

w12-Lec

Other Collection Types: Sets and Dictionaries

for 204111

Kittipitch Kuptavanich

Sets

ได้สิ่งของมาแล้ว! ไม่ซ้ำกัน

๕

- Sets เป็นวัตถุประเภท Collection ที่มีสมาชิกไม่ซ้ำกัน (ลักษณะคล้าย Set ทางคณิตศาสตร์) ประกอบด้วย

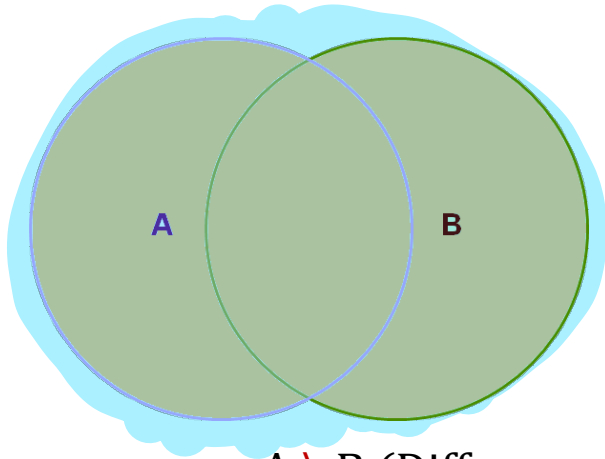
- `set`
- `frozenset` (Immutable Collection)

- เราสามารถใช้ Set เพื่อทำ Operation ต่าง ๆ ได้แก่

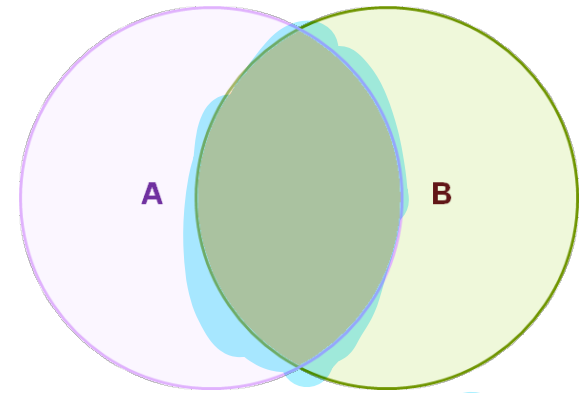
- การกำจัด Element ที่ซ้ำออก
- การหายูเนียน (Union: \cup)
- การหาอินเตอร์เซกชัน (Intersection: \cap)
- การหาผลต่างของเซต (Difference: \setminus)
- การหาผลต่างสมมาตร (Symmetric Difference: \oplus)

Common Set Operations

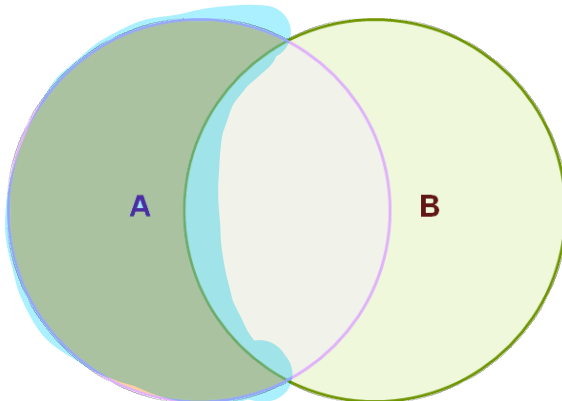
$A \cup B$ (Union) |



$A \cap B$ (Intersection) &

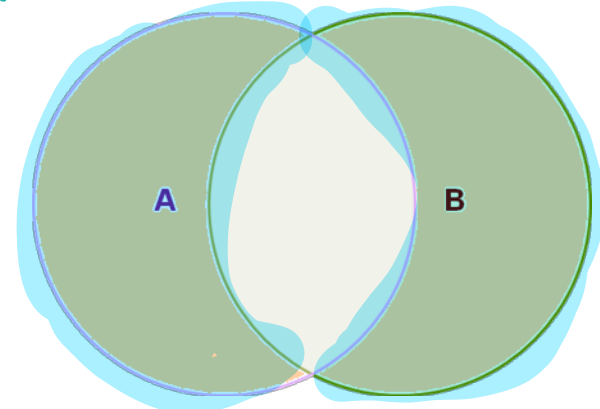


$A \setminus B$ (Difference) -



$A \oplus B$ (Symmetric Difference) ^

$(A \cup B) - (A \cap B)$



Examples

- เราใช้เครื่องหมายปีกกา `{}` และ Comma `,` เพื่อสร้าง Set

```

01 s = {2, 3, 5, 7, 9}
02 print(3 in s)           # prints True
03 print(4 in s)           # prints False
04 for x in range(10):
05     if (x not in s):
06         print(x, end="") # prints 0 1 4 6 8

