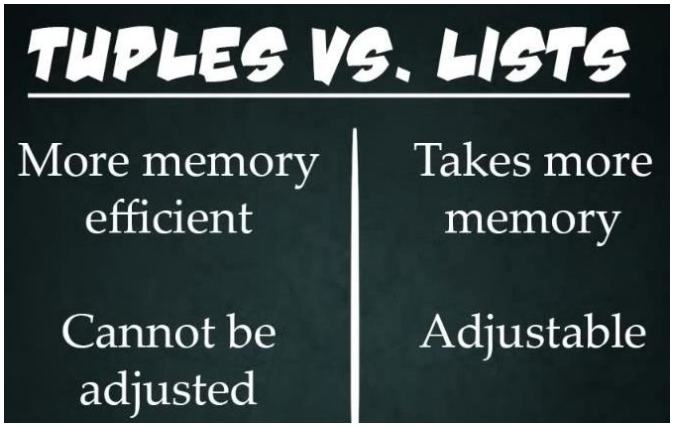
Ch 7: Collection Data Types in Python

- Tuple
- Set
- Dictionary
- One More Small Thing about * and **

Tuple in Python

```
>>> t1 = ()
>>> t2 = (1,)
>>> t3 = (1, 2, 3)
>>> t4 = 1, 2, 3
>>> t5 = ('a', 'b', ('ab', 'cd'))
```

- Tuple is similar to list
  But, Tuple is *immutable*



1. 튜플 요소값 삭제 시 오류

```
>>> t1 = (1, 2, 'a', 'b')
>>> del t1[0]
Traceback (innermost last):
File "", line 1, in ?del t1[0]
TypeError: object doesn't support item deletion
```

2. 튜플 요소값 변경 시 오류

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0] = 'c'
Traceback (innermost last):
File "", line 1, in ?t1[0] = 'c'
TypeError: object doesn't support item assignment
```

Tuple Operations [1/3] Applying Basic Operators

Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.

Python Expression	Results	Description
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print(x)	1 2 3	Iteration

튜플 더하기

```
>>> t1 = (1, 2, 'a', 'b')
>>> t2 = (3, 4)
>>> t1 + t2
(1, 2, 'a', 'b', 3, 4)
```

튜플 곱하기

```
>>> t2 = (3, 4)
>>> t2 * 3
(3, 4, 3, 4, 3, 4)
```

Tuple Operations [2/3] Indexing and Slicing

Because tuples are sequences, indexing and slicing work the same way for tuples as they do for strings. Assuming following input –

```
L = ('spam', 'Spam', 'SPAM!')
```

Python Expression	Results	Description
L[2]	'SPAM!'	Offsets start at zero
L[-2]	'Spam'	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

인덱싱하기

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0]
1
>>> t1[3]
'b'
```

슬라이싱하기

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[1:]
(2, 'a', 'b')
```

Tuple Operations [3/3] Applying Built-in Functions

len(tuple)

Gives the total length of the tuple.

max(tuple)

Returns item from the tuple with max value.

min(tuple)

Returns item from the tuple with min value.

tuple(seq)

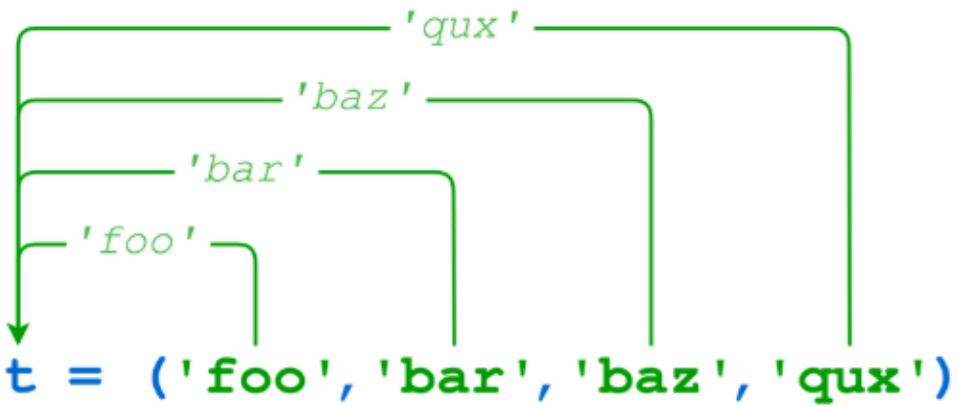
Converts a list into tuple.

