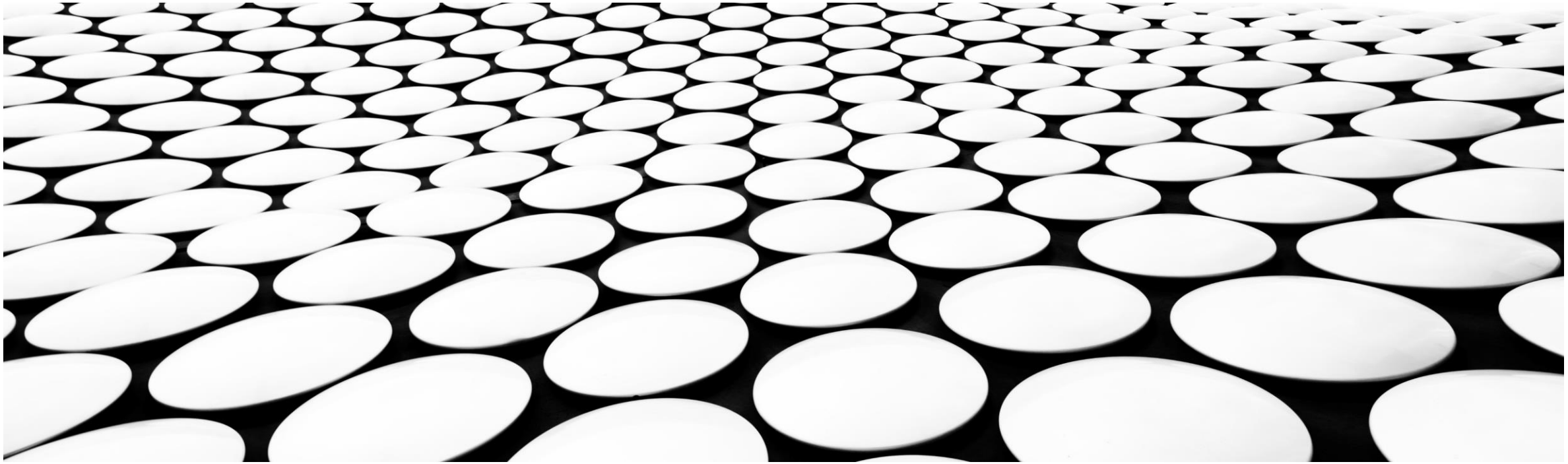
Python 字串與群集型態

MARs教育團隊 – 孫汶琳 交大電機博士候選人



目錄

- 字串 String
- 串列 List
- 資料組 Tuple
- 集合 Set
- 字典 Dictionary

字串 (String)

String方法列表: https://www.w3schools.com/python/python_ref_string.asp

- 在Python中，String可以使用雙引號或者單引號
- 在學過函式(Function) 之後，我們可以分析最常使用的函式 print() :

```
print('Hello, world!')
```

函式名稱　String型態的參數

- 賦予一個變數 String 型態的值
  - 單行

```
str = 'Hello'
```

或

```
str = "Hello"
```

  - 多行

```
str = ' ' ' Hello, everyone!
I am Winnie.
I love Python!!!!' ' '
```

或

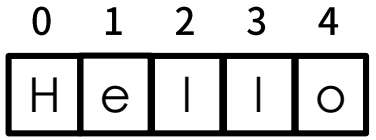
```
str = """Hello, everyone!
I am Winnie.
I love Python!!!!"""
```

- 如果單引號和雙引號內什麼都沒有，就是空字串的意思
  - 之後在使用不同的字串處理函式時會常看到

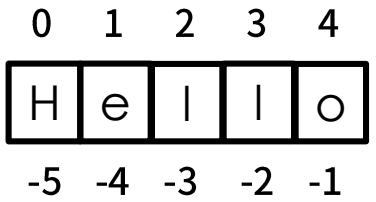
字串 (String) - 編號 (Index)

- 在Python中，String是一種陣列 (arrays)，也就是由一格一格的字母照順序組成的
- 我們可以使用中括號 [**編號**] 去選擇我們所要使用的格子，如 str[2]
  - 陣列的編號都是從 0 開始的唷!

```
str = 'Hello'
```



- Python的編號可以是負數，也就是說如 str[-1]，這是合法且有意義的!