```

- เราสามารถใช้ Operation เช่น `in` หรือ Loop ได้เหมือนใน Collection Type อื่น ๆ

Properties of Sets

- **set** เป็น **Collection Type** แบบไม่มีลำดับ

```
09 s = set([2, 4, 8])
10 print(s)           # prints {8, 2, 4}
11 for element in s:
12     print(element, end="") # prints 8 2 4
```

- ลำดับสมาชิกที่แสดงผล ใน **set** ขึ้นกับการ **algorithm** ในการเรียงลำดับสมาชิกที่แตกต่างกันไปตามการ **implement** มาตรฐาน **Python (CPython vs Jython)** และสามารถมีความแตกต่างกันไปในแต่ละ **version (CPython 3.x vs CPython3.y)**

```
>>> # notice how the orders are unpredictable (although static)
>>> {x*7 for x in range(10)}
{0, 35, 7, 42, 14, 49, 21, 56, 28, 63}
>>> {x for x in range(10)}
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Properties of Sets [2]

- Element แต่ละตัวจะไม่ซ้ำกัน

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

- แต่ละ Element ต้องมีคุณสมบัติ Immutable เช่น `str`, `int` หรือ Atomic Type อื่น ๆ

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

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

- Set ไม่สามารถมี Element เป็นชนิด List หรือ Set ได้
 - Set _____ Element เป็นชนิด Tuple ได้

Sets are Efficient

```
05 import time
06 n = 1000
07 l = list(range(2, n+1, 2))
08
09 def count_member(c, n, use_set=False):
10     a = c
11     if use_set:
12         a = set(c)
13
14     start = time.time()
15     count = 0
16     for x in range(n+1):
17         if x in a:
18             count += 1
19     elapsed1 = time.time() - start
20     print(f'count={count} and time = {elapsed1:0.5f} seconds')
21
22 print("Using a list... ", end="")
23 count_member(l, n)
24 print("Using a set.... ", end="")
25 count_member(l, n, True)
```

Using a list... count=500 and time = 0.00304 seconds
Using a set.... count=500 and time = 0.00005 seconds

Creating Sets

```
>>> # Create an empty set
>>> s = set()
>>> print(s)      # prints set()
set()

>>> # Create a set from a list
>>> s = set(["cat", "cow", "dog"])
>>> print(s)
{'cow', 'cat', 'dog'}

>>> # Create a set from any iterable object
>>> s = set("wahoo")
>>> print(s)
{'a', 'h', 'w', 'o'}
```


Set Operations in Python

Operation	Result	Notes
<code>len(s)</code>	cardinality of set <i>s</i>	
<code>s.copy()</code>	new set with a shallow copy of <i>s</i>	
<code>s.pop()</code>	remove and return an arbitrary element from <i>s</i> ; raises <code>KeyError</code> if empty	
<code>s.clear()</code>	remove all elements from set <i>s</i>	
<code>x in s</code>	test <i>x</i> for membership in <i>s</i>	
<code>x not in s</code>	test <i>x</i> for non-membership in <i>s</i>	
<code>s.add(x)</code>	add element <i>x</i> to set <i>s</i>	
<code>s.remove(x)</code>	remove <i>x</i> from set <i>s</i> ; raises <code>KeyError</code> if not present	
<code>s.discard(x)</code>	Remove element <i>x</i> from the set if it is present.	

Set Operations in Python [2]

Operation	\equiv	Result	Notes
<code>s.issubset(t)</code>	<code>s <= t</code>	test whether every element in <i>s</i> is in <i>t</i>	
<code>s.issuperset(t)</code>	<code>s >= t</code>	test whether every element in <i>t</i> is in <i>s</i>	
<code>s.union(t)</code>	<code>s t</code>	<u>new set</u> with elements from both <i>s</i> and <i>t</i>	
<code>s.intersection(t)</code>	<code>s & t</code>	<u>new set</u> with elements common to <i>s</i> and <i>t</i>	
<code>s.symmetric_difference(t)</code>	<code>s ^ t</code>	<u>new set</u> with elements in either <i>s</i> or <i>t</i> but <u>not both</u>	
<code>s.update(t)</code>	<code>s = t</code>	Same as <code>s = s t</code>	
<code>s.intersection_update(t)</code>	<code>s &= t</code>	Same as <code>s = s & t</code>	
<code>s.difference_update(t)</code>	<code>s -= t</code>	Same as <code>s = s - t</code>	
<code>s.symmetric_difference_update(t)</code>	<code>s ^= t</code>	Same as <code>s = s ^ t</code>	

Example Using Sets

