서 element를 중간에서 remove 가능하다!

Result

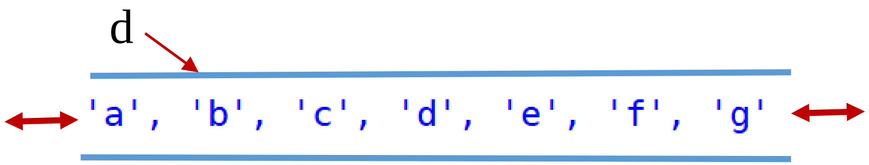


```
Deque: deque(['a', 'b', 'c', 'd', 'e', 'f', 'g'])
Length: 7
Left end: a
Right end: g
remove(c): deque(['a', 'b', 'd', 'e', 'f', 'g'])
append(h): deque(['a', 'b', 'd', 'e', 'f', 'g', 'h'])
appendleft(X): deque(['X', 'a', 'b', 'd', 'e', 'f', 'g', 'h'])
```

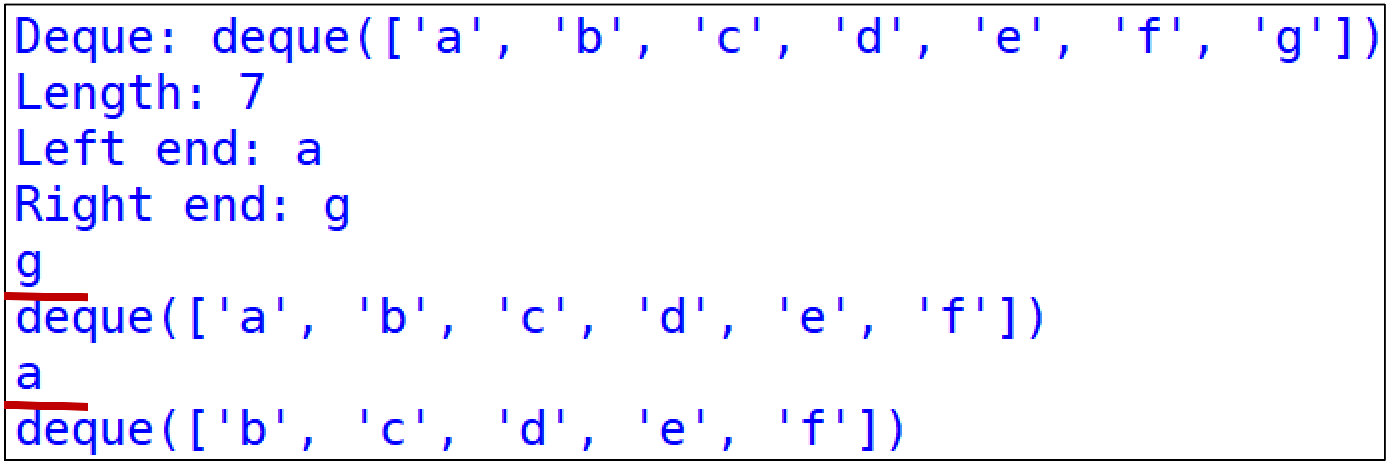
“collections” Module – deque( ) Function [2/3]



```
import collections
d = collections.deque('abcdefg')
print ('Deque:', d)
print ('Length:', len(d))
print ('Left end:', d[0])
print ('Right end:', d[-1])

print (d.pop())      # pop rightmost element
print (d)

print (d.popleft()) # pop leftmost element
print (d)
```

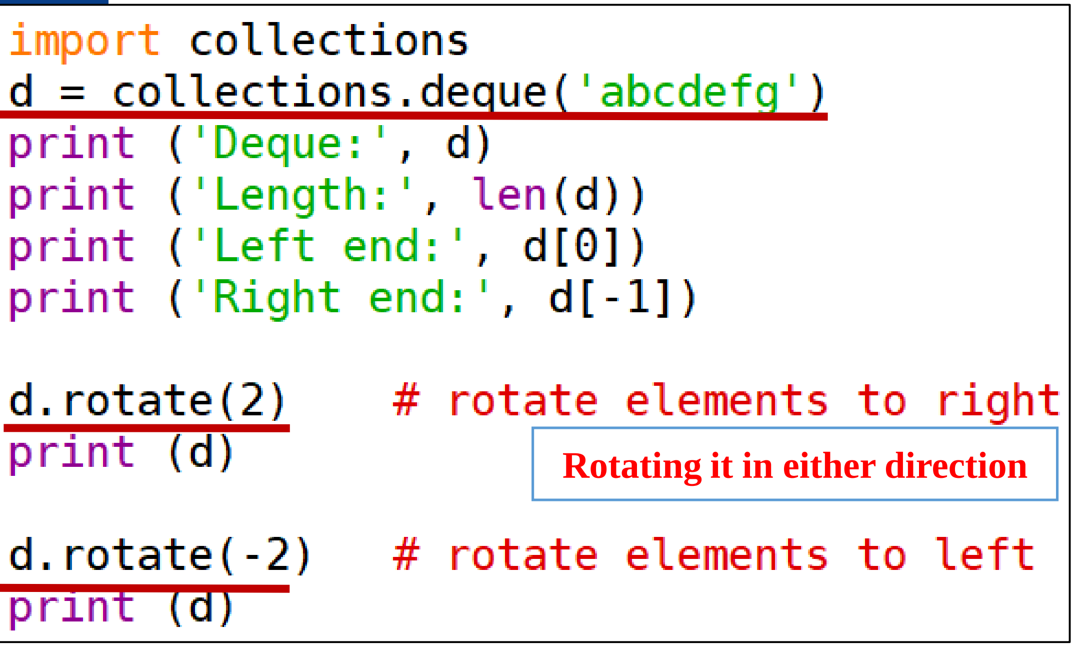


Result



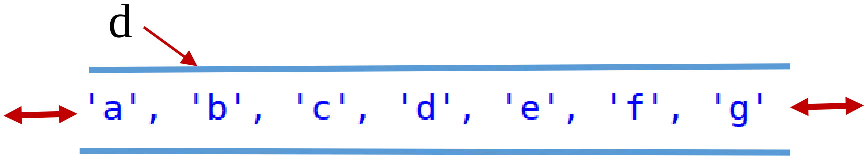
```
Deque: deque(['a', 'b', 'c', 'd', 'e', 'f', 'g'])
Length: 7
Left end: a
Right end: g
g
deque(['a', 'b', 'c', 'd', 'e', 'f'])
a
deque(['b', 'c', 'd', 'e', 'f'])
```

“collections” Module – deque( ) Function [3/3]

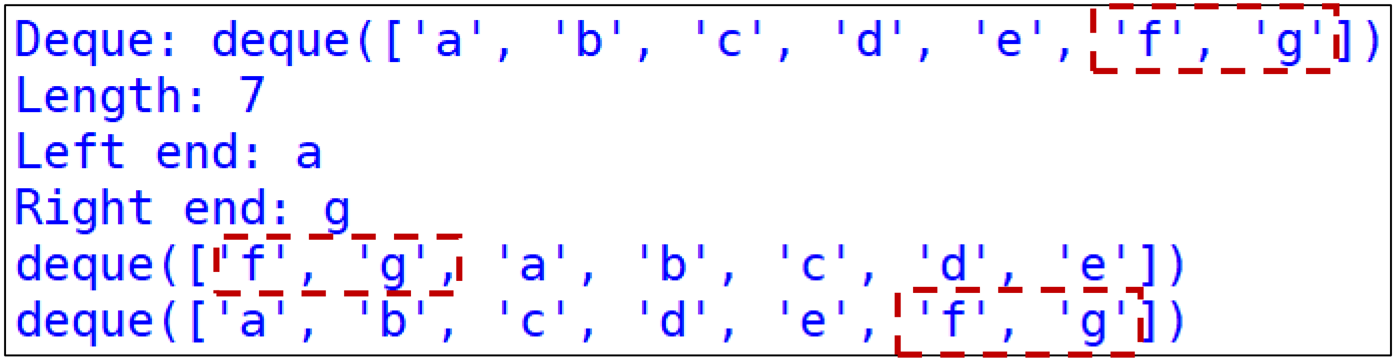


```
import collections
d = collections.deque('abcdefg')
print ('Deque:', d)
print ('Length:', len(d))
print ('Left end:', d[0])
print ('Right end:', d[-1])

d.rotate(2)     # rotate elements to right
print (d)

d.rotate(-2)    # rotate elements to left
print (d)
```

Rotating it in either direction



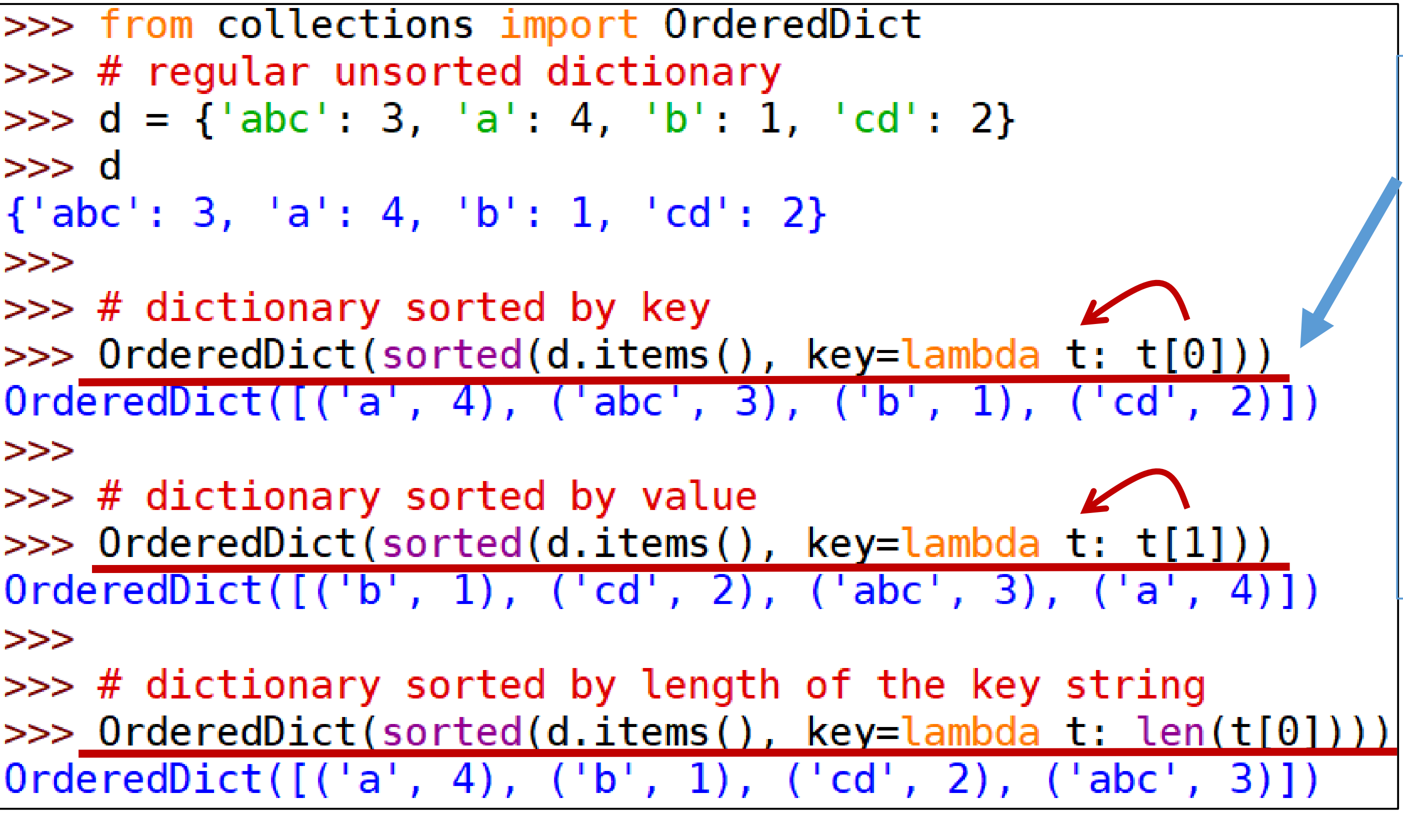
Result



```
Deque: deque(['a', 'b', 'c', 'd', 'e', 'f', 'g'])
Length: 7
Left end: a
Right end: g
deque(['f', 'g', 'a', 'b', 'c', 'd', 'e'])
deque(['a', 'b', 'c', 'd', 'e', 'f', 'g'])
```

“collections” Module – OrderedDict Class

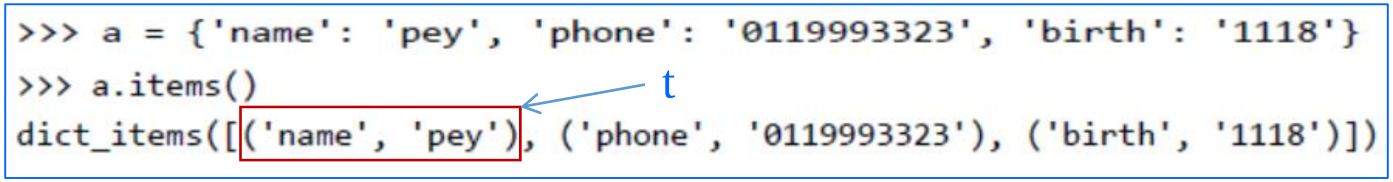
- A regular **dictionary** does not tract the order of key:value pairs, and iterating over it produces the values in an arbitrary order
- We can give a rule how the items of the orderedDict object should be **ordered**



```
>>> from collections import OrderedDict
>>> # regular unsorted dictionary
>>> d = {'abc': 3, 'a': 4, 'b': 1, 'cd': 2}
>>> d
{'abc': 3, 'a': 4, 'b': 1, 'cd': 2}
>>>
>>> # dictionary sorted by key
>>> OrderedDict(sorted(d.items(), key=lambda t: t[0]))
OrderedDict([('a', 4), ('abc', 3), ('b', 1), ('cd', 2)])
>>>
>>> # dictionary sorted by value
>>> OrderedDict(sorted(d.items(), key=lambda t: t[1]))
OrderedDict([('b', 1), ('cd', 2), ('abc', 3), ('a', 4)])
>>>
>>> # dictionary sorted by length of the key string
>>> OrderedDict(sorted(d.items(), key=lambda t: len(t[0])))
OrderedDict([('a', 4), ('b', 1), ('cd', 2), ('abc', 3)])
```

sorted( ) 로 complex object 들을 sort할때 object들의 index를 key로 이용할수 있다.

d.items( ) 에서 dictionary items 들이 key:value pair 이므로 t[0] 는 key , t[1]은 value



```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
>>> a.items()
dict_items([('name', 'pey'), ('phone', '0119993323'), ('birth', '1118')])
```

(Ch 22) Data Structure 관련 Modules

- collections
- array
- queue
- heapq
- bisect

“array” Module

Python의 List기반 2D Array와 3D Array보다 more efficient하게!!