字串 (String) - 切割 (Slicing)

- Python可以透過簡潔的語法來選擇所要的字串區段

[起始編號 : 結束編號]

所選擇的範圍包含起始編號的格子，
但不包含結束編號的格子唷!

- 常見的使用方式:

```
str = 'Hello'
```

  - 選擇特定區段

```
print(str[1:3])
```

```
'el'
```

  - 從頭到特定編號

```
print(str[:3])
```

```
'Hel'
```

  - 從特定編號到尾

```
print(str[1:])
```

```
'ello'
```

亦可使用負數的編號，不過
編號仍要由小至大的唷!
例如: str[-5:-2]，就是指'Hel'

字串 (String) - 調整

```
str = 'How are you?'
```

- Python內建多樣的字串處理方法(methods)，使用方式為: 變數.方法
- 這些方法會 **回傳** 該變數經過指定方法處理後的結果，所以並不會改變原本的字串變數
  - 轉換大寫 **upper()**、小寫 **lower()**

```
print(str.upper())
print(str.lower())
```

```
'HOW ARE YOU?'
'how are you?'
```

  - 移除空白符號 **strip()**

```
print(str.strip())
```

```
'Howareyou?'
```

  - 字串取代 **replace()**，第一個參數為目標字串，第二個參數為替代的字串

```
print(str.replace('How', 'What'))
```

```
'What are you?'
```

  - 字串切割 **split()**，參數代表以什麼字串分割，會回傳一個由切割後片段組成的串列 (List)

```
print(str.split('?'))
print(str.split(' ')) # 用空白鍵分割
```

```
['How are you', '']   # 第二個為空字串
['How', 'are', 'you?']
```

字串 (String) - 串接

- 一般來說，加號(+)式用來做數學運算的，不過Python也可以透過加號來串接字串

```
question = 'How are you?'
name = 'Winnie'

str1 = question + name
str2 = question + ' ' + name
str3 = question[:-1] + ', ' + name + '?'

print(str1)  # 'How are you?Winnie'
print(str2)  # 'How are you? Winnie'
print(str3)  # 'How are you, Winnie?'
```

更多使用方法: https://www.w3schools.com/python/python_string_formatting.asp

字串 (String) - 格式化(Format)

- 雖然可以透過加號來串接字串，但是若是所要串接的變數不是字串型態，就會變得很麻煩...

```
str1 = 'I am '
str2 = ' years old.'
age = 18

print(str1 + age + str2)        # 錯誤! 因為age不是字串型態!
print(str1 + str(age) + str2)  # 可以! 因為大家都是字串型態了
```

- 為此，Python提供 **format()** 這個方法，分開陳述所要輸出的語句和變數

```
str = 'I am {} years old.'
age = 18
print(str1.format(age))    # 'I am 18 years old.'
```

字串 (String) - 格式化(Format)

- 若沒有為字串內的 {} 加上編號，則變數會照所列順序一一對應填上

```
str = 'My name is {}. I am {} years old. I love {}.'
lover = 'Alen'
age = 18
name = 'Winnie'
print(str.format(lover, age, name))
# 'My name is Alen. I am 18 years old. I love Winnie.'
```

- 若有加上編號，則變數會依序填入對應的位置

```
str = 'My name is {2}. I am {1} years old. I love {0}.'
love = 'Alen'
age = 18
name = 'Winnie'
print(str.format(lover, age, name))
# 'My name is Winnie. I am 18 years old. I love Alen.'
```

字串 (String) - 跳脫字元(Escape Character)

- 有時候我們想要在字串內加入像是單引號或雙引號等保留的符號，直接使用是不行的

```
str = "My nickname is "Pooh"."      # 錯誤!
```

- 這時就可以在該字元前加上反斜線(\)就會變成跳脫字元，跳脫原本保留符號的束縛!

```
str = "My nickname is \"Pooh\"."   # 可以!
```

跳脫字元	說明
\'	單引號
\\	反斜線
\n	換行
\r	迴車
\t	Tab
\ooo	8進位值，oo部分可填值
\xhh	16進位值，hh部分可填值

串列 (List)

List方法列表 https://www.w3schools.com/python/python_lists_methods.asp

- List 用來在一個變數中儲存多個物品，是Python內建的四種容器型態之一
- 使用中括號 [ ] 來創建和獲取其中的物品

```
fruits = []  # 這是一個空的 List
fruits = [ 'apple', 'banana', 'cherry' ]  # 這是 List 給值的方式
fruits = [ 'apple', 'banana', 'cherry', 'banana' ]  # 物品可以重複
fruits = [ 'apple', True, 10, 'banana' ]  # 物品可以是不同資料型態的
```