```

01 def repeats(a):
02     seen = set()
03     seen_again = set()
04     for element in a:
05         if (element in seen):
06             seen_again.add(element)
07             seen.add(element)
08     return sorted(list(seen_again))

```

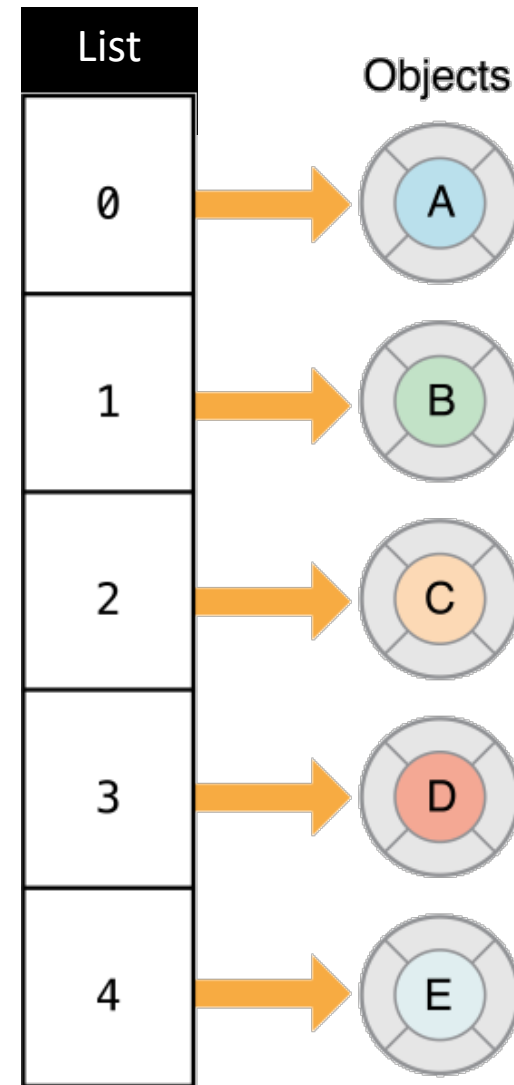
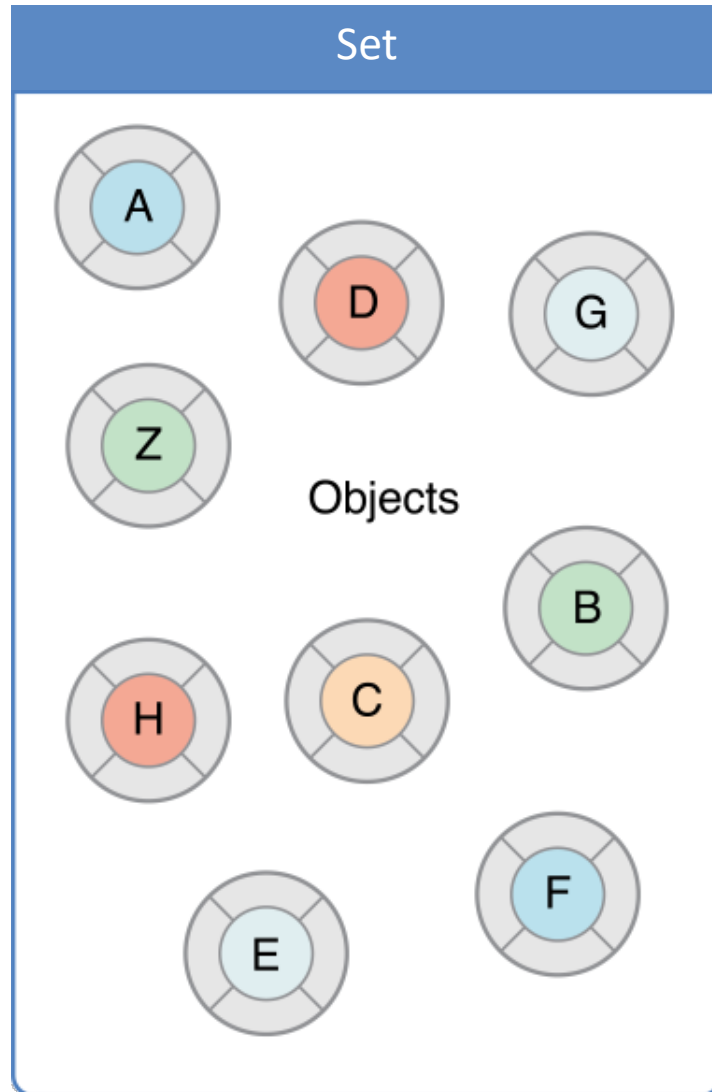
- ฟังก์ชัน `repeats()` ทำหน้าที่ แสดงสมาชิกซ้ำ เป็น list

