Data Structures with Python

C. Alex Hu, PhD
@ *bigDataSpark Forum 2018/02/22*

REFERENCE

Python 官方網站： “**The Python Tutorial — 5. Data Structures**”
https://docs.python.org/3/tutorial/datastructures.html

1. 資料型態 list 的相關方法

(1) list.**append**(*x*)

```
##   先產生下列三筆資料：
listNumbers = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
listLetters = ['A', 'b', 'C', 'd', 'E']
listConstants = [ 1.41421, 3.14159, 2.71828]

list0 = listNumbers
list0           #  [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]

list0.append('a')
list0           #  [1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 'a']

list1 = None
list1           #  注意：沒有輸出結果
list1.append(list0)
##   錯誤訊息如下：
##   Traceback (most recent call last):
##     File "<pyshell#11>", line 1, in <module>
##       list1.append(list0)
##   AttributeError: 'NoneType' object has no attribute 'append'

list1 = []
list1      #  []

list1.append(1000)
list1      #  [1000]
```

```
[ NOTE ]:
##  list[len(list):] = [x] 等同於 list.append(x)
list0[len(list0):] = ['b']  # 等同於 list0.append('b')
list0   #  [1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 'a', 'b']
```

```
list1.append(list0)
list1      #  [1000, [1, 2, 3, 4, 5, 6, 7, 8, 9, 0, ‘a’]]
```

(2) list.**extend**(*iterable*) & list.**copy**()

```
##   先產生下列三筆資料：
listNumbers = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
listLetters = ['A', 'b', 'C', 'd', 'E']
listConstants = [ 1.41421, 3.14159, 2.71828]

list0 = []
##  [extend() 的用法]
list0 = listLetters
list0          #  ['A', 'b', 'C', 'd', 'E']
list0.extend(listNumbers[2:5])
list0        #  ['A', 'b', 'C', 'd', 'E', 3, 4, 5]

##  But…  listLetters 原始資料被更改！  WHY ???
listLetters  #  ['A', 'b', 'C', 'd', 'E', 3, 4, 5]  What happened?

list0.append(1000.0)
list0          #  ['A', 'b', 'C', 'd', 'E', 3, 4, 5, 1000.0]
listLetters    #  ['A', 'b', 'C', 'd', 'E', 3, 4, 5, 1000.0] ???
listNumbers    #  [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]  OK!
```

Q1： 請問，上列程式中，listLetters 內的資料為什麼會改變？

[HINT：(1) **list0 = listLetters**， (2) *local symbol table*]

Q2： 請問，上列程式中，如何避免 listLetters 內的資料改變？

A： (1) **修改 extend() 的用法 (如下)**， (2) **list.copy()**]

```
list1 = []
list1.extend(listNumbers)      #  Using extend() to duplicate…
list1          #  [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]  OK!
listNumbers    #  [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]  OK!

list2 = []
list2 = list0.copy()      #  Using copy() to make a shallow copy…
list2      #  ['A', 'b', 'C', 'd', 'E', 3, 4, 5, 1000.0]
list2.append(2000.0)
list2      #  ['A', 'b', 'C', 'd', 'E', 3, 4, 5, 1000.0, 2000.0]
list0      #  ['A', 'b', 'C', 'd', 'E', 3, 4, 5, 1000.0]  OK!
```

<< append() vs. extend() >>

```
list1 = []
list2 = []
list1 = listNumbers.copy()
list1          #  [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
list2 = list1.copy()
list2          #  [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]

list1.append(listConstants)
list1
#  [1, 2, 3, 4, 5, 6, 7, 8, 9, 0, [1.41421, 3.14159, 2.71828]]

list2.extend(listConstants)
list2
#  [1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1.41421, 3.14159, 2.71828]
```

[NOTE]: list.extend(*iterable*) & list.copy() 的替代用法

(1) **list[len(list):] = *iterable*** 等同於 list.**extend**(*iterable*)

```
list0[len(list0):] = listNumbers[3:6]  # 等同於 list2.append('b')
list0   #  ['A', 'b', 'C', 'd', 'E', 3, 4, 5, 1000.0, 4, 5, 6]
```

(2) **list[:]** 等同於 list.**copy**()

```
list1 = list0[:]
list1   #  [‘A', 'b', 'C', 'd', 'E', 3, 4, 5, 1000.0, 4, 5, 6]
```