```
In [9]: t1 = (1, 2)

In [10]: t2 = (2, 3)

In [11]: len(t1)
Out[11]: 2

In [12]: max(t2)
Out[12]: 3

In [13]: min(t1)
Out[13]: 1

In [14]: tuple([3, 2, 1])
Out[14]: (3, 2, 1)

In [15]: tuple({2, 5, 8})
Out[15]: (8, 2, 5)
```

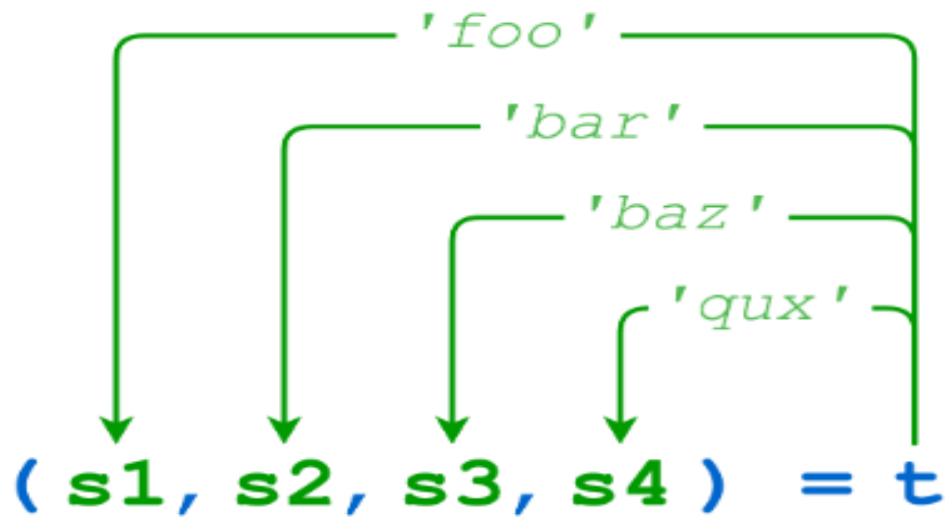
List의 built-in function: pop( ), append( ), insert( ), remove( ), append( ), extend( ) 등등은 Tuple 에 적용되지 않는다!

Tuple Assignment, Packing and Unpacking

- Tuple Assignment = Packing items into an Object



- Unpacking items from an Object



(Ch8) Collection Data Types in Python

- Tuple
- Set
- Dictionary
- One More Small Thing about * and **

Set in Python

Quick Example

```
s = set([2,3,5])
print(3 in s)          # prints True
print(4 in s)          # prints False
for x in range(7):
    if (x not in s):
        print(x)       # prints 0 1 4 6
```

Create an empty set

```
s = set()
print(s)      # prints set()
```

Create a set from a list

```
s = set(["cat", "cow", "dog"])
print(s)     # prints {'cow', 'dog', 'cat'}
```

Create a set from any iterable object

```
s = set("wahoo")
print(s)      # surprised?
```

Create a statically-allocated set

```
s = { 2, 3, 5 }
print(s)    # prints { 2, 3, 5 }
```

Caution: { } is not an empty set!

Empty set ➔ set()

```
s = { }
print(type(s) == set)  # False!
print(type(s))         # This is a dict (we'll learn about those soon)
```

Sets are Unordered

```
s = set([2,4,8])
print(s)           # prints {8, 2, 4} in standard Python
for element in s: # prints 8, 2, 4
    print(element)
```

Elements are Unique

```
s = set([2,2,2])
print(s)           # prints {2}
print(len(s))      # prints 1
```

This is Python set: (only immutable data!)
{ 3, 4, "Kim"}

The followings are not Python set
{4, 9, [3, 4]}
{ 3, 7, {5, 10}}

Set elements must be hashable!

Elements Must Be Immutable