```

>>> print(repeats([2, 5, 3, 4, 6, 4, 2]))
[2, 4]

```

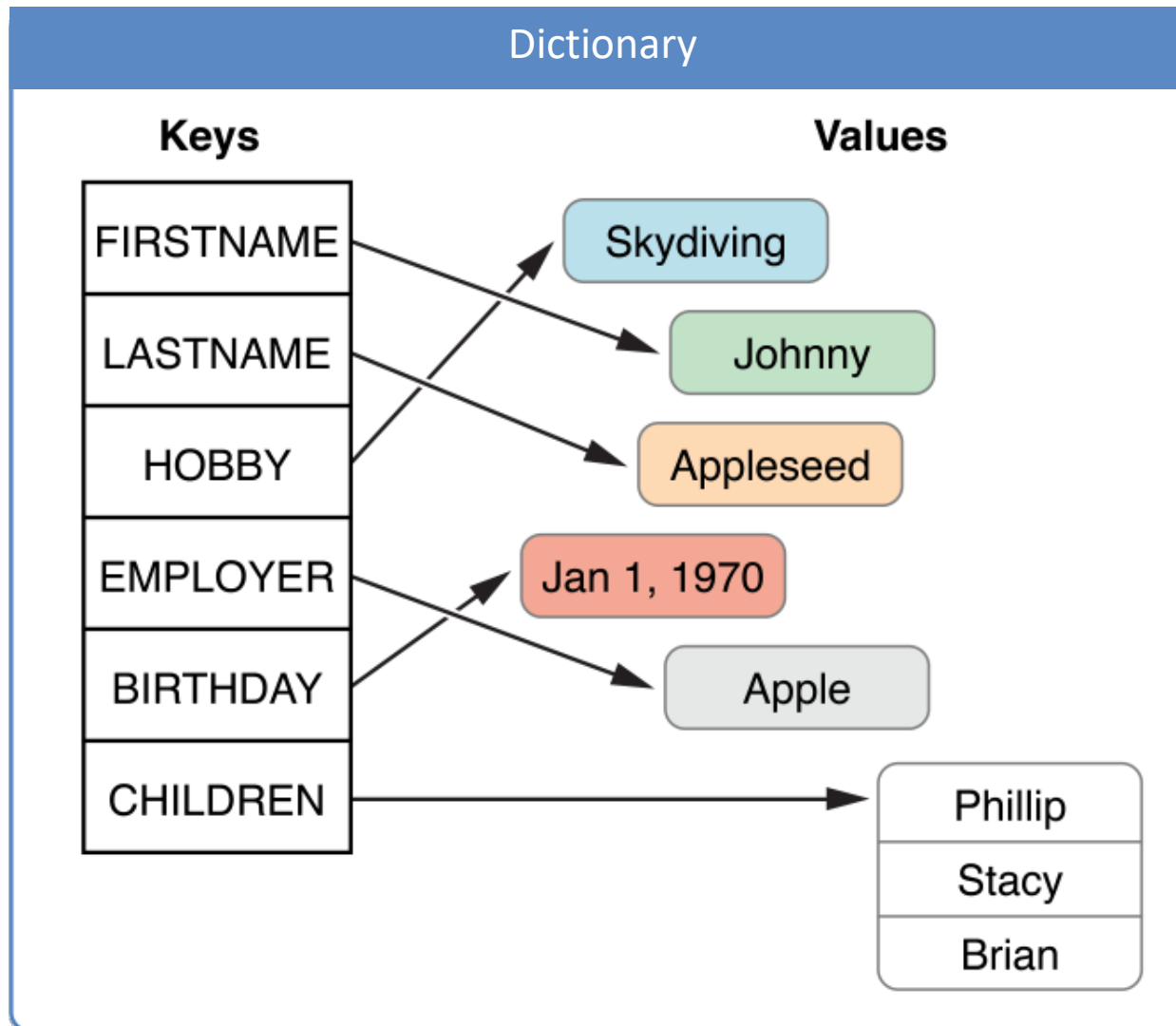
Sets vs Lists



Dictionaries

- Dictionary (หรือ Hash Map) มีลักษณะคล้าย List
 - Index ใน List ต้องเป็นเลขจำนวนเต็ม (≥ 0)
 - แต่ Index ใน Dictionary สามารถเป็นข้อมูลได้ (เกือบ) ทุกชนิด
- เราเรียก Index ใน Dictionary ว่า *key*
- และเรียกค่าที่เก็บไว้ใน Index นั้น ๆ ว่า *value*
- Dictionary เป็นข้อมูลประเภท Mapping Type คือ เป็นการ Map *key* \rightarrow *value*

Dictionaries [2]



Dictionaries [3]

- เราใช้ฟังก์ชัน Built-in `dict()` ในการสร้าง Dictionary เปล่า
- เช่นหากต้องการสร้าง Dictionary สำหรับเก็บหมายเลขโทรศัพท์

โทรศัพท์ *empty dictionary*

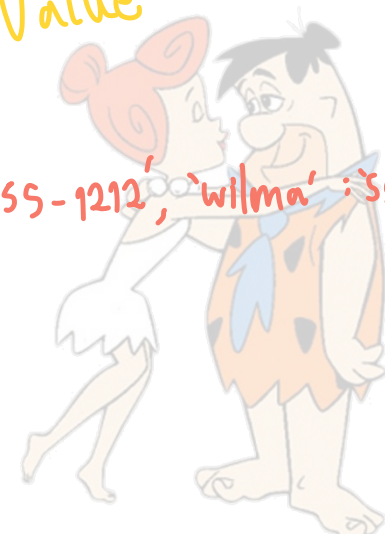
```
>>> d = dict()
>>> d['fred'] = '555-1212'
>>> d['wilma'] = '555-3456'
>>> print(d['fred'])
'555-1212'
```

d {key} = value

d: { 'fred': '555-1212', 'wilma': '555-3456' }