- Purpose : Making array of fixed-type numerical data efficiently
- 1D Array, 2D Array, 3D Array are all emulated with Python List

- For large amounts of data, an **array** may make more efficient use of memory than a **list**

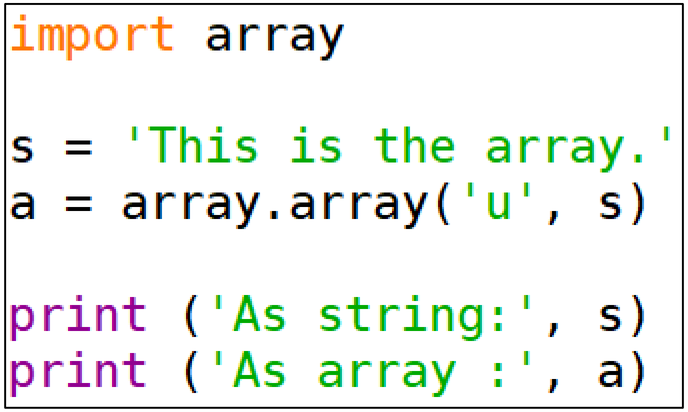
- Since the **array** is limited to a single data type, it can use a more compact memory representation than a general purpose **list**

- Uses the library of C array
  - Needs to define type before using
  - Faster than Python List
    - But if you **really need speed**, consider using pure C

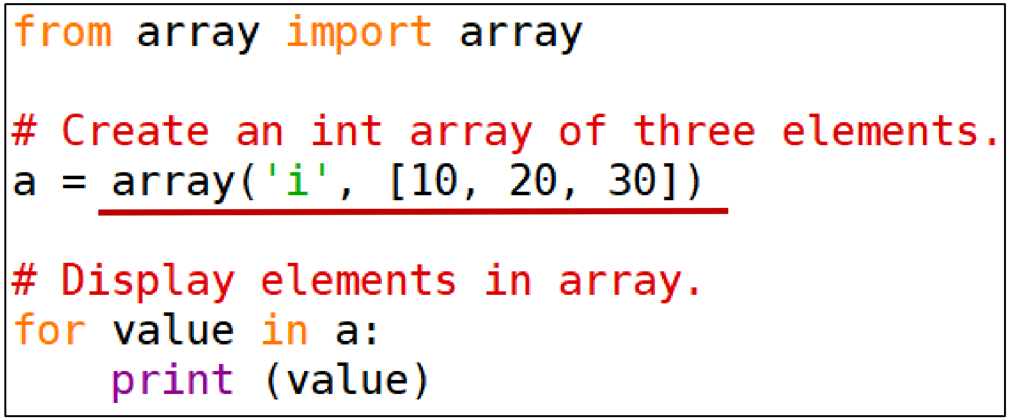
- Python list data type은 다른종류의 item들을 keep할수 있다
  - [“SNU”, 59, [10, 20]]

“array” Module – Initialization with array( )

typecode : ‘u’ for string, ‘i’ for integer

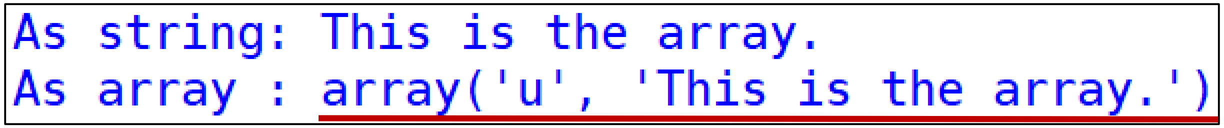


```
import array

s = 'This is the array.'
a = array.array('u', s)

print ('As string:', s)
print ('As array :', a)
```



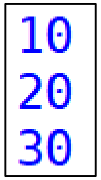
```
from array import array

# Create an int array of three elements.
a = array('i', [10, 20, 30])

# Display elements in array.
for value in a:
    print (value)
```

Result

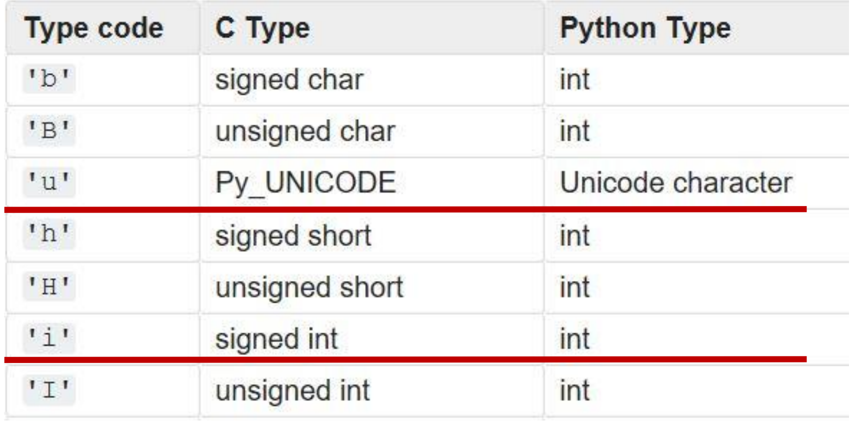


```
As string: This is the array.
As array : array('u', 'This is the array.')
```

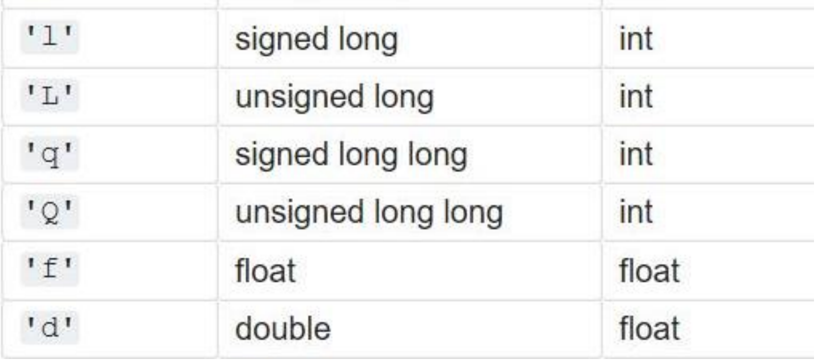
Result



```
10
20
30
```



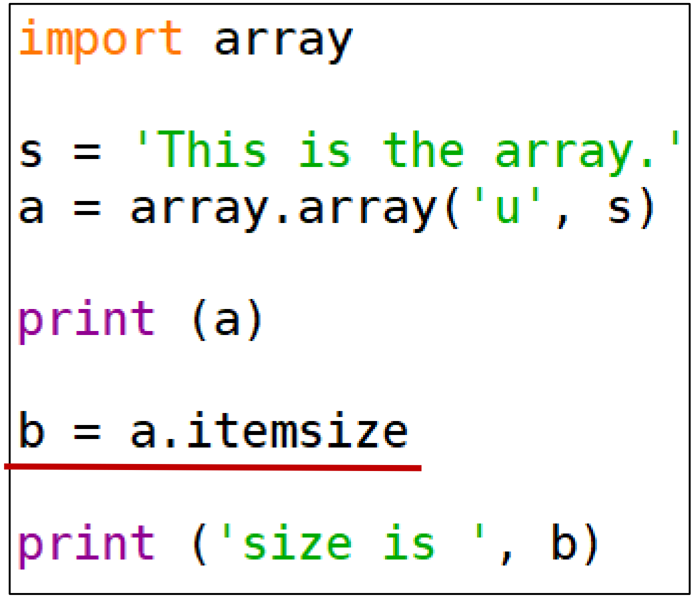
Type code	C Type	Python Type
'b'	signed char	int
'B'	unsigned char	int
'u'	Py_UNICODE	Unicode character
'h'	signed short	int
'H'	unsigned short	int
'i'	signed int	int
'I'	unsigned int	int



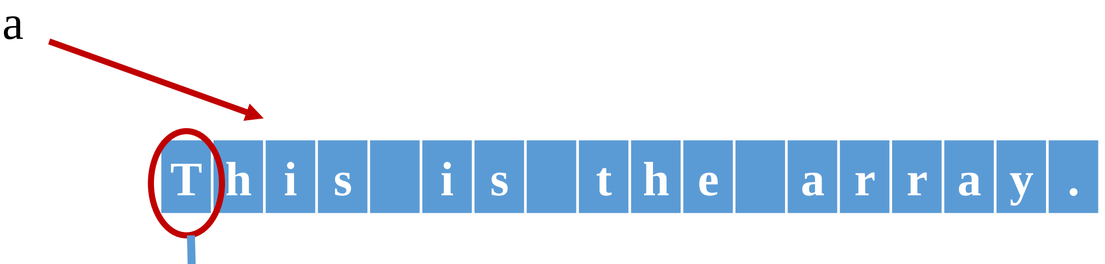
Type code	C Type	Python Type
'l'	signed long	int
'L'	unsigned long	int
'q'	signed long long	int
'Q'	unsigned long long	int
'f'	float	float
'd'	double	float

“array” Module – itemize attribute [1/7]

- array.**itemsize** : the length in bytes of one array item in the internal representation



```python
import array

s = 'This is the array.'
a = array.array('u', s)

print (a)

b = a.itemsize

print ('size is ', b)
```



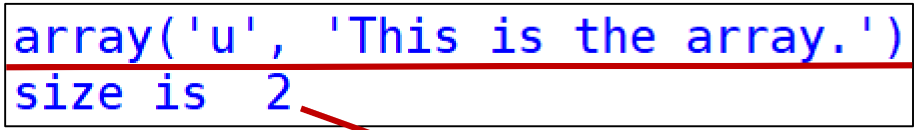
a

T	h	i	s		i	s		t	h	e		a	r	r	a	y	.

2 bytes로 표현된다

```
>>> print(a[0])
T
```

Result

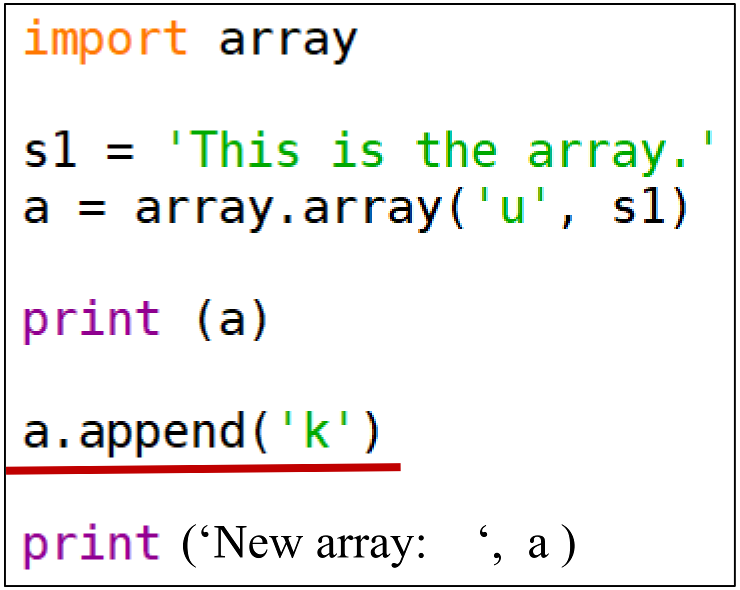


```
array('u', 'This is the array.')
size is  2
```

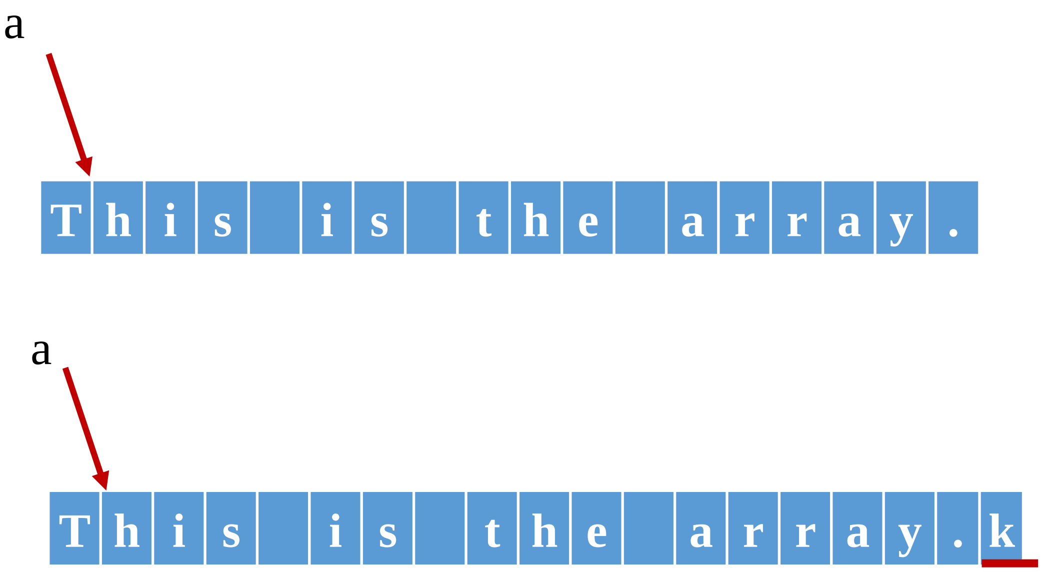
One array item의 size

“array” Module – append( ) [2/7]