```
a = ["lists", "are", "mutable"]
s = set([a])        # TypeError: unhashable type: 'list'
print(s)
```

```
a = ["lists", "are", "mutable"]
s = set(a)
print(a)
```

This is OK

Another example:

```
s1 = set(["sets", "are", "mutable", "too"])
s2 = set([s1])      # TypeError: unhashable type: 'set'
print(s)
```

Sets are Very Efficient

Comparing Membership-Checking Performance in List and Set

```
# 0. Preliminaries
import time
n = 1000

# 1. Create a list [2,4,6,...,n] then check for membership
# among [1,2,3,...,n] in that list.

# don't count the list creation in the timing
a = list(range(2,n+1,2))

print("Using a list... ", end="")
start = time.time()
count = 0
for x in range(n+1):
    if x in a:
        count += 1
end = time.time()
elapsed1 = end - start
print("count=", count," and time = %0.4f seconds" % elapsed1)
```

```
# 2. Repeat, using a set
print("Using a set.... ", end="")
start = time.time()
s = set(a)
count = 0
for x in range(n+1):
    if x in s:
        count += 1
end = time.time()
elapsed2 = end - start
print("count=", count," and time = %0.4f seconds" % elapsed2)
print("With n=%d, sets ran about %0.1f times faster than lists!" %
      (n, elapsed1/elapsed2))
print("Try a larger n to see an even greater savings!")
```

Length of Set & Copy of Set

Operation	Result
len(s)	cardinality (size) of set s

Example

```
s = { 2, 3, 2, 4, 3 }
print(len(s))
```

s.copy()	new set with a shallow copy of s

```
s = { 1, 2, 3 }
t = s.copy()
s.add(4)
print(s)
print(t)
```

S → { 1, 2, 3}

T → {1, 2, 3}

S → { 1, 2, 3}

T → {1, 2, 3, 4}

Set Membership Checking

Operation	Result	Example

x in s	test x for membership in s	`s = { 1, 2, 3 }`
		`print(0 in s)`
		`print(1 in s)`

x not in s	test x for non-membership in s	`s = { 1, 2, 3 }`
		`print(0 not in s)`
		`print(1 not in s)`

Single Set Update Functions [1/2]

s.pop()	remove and return an arbitrary element from s; raises KeyError if empty

```
s = { 2, 4, 8 }
print(s.pop())  # unpredictable!
print(s)
```

s.clear()	remove all elements from set s

```
s = { 1, 2, 3 }
s.clear()
print(s, len(s))
```

Single Set Update Functions [2/2]

s.add(x)	add element x to set s

```
s = { 1, 2, 3 }
print(s, 4 in s)
s.add(4)
print(s, 4 in s)
```

s.remove(x)	remove x from set s; raises KeyError if not present

```
s = { 1, 2, 3 }
print(s, 3 in s)
s.remove(3)
```

s.discard(x)	removes x from set s if present

```
s = { 1, 2, 3 }
print(s, 3 in s)
s.discard(3)
print(s, 3 in s)
s.discard(3) # does not crash!
print(s, 3 in s)
```

Subset Checking & Superset Checking

s.issubset(t)	s <= t

Test whether every element in s is in t

```
print({1,2} <= {1},      {1,2}.issubset({1}))
print({1,2} <= {1,2},    {1,2}.issubset({1,2}))
print({1,2} <= {1,2,3}, {1,2}.issubset({1,2,3}))
```

s.issuperset(t)	s >= t

Test whether every element in t is in s

```
print({1,2} >= {1},      {1,2}.issuperset({1}))
print({1,2} >= {1,2},    {1,2}.issuperset({1,2}))
print({1,2} >= {1,2,3}, {1,2}.issuperset({1,2,3}))
```

Dual Set Functions : No Change on S and T [1/2]

s.union(t)	s | t

New set with elements from both s and t