```
>>> # now fred and wilma get married, so...
>>> d['fred'] = d['wilma']
>>> print(d['fred'])
'555-3456'
```

print(d[key])



Properties of Dictionaries

- Dictionary เป็นการผูกค่าระหว่าง **key** และ **value**
 - **key** เป็น Set ดังนั้น **key** แต่ละตัวต้องมีลักษณะ **Immutable, ไม่ซ้ำกัน และไม่มีลำดับ (Unordered)**
 - **Strings, number**
 - **Tuples (ที่มี Element เป็น Immutable)**
 - ดังนั้น ~~([2], 3)~~ เป็น **key** ไม่ได้ เนื่องจากเป็น Tuple ที่ประกอบด้วย List (Mutable)
 - **value** สามารถเป็นข้อมูลประเภทใดก็ได้ ไม่มีข้อจำกัด

Creating a Dictionary

- ใช้เครื่องหมายปีกกา {}

```
>>> # Empty dictionary
```

```
>>> d = {}
```

```
>>> print(d)
```

```
{}
```

```
>>> # key:value
```

```
>>> d = {"cow": 5, "dog": 98, "cat": 1}
```

```
>>> print(d)
```

```
{'dog': 98, 'cow': 5, 'cat': 1}
```

Order in a Dictionary's keys

- ลำดับสมาชิกที่แสดงผล ใน `dict` ขึ้นกับลำดับของ Operation (e.g. `insert/delete`) ของ dictionary นั้น ๆ (Python 3.7+)

```
>>> d1 = dict()
>>> d2 = dict()
>>> d1['a'] = 3; d1['b'] = 5
>>> d2['b'] = 5; d2['a'] = 3
>>> d1 == d2
True
>>> print(d1)
{'a': 3, 'b': 5}
>>> print(d2)
{'b': 5, 'a': 3}
```

Creating a Dictionary [2]

- ใช้ฟังก์ชัน `dict()`

```
>>> d = dict()           # Empty dictionary
>>> print(d)
{}

>>> # From a Collection of Tuple (key, value) i.e. a List
>>> pairs = [("one", 1), ("two", 2), ("three", 3)]
>>> d = dict(pairs)
>>> print(d)
{'one': 1, 'three': 3, 'two': 2}