- array.**append**(x) : Append new item x to the end of the array (only unicode)

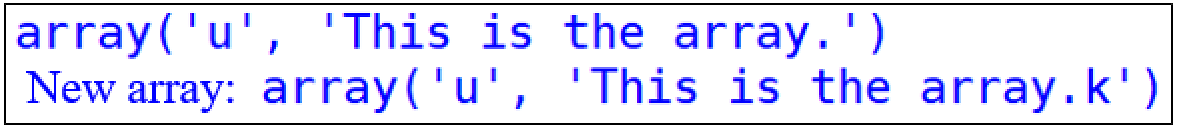


```
import array

s1 = 'This is the array.'
a = array.array('u', s1)

print (a)

a.append('k')

print (‘New array:   ‘, a )
```



a
T h i s   i s   t h e   a r r a y .
a
T h i s   i s   t h e   a r r a y . k

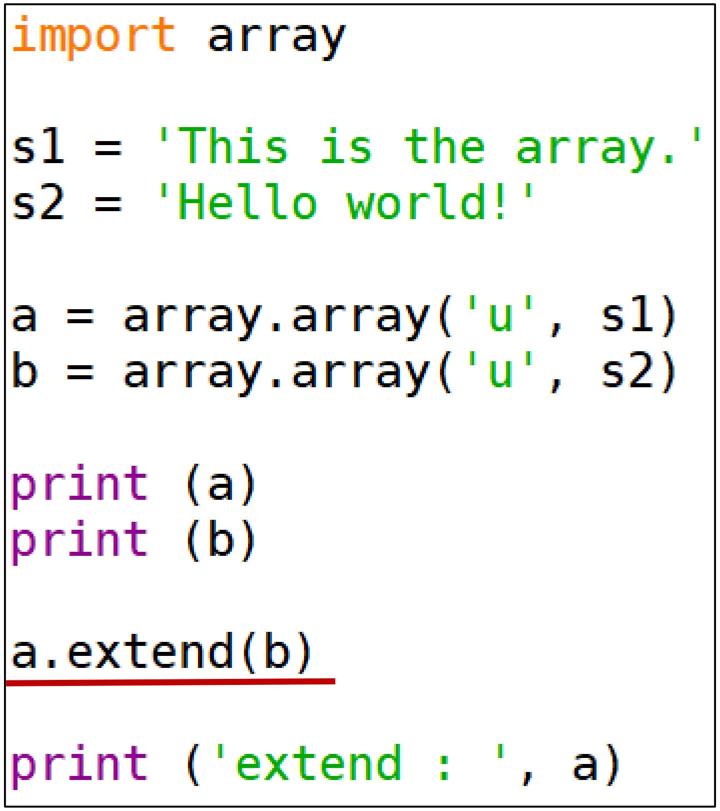
Result



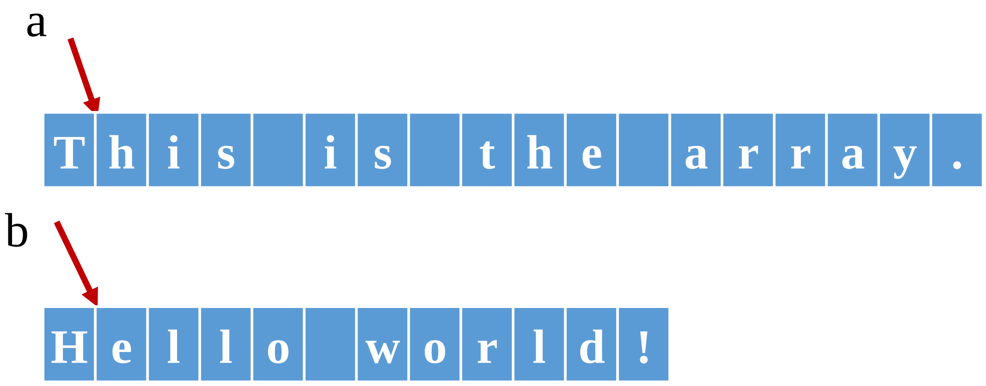
```
array('u', 'This is the array.')
 New array: array('u', 'This is the array.k')
```

“array” Module – extend( ) [3/7]

- array.**extend( )** : append items from an iterable to the end of the array
- If an iterable is another array, it must have exactly the same type



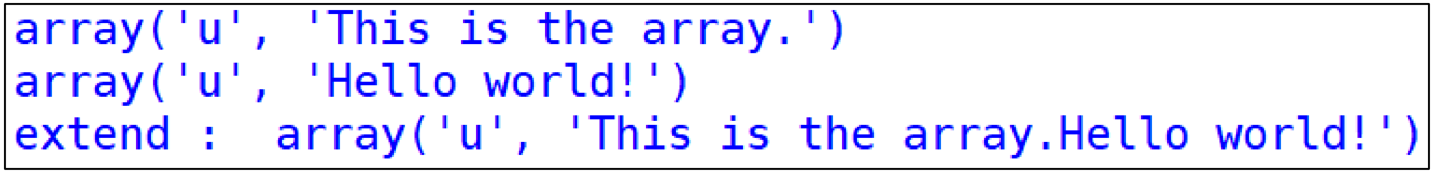
```
import array

s1 = 'This is the array.'
s2 = 'Hello world!'

a = array.array('u', s1)
b = array.array('u', s2)

print (a)
print (b)

a.extend(b)

print ('extend : ', a)
```



T	h	i	s		i	s		t	h	e		a	r	r	a	y	.

H	e	l	l	o		w	o	r	l	d	!

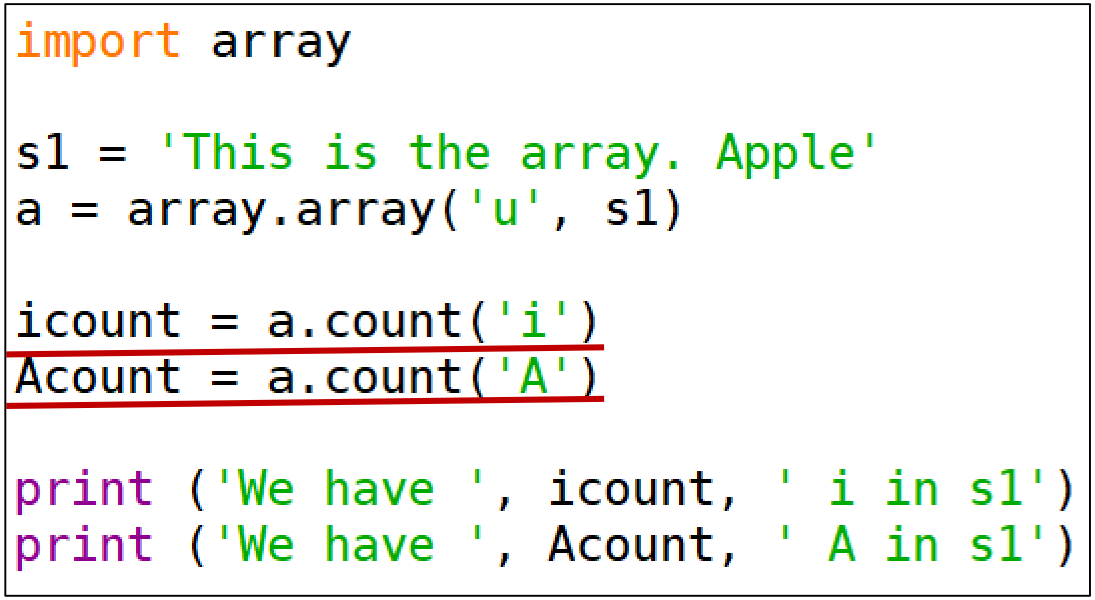
Result



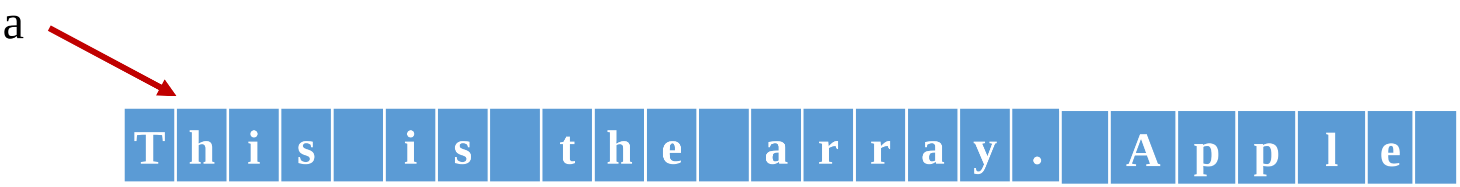
```
array('u', 'This is the array.')
array('u', 'Hello world!')
extend :  array('u', 'This is the array.Hello world!')
```

“array” Module – count( ) [4/7]

- array.**count( )** : Return the number of occurrences of x in the array

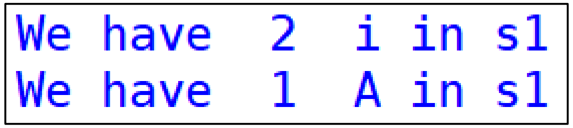


```python
import array

s1 = 'This is the array. Apple'
a = array.array('u', s1)

icount = a.count('i')
Acount = a.count('A')

print ('We have ', icount, ' i in s1')
print ('We have ', Acount, ' A in s1')
```



T	h	i	s		i	s		t	h	e		a	r	r	a	y	.		A	p	p	l	e	

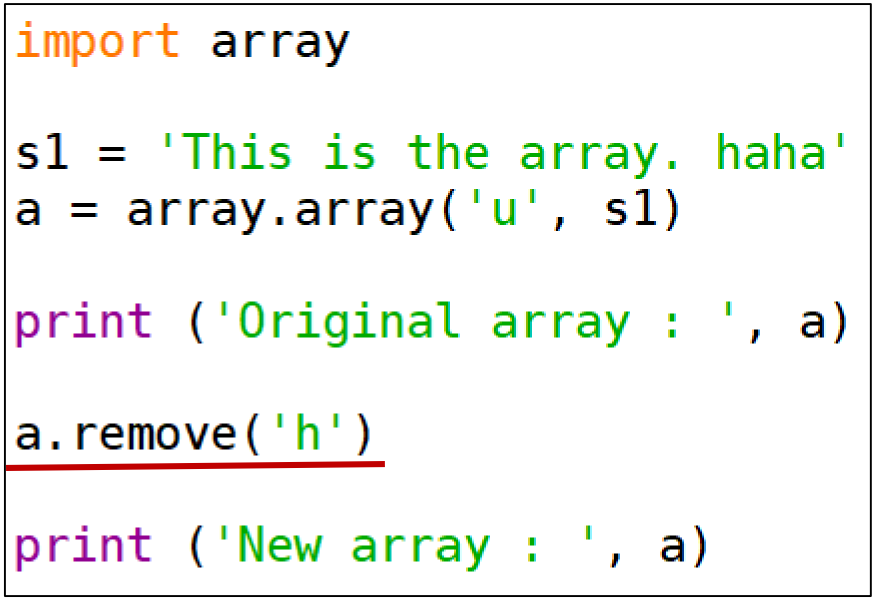
Result



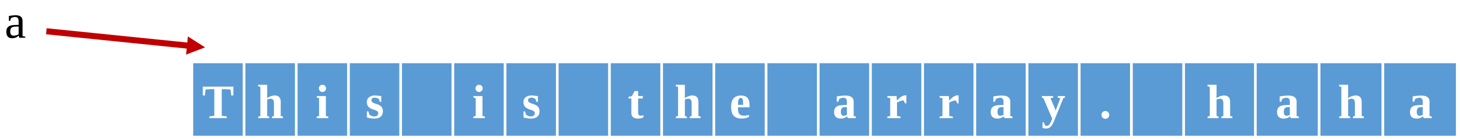
```
We have  2  i in s1
We have  1  A in s1
```

“array” Module – remove( ) [5/7]

- array.**remove**(x) : Remove the first occurrence of x from the array



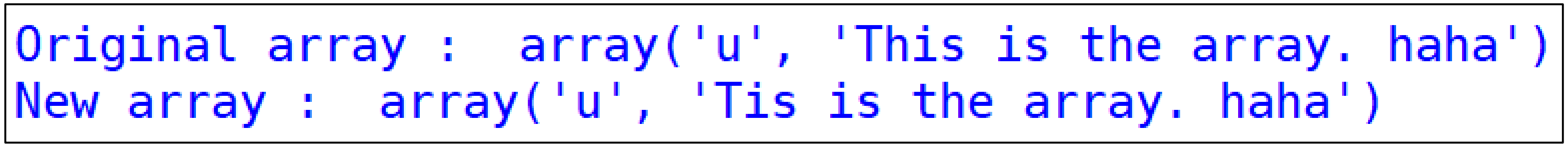
```
import array

s1 = 'This is the array. haha'
a = array.array('u', s1)

print ('Original array : ', a)

a.remove('h')

print ('New array : ', a)
```



a

T	h	i	s		i	s		t	h	e		a	r	r	a	y	.		h	a	h	a

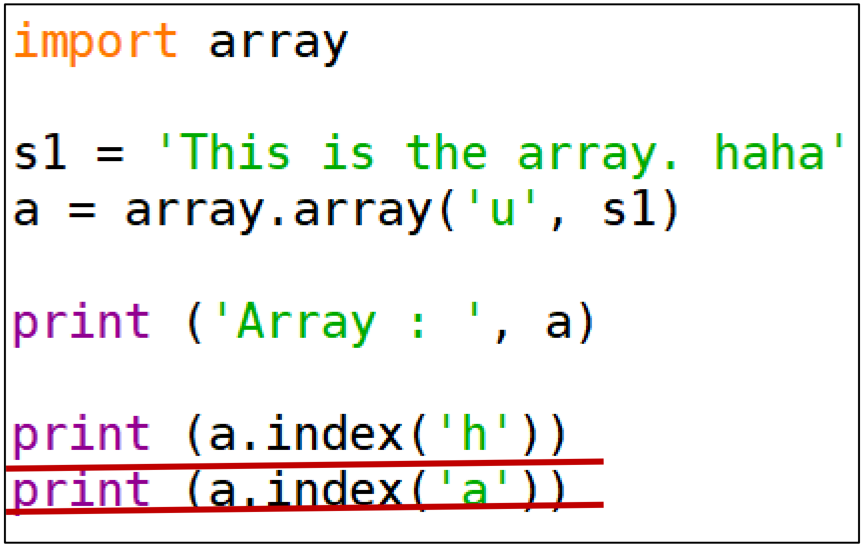
Result



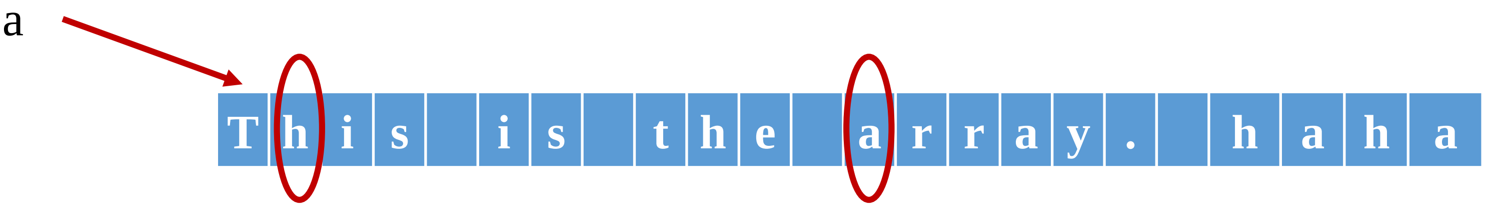
```
Original array :  array('u', 'This is the array. haha')
New array :  array('u', 'Tis is the array. haha')
```