(3) **list** 的相關方法

list 的相關方法	敘述	替代用法
`list.`**`append`**`(`*`x`*`)`	附加一個項目 *x* 至 list 結尾之後	`list[len(list):]=[`*`x`*`]`
`list.`**`extend`**`(`*`iterable`*`)`	將一組連續的項目直接附加至 list 結尾之後，以延伸 list 的資料	`list[len(list):] = `*`iterable`*
`list.`**`copy`**`()`	回傳一個 list 的淺層複製(shallow copy)	`list[:]`
`list.`**`insert`**`(`*`i, x`*`)`	在 list 中，插入項目 *x* 於第 *i* 個位置之前	
`list.`**`remove`**`(`*`x`*`)`	移除 list 中第一個 *x* 項目。（如果 list 中無 *x* 項目，將顯示錯誤訊息）	
`list.`**`clear`**`(`*`x`*`)`	移除 list 中的所有 *x* 項目。	`del list[:]`
`list.`**`pop`**`([`*`i`*`])` 其中，[*i*] 代表一個可有可無的參數選項	移除在 *i* 位置上的特定項目。 **[NOTE]:**如果沒有標示 *i* 位置（亦即，list.pop()），則預設上，將會移除 list 中最後一個項目。	
`list.`**`index`**`(`*`x[,start[, end]]`*`)`	回傳項目 *x* 在 list 中第一次出現的位置（亦即，索引值）。（如果 list 中無 *x* 項目，將顯示錯誤訊息） 其中，*[,start[,end]]* 分別代表搜尋範圍的起始和結束位置，屬於可有可無的參數選項	
`list.`**`count`**`(`*`x`*`)`	回傳項目 *x* 在 list 中出現的次數。	
`list.`**`reverse`**`()`	反向排列 list 中的元素。	
`list.`**`sort`**`(`*`key=None, reverse=False`*`)`	針對 list 中的元素，進行排序(sorting)	

[Example 1.1 Methods for Lists] — 從 Python 3.6.2 Shell 執行下列指令:

```
>>> words = ['Fly', 'robin', 'fly', 'Up', 'up', 'to', 'the', 'sky']
>>> words.index('up')
4
>>> words.index('down')
Traceback (most recent call last):
  File "<pyshell#128>", line 1, in <module>
    words.index('down')
ValueError: 'down' is not in list

>>> words.count('fly')
1
>>> w = []
>>> w.extend(words[0:3])
>>> w
['Fly', 'robin', 'fly']

>>> words.insert(3,w)
>>> words
['Fly', 'robin', 'fly', ['Fly', 'robin', 'fly'], 'Up', 'up', 'to', 'the', 'sky']

>>> words.remove(w)
>>> words
['Fly', 'robin', 'fly', 'Up', 'up', 'to', 'the', 'sky']

>>> words.insert(3,'Fly')
>>> words
['Fly', 'robin', 'fly', 'Fly', 'Up', 'up', 'to', 'the', ‘sky']

>>> del words[3]   # del function
>>> words
['Fly', 'robin', 'fly', 'Up', 'up', 'to', 'the', 'sky']
```

< EXERCISE 1-1 >

請利用上列程式中的 w 值 (= ['Fly', 'robin', ‘fly’])，將其插入 words (= ['Fly', 'robin', 'fly', 'Up', 'up', 'to', 'the', ‘sky'])，使得 words 的輸出結果如下：

```
>>> words
['Fly', 'robin', 'fly', 'Fly', 'robin', 'fly', 'Fly', 'robin', 'fly',
'Up', 'up', 'to', 'the', 'sky']
```

< EXERCISE 1-1 > 參考程式解答

```
>>> w.reverse()
['fly', 'robin', ‘Fly']
>>> for i in range(2):
      for x in w:
            words.insert(0,x)  # Or, words.insert(3,x)


>>> words
['Fly', 'robin', 'fly', 'Fly', 'robin', 'fly', 'Fly', 'robin', 'fly',
'Up', 'up', 'to', 'the', 'sky']
```

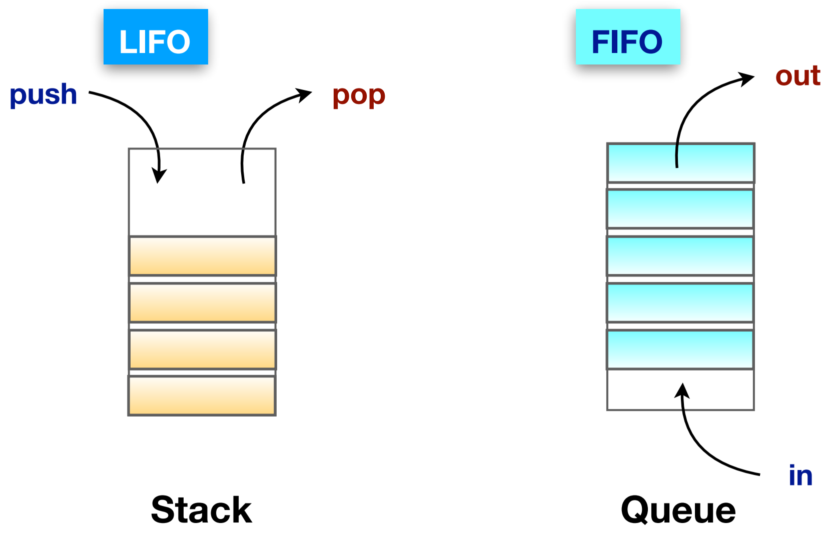
[Example 1.2 Methods for Lists] — `list.sort()` & `list.pop()`