```
print({1,2} | {1},      {1,2}.union({1}))
print({1,2} | {1,3},    {1,2}.union({1,3}))
s = {1,2}
t = s | {1,3}
print(s, t)
```

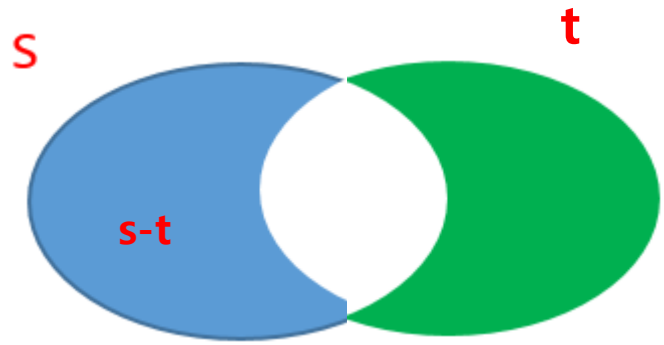
s.intersection(t)	s & t

New set with elements common to s and t

```
print({1,2} & {1},      {1,2}.intersection({1}))
print({1,2} & {1,3},    {1,2}.intersection({1,3}))
s = {1,2}
t = s & {1,3}
print(s, t)
```

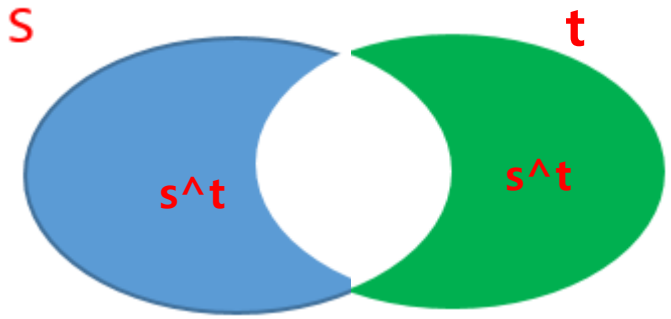
Dual Set Functions : No Change on S and T [2/21]

s.difference(t)	s - t



New set with elements from in s but not in t

```
print({1,2} - {1},      {1,2}.difference({1}))
print({1,2} - {1,3},    {1,2}.difference({1,3}))
s = {1,2}
t = s - {1,3}
print(s, t)
```



s.symmetric_difference(t)	s ^ t

New set with elements in either s or t but not both

```
print({1,2} ^ {1},      {1,2}.symmetric_difference({1}))
print({1,2} ^ {1,3},    {1,2}.symmetric_difference({1,3}))
s = {1,2}
t = s ^ {1,3}
print(s, t)
```

Dual Set Update Functions : Update on S [1/2]

s.update(t)	s |= t	modify s adding all elements found in t

```
s = {1,2}
t = {1,3}
u = {2,3}
s.update(u)
t |= u
print(s, t, u)
```

```
In [4]: s.update(u)

In [5]: s
Out[5]: {1, 2, 3}

In [6]: t |= u

In [7]: t
Out[7]: {1, 2, 3}
```

s.update(t)

=

s = s.union(t)

s.intersection_update(t)	s &= t	modify s keeping only elements also found in t

```
s = {1,2}
t = {1,3}
u = {2,3}
s.intersection_update(u)
t &= u
print(s, t, u)
```

```
In [11]: s.intersection_update(u)

In [12]: s
Out[12]: {2}

In [13]: t &= u

In [14]: t
Out[14]: {3}
```

Dual Set Update Functions : Update on S [2/2]

s.difference_update(t)	s -= t	modify s removing all elements found in t

```
s = {1,2}
t = {1,3}
u = {2,3}
s.difference_update(u)
t -= u
print(s, t, u)
```

```
In [18]: s.difference_update(u)

In [19]: s
Out[19]: {1}

In [20]: t -= u

In [21]: t
Out[21]: {1}
```

s.symmetric_difference_update(t)	s ^= t

Modify s keeping elements from s or t but b=not both

```
s = {1,2}
t = {1,3}
u = {2,3}
s.symmetric_difference_update(u)
t ^= u
print(s, t, u)
```

```
In [25]: s.symmetric_difference_update(u)

In [26]: s
Out[26]: {1, 3}

In [27]: t ^= u

In [28]: t
Out[28]: {1, 2}
```

(Ch 8) Collection Data Types in Python

- Tuple
- Set
- Dictionary
- One More Small Thing about * and **

Dictionary Data Type