“array” Module – index( ) [6/7]

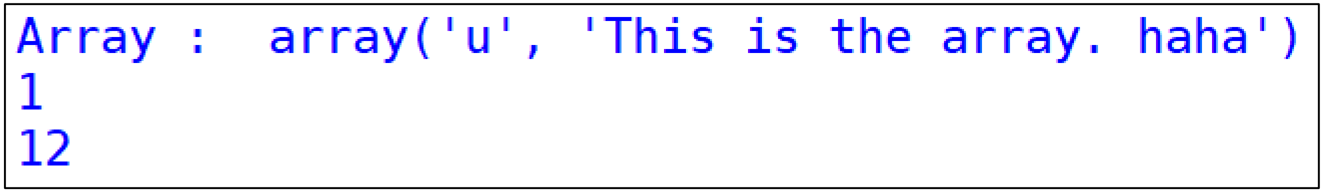
- array.**index(x)** : Return the smallest i such that i is the index of the first occurrence of x in the array



```
import array

s1 = 'This is the array. haha'
a = array.array('u', s1)

print ('Array : ', a)

print (a.index('h'))
print (a.index('a'))
```



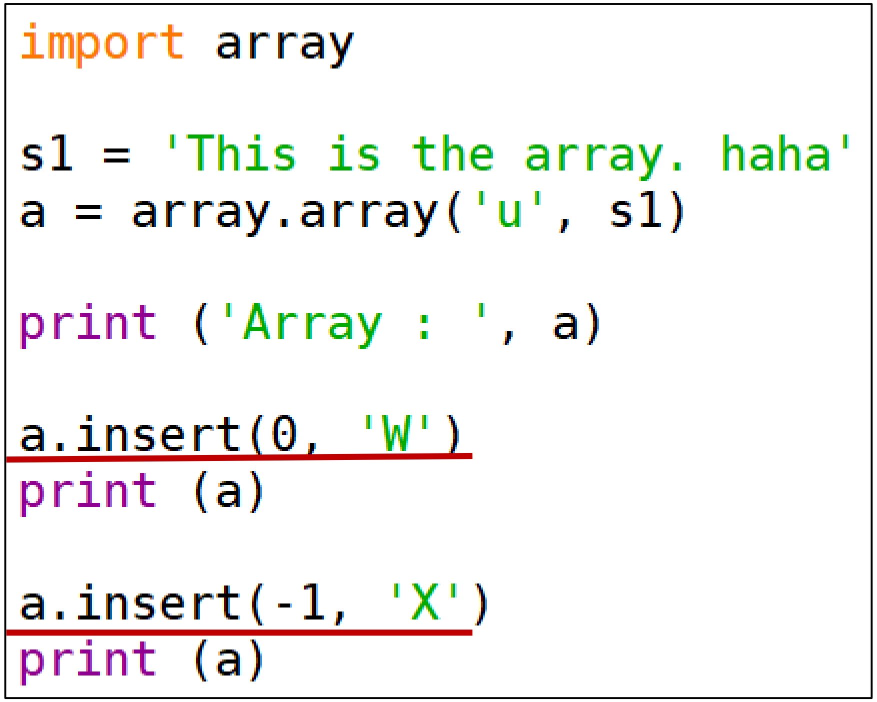
Result



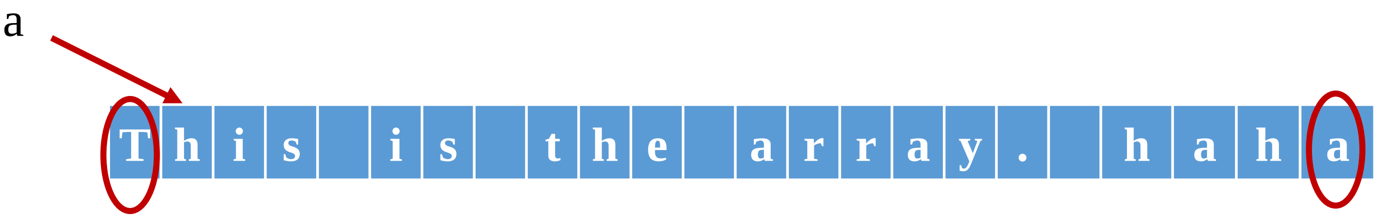
```
Array :  array('u', 'This is the array. haha')
1
12
```

“array” Module – insert( ) [7/7]

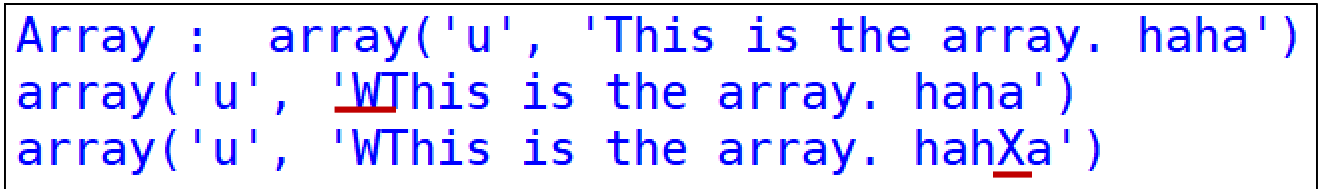
array.**insert**(i, x) : insert a new item with value x in the array before position i. Negative values are treated as being relative to the end of the array



```
import array

s1 = 'This is the array. haha'
a = array.array('u', s1)

print ('Array : ', a)

a.insert(0, 'W')
print (a)

a.insert(-1, 'X')
print (a)
```

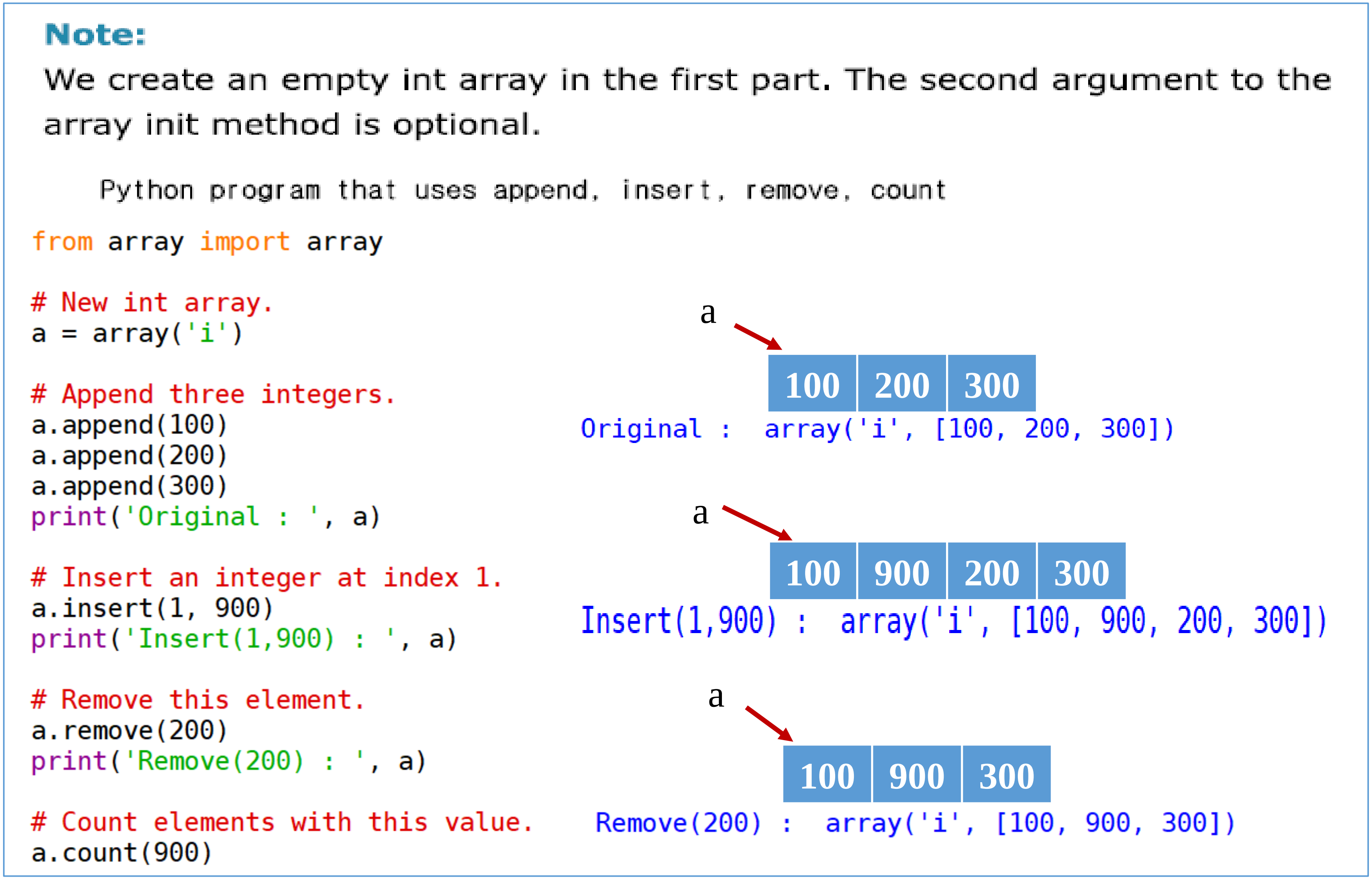


Result



```
Array :  array('u', 'This is the array. haha')
array('u', 'WThis is the array. haha')
array('u', 'WThis is the array. hahXa')
```

“array” Module Application



Note:

We create an empty int array in the first part. The second argument to the array init method is optional.

Python program that uses append, insert, remove, count

```
from array import array

# New int array.
a = array('i')

# Append three integers.
a.append(100)
a.append(200)
a.append(300)
print('Original : ', a)

# Insert an integer at index 1.
a.insert(1, 900)
print('Insert(1,900) : ', a)

# Remove this element.
a.remove(200)
print('Remove(200) : ', a)

# Count elements with this value.
a.count(900)
```

a

100	200	300

```
Original :  array('i', [100, 200, 300])
```

a

100	900	200	300

```
Insert(1,900) :  array('i', [100, 900, 200, 300])
```

a

100	900	300

```
Remove(200) :  array('i', [100, 900, 300])
```

(Ch 22) Data Structure 관련 Modules

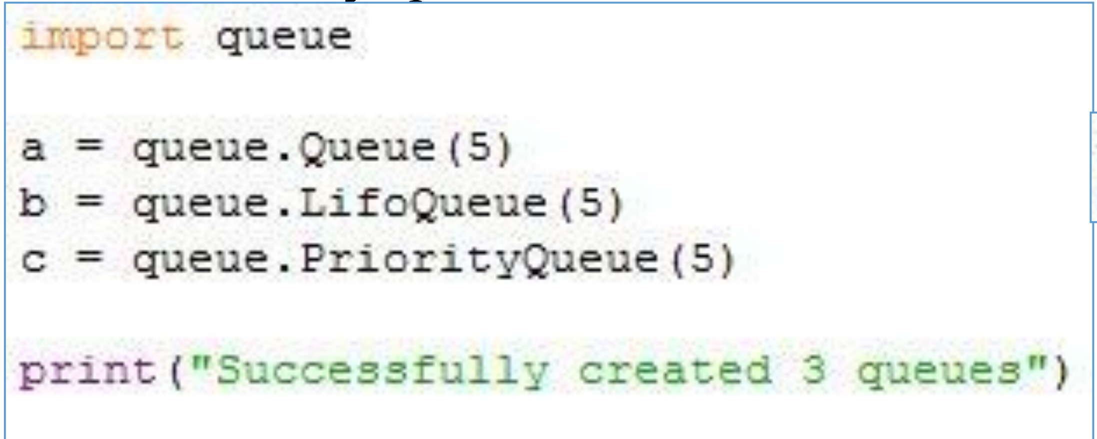
- **collections**
- **array**
- **queue**
- **heapq**
- **bisect**

“queue” Python Standard Library

List, Set, Dict같은것으로 Queue를 지원하는것은 불편하고, 세가지 대표 queue종류을 한번에 지원!!