## 承上述 < **EXERCISE 1-1** > 結果:

```
>>> words.sort()    # words 進行排序 (sorting)
>>> words
['Fly', 'Fly', 'Fly', 'Up', 'fly', 'fly', 'fly', 'robin', 'robin',
'robin', 'sky', 'the', 'to', 'up']

>>> words.pop()     # 移除 words 最後一項
'up'
>>> words
['Fly', 'Fly', 'Fly', 'Up', 'fly', 'fly', 'fly', 'robin', 'robin',
'robin', 'sky', 'the', 'to']

>>> words.pop(3)    # 移除 words 中的 'Up' (index = 3)
'Up'
>>> words
['Fly', 'Fly', 'Fly', 'fly', 'fly', 'fly', 'robin', 'robin', 'robin',
'sky', 'the', 'to']
```

[Example 1.3 Stack (堆疊) by Lists] — `list.append()` & `list.pop()`

```
def Stack_LIFO(prompt=' Push to Stack (type null if done): '):
    stack = []
    while True:
        top = input(prompt)
        if top in ('n', 'nu', 'nul', 'null'):
            break
        else:
            stack.append(top)

    print('\n Within Stack: ', stack, end='\n\n')
    print(' <<< Pop the Stack --> LIFO : Last-In-First-Out ')

    for top in stack[:]:    # Could we use "for top in stack:" instead?
            print(' pop Stack : ', stack.pop())
    print('\n *******  Stack = ', stack)
```



< EXERCISE 1-2 > Queue (佇列) 實作 — FIFO

請修改上列 Stack 程式，實作 “佇列 (queue)” 資料結構。

[Hint : pop() 的參數列要如何修改？]

2. List Comprehension (列表精簡)

Python 採用 “List Comprehension (列表精簡)“ 的概念；方便我們產生 list(列表) 資料。 [註： comprehension 有「理解；包含」的意思。]

一般而言，如果我們要使用數學方式表示一個集合(set)，可以採用下列方式：

A = {2n+1| 0<n<10}   ## A = {3,5,7,9,11,13,15,17,19}

B = {x^2| x<12 and x in A}   ## B = { 9, 25, 49, 81, 121}

Python 的 “List Comprehension (列表精簡)“ 概念，採用近似數學表示方式，撰寫 A 和 B 兩個集合列表的表示式，例如：

```
A = [2*n+1 for n in range(1,10)]
B = [x**2 for x in A if x < 12]
```

[Example 2.1 List Comprehension]

```
    ##  List Comprehension
    def list_Compreh():
        A = [2*n+1 for n in range(1,10)]
        B = [x**2 for x in A if x < 12]
        print(A)    #  [3, 5, 7, 9, 11, 13, 15, 17, 19]
        print(B)    #  [9, 25, 49, 81, 121]
```

List Comprehension 的表示法可以幫助我們精簡程式，例如，下列指令：

```
>>> pairs = []
>>> for i in [‘Apple','Android','Asus']:        ##  Nested loops…
      for j in ['Google','Apple','Microsoft']:
          if i != j:
              pairs.append((i, j))

>>> pairs
[('Apple', 'Google'), ('Apple', 'Microsoft'), ('Android', 'Google'),
('Android', 'Apple'), ('Android', 'Microsoft'), ('Asus', 'Google'),
('Asus', 'Apple'), ('Asus', 'Microsoft')]
```

可以簡化為

```
>>> [(i,j) for i in ['Apple', 'Android', 'Asus'] for j in ['Google',
'Apple', 'Microsoft'] if i != j]
[('Apple', 'Google'), ('Apple', 'Microsoft'), ('Android', 'Google'),
('Android', 'Apple'), ('Android', 'Microsoft'), ('Asus', 'Google'),
('Asus', 'Apple'), ('Asus', 'Microsoft')]
```

[Example 2.2   List Comprehension]

```
>>> from math import exp
>>> [str(round(exp(1), i)) for i in range(1, 6)]
['2.7', '2.72', '2.718', '2.7183', ‘2.71828']

>>> from math import pi
>>> pi
3.141592653589793
>>> [float(round(pi*10**2, i)) for i in range(1, 6)]     # pi*r²
[314.2, 314.16, 314.159, 314.1593, 314.15927]
```

< EXERCISE 2-1 >

請利用 list comprehension 表示方式，簡化下列運算

```
>>> pairs = []
>>> for i in [1,3,5]:
        for j in [4,1,2]:
            if i != j:
                pairs.append((i, j, (i+j)/2))

>>> pairs
[(1, 4, 2.5), (1, 2, 1.5), (3, 4, 3.5), (3, 1, 2.0), (3, 2, 2.5),
 (5, 4, 4.5), (5, 1, 3.0), (5, 2, 3.5)]
```

< EXERCISE 2-2 >

(1) 請找出 1 ~ 100 之間的 “質數(prime number)”。

(2) 之後，利用 list comprehension 表示方式，簡化程式。

EXERCISE 2-2 參考解答：

```
[x for x in range(2, 100) if x not in (j for i in range(2, 11) for j in
range(i*2, 100, i))]

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
71, 73, 79, 83, 89, 97]
```

3. Basic Sequence Types — `list, tuple, range`

[Ref: https://docs.python.org/3/library/stdtypes.html#typesseq]

