

Advanced Data Types in Python

- Tuple
- Set
- Dictionary

Tuple in Python

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

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 is similar to list
But, Tuple is *immutable*

TUPLES VS. LISTS

More memory
efficient

Takes more
memory

Cannot be
adjusted

Adjustable

Tuple Operations [1/3]

Basic Tuples Operations

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.

In fact, tuples respond to all of the general sequence operations we used on strings in the prior chapter –

Python Expression	Results	Description
<code>len((1, 2, 3))</code>	3	Length
<code>(1, 2, 3) + (4, 5, 6)</code>	<code>(1, 2, 3, 4, 5, 6)</code>	Concatenation
<code>('Hi!') * 4</code>	<code>('Hi!', 'Hi!', 'Hi!', 'Hi!')</code>	Repetition
<code>3 in (1, 2, 3)</code>	True	Membership
<code>for x in (1, 2, 3): print x,</code>	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, Slicing, and Matrixes

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] Built-in Tuple Functions

SN	Function with Description
1	cmp(tuple1, tuple2) ↗ Compares elements of both tuples.
2	len(tuple) ↗ Gives the total length of the tuple.
3	max(tuple) ↗ Returns item from the tuple with max value.
4	min(tuple) ↗ Returns item from the tuple with min value.
5	tuple(seq) ↗ Converts a list into tuple.

Advanced Data Types in Python

- Tuple
- Set
- Dictionary

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!

```
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
```

Elements Must Be Immutable

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

Another example:

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

Sets are Very Efficient

```
# 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!")
```

Set Operations [1/5]

Operation	Result
<code>len(s)</code>	cardinality (size) of set s

Example

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

<code>s.copy()</code>	new set with a shallow copy of s
-----------------------	----------------------------------

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

Set Operations [1/5]

<code>s.pop()</code>	remove and return an arbitrary element from <code>s</code> ; raises <code>KeyError</code> if empty
----------------------	----------------------------------------------------------------------------------------------------

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

<code>s.clear()</code>	remove all elements from set <code>s</code>
------------------------	---------------------------------------------

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

Set Operations [2/5]

Operation	Result	Example
<code>x in s</code>	test x for membership in s	<pre>s = { 1, 2, 3 } print(0 in s) print(1 in s)</pre>
<code>x not in s</code>	test x for non-membership in s	<pre>s = { 1, 2, 3 } print(0 not in s) print(1 not in s)</pre>
<code>s.add(x)</code>	add element x to set s	<pre>s = { 1, 2, 3 } print(s, 4 in s) s.add(4) print(s, 4 in s)</pre>

Set Operations [3/5]

<code>s.remove(x)</code>	remove x from set s; raises <code>KeyError</code> if not present
--------------------------	------------------------------------------------------------------

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

<code>s.discard(x)</code>	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)
```


Set Operations [3/5]

`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}))
```

Set Operations [4/5]

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)
```

Set Operations [4/5]

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)
```

Set Operations [5/5]

`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)
```

`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)
```

Set Operations [5/5]

`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)
```

`s.symmetric_difference_update(t)` | `s ^= t`

Modify s keeping elements from s or t but not both

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

Advanced Data Types in Python

- Tuple
- Set
- Dictionary

Dictionary 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)
```

```
>>> 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
```


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!
```

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 }
```

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'
```

Dictionary Operations [1/7]

```
>>> 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'])
```

Dictionary Operations [2/7]

```
>>> 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  
{}
```

Dictionary Operations [3/7]

`dict_keys` 객체는 다음과 같이 사용할 수 있다. 리스트를 사용하는 것과 차이가 없지만, 리스트 고유의 함수인 `append`, `insert`, `pop`, `remove`, `sort` 등의 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']
```

Dictionary Operations [4/7]

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 Operations [5/7]

Operation	Result	Example
<code>len(d)</code>	the number of items (key-value pairs) in dictionary <code>d</code>	<pre>d = { 1:[1,2,3,4,5], 2:"abcd" } print(len(d))</pre>
<code>d.copy()</code>	new dictionary with a shallow copy of <code>d</code>	<pre>d1 = { 1:"a" } d2 = d1.copy() d1[2] = "b" print(d1) print(d2)</pre>
<code>d.popitem()</code>	remove and return an arbitrary (key,value) pair from <code>d</code> ; raises <code>KeyError</code> if empty	<pre>d = { 1:"a", 2:"b" } print(d.popitem()) # unpredictable print(d)</pre>
<code>d.clear()</code>	remove all items from dictionary <code>d</code>	<pre>d = { 1:"a", 2:"b" } d.clear() print(d, len(d))</pre>

Dictionary Operations [6/7]

for key in d	iterate over all keys in d.	<pre>d = { 1:"a", 2:"b" } for key in d: print(key, d[key])</pre>
key in d	test if d has the given key	<pre>d = { 1:"a", 2:"b" } print(0 in d) print(1 in d) print("a" in d) # surprised?</pre>
key not in d	test if d does not have the given key	<pre>d = { 1:"a", 2:"b" } print(0 not in d) print(1 not in d) print("a" not in d)</pre>
d[key]	the item of d with the given key. Raises a KeyError if key is not in the map.	<pre>d = { 1:"a", 2:"b" } print(d[1]) print(d[3]) # crash!</pre>

Dictionary Operations [7/7]

<code>d[key] = value</code>	set <code>d[key]</code> to value.	<pre>d = { 1:"a", 2:"b" } print(d[1]) d[1] = 42 print(d[1])</pre>
<code>get(key[,default])</code>	the value for key if key is in the dictionary, else default (or None if no default is provided).	<pre>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</pre>
<code>del d[key]</code>	remove <code>d[key]</code> from d. Raises <code>KeyError</code> if key not in d.	<pre>d = { 1:"a", 2:"b" } print(1 in d) del d[1] print(1 in d) del d[1] # crash!</pre>