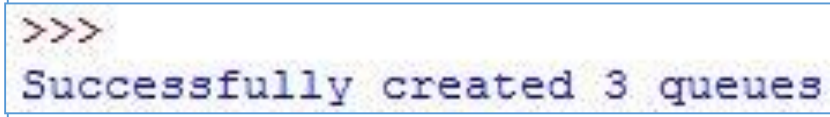
- The queue module provides a safe implementation of FIFO structure
  - Queue class implemented all the required locking semantics

- There are 3 types of Queue, which differ in the order of the entities retrieved
  - FIFO queue ➔ Queue Class
  - LIFO queue (Works like a stack) ➔ LifoQueue Class
  - Priority queue ➔ PriorityQueue Class



```
import queue

a = queue.Queue(5)
b = queue.LifoQueue(5)
c = queue.PriorityQueue(5)

print("Successfully created 3 queues")
```

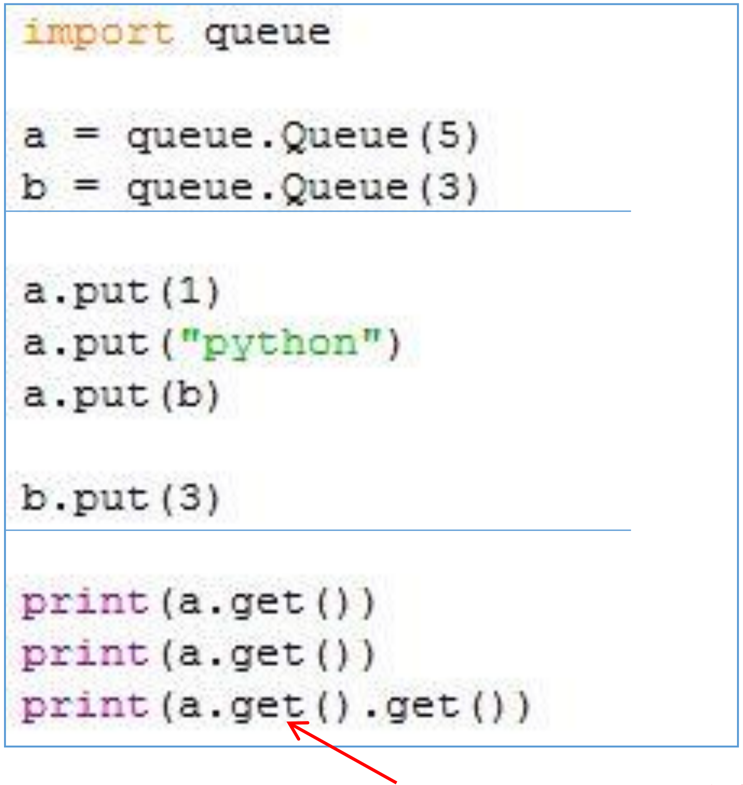
Result



```
>>>
Successfully created 3 queues
```

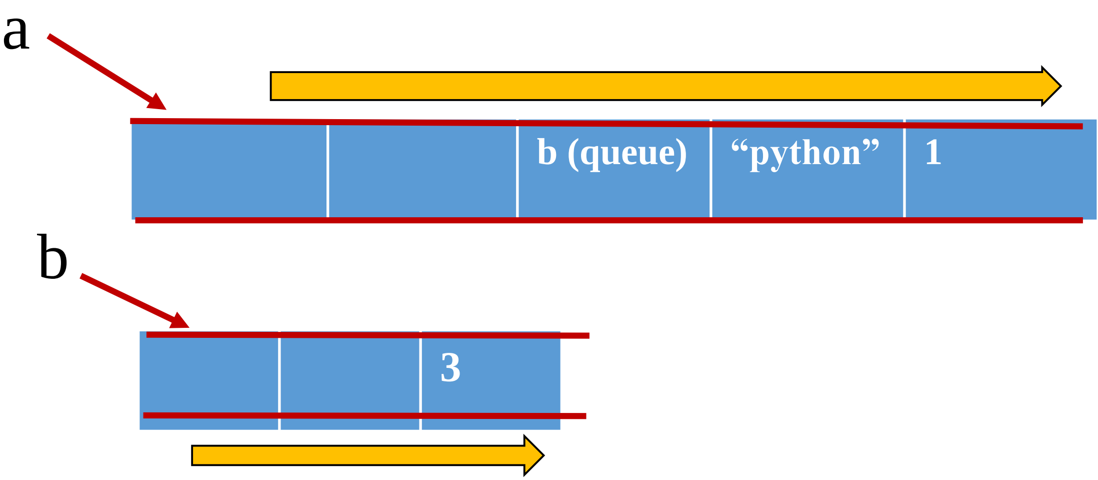
“queue” Module: Queue Class for Queue [1/2]

- queue.**Queue**(x) : Construct a FIFO queue of size ‘x’
- queue.**Queue**() : Construct a FIFO queue of infinite size
- queue.**put**(x) : Put item into the queue. Item can be anything
- queue.**get**(x) : Delete the item and return that item

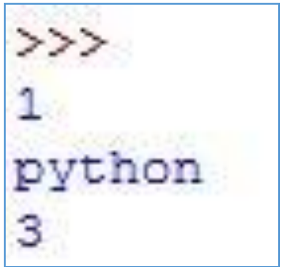


```
import queue

a = queue.Queue(5)
b = queue.Queue(3)

a.put(1)
a.put("python")
a.put(b)

b.put(3)

print(a.get())
print(a.get())
print(a.get().get())
```

Return the queue ‘b’



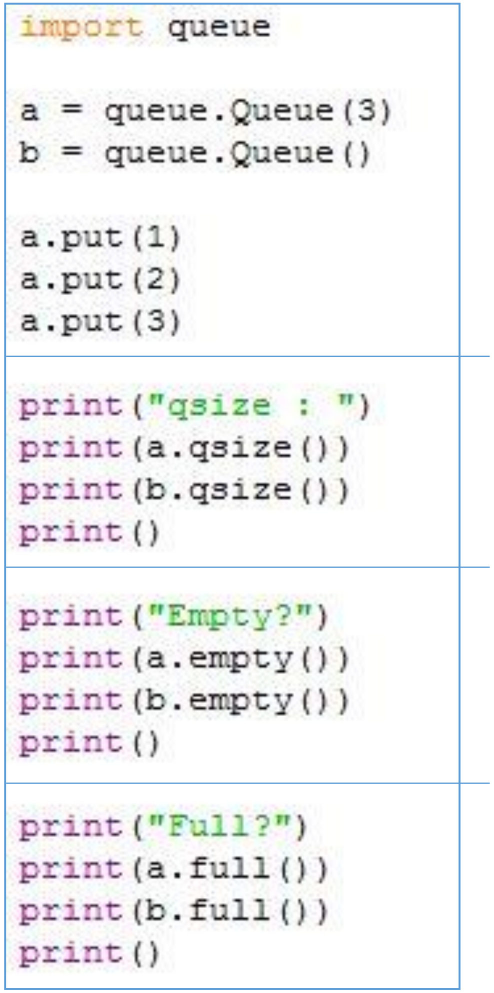
Result



```
>>>
1
python
3
```

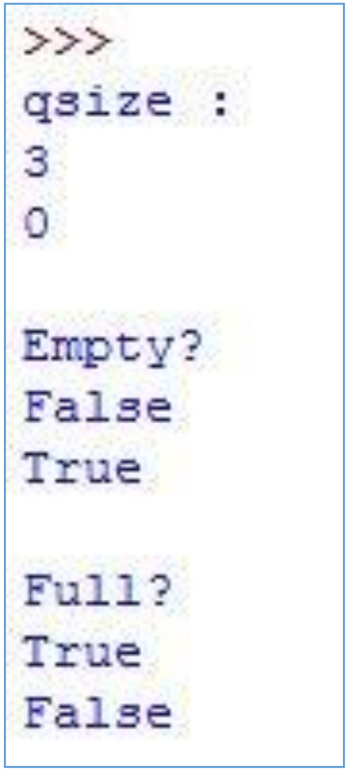
“queue” Module: Queue Class for Queue [2/2]

- queue.**qsize()** : Return the number of items in the queue
- queue.**empty()** : Return True if the queue is empty, False otherwise
- queue.**full()** : Return True if the queue is full, False otherwise

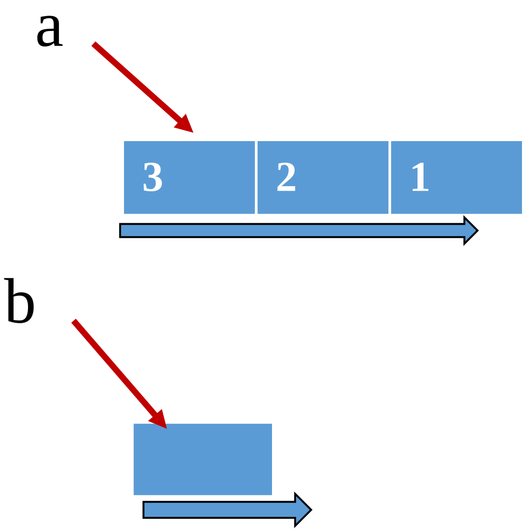


```
import queue

a = queue.Queue(3)
b = queue.Queue()

a.put(1)
a.put(2)
a.put(3)

print("qsize : ")
print(a.qsize())
print(b.qsize())
print()

print("Empty?")
print(a.empty())
print(b.empty())
print()

print("Full?")
print(a.full())
print(b.full())
print()
```

Result

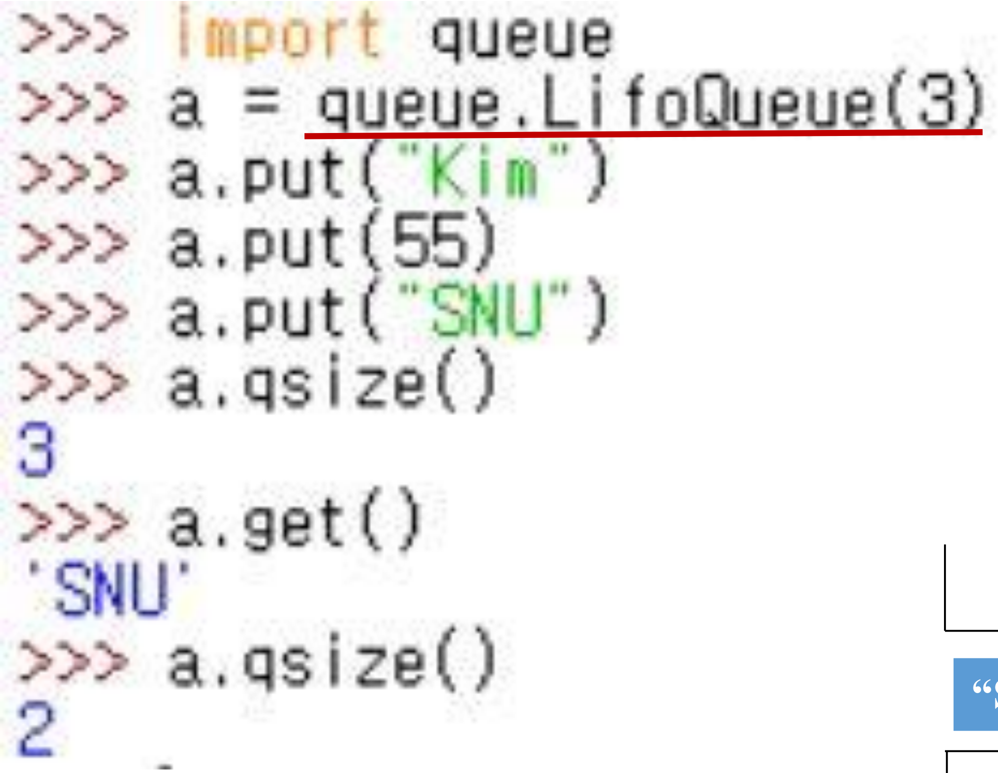


```
>>>
qsize :
3
0

Empty?
False
True

Full?
True
False
```

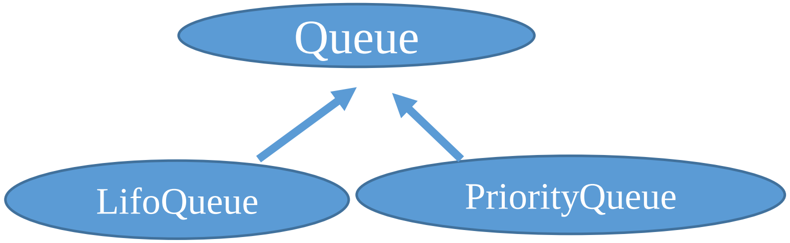


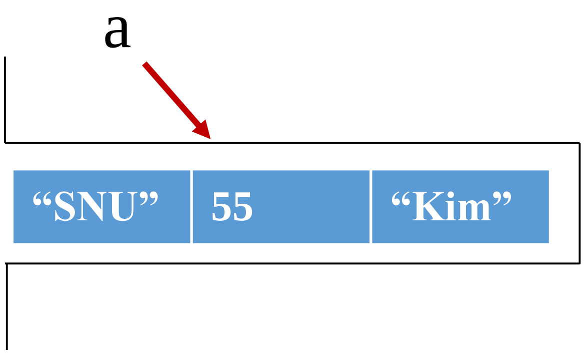
“queue” Module: LiFoQueue Class for Stack

- Subclass of Queue class
- put(x), get(x), qsize(), empty(), full() are all similar with that of Queue class