三種基本的序列(sequence)型別： `list` (列表), `tuple` (元組), `range` (範圍)。

3.1 一般序列運算 (Common Sequence Operations)

1. `in` *vs.* `not in` **[NOTE]: `in` 和 `not in` 具有與其他比較運算子相同的優先權！**

```
>>> [x for x in [1,10] if x not in [x*2, 10]]
[1]
```

2. `*(repetition)`運算 — `seq*n` *or* `n*seq`

[Example 3.1 `*(repetition)`運算]

```
>>> list1 = [[1]]
>>> list2 = list1 * 2
>>> list2
[[1], [1]]
>>> list2[0].append(2)
>>> list2
[[1, 2], [1, 2]]     # Q: Why? A: References to list1, [[1]].
>>> list3 = list2 * (-1)
>>> list3
[]             # Q: How come did the elements disappear?
```

請比較下列二者輸出結果：

```
>>> list2[0].append(2)
>>> list2.append(2)
```

[Example 3.2 與 `*(repetition)`運算 比較]

```
>>> list3 = [[] for i in range(4)]
>>> list3
[[], [], [], []]     # Q: References to an empty list ???
>>> for i in range(4):
        list3[i].append(2*i)

>>> list3
[[0], [2], [4], [6]]
```

請比較下列指令輸出結果：

```
>>> list1 = [[]] * 4
>>> list1
[[], [], [], []]
>>> for i in range(4):
        list1[i].append(2*i)

>>> list1    # WHY???
[[0, 2, 4, 6], [0, 2, 4, 6],
[0, 2, 4, 6], [0, 2, 4, 6]]
```

3. 索引(index) — seq(*i*), seq(*i*:*j*), seq(*i*:*j*:*k*)

[Example 3.3]

```
>>> seq = [i**2 for i in range(10)]
>>> seq
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> seq[3]
9
>>> seq[3:9]
[9, 16, 25, 36, 49, 64]
>>> seq[3:9:2]
[9, 25, 49]
```

4. +(concatenation)運算 — seq1 + seq2

[Example 3.4 +(concatenation)運算]

```
>>> seq1 = [chr(i) for i in range(65,71)]
>>> seq1           # integer values to ASCII codes
['A', 'B', 'C', 'D', 'E', ‘F']
>>> seq2 = [chr(i) for i in range(97,103)]
>>> seq2
['a', 'b', 'c', 'd', 'e', 'f']
>>> seq1 + seq2
['A', 'B', 'C', 'D', 'E', 'F', 'a', 'b', 'c', 'd', 'e', ‘f’]

>>> [ord(seq1[i]) for i in range(6)]   # ASCII to integer values
[65, 66, 67, 68, 69, 70]
>>> [ord(seq1[i]) for i in range(6)] + [ord(seq2[i]) for i in range(6)]
[65, 66, 67, 68, 69, 70, 97, 98, 99, 100, 101, 102]
```

< EXERCISE 3-1 >

請問，下列程式的輸出結果分別為何？

```
>>> [ord(seq1[i]) + ord(seq2[i]) for i in range(6)]

>>> [[ord(seq1[i])] + [ord(seq2[i])] for i in range(6)]
```

3.2 可改變的序列型別 (Mutable Sequence Types)

可改變的序列 (Mutable Sequences) 運算，如下：

```
seq[i] = x
seq[i:j] = iterable          seq[i:j:k] = iterable
del seq[i:j]                 del seq[i:j:k]
seq.append(x)
seq.clear()
seq.copy()
seq.insert(i, x)
seq.pop([i])
seq.remove(x)
seq.reverse()
seq.extend(iterable) (or, seq += iterable)
seq *= n
```

3.3 Lists (列表)

（請參考本文件 **“1. 資料型態 list 的相關方法**” 和 “**2. List Comprehension**” 章節。）

3.4 Tuples (元組)

一般而言，`tuple`(元組)是不可改變的序列型別 (Immutable Sequence Types)，但是，`tuple` 可以有可變的元素(例如：`list`)。

`tuple` 通常用於儲存異質資料(heterogeneous data)，亦即不同資料型別的資料集。

class **tuple**([*iterable*])

`tuple` 可以下列方式產生：

— 利用 小括號(parentheses) 產生空元組(empty tuple)： `()`
— 利用緊隨的逗號，產生單一物件元組(a singleton tuple)： `a,` 或 `(a,)`
— 利用逗號區隔，產生 tuple： `a, b, c` 或 `(a, b, c)`
— 利用內建的 `tuple()` 建構子產生元組： `tuple()` 或 `tuple(`*iterable*`)`
— 利用內建的列舉函數 `enumerate(`*iterable, start=0*`)` 回傳 `tuple`

[Example 3.5 tuple packing 運算]

```
>>> t = ()    # Creating an empty tuple…
>>> t
()

>>> t = 'Python',  # Creating a singleton…
>>> t
('Python',)
```