>>> # function parameters
>>> d = dict(one=1, two=2, three=3)
>>> print(d)
{'one': 1, 'three': 3, 'two': 2}
```

Creating a Dictionary [3]

```

05 person1 = {
06     'Name': "Allan Smith",
07     'Man': True,
08     "age": 45,
09     'siblings': ["Yae", "Samuel", "Liz", "Ammon"],
10     "Pet": None
11 }

```

- Can be created with the `dict()` function

```

15 person1 = dict( 'Name': "Allan Smith"
16                  'Man' = True
17                  "age" = 45
18                  'siblings' = ["Yae", "Samuel", "Liz", "Ammon"]
19                  "Pet" = None

```

Dictionary Operations

Operation	Result	Notes
<code>len(d)</code>	Return the number of items (key-value pairs) in the dictionary <i>d</i> .	
<code>d.clear()</code>	Remove all items from the dictionary <i>d</i> .	
<code>d.copy()</code>	Return a <u>shallow copy</u> of the dictionary <i>d</i> .	
<code>d.keys()</code>	Return a new view of the dictionary's keys.	
<code>d.popitem()</code>	Remove and return an arbitrary (key, value) pair from the dictionary. If the dictionary is empty, calling <code>popitem()</code> raises a <u><code>KeyError</code></u> .	
<code>for key in d</code>	Iterate over all keys in <i>d</i> For example: <pre>d = {"cow": 5, "dog": 98, "cat": 1} for key in d: print(key, d[key])</pre>	

Dictionary Operations [2]

Operation	Result	Notes
<code>key in d</code>	Return True if <i>d</i> has a key <i>key</i> , else False.	
<code>key not in d</code>	Equivalent to not <i>key</i> in <i>d</i> .	
<code>d[key]</code>	Return the item of <i>d</i> with key <i>key</i> . Raises a <u>KeyError</u> if <i>key</i> is not in the map.	
<code>get(key[,default])</code>	Return the value for <i>key</i> if <i>key</i> is in the dictionary, else <i>default</i> . If <i>default</i> is not given, it defaults to <u>None</u> , so that this method never raises a <u>KeyError</u> .	
<code>d[key] = value</code>	Set <i>d[key]</i> to <i>value</i> .	
<code>del d[key]</code>	Remove <i>d[key]</i> from <i>d</i> . Raises a <u>KeyError</u> if <i>key</i> is not in the map.	
<code>update([other])</code>	Update the dictionary with the <i>key/value</i> pairs from <i>other</i> , overwriting existing keys. Return <u>None</u> .	

Example Operations

ถ้ามีใน dict key มี ๑ = เพิ่ม key หรือ value

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127 → เพิ่ม key ใหม่ dict
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack'] → แสดงค่า
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}

>>> list(tel.keys())
['irv', 'guido', 'jack']
>>> sorted(tel.keys())
['guido', 'irv', 'jack']

>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

Example Using Dictionaries


```
02 def most_frequent(a):
03     max_value = None
04     max_count = 0
05     frequency = dict()
06     for element in a:
07         if element in frequency:
08             count = frequency[element]
09         else:
10             count = 0
11         count += 1
12         frequency[element] = count
13         if (count > max_count):
14             max_count = count
15             max_value = element
16     return max_value
17
18 print(most_frequent([2, 5, 3, 4, 6, 4, 2, 4, 5])) # 4
```


Example Using Dictionaries [2]

```

02 def most_frequent(a):
03     max_value = None
04     max_count = 0
05     frequency = dict()
06     for element in a:
07         count = 1 + frequency.get(element, 0)
08         frequency[element] = count
09         if (count > max_count):
10             max_count = count
11             max_value = element
12     return max_value
13
14 print(most_frequent([2, 5, 3, 4, 6, 4, 2, 4, 5])) # 4

```



- สังเกตการใช้ Method `dict.get()` โดยมี 0 เป็นค่า Default ที่บรรทัดที่ 07

map and filter with dict

mapping

```
>>> a = {'cat':1, 'bat':2}
```

```
>>> dict(map(lambda x: (x[0], x[1]**2), a.items()))
{'cat': 1, 'bat': 4}
```

list of value in dict a

filtering on keys

```
>>> a = {'cat':1, 'bat':2}
```

```
>>> dict(filter(lambda x: 'c' in x[0], a.items()))
{'cat': 1}
```

filtering on values