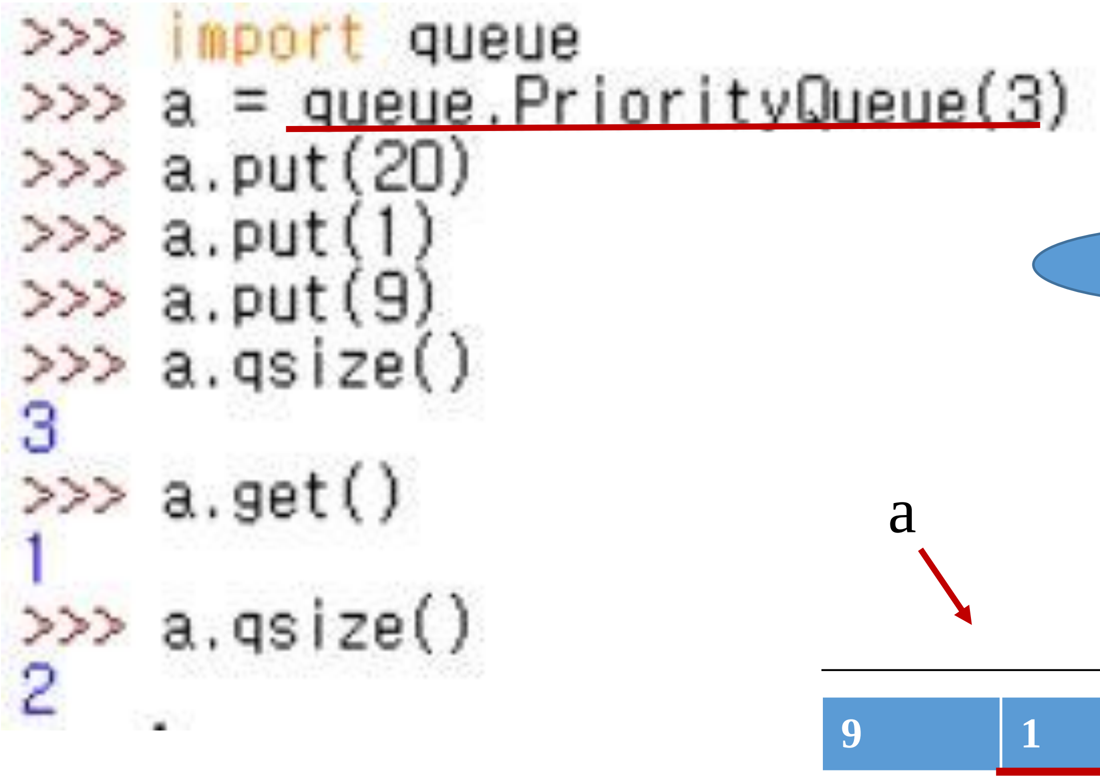
```
>>> import queue
>>> a = queue.LifoQueue(3)
>>> a.put("Kim")
>>> a.put(55)
>>> a.put("SNU")
>>> a.qsize()
3
>>> a.get()
'SNU'
>>> a.qsize()
2
```



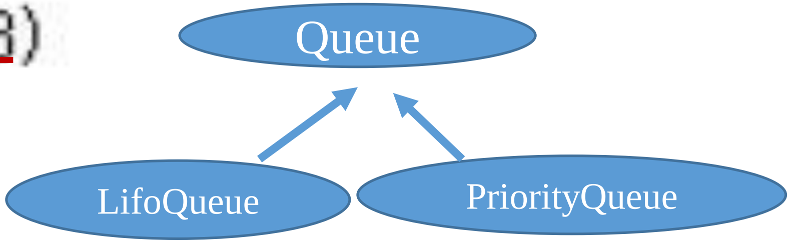


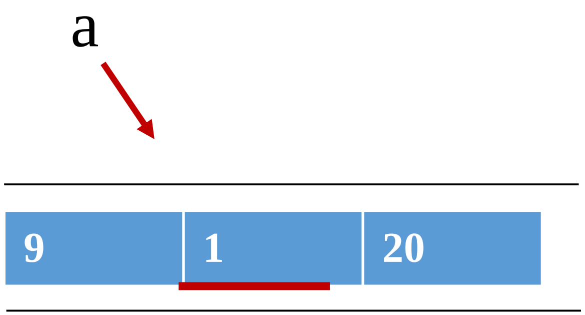
“queue” Module : PriorityQueue Class

- A subclass of Queue class, retrieves entries in priority order (lowest first)
- put(x), get(x), qsize(), empty(), full() are all similar with that of Queue class



```
>>> import queue
>>> a = queue.PriorityQueue(3)
>>> a.put(20)
>>> a.put(1)
>>> a.put(9)
>>> a.qsize()
3
>>> a.get()
1
>>> a.qsize()
2
```





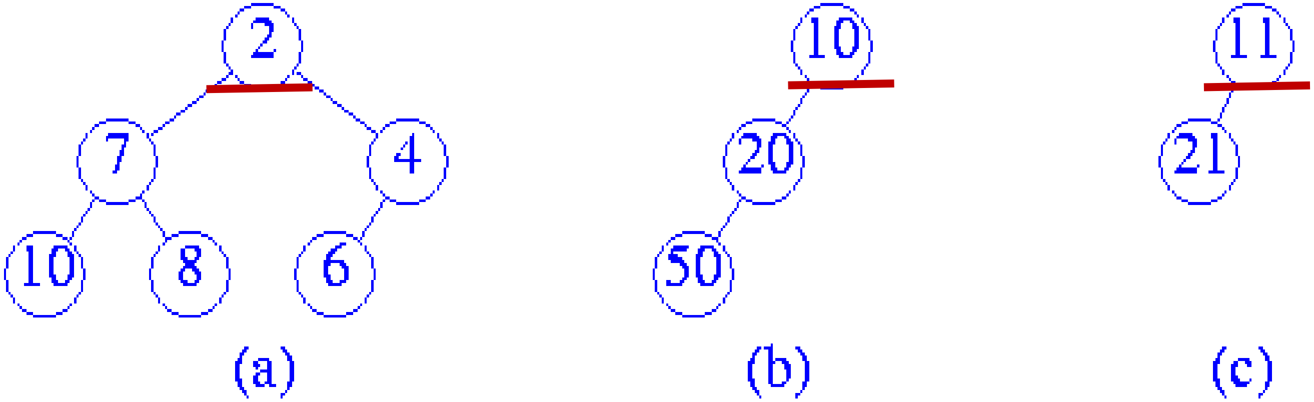
(Ch 22) Data Structure 관련 Modules

- **collections**
- **array**
- **queue**
- **heapq**
- **bisect**

“heapq” Module

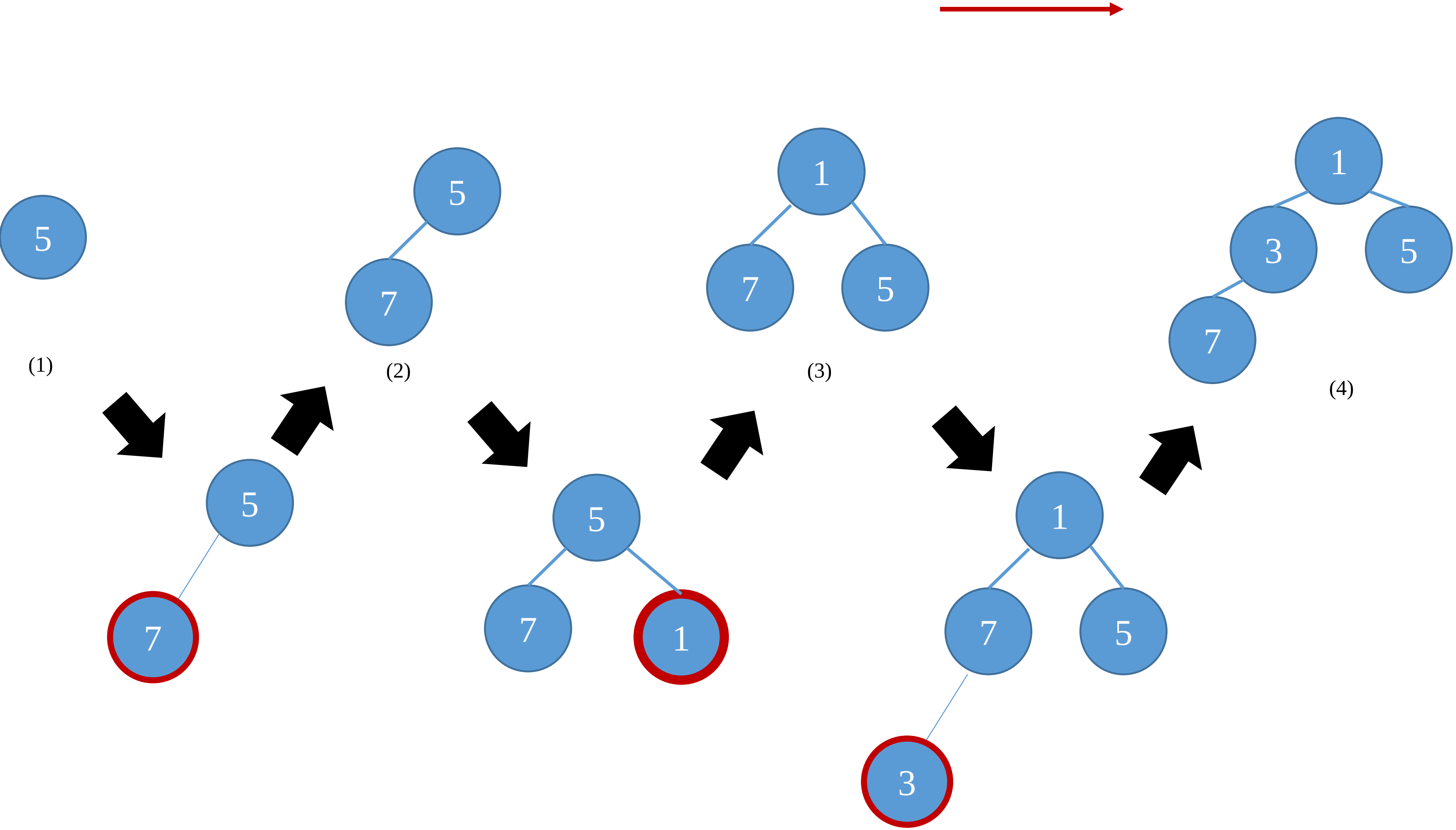
- **heapq** module provides an implementation of the **heap queue** algorithm, also known as the priority queue algorithm
  - Class는 없고, Heap 관련 function들을 지원

- **Heaps** are **complete binary trees** for which every parent node has a value less than or equal to any of its children
  - Sometimes called, Min Heap or Priority Queue
  - smallest element is always the root, heap[0]



Constructing Heap

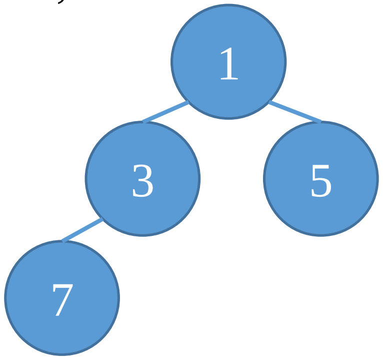
- Suppose the data is arrived in the following sequence [5, 7, 1, 3]



“heapq” module – Functions [1/4]

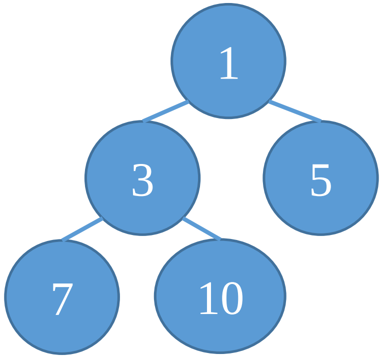
- heapq.**heapify**(*x*): transform list x into a heap, in-place, in linear time

```
>>> from heapq import *
>>> qdata = [5, 7, 1, 3]
>>> heapify(qdata)
>>> qdata
[1, 3, 5, 7]
```



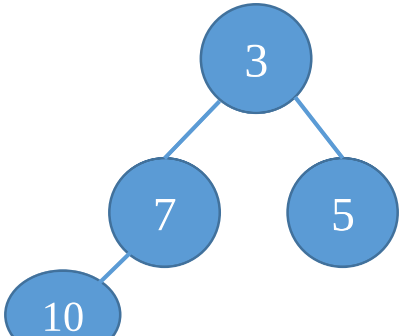
- heapq.**heappush**(*heap*, *item*): push the value item onto the heap

```
>>> heappush(qdata, 10)
>>> qdata
[1, 3, 5, 7,10]
```



- heapq.**heappop**(*heap*): pop and return the smallest item from the heap

```
>>> heappop(qdata)
>>> qdata
[3, 7, 5, 10]
```



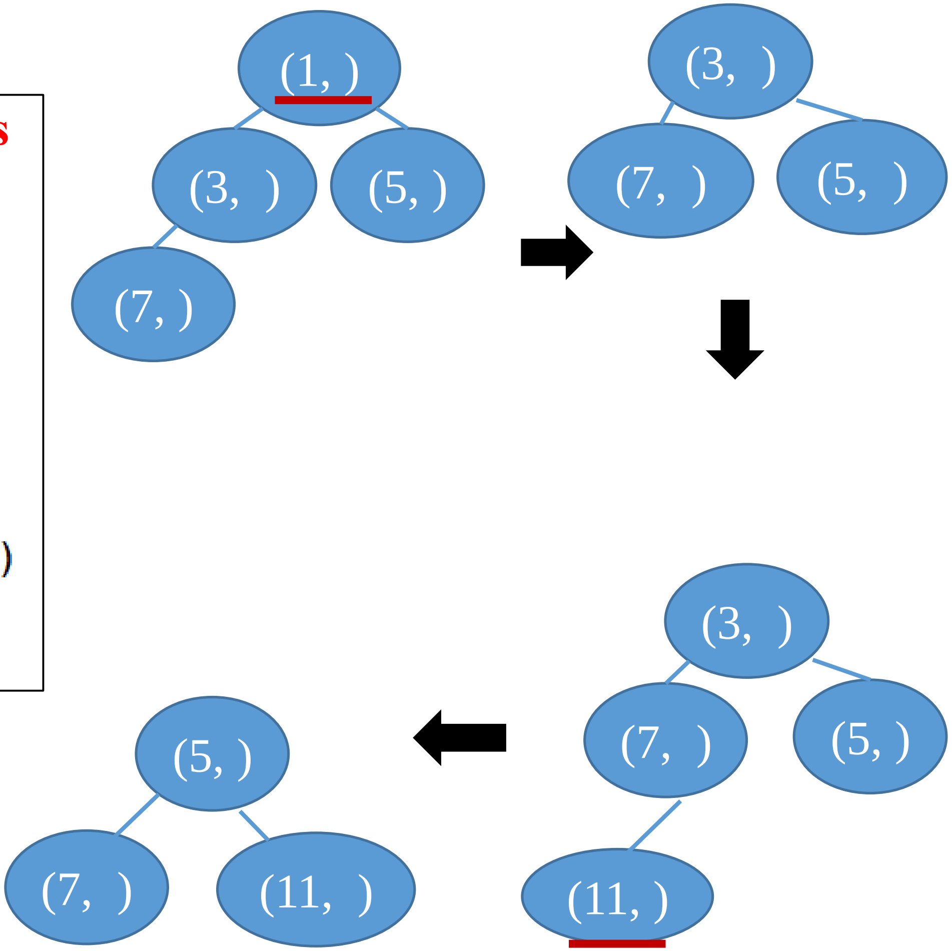
“heapq” module – Functions [2/4]