- 可以使用內建的 len() 函式來確認 List 的長度(物品數量)

```
fruits = [ 'apple', 'banana', 'cherry' ]
print( len(fruits) )  # 3
```

串列 (List) - 取得物品

```
fruits = [ 'apple', 'banana', 'cherry', 'durian', 'egg fruit', 'fig' ]
```

- List 中的物品是有順序的，可以用編號的選取要取用的物品，而編號從 0 開始，亦可使用負的編號

```
print( fruits[2] )  # 'cherry'
print( fruits[-2] )  # 'banana'
```

- 適用 String 選取片段 (切割) 的方式，亦不會改變原來的 List

```
print( fruits[2:4] )  # [ 'cherry', 'durian' ]
print( fruits[:3] )    # [ 'apple', 'banana', 'cherry' ]
print( fruits[3:] )    # [ 'durian', 'egg fruit', 'fig' ]
print( fruits[-4:-1] )    # ['cherry', 'durian', 'egg fruit' ]
```

- 可使用 **in** 和 **not in** 檢查物品是否在 List 中

```
if 'apple' in fruits :
    print('Yes, it is in our fruit list!')
```

串列 (List) – 改變物品

```
fruits = [ 'apple', 'banana', 'cherry', 'durian' ]
```

- 改變特定物品的值

```
fruits[1] = 'black berry'
print(fruits)  # [ 'apple', 'black berry', 'cherry', 'durian' ]
```

- 改變特定範圍內物品的值
  - 前後物品數量一樣多的話就單純取代

```
fruits[1:3] = [ 'black berry', 'cantaloupe' ]
print(fruits)  # [ 'apple', 'black berry', 'cantaloupe', 'durian' ]
```

  - 如果前後物品數量不一致，選取區間內的物品移除，替代物品加入，List 長度會隨之更動

```
fruits[1:2] = [ 'black berry', 'cantaloupe' ]
print(fruits)  # [ 'apple', 'black berry', 'cantaloupe', 'cherry', 'durian' ]
```

串列 (List) – 加入物品

```
fruits = [ 'apple', 'banana', 'cherry' ]
```

- 從後面加入物品 **append()**

```
fruits.append('durian')
print(fruits)  # [ 'apple', 'banana', 'cherry', 'durian' ]
```

- 插入一樣物品 **insert()**

```
fruits.insert(1, 'black berry')
print(fruits)  # [ 'apple', 'black berry', 'banana', 'cherry' ]
```

- 插入一個容器中的所有物品 **extend()**
  - 只要容器是可疊代的(iterable) 都可以作為 extend 的參數，像是 String, List, Tuple, Dictionary, Set 都是可以的

```
another_fruits = [ 'durian', 'egg fruit' ]
fruits.extend(another_fruits)
print(fruits)  # [ 'apple', 'banana', 'cherry', 'durian', 'egg fruit' ]
```

串列 (List) – 移除物品

```
fruits = [ 'apple', 'banana', 'cherry', 'banana' ]
```

- 移除特定物品 **remove()** ，只有最先遇到的會移除

```
fruits.remove('banana')
print(fruits)  # [ 'apple', 'cherry', 'banana' ]
```

- 移除特定編號的物品 **insert()** ，若沒有指定編號，則會移除最後一個物品

```
fruits.pop(1)
print(fruits)  # [ 'apple', 'cherry', 'banana' ]
fruits.pop()
print(fruits)  # [ 'apple', 'cherry' ]
```

- 清空整個 List **clear()**

```
fruits.clear ()
print(fruits)  # []
```

串列 (List) – 使用迴圈遍歷物品

```
fruits = [ 'apple', 'banana', 'cherry', 'banana' ]
```

- 使用 for 迴圈，遍歷整個 List

```
for fruit in fruits :
    print(fruit)
```

- 使用 for 迴圈，透過編號遍歷整個 List

```
for idx in range(len(fruits)) :
    print(fruits[idx])
```

- 使用 while 迴圈，透過編號遍歷整個 List

```
idx = 0
while idx < len(fruits) :
    print(fruits[idx])
    idx = idx + 1
```

- 使用**串列的綜合表達式 (List Comprehension)**，簡潔的遍歷整個 List

```
[ print(fruit) for fruit in fruits ]
```