請注意下列輸出結果是一個字串：

```
>>> t = 'Python'
>>> t
'Python'    # 字串(string)
```

```
>>> t = 'Python', 123, 3.1416    #  t = ('Python', 123, 3.1416)
>>> t
('Python', 123, 3.1416)
>>> for i in range(len(t)):
        print(t[i])

Python
123
3.1416


>>> t1 = ('ABC','NBC','CBS'), t, (2.71828, 1.732)  #  Nested tuple…
>>> t1
(('ABC', 'NBC', 'CBS'), ('Python', 123, 3.1416), (2.71828, 1.732))

>>> t2 = ['a','b','c', 1, 2], [11, 22], ('Python', 3.6)
>>> t2                           #  tuple 可以有可變的元素(例如：list)
(['a', 'b', 'c', 1, 2], [11, 22], ('Python', 3.6))

>>> t2[0] = [97, 98, 99]     #  tuple 的元素是不可改變的(immutable)
Traceback (most recent call last):
  File "<pyshell#100>", line 1, in <module>
    t2[0] = [97, 98, 99]
TypeError: 'tuple' object does not support item assignment

>>> t2[2] = 11               #  tuple 的元素是不可改變的(immutable)
Traceback (most recent call last):
  File "<pyshell#102>", line 1, in <module>
    t2[2] = 11
TypeError: 'tuple' object does not support item assignment
```

[Example 3.6 `sequence unpacking` 運算]

```
>>> t1
(('ABC', 'NBC', 'CBS'), ('Python', 123, 3.1416), (2.71828, 1.732))
>>> tv, py, math = t1    # reverse operation: (tv, py, math) = t1
>>> tv
('ABC', 'NBC', 'CBS')
>>> py
('Python', 123, 3.1416)
>>> math
(2.71828, 1.732)

>>> t2
(['a', 'b', 'c', 1, 2], [11, 22], ('Python', 3.6))
>>> an, nu, py3 = t2     # reverse operation: (an, nu, py3) = t2
>>> an
['a', 'b', 'c', 1, 2]
>>> nu
[11, 22]
>>> py3
('Python', 3.6)
```

[Example 3.7 `enumerate(`*`iterable, start=0`*`)` 列舉函數運算]

```
>>> languages = ['Python', 'R', 'Scala']
>>> list(enumerate(languages))
[(0, 'Python'), (1, 'R'), (2, ‘Scala')]  # enumerate() returns tuples…

>>> list(enumerate(languages, start=1))
[(1, 'Python'), (2, 'R'), (3, 'Scala')]

>>> for index, element in enumerate(languages, start = 1):
        print(index, element)

1 Python
2 R
3 Scala
```

[Example 3.8 tuple() 建構子運算]

```
>>> t = tuple()          # Creating an empty tuple…
>>> t
()
>>> t = tuple(range(2, 7))  #  tuple(iterable)
>>> t
(2, 3, 4, 5, 6)

>>> t1
(('ABC', 'NBC', 'CBS'), ('Python', 123, 3.1416), (2.71828, 1.732))
>>> t = tuple(t1[1:2])  #  Using tuple(iterable) to create a singleton
>>> t
(('Python', 123, 3.1416),)

>>> tuple('abc')   # String argument: return a tuple with characters.
('a', 'b', 'c')
>>> tuple('a','b','c')
Traceback (most recent call last):
  File "<pyshell#153>", line 1, in <module>
    tuple('a','b','c')
TypeError: tuple() takes at most 1 argument (3 given)
>>> tuple([‘a','b','c'])
('a', 'b', 'c')
>>> tuple([97, 98, 99])
(97, 98, 99)
>>> tuple(('a','b','c')) # Tuple argument: return the unchanged tuple.
('a', 'b', 'c')

##  Q: 下列二指令都回傳相同 tuple 結果，請問二者差異為何？
>>> tuple([('a','b'),(97,98)])
(('a', 'b'), (97, 98))
>>> tuple((('a','b'),(97,98)))
(('a', 'b'), (97, 98))
```

[Example 3.9 “最大公因數” 和 “費氏數列” — tuple 型別運算應用]

```
# GCD: greatest common divisor
a, b = 21, 144
while a != 0:
   print(' a =\t', a,'\t b =\t',b)
   a, b = b % a, a    # tuple types for SWAP: (a, b) = (b % a, a)
print(" GCD = ", b, end=‘\n')

# Fibonacci series: F(i) = F(i-1) + F(i+1)
a, b = 0, 1
while b < 50:
   print(b, end=' ')
   a, b = b, a+b      # tuple types for SWAP: (a, b) = (a, a+b)
```

[NOTE]: tuple 是利用逗號區隔方式產生；一般而言，小括號只是一個選項，除非用於標示 empty tuple 或者避免造成混淆狀況。

3.5 Text Sequence Type (文字序列資料型別) — `str`

(參考 https://docs.python.org/3/library/stdtypes.html#textseq)

在 Python 程式語言中，文字資料是透過 `str` 物件來進行字串 (strings) 處理；其中，字串 (strings) 是以 Unicode 方式編碼的 不可改變資料序列 (Immutable Sequences)。