- heapq.**heappushpop**(*heap*, *item*): push item on the heap, then pop and return the smallest item from the heap.

source

```
from heapq import *

h = []
heappush(h, (5, 'write code'))
heappush(h, (7, 'release product'))
heappush(h, (1, 'write sepc'))
heappush(h, (3, 'create tests'))

print(heappop(h))
print(heappushpop(h, (11, 'push pop')))

print(h)
```

Uses Python’s Default List

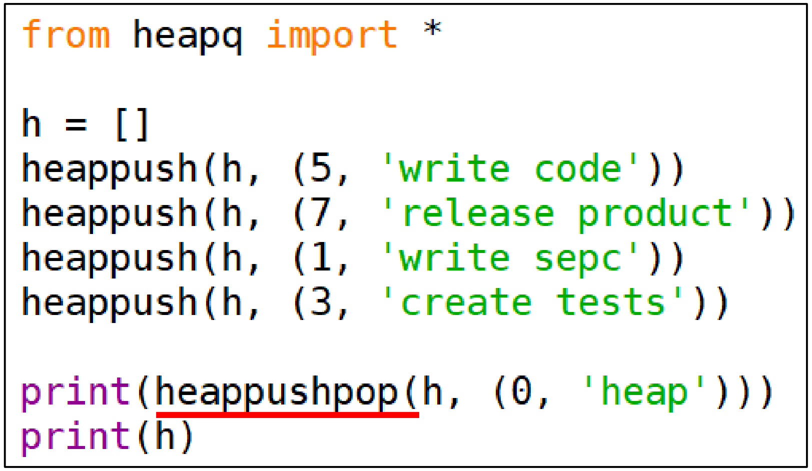


result

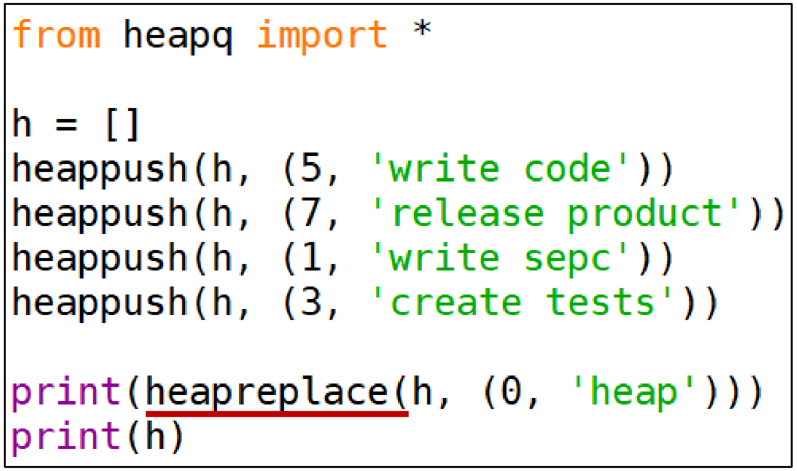
```
(1, 'write sepc')
(3, 'create tests')
[(5, 'write code'), (7, 'release product'), (11, 'push pop')]
```

“heapq” module – Functions   [3/4]

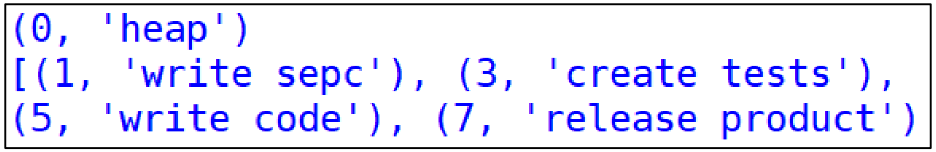
- heapq.**heapreplace**(*heap, item*): pop and return the smallest item from the heap, and also push the new item



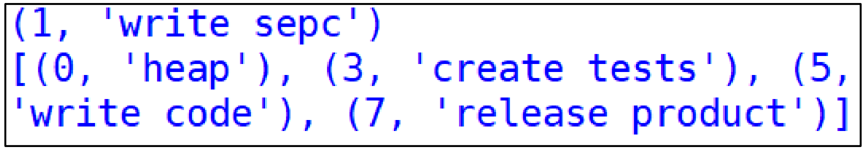
```
from heapq import *

h = []
heappush(h, (5, 'write code'))
heappush(h, (7, 'release product'))
heappush(h, (1, 'write sepc'))
heappush(h, (3, 'create tests'))

print(heappushpop(h, (0, 'heap')))
print(h)
```



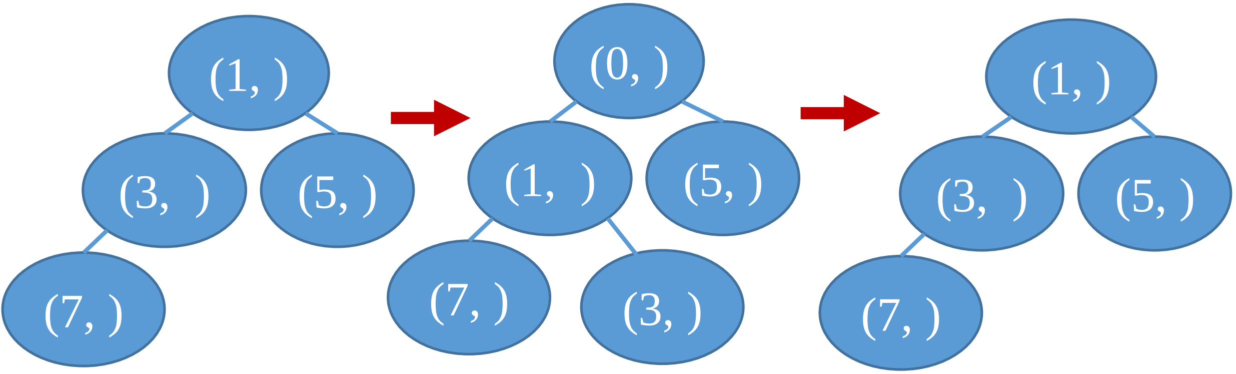
```
from heapq import *

h = []
heappush(h, (5, 'write code'))
heappush(h, (7, 'release product'))
heappush(h, (1, 'write sepc'))
heappush(h, (3, 'create tests'))

print(heapreplace(h, (0, 'heap')))
print(h)
```

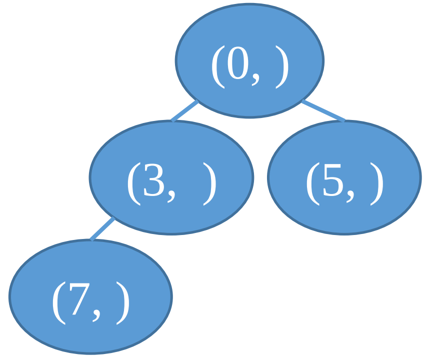


```
(0, 'heap')
[(1, 'write sepc'), (3, 'create tests'),
(5, 'write code'), (7, 'release product')
```



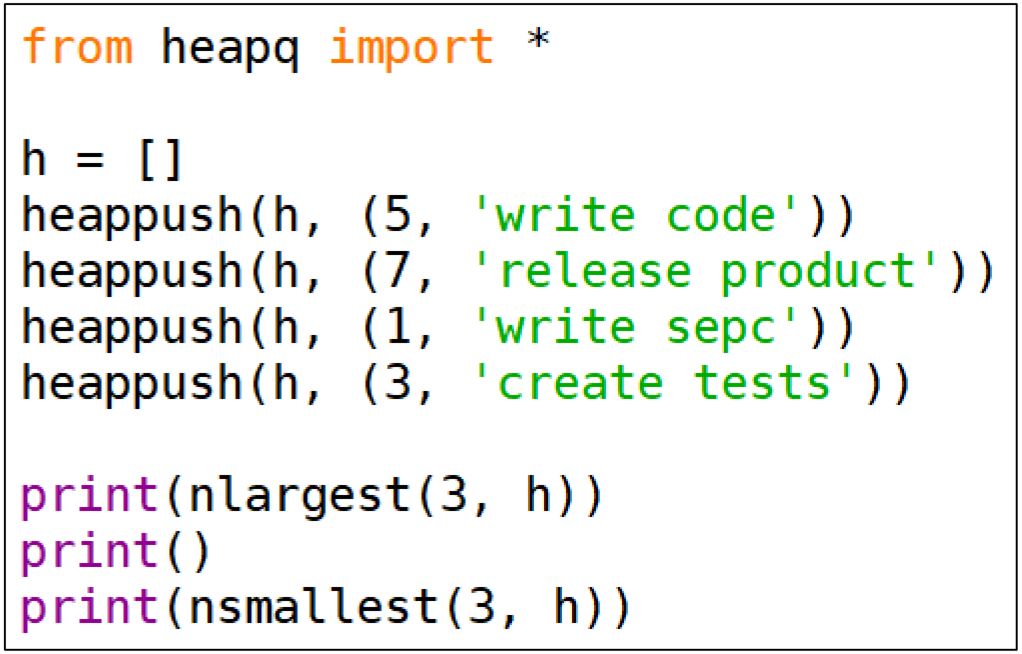
```
(1, 'write sepc')
[(0, 'heap'), (3, 'create tests'), (5,
'write code'), (7, 'release product')]
```



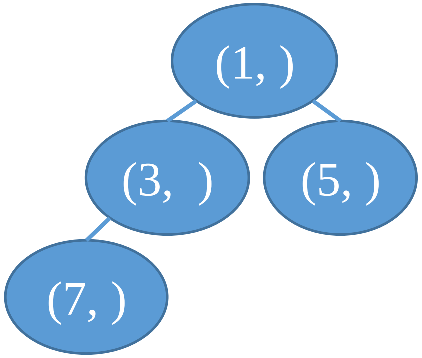


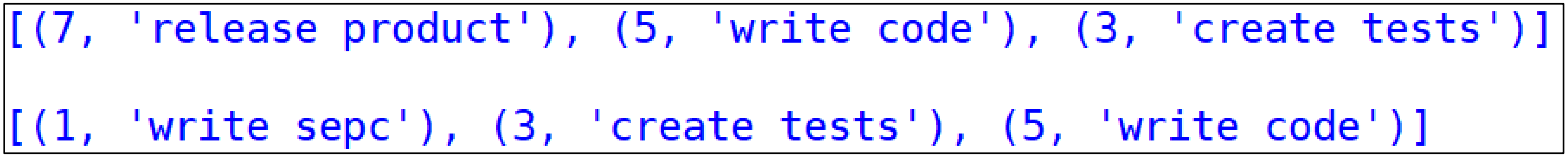
“heapq” module – Functions [4/4]

- heapq.**nlargest**(*n, iterable, key=None*): return a list with the n largest elements from the dataset defined by *iterable*
- heapq.**nsmallest**(*n, iterable, key=None*): return a list with the n smallest elements from the dataset defined by *iterable*



```python
from heapq import *

h = []
heappush(h, (5, 'write code'))
heappush(h, (7, 'release product'))
heappush(h, (1, 'write sepc'))
heappush(h, (3, 'create tests'))

print(nlargest(3, h))
print()
print(nsmallest(3, h))
```





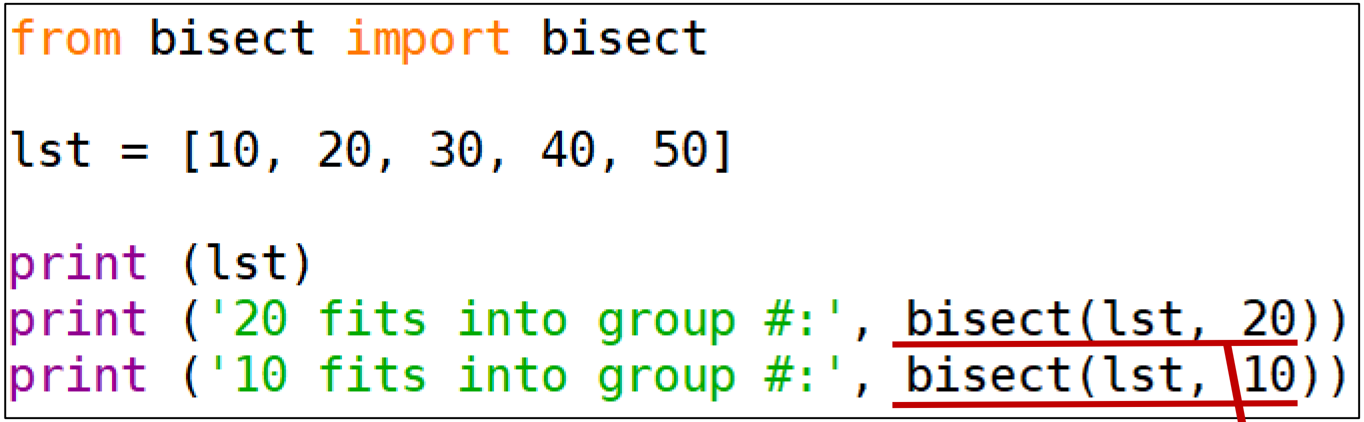
```
[(7, 'release product'), (5, 'write code'), (3, 'create tests')]

[(1, 'write sepc'), (3, 'create tests'), (5, 'write code')]
```