```
>>> a = {'cat':1, 'bat':2}
```

```
>>> dict(filter(lambda x: x[1] % 2 == 0, a.items()))
{'bat': 2}
```

JSON – An Example of a Dictionary Daily Usage

- JavaScript Object Notation (JSON) is one of the most popular ways to transmit data between applications.
- A basic unit of JSON is a *key*:*value* pair, for example:

```
05 {  
06     "name":  
07     [  
08         "Steve",  
09         "Steven",  
10         "Esteban"  
11     ],  
12     "fingers": 10,  
13     "street": "Keri Drive"  
14 }
```

JSON – An Example from a Real API

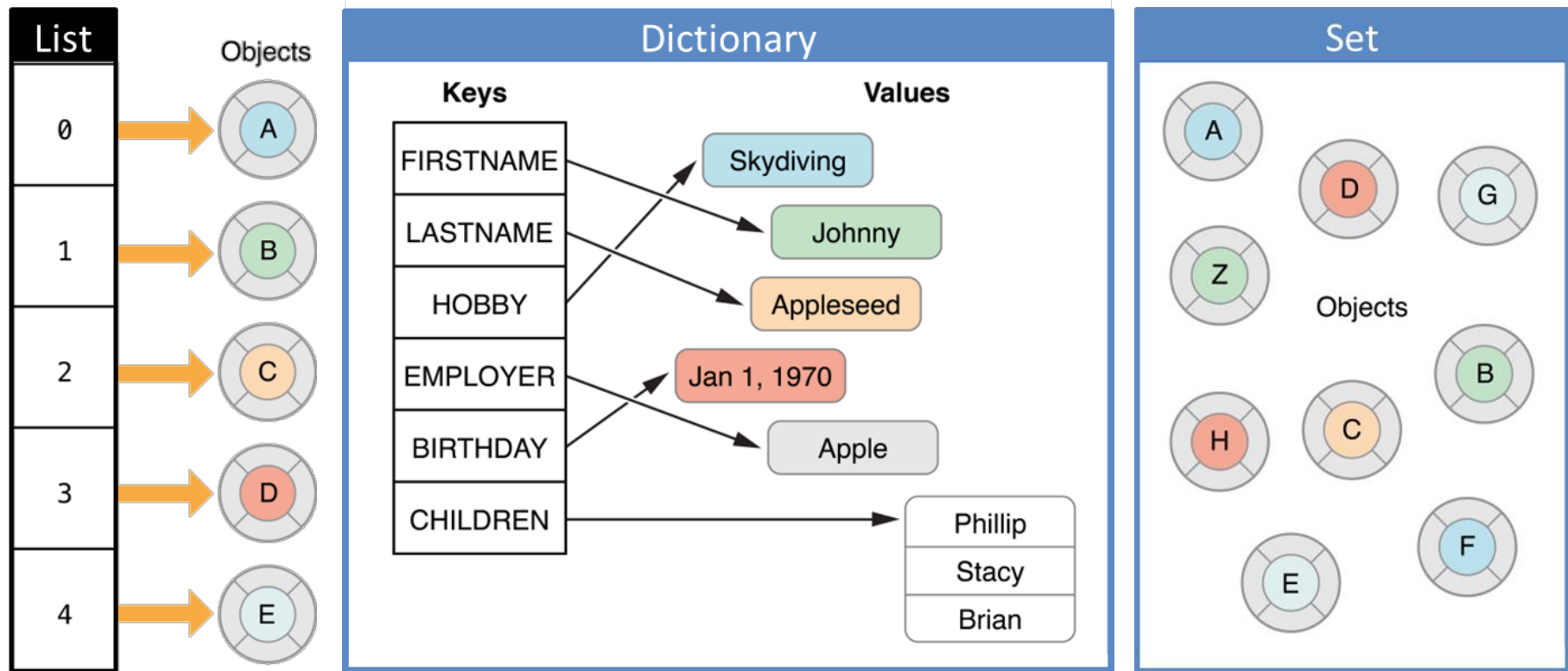
```
{
  "coord": {
    "lon": 10.99,
    "lat": 44.34
  },
  "weather": [
    {
      "id": 501,
      "main": "Rain",
      "desc": "moderate rain",
      "icon": "10d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 298.48,
    "feels_like": 298.74,
    "temp_min": 297.56,
    "temp_max": 300.05,
    "pressure": 1015,
    "humidity": 64,
    "sea_level": 1015,
```