字串常值 (string literals) 可以下列方式呈現：

— 單引號 (single quotes)： **`'allows embedded "double" quotes'`**
— 雙引號 (double quotes)： **`"allows embedded 'single' quotes"`**
— 三重引號 (triple quoted)：
  `'''Three single quotes''', """Three double quotes"""`

[NOTE] : 三重引號 (triple quoted) 通常用於一個包含多行的字串 — 所有空格 (whitespace) 也會被包含於字串常值 (string literals)內。

class **str**(*object=''*)

class **str**(*object=b'', encoding='utf-8', errors='strict'*)

(參考 https://docs.python.org/3/library/stdtypes.html#textseq)

3.5.1 字串方法

字串的運算包括 3.1 節 “一般序列運算 (Common Sequence Operations)”之外，也包括下列方法 (部份)： (完整的字串方法，參考 https://docs.python.org/3/library/stdtypes.html#textseq)

str.**capitalize**()	str.**casefold**()
str.**center**(*width*[, *fillchar*])	str.**count**(*sub*[, *start*[, *end*]])
str.**encode**(*encoding="utf-8", errors="strict"*)	
str.**find**(*sub*[, *start*[, *end*]])	str.**index**(*sub*[, *start*[, *end*]])
str.**format**(**args, **kwargs*)	str.**join**(*iterable*)
str.**lower**()	str.**upper**()
str.**strip**([*chars*])	str.**rstrip**([*chars*])
str.**replace**(*old, new*[, *count*])	
str.**split**(*sep=None, maxsplit=-1*)	
str.**splitlines**([*keepends*])	

[Example 3.10 — str 型別運算應用]

```
>>> string = '''Towers of gold are still too little
These hands could hold the world but it'll
Never be enough
Never be enough
For me'''

>>> string
"Towers of gold are still too little\nThese hands could hold the
world but it'll\nNever be enough\nNever be enough\nFor me"

>>> string.lower()
"towers of gold are still too little\nthese hands could hold the
world but it'll\nnever be enough\nnever be enough\nfor me"

>>> string.split(sep=' ')
['Towers', 'of', 'gold', 'are', 'still', 'too', 'little\nThese',
'hands', 'could', 'hold', 'the', 'world', 'but', "it'll\nNever",
'be', 'enough\nNever', 'be', 'enough\nFor', 'me']

>>> string.split(sep='\n')
['Towers of gold are still too little', "These hands could hold
the world but it'll", 'Never be enough', 'Never be enough', 'For
me']
>>> string_list = string.splitlines()
>>> string_list
['Towers of gold are still too little', "These hands could hold
the world but it'll", 'Never be enough', 'Never be enough', 'For
me']

>>> string1 = []
>>> for line in string_list:
        string1.append(line.split(sep=' '))

>>> string1
[['Towers', 'of', 'gold', 'are', 'still', 'too', 'little'], ['These',
'hands', 'could', 'hold', 'the', 'world', 'but', "it'll"], ['Never',
'be', 'enough'], ['Never', 'be', 'enough'], ['For', 'me']]
```

[NOTE]:

有關於 "Binary Sequence Types — bytes, bytearray, memoryview"
請參考 https://docs.python.org/3/library/stdtypes.html#binaryseq

4. Sets (集合)

set(集合)是 Python 語言的一種資料型別(data type)。

在一個 set(集合)中，其元素不會重複出現，也不會依次序儲存(unordered collection)。

set 通常應用於成員測試(membership testing) 和 去除重複的條目(duplicate entries)。

set 物件也用於數學運算，例如：聯集(union)、交集(intersection)、...

set 物件產生方式如下：

— 利用 大括號(curly braces，{}) 或 set() 函數產生集合物件

— 空集合(empty set) 一定要使用 set() 函數產生，不可以使用空的大括號{}；因為後者將用於產生一個空字典(empty dictionary)。

(參見下一節有關 “dictionary” 說明)

[Example 4.1 set 物件產生]

```
>>> words = {'Hello','world','hello','Python','hello','sets'}
>>> words            #  去除重複的條目
{'Python', 'hello', 'sets', 'world', 'Hello'}  # unordered collection

>>> 'sets' in words        #  成員測試(membership testing)
True
>>> 'Sets' in words
False

>>> word_py1 = {'Python'}
>>> word_py2 = {'Pythagoras'}
>>> word_py1
{'Python'}
>>> word_py2
{‘Pythagoras'}

>>> word_py1 = set('Python')
>>> word_py1
{'t', 'P', 'y', 'n', 'h', 'o'}
>>> word_py2 = set('Pythagoras')
>>> word_py2
{'t', 's', 'P', 'r', 'y', 'h', 'g', 'o', 'a'}
```

[Example 4.2 set 物件數學運算]

```
>>> word_py1 = set('Python')
>>> word_py2 = set('Pythagoras')
>>> word_py1
{'t', 'P', 'y', 'n', 'h', 'o'}
>>> word_py2
{'t', 's', 'P', 'r', 'y', 'h', 'g', 'o', 'a'}

>>> word_py2 & word_py1         #  intersection
{'t', 'P', 'y', 'h', 'o'}
>>> word_py2 | word_py1         #  union
{'t', 's', 'P', 'r', 'y', 'n', 'h', 'g', 'o', 'a'}
>>>
>>> word_py2 - word_py1         #  difference : word_py2 - word_py1
{'g', 's', 'r', 'a'}
>>> word_py1 - word_py2         #  difference : word_py1 - word_py2
{'n'}
>>> word_py2 ^ word_py1         #  symmetric difference
{'s', 'r', 'g', 'a', 'n'}
```