(Ch 22) Data Structure 관련 Modules

- **collections**
- **array**
- **queue**
- **heapq**
- **bisect**

“bisect” Module

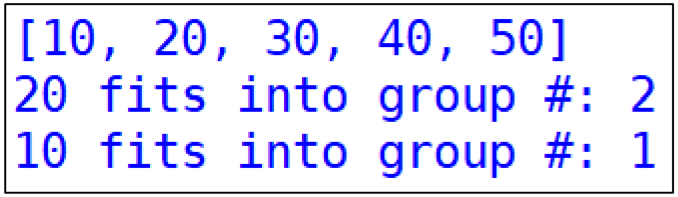
- The bisect(list_of_breakpoints, point) function is generally useful for categorizing point (numeric data) among the list of breakpoint
  - It uses a basic bisection algorithm to do its work
- For long lists of items with expensive comparison operations, this can be an improvement over the more common approach



```
from bisect import bisect

lst = [10, 20, 30, 40, 50]

print (lst)
print ('20 fits into group #:', bisect(lst, 20))
print ('10 fits into group #:', bisect(lst, 10))
```

bisect( ) returns

50 ~ +∞ :	→	5
40 ~ 49 :	→	4
30 ~ 39 :	→	3
20 ~ 29 :	→	2
10 ~ 19 :	→	1
-∞ ~ 9 :	→	0

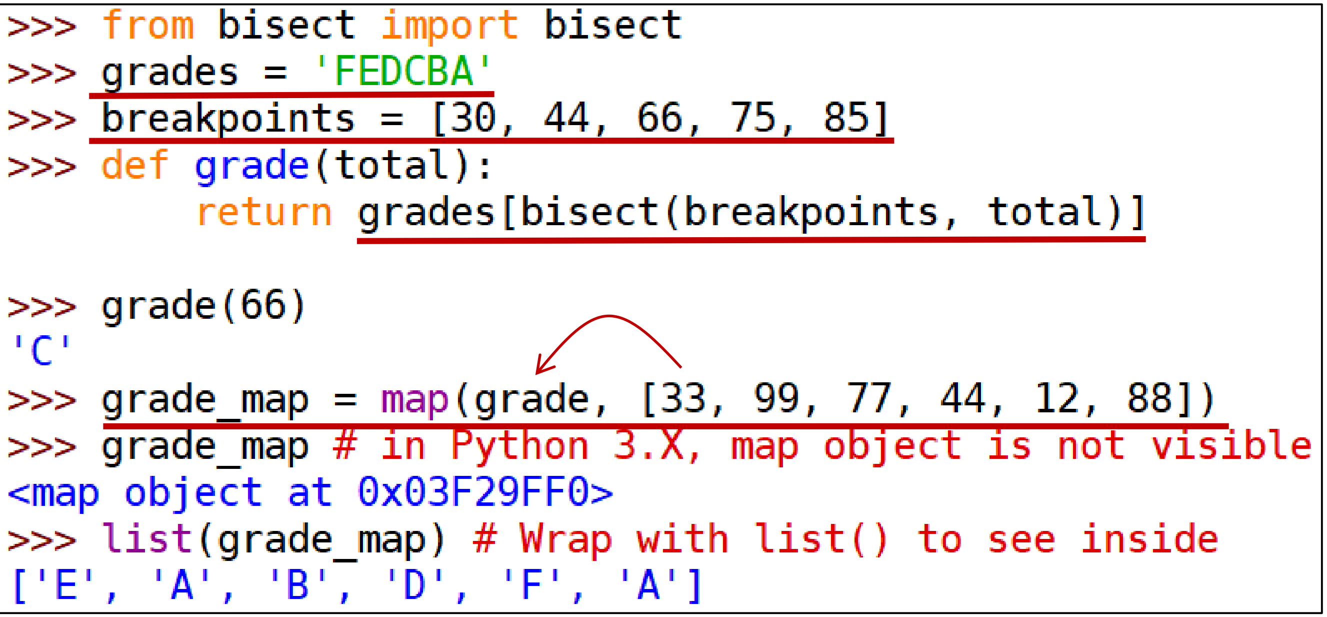


```
[10, 20, 30, 40, 50]
20 fits into group #: 2
10 fits into group #: 1
```

categorizing 20 among lst

“bisect” Module – bisect( ) [1/3]

- bisect.**bisect**(*a*, *x*, *lo=0*, *hi=len(a)*)
  - The returned insertion point i partitions the array a into two halves so that all **(val <= x for val in a[lo:i])** for the left side and all **(val > x for val in a[i:hi])** for the right side.



```
>>> from bisect import bisect
>>> grades = 'FEDCBA'
>>> breakpoints = [30, 44, 66, 75, 85]
>>> def grade(total):
        return grades[bisect(breakpoints, total)]

>>> grade(66)
'C'
>>> grade_map = map(grade, [33, 99, 77, 44, 12, 88])
>>> grade_map # in Python 3.X, map object is not visible
<map object at 0x03F29FF0>
>>> list(grade_map) # Wrap with list() to see inside
['E', 'A', 'B', 'D', 'F', 'A']
```

grades = “FEDCBA”

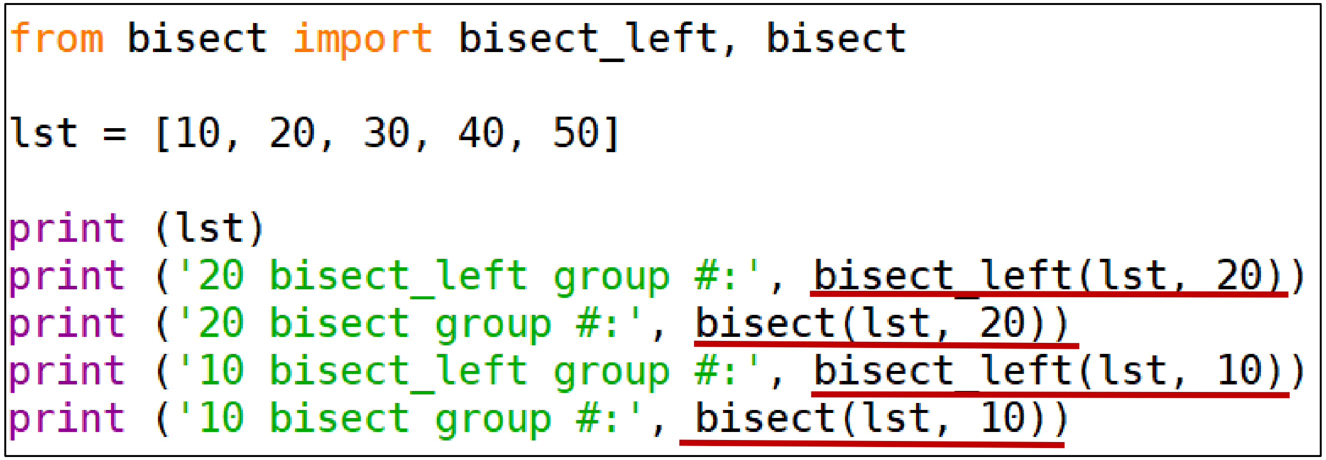
grades[0] ➔ F
grades[1] ➔ E
grades[2] ➔ D
grades[3] ➔ C
grades[4] ➔ B
grades[5] ➔ A

bisect( ) returns

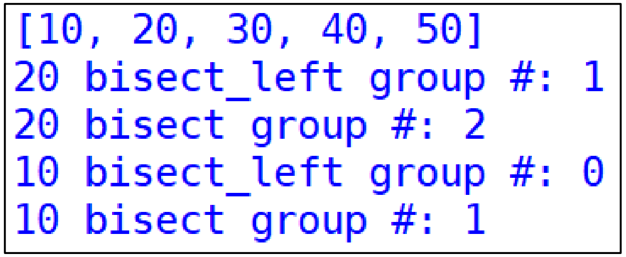
80 ~ +∞ :	➔	5	(A)
75 ~ 84 :	➔	4	(B)
66 ~ 74 :	➔	3	(C)
44 ~ 65 :	➔	2	(D)
30 ~ 43 :	➔	1	(E)
-∞ ~ 29 :	➔	0	(F)

“bisect” Module – bisect_left( ) [2/3]

- bisect.**bisect_left**(*a*, *x*, *lo=0*, *hi=len(a)*)
  - Locate the insertion point for *x* in *a* to maintain sorted order
  - The returned insertion point *i* partitions the array a into two halves so that **all (val < x for val in a[*lo*:*i*])** for the left side and **all (val >= x for val in a[*i*:*hi*])** for the right side



```python
from bisect import bisect_left, bisect

lst = [10, 20, 30, 40, 50]

print (lst)
print ('20 bisect_left group #:', bisect_left(lst, 20))
print ('20 bisect group #:', bisect(lst, 20))
print ('10 bisect_left group #:', bisect_left(lst, 10))
print ('10 bisect group #:', bisect(lst, 10))
```



```
[10, 20, 30, 40, 50]
20 bisect_left group #: 1
20 bisect group #: 2
10 bisect_left group #: 0
10 bisect group #: 1
```

bisect

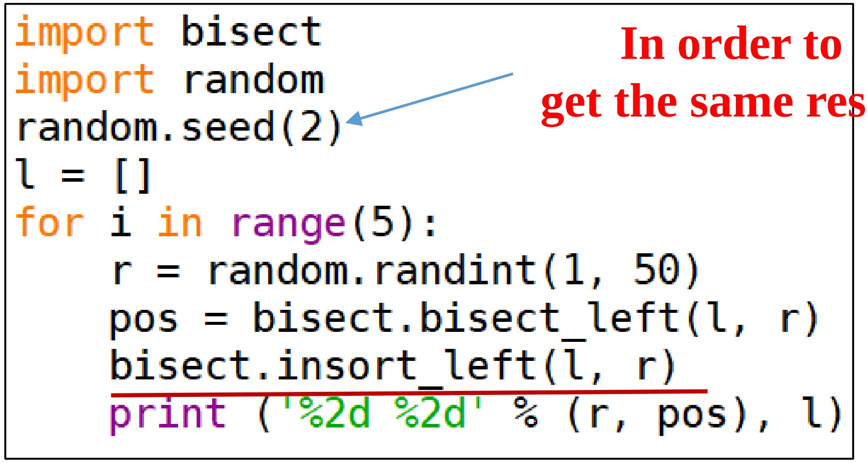
50 ~ +∞ :	➔	5
40 ~ 49 :	➔	4
30 ~ 39 :	➔	3
20 ~ 29 :	➔	2
10 ~ 19 :	➔	1
-∞ ~ 9 :	➔	0

bisect_left

51 ~ +∞ :	➔	5
41 ~ 50 :	➔	4
31 ~ 40 :	➔	3
21 ~ 30 :	➔	2
11 ~ 20 :	➔	1
-∞ ~ 10 :	➔	0

“bisect” Module – insort( ) [3/3]

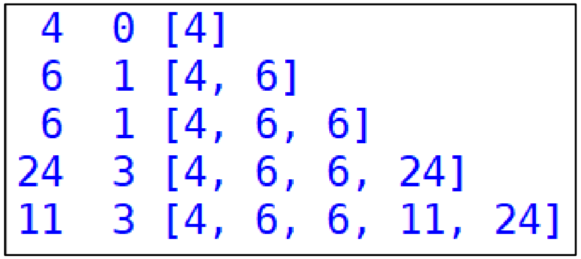
- bisect.**insort_left**(*a, x, lo=0, hi=len(a)*): insert *x* into *a* in sorted order
- bisect.**insort**(*a, x, lo=0, hi=len(a)*): Similar to insort_left(), but inserting *x* into *a* after any existing entries of *x*.



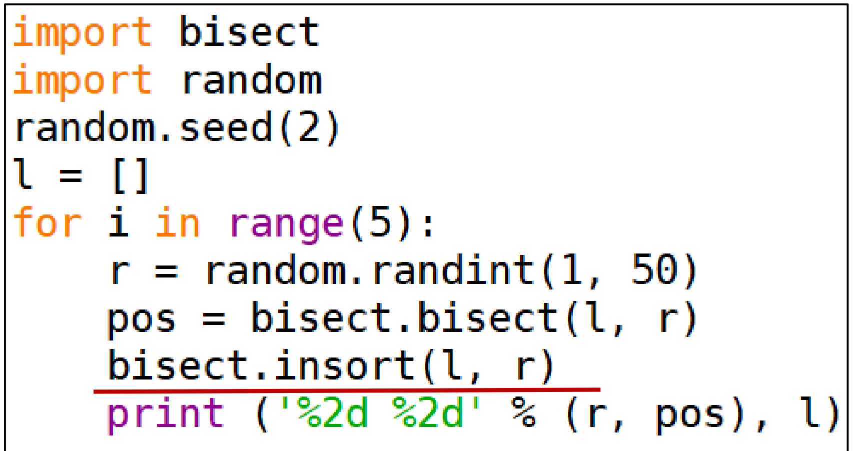
```
import bisect
import random
random.seed(2)
l = []
for i in range(5):
    r = random.randint(1, 50)
    pos = bisect.bisect_left(l, r)
    bisect.insort_left(l, r)
    print ('%2d %2d' % (r, pos), l)
```

In order to get the same result

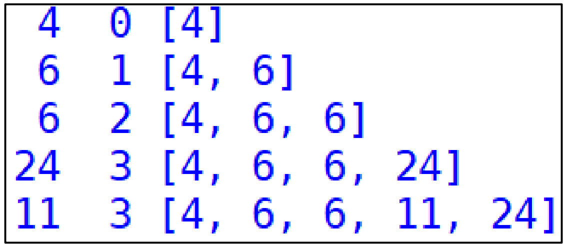
l → [ ]



```
 4  0 [4]
 6  1 [4, 6]
 6  1 [4, 6, 6]
24  3 [4, 6, 6, 24]
11  3 [4, 6, 6, 11, 24]
```



```
import bisect
import random
random.seed(2)
l = []
for i in range(5):
    r = random.randint(1, 50)
    pos = bisect.bisect(l, r)
    bisect.insort(l, r)
    print ('%2d %2d' % (r, pos), l)
```



```
 4  0 [4]
 6  1 [4, 6]
 6  2 [4, 6, 6]
24  3 [4, 6, 6, 24]
11  3 [4, 6, 6, 11, 24]
```