```
>>> dic = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
```

key	value
name	pey
phone	01199993323
birth	1118

```
>>> dic['name']
'pey'
>>> dic['phone']
'0119993323'
>>> dic['birth']
'1118'
```

Insert and Delete in Dictionary

```
>>> a = {1: 'a'}
>>> a[2] = 'b'
>>> a
{2: 'b', 1: 'a'}
```

```
>>> a['name'] = 'pey'
{'name':'pey', 2: 'b', 1: 'a'}
```

```
>>> a[3] = [1,2,3]
{'name': 'pey', 3: [1, 2, 3], 2: 'b', 1: 'a'}
```

```
>>> del a[1]
>>> a
{'name': 'pey', 3: [1, 2, 3], 2: 'b'}
```

Dictionary 만들 때 주의사항

중복되는 Key 값은 금지

```
>>> a = {1:'a', 1:'b'}
>>> a
{1: 'b'}
```

Key 값은 immutable value만 허락

```
>>> a = {[1,2] : 'hi'}
Traceback (most recent call last):
File "", line 1, in ?
TypeError: unhashable type
```

Dictionary Example in Python

Quick Example

```
stateMap = { 'pittsburgh':'PA', 'chicago':'IL', 'seattle':'WA', 'boston':'MA' }
city = input("Enter a city name --> ").lower()
if (city in stateMap):
    print(city.title(), "is in", stateMap[city])
else:
    print("Sorry, never heard of it.")
```

Another Example:

```
counts = dict()
while True:
    n = int(input("Enter an integer (0 to end) --> "))
    if (n == 0): break
    if (n in counts):
        counts[n] += 1
    else:
        counts[n] = 1
    print("I have seen", n, "a total of", counts[n], "time(s)")
print("Done, counts:", counts)
```

Another Dictionary Example

```
>>> sam = {}
>>> sam["weapon"] = "chainsaw"
>>> sam["health"] = 10
>>> sam
{'weapon': 'chainsaw', 'health': 10}
>>> sam["weapon"]
'chainsaw'
>>> del sam["health"]
>>> sam
{'weapon': 'chainsaw'}
>>>
```

```
>>> myDict = { "key1": 10, "key2": 20, "key5": 45}
>>> myDict["key8"] = 60          # add a "key8:60" pair
>>> myDict["key2"]               # retrieve the value part of key2
>>> del myDict["key5"]           # delete the "key5:data5" pair
```

Creating Dictionary [1/3]

Create an empty dictionary

```
d = dict()
print(d)    # prints {}
```

Create an empty dictionary using braces syntax

```
d = { }
print(d)    # prints {}
```

Create a dictionary from a list of (key, value) pairs

```
pairs = [("cow", 5), ("dog", 98), ("cat", 1)]
d = dict(pairs)
print(d)    # unpredictable order!
```

Statically-allocate a dictionary

```
d = { "cow":5, "dog":98, "cat":1 }
print(d)    # ditto!
```

Creating Dictionary [2/3]

Dictionaries Map Keys to Values

```
ages = dict()
key = "fred"
value = 38
ages[key] = value  # "fred" is the key, 38 is the value
print(ages[key])
```

Keys are unordered

```
d = dict()
d[2] = 100
d[4] = 200
d[8] = 300
print(d)  # unpredictable order
```

Keys are unique

```
d = dict()
d[2] = 100
d[2] = 200
d[2] = 400
print(d)  # { 2:400 }
```

Creating Dictionary [3/3]

Keys must be immutable

```
d = dict()
a = [1] # lists are mutable, so...
d[a] = 42 # Error: unhashable type: 'list'
```

Values are Unrestricted

```
# values may be mutable
d = dict()
a = [1,2]
d["fred"] = a
print(d["fred"])
a += [3]
print(d["fred"]) # sees change in a!

# but keys may not be mutable
d[a] = 42       # TypeError: unhashable type: 'list'
```

Extracting Keys and Values From Dictionary [1/2]

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
```

Key 리스트 만들기(keys)

```
>>> a.keys()
dict_keys(['name', 'phone', 'birth'])
```

a.keys()는 딕셔너리 a의 Key만을 모아서 dict_keys라는 객체를 리턴한다.