[NOTE] : 對稱差集 (symmetric difference) —> XOR

兩個集合 A 和 B 中，未曾重複出現的元素將會留下，重複出現的元素將被移除。

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

[Example 4.3 set 可以使用 list comprehension 運算]

```
>>> word_py1 = set('Python')
>>> word_py2 = set('Pythagoras')
>>> {x for x in word_py2 if x in word_py1}       #  intersection
{'t', 'P', 'y', 'h', 'o'}
>>> {x for x in word_py2 if x not in word_py1}   #  difference
{'g', 's', 'r', 'a'}
>>> {x for x in word_py1 if x not in word_py2}   #  difference
{'n'}
```

< EXERCISE 4-1 >

在 **Example 4.2** 中的 set intersection 和 difference 改寫成 list comprehension 的結果，分別顯示於 **Example 4.3** 中。

請問，如何改寫 set union 和 symmetric difference 程式部份？

5. Dictionaries (字典)

- dictionary(字典)是 Python 常用的一種資料型別(data type)，屬於 mapping type。
- 因此，可將 dictionary 視為一個不依次序儲存的 鍵值對 (key-value pairs)資料集。
- 由於 鍵值對 (key-value pairs) 中的 key 是不可修改的 (immutable) 資料，所以，任何 immutable 的資料型別都可以做為 key 使用 (例如：*numbers, strings, tuples, …*) 。
- 若要將 tuple 當作 key 使用時，要注意其元素不可以是 mutable 的資料型別 (例如：*lists*) 。
- list(d.keys()) : 將回傳一個 dictionary d 的 key 列表 (任意存放)
- sorted(d.keys()) : 將回傳一個 dictionary d 的 key 排序列表
- 使用 in 來檢查 key 是否在 dictionary 中。
- *class* **dict**(***kwarg*)
- *class* **dict**(*mapping, **kwarg*)
- *class* **dict**(*iterable, **kwarg*)
- dictionary 物件產生方式如下：

  — 利用 一對大括號(curly braces，{}) 產生空字典 (empty dictionary)

  — 在一對大括號之間，以逗號區隔鍵值對方式，來產生 dictionary 物件

  — 利用內建的 dict() 建構子產生 dictionary 物件

[Example 5.1 dictinary 物件產生方式]

```
>>> gpa0 = {'C':2.0, 'B':3.0, 'A':4.0, 'F':'Fail'}
>>> gpa0
{'C': 2.0, 'B': 3.0, 'A': 4.0, 'F': ‘Fail'}
>>> gpa1 = dict(A=4.0, B=3.0, C=2.0, F=‘Fail')
{'A': 4.0, 'B': 3.0, 'C': 2.0, 'F': ‘Fail'}

>>> gpa2 = dict({'B':3.0, 'A':4.0, 'F':'Fail', ‘C':2.0})
>>> gpa3 = dict([('F', 'Fail'), ('C', 2.0), ('A', 4.0), ('B', 3.0)])
>>> gpa4 = dict(zip(['F', 'C', 'B', 'A'], ['Fail', 2.0, 3.0, 4.0]))
>>> gpa0 == gpa1 == gpa2 == gpa3 == gpa4
True
```

[Example 5.2 dictinary 運算]

```
>>> fastfood = {'Cheeseburger':50, 'fries':25, 'Coke':35, 'coffee':45}
>>> items = fastfood.keys()
>>> prices = fastfood.values()
>>> items
dict_keys(['Cheeseburger', 'fries', 'Coke', 'coffee'])
>>> prices
dict_values([50, 25, 35, 45])
>>> sum(prices)
155

>>> list(items)
['Cheeseburger', 'fries', 'Coke', ‘coffee']
>>> sorted(items)
['Cheeseburger', 'Coke', 'coffee', 'fries']
>>> list(prices)
[50, 25, 35, 45]
>>> sorted(prices)
[25, 35, 45, 50]

>>> del fastfood[‘coffee']
>>> fastfood
{'Cheeseburger': 50, 'fries': 25, 'Coke': 35}
>>> list(fastfood)
['Cheeseburger', 'fries', 'Coke']
>>> list(items)
['Cheeseburger', 'fries', 'Coke']
>>> list(prices)
[50, 25, 35]

>>> fastfood & {'Hamburger','OJ','Coke'}
Traceback (most recent call last):
  File "<pyshell#240>", line 1, in <module>
    fastfood & {'Hamburger','OJ','Coke'}
TypeError: unsupported operand type(s) for &: 'dict' and ‘set'

##  Set operations for keys……
>>> items & {'Hamburger','OJ','Coke'}
{‘Coke'}
>>> items | {'Hamburger','OJ','Coke'}
{'OJ', 'Hamburger', 'Cheeseburger', 'Coke', 'fries'}
```