```
    "grnd_level": 933
  },
  "visibility": 10000,
  "wind": {
    "speed": 0.62,
    "deg": 349,
    "gust": 1.18
  },
  "rain": {
    "1h": 3.16
  },
  "clouds": {
    "all": 100
  },
  "sys": {
    "type": 2,
    "id": 2075663,
    "country": "IT",
    "sunrise": 1661834187,
    "sunset": 1661882248
  }
}
```

JSON – An Example [2]

```
05 import json
06
07 api_res = '''
08 {
09     "coord": {
10         "lon": 10.99,
11         "lat": 44.34
12     },
13     ...
14 }
15 '''
16
17 data = json.loads(api_res)
18 print(data['visibility'])
19
20 hum = data['main']['humidity']
21 print(hum)
22
23 id_ = data['weather'][0]['id']
24 print(id_)
```

Collections Recap



Practice 1

เขียนฟังก์ชัน `most_common_name(names)` ที่รับ parameter `names` เป็น `list` ของชื่อ เช่น `["Jane", "Aaron", "Cindy", "Aaron"]` และ คืนค่าชื่อที่ซ้ำกันมากที่สุด โดยหากมีเพียงชื่อเดียวให้คืนค่าเป็น `string` หากมีมากกว่าหนึ่งชื่อ ให้คืนค่าเป็น `set`

```
>>> most_common_name(["Jane", "Aaron", "Cindy", "Aaron"])
"Aaron"
>>> most_common_name(["Jane", "Aaron", "Jane", "Aaron"])
{"Aaron", "Jane"}
```

Practice 2

เขียนฟังก์ชัน `most_popular_friend(name_dict)` ที่รับ parameter `name_dict` เป็น `dict` ที่มี `key` เป็นชื่อคน และ `value` เป็น `set` รายชื่อเพื่อนของคนนั้น ๆ แล้วให้คืนค่าชื่อคนที่ถูกระบุว่าเป็นเพื่อนของอื่น ที่ซ้ำกันมากที่สุด (มีชื่อเดียวเสมอ)
 ทั้งนี้ กำหนดให้ความเป็นเพื่อน เป็น One Way Relationship (Betty อาจจะไม่เห็น Wilma เป็นเพื่อนก็ได้ 😞)

```
>>> d = dict()
>>> d["fred"] = set(["wilma", "betty", "barney"])
>>> d["wilma"] = set(["fred", "betty", "dino"])
>>> most_popular_friend(d)
'betty'
```


Practice 3

เขียนฟังก์ชัน `word_count(text)` เพื่อนับจำนวนคำที่ปรากฏในสายอักขระ (String) `text` โดยฟังก์ชันจะ คืนค่าเป็น `dict` โดยมี `key` เป็นแต่ละคำที่ปรากฏใน `text` และมี `value` เป็นความถี่ ทั้งนี้ตัวอักษรที่อยู่ใน `key` จะต้องเป็นตัวอักษรพิมพ์เล็ก

Input

```
"This jacket costs $5 MORE
than that jacket, REALLY?"
```

Output

```
{'this': 1,
 'jacket': 2,
 'costs': 1,
 '5': 1,
 'more': 1,
 'than': 1,
 'that': 1,
 'really': 1}
```

Practice 3 [2]

ข้อกำหนด

- การนับความถี่จะเป็นแบบ **Case Insensitive** ('ant' และ 'Ant' ถือเป็นคำเดียวกัน)
- ข้อความใน *text* จะเป็นภาษาอังกฤษมาตรฐานในรูปแบบที่ถูกต้อง (**well-formed English**)
- ไม่พิจารณาเครื่องหมายวรรคตอนต่าง ๆ เฉพาะที่ล้อมรอบคำ เช่น
!"#\$%&'()*+,-./:;<=>?@[\\]^_`{|}~
- ลำดับในการแสดงผลใน **output** ไม่จำเป็นต้องเหมือนตัวอย่าง

References

- <https://docs.python.org/3/library/stdtypes.html#set>
- <https://docs.python.org/3/tutorial/datastructures.html#sets>
- <http://www.cs.cmu.edu/~112/notes/notes-sets.html>
- <https://docs.python.org/3/library/stdtypes.html#mapping-types-dict>
- <https://docs.python.org/3/tutorial/datastructures.html#dictionaries>
- <http://www.cs.cmu.edu/~112/notes/notes-maps.html>
- <https://medium.com/analytics-vidhya/python-dictionary-and-json-a-comprehensive-guide-ceed58a3e2ed>
- <https://heardlibrary.github.io/digital-scholarship/script/python/json/>
- <https://openweathermap.org/current>
- <https://www.kosbie.net/cmu/fall-21/15-112/>