Value 리스트 만들기(values)

```
>>> a.values()
dict_values(['pey', '0119993323', '1118'])
```

a.values( ) 는 disctionary a의 value만을 모아서 dict_values라는 객체를 리턴

Extracting Keys and Values From Dictionary [2/2]

dict_keys 객체는 다음과 같이 사용할 수 있다. 리스트를 사용하는 것과 차이가 없지만, 리스트 고유의 함수인 append, insert, pop, remove, sort등의 함수를 수행할 수는 없다.

```
>>> for k in a.keys():
...     print(k)
...
phone
birth
name
```

dict_keys 객체를 리스트로 변환하려면 다음과 같이 하면 된다.

```
>>> list(a.keys())
['phone', 'birth', 'name']
```

Dealing with Key-Value Items in Dictionary

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
```

Key, Value 쌍 얻기(items)

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

items 함수는 key와 value의 쌍을 튜플로 묶은 값을 dict_items 객체로 돌려준다.

Key: Value 쌍 모두 지우기(clear)

```
>>> a.clear()
>>> a
{}
```

Getting Value using Key in Dictionary

Key로 Value얻기(get)

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> a.get('name')
'pey'
>>> a.get('phone')
'0119993323'
```

해당 Key가 딕셔너리 안에 있는지 조사하기(in)

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> 'name' in a
True
>>> 'email' in a
False
```

Dictionary Function Review [1/3]

Operation	Result	Example
len(d)	the number of items (key-value pairs) in dictionary d	`d = { 1:[1,2,3,4,5], 2:"abcd" }` `print(len(d))`

d.copy()	new dictionary with a shallow copy of d	`d1 = { 1:"a" }`
		`d2 = d1.copy()`
		`d1[2] = "b"`
		`print(d1)`
		`print(d2)`

d.popitem()	remove and return an arbitrary (key,value) pair from d; raises KeyError if empty	`d = { 1:"a", 2:"b" }`
		`print(d.popitem())  # unpredictable`
		`print(d)`

d.clear()	remove all items from dictionary d	`d = { 1:"a", 2:"b" }`
		`d.clear()`
		`print(d, len(d))`

Dictionary Function Review [2/3]

for key in d	Iterate over all keys in d.	`d = { 1:"a", 2:"b" }`
		`for key in d:`
		`  print(key, d[key])`

key in d	test if d has the given key	`d = { 1:"a", 2:"b" }`
		`print(0 in d)`
		`print(1 in d)`
		`print("a" in d) # surprised?`

key not in d	test if d does not have the given key	`d = { 1:"a", 2:"b" }`
		`print(0 not in d)`
		`print(1 not in d)`
		`print("a" not in d)`

d[key]	the item of d with the given key. Raises a KeyError if key is not in the map.	`d = { 1:"a", 2:"b" }`
		`print(d[1])`
		`print(d[3]) # crash!`

Dictionary Function Review [3/3]

d[key] = value	set d[key] to value.	`d = { 1:"a", 2:"b" }`
		`print(d[1])`
		`d[1] = 42`
		`print(d[1])`

get(key[,default])	the value for key if key is in the dictionary, else default (or None if no default is provided).	`d = { 1:"a", 2:"b" }`
		`print(d.get(1))` `# works like d[1] here`
		`print(d.get(1, 42))` `# default is ignored`
		`print(d.get(0))` `# doesn't crash!`
		`print(d.get(0, 42))` `# default is used`

del d[key]	remove d[key] from d. Raises KeyError if key not in d.	`d = { 1:"a", 2:"b" }`
		`print(1 in d)`
		`del d[1]`
		`print(1 in d)`
		`del d[1]` `# crash!`

set append, set extend, are not in dictionary

(Ch 8) Collection Data Types in Python

- Tuple
- Set
- Dictionary
- One More Small Thing about * and **

* and ** in Function Call [1/2]

The single star * unpacks the sequence/collection into positional argument