[Example 5.3 dictinary 運算 - 2]

```
>>> items.append('coffee')
Traceback (most recent call last):
  File "<pyshell#248>", line 1, in <module>
    items.append('coffee')
AttributeError: 'dict_keys' object has no attribute 'append'

>>> fastfood.append({'coffee':25})
Traceback (most recent call last):
  File "<pyshell#251>", line 1, in <module>
    fastfood.append({'coffee':25})
AttributeError: 'dict' object has no attribute ‘append’

##  Adding a key-value pair into a dictionary……
>>> fastfood['coffee'] = 25
>>> fastfood
{'Cheeseburger': 50, 'fries': 25, 'Coke': 35, 'coffee': 25}

>>> 'coffee' in fastfood
True
>>> 'OJ' in fastfood
False
```

[Example 5.4 dict comprehension]

```
>>> {x : x**2+2*x+1 for x in range(0,6)}
{0: 1, 1: 4, 2: 9, 3: 16, 4: 25, 5: 36}

>>> from math import log
>>> {x : round(log(x, 10), 4) for x in (1,2,3,5,10,50,100)}
{1: 0.0, 2: 0.301, 3: 0.4771, 5: 0.699, 10: 1.0, 50: 1.699, 100: 2.0}
```

< EXERCISE 5-1 >

(1) 請利用 dictionary 建立一組 “帳號：密碼” (鍵值對)，可用於會員登入、ATM、email 登入...

(2) 請撰寫程式，可輸入個人帳號、密碼，並執行帳密驗證，輸出正確與否訊息。

6. Looping Techniques [Ref: 5.6.]

- When looping through dictionaries, the key and corresponding value can be retrieved at the same time using the **`items()`** method.

[Example 6.1]

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
        print(k, v)

gallahad the pure
robin the brave
```

- When looping through a sequence, the position index and corresponding value can be retrieved at the same time using the **`enumerate()`** function.

[Example 6.2]

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
        print(i, v)

0 tic
1 tac
2 toe
```

- To loop over two or more sequences at the same time, the entries can be paired with the **`zip()`** function.

[Example 6.3]

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
        print('What is your {0}?  It is {1}.'.format(q, a))

What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

- To loop over a sequence in reverse, first specify the sequence in a forward direction and then call the **`reverse()`** function.

[Example 6.4]

```
>>> for i in reversed(range(1, 10, 2)):
        print(i)

9
7
5
3
1
```

- To loop over a sequence in sorted order, use the **`sorted()`** function which returns a new sorted list while leaving the source unaltered.

[Example 6.5]

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> for f in sorted(set(basket)):
        print(f)

apple
banana
orange
pear
```

- It is sometimes tempting to change a list while you are looping over it; however, it is often simpler and safer to create a new list instead.

[Example 6.6]

```
>>> import math
>>> raw_data = [56.2, float('NaN'), 51.7, 55.3, 52.5, float('NaN'), 47.8]
>>> filtered_data = []
>>> for value in raw_data:
        if not math.isnan(value):
            filtered_data.append(value)

>>> filtered_data
[56.2, 51.7, 55.3, 52.5, 47.8]
```

7. More on Conditions [Ref: 5.7.]

- When The Boolean operators **`and`** and **`or`** are so-called **short-circuit operators**: their arguments are evaluated from left to right, and evaluation stops as soon as the outcome is determined.
- For example, if `A` and `C` are true but `B` is false, `A and B and C` does not evaluate the expression `C`.
- When used as a general value and not as a Boolean, the return value of a short-circuit operator is the last evaluated argument.

[Example 7.1]

```
>>> string1, string2, string3 = '', 'Trondheim', 'Hammer Dance'
>>> non_null = string1 or string2 or string3
>>> non_null
'Trondheim'
```

8. Comparing Sequences & Other Types [Ref: 5.8.]

- Sequence objects may be compared to other objects with the same sequence type.
- The comparison uses **lexicographical ordering**:
  1. first the first two items are compared, and if they differ this determines the outcome of the comparison; if they are equal, the next two items are compared, and so on, until either sequence is exhausted.
  2. If two items to be compared are themselves sequences of the same type, the lexicographical comparison is carried out recursively.
  3. If all items of two sequences compare equal, the sequences are considered equal.
  4. If one sequence is an initial sub-sequence of the other, the shorter sequence is the smaller (lesser) one.
  5. Lexicographical ordering for strings uses the Unicode code point number to order individual characters.

[Example 8.1]

```
(1, 2, 3)                < (1, 2, 4)
[1, 2, 3]                < [1, 2, 4]
'ABC' < 'C' < 'Pascal' < 'Python'
(1, 2, 3, 4)             < (1, 2, 4)
(1, 2)                   < (1, 2, -1)
(1, 2, 3)               == (1.0, 2.0, 3.0)
(1, 2, ('aa', 'ab'))     < (1, 2, ('abc', 'a'), 4)
```

Each output of these statements is **True**.