```
>>> data = ( 1, 2, 3)
>>> print(*data)
>>> 1 2 3

>>> data = [1, 2, 3]
>>> print(*data)
>>> 1 2 3

>>> data = {1, 2, 3}
>>> print(*data)
>>> 1 2 3

>>> data = { 1: 'kim', 2: 'Lee'}
>>> print(*data)
>>> 1 2
```

```
def sum(a, b):
    return a + b

values = (1, 2)

s = sum(*values)
```

sum() 의 a, b는 의 단일 입력값으로 받겠다는데

sum() 의 입력값으로 하고 싶다면

This will unpack the tuple so that it actually executes as:

```
s = sum(1, 2)
```

* and ** in Function Call [2/2]

```
def sum(a, b):
    return a + b
```

sum() 의 a, b는 의 단일 입력값으로 받겠다는데

The double star `**` does the same, only using a dictionary

```
values = { 'a': 1, 'b': 2 }
s = sum(**values)
```

sum() 의 입력값으로 하고 싶다면

Will be executed as
s = sum(a=1, b=2)

참고로

```
>>> print(*values)
>>> a b
>>> print(**values)
>>> Error
```

You can also combine:

```
def sum(a, b, c, d):
    return a + b + c + d

values1 = (1, 2)
values2 = { 'c': 10, 'd': 15 }
s = sum(*values1, **values2)
```

will execute as:

```
s = sum(1, 2, c=10, d=15)
```

* and ** in Function Definition [1/4]

Remember?

여러 개의 입력값을 받는 함수 [1/2]

- 여러 개의 입력값을 모두 더하는 함수를 만들려 한다.
- 예, sum_many(1, 2) returns 3 sum_many(1,2,3,4,5) returns 15

```
>>> def sum_many(*args):
...     sum = 0
...     for i in args:
...         sum = sum + i
...     return sum
...
>>>
```

여러 개의 입력값이 들어오는것을 기대

```
>>> result = sum_many(1,2,3)
>>> print(result)
6
>>> result = sum_many(1,2,3,4,5,6,7,8,9,10)
>>> print(result)
55
```

List값을 입력값으로 하고 싶다면!!

```
>>> input_data = [1, 2, 3]
>>>
sum_many( *input_data )
>>> 6
```

* and ** in Function Definition [2/4]

Remember?

여러 개의 입력값을 받는 함수 [2/2]

```
>>> def sum_mul(choice, *args):
...     if choice == "sum":
...         result = 0
...         for i in args:
...             result = result + i
...     elif choice == "mul":
...         result = 1
...         for i in args:
...             result = result * i
...     return result
...
>>>
```

```
>>> result = sum_mul('sum', 1,2,3,4,5)
>>> print(result)
15
>>> result = sum_mul('mul', 1,2,3,4,5)
>>> print(result)
120
```

List값을 입력값으로 하고 싶다면!!

```
>>> input_data = [1, 2, 3]
>>> sum_mul( 'sum',  *input_data)
>>> 6
```

* and ** in Function Definition [3/4]

Remember?

여러 개의 Parameter = Value 형태를 받는 함수 [1/2]

```
def sum(*values, **options):
    s = 0
    for i in values:
        s = s + i
    if "neg" in options:
        if options["neg"]:
            s = -s
    return s

s = sum(1, 2, 3, 4, 5)                # returns 15
s = sum(1, 2, 3, 4, 5, neg=True)      # returns -15
s = sum(1, 2, 3, 4, 5, neg=False)     # returns 15
```

여러개의 parameter = value 형태로 들어오는것을 기대

```
>>> input_data = [1, 2, 3]
>>> option_data = { neg: True }
>>> sum(*input_data,  **option_data)
>>> -6
```

List값과 Dictionary값을 입력값으로 하고 싶다면!!

* and ** in Function Definition [4/4]

Remember?

여러 개의 Parameter = Value 형태를 받는 함수 [2/2]

여러개의 parameter = value
형태로 들어오는것을 기대

```
>>> def foo(**option):
        if 'b' in option:
            print("Mr. Lee")
        else:
            print("None")

>>> foo(a = 'Kim')
>>> None
>>> foo(a= 'Kim', b = 'Lee')
>>> Mr. Lee
>>>
>>> data = {'a': 'kim', 'b': 'Lee'}
>>> foo(**data)
>>> Mr. Lee
```