串列 (List) – 排序物品 I

```
fruits = [ 'apple', 'cherry', 'Banana' ]
nums = [ 1, 10, 5 ]
```

- 預設是遞增排序

```
fruits.sort()  # 照字母排序，大小寫有別，大寫在前
print(fruits)  # [ 'Banana', 'apple', 'cherry' ]
nums.sort()  # 照數字大小排序
print(nums)  # [ 1, 5, 10 ]
```

- 用有名字的參數讓他遞減排序

```
fruits.sort(reverse = True)  # 照字母排序，大小寫有別
print(fruits)  # [ 'cherry', 'apple', 'Banana' ]
nums.sort(reverse = True)  # 照數字大小排序
print(nums)  # [ 10, 5, 1 ]
```

- 預設是大小寫有別(case-sensitive)，若要變成忽略大小寫(case-insensitive)，則可以設定 key 參數

```
fruits.sort(key = str.lower)  # 照字母排序，忽略大小寫
print(fruits)  # [ 'apple', 'Banana', 'cherry' ]
```

串列 (List) – 排序物品 II

```
fruits = [ 'cherry', 'apple', 'banana']
nums = [ 1, 10, 5 ]
```

- 把整個序列的物品順序倒過來 **reverse()**

```
fruits.reverse()
print(fruits)  # [ 'banana', 'apple', 'cherry' ]
nums.reverse()
print(nums)  # [ 5, 10, 1 ]
```

- 自定義規則，使用函式的回傳值還作為排序的依據，由小到大排列

```
def myfunc(n) :  # 離 5 越近的值排越前面
   return abs(n - 5)

nums.sort(key = myfunc)
print(nums)  # [ 5, 1, 10 ]
```

串列 (List) – 複製串列

- 單純的使用等號賦值是不能真正的複製串列的!

```
fruits = [ 'apple', 'banana', 'cherry' ]
fruits2 = fruits   # 不是真正的複製，兩個變數指向同一個串列!
fruits2.pop()
print(fruits)  # [ 'apple', 'banana' ]
```

- 若要真正的複製，可以使用 List 內建的方法 copy() 或者 Python內建的函式 list()!

```
fruits = [ 'apple', 'banana', 'cherry' ]
fruits2 = fruits.copy()   # 真正的複製一個新的串列
fruits3 = list(fruits)      # 真正的複製一個新的串列
fruits2.pop()
fruits3.pop(1)
print(fruits)    # [ 'apple', 'banana', 'cherry' ]
print(fruits2)  # [ 'apple', 'banana' ]
print(fruits3)  # [ 'apple', 'cherry' ]
```

串列 (List) – 串接串列

```
fruits = [ 'cherry', 'apple', 'banana']
nums = [ 1, 10, 5 ]
```

- 可以直接使用加號 + 串接

```
mix = fruits + nums
print(mix)  # [ 'banana', 'apple', 'cherry', 1, 10, 5 ]
```

- 亦可使用 **extend()** 這個方法串接

```
fruits.extend(nums)
print(fruits)  # [ 'banana', 'apple', 'cherry', 1, 10, 5 ]
```

- 當然也可以用迴圈逐一把物品加進去另一個串列

```
for num in nums :  # 離 5 越近的值排越前面
    fruits.append(num)
print(fruits) # [ 'banana', 'apple', 'cherry', 1, 10, 5 ]
```

資料組(Tuple) Tuple方法列表 https://www.w3schools.com/python/python_tuples_methods.asp

- Tuple 基本上和 List 在概念和用法和上很像，唯一的區別是 Tuple 創建後是**不可以更動**的 (unchangeable)!
  - 也就是說它不能像 List 一樣新增、移除、更新內容物!
- Tuple 的創建使用的是**小括號 ()** ，不過當內容物只有一個的時候要在後面加上**逗號 ,** 才能順利創建成 Tuple 唷!

```
tuple1 = ('cherry', 'apple', 'banana')  # 這是一個長度 3 的Tuple
tuple2 = ('cherry',)                    # 這是一個長度 1 的Tuple
tuple1 = ('cherry')                     # 這不是一個 Tuple!
```

- 和 List 一樣，是有序的，編號從 0 開始，編號可以為負數，使用**中括號 [編號]** 來獲取特定編號的物品
- 也和 List 一樣，內容物可以重複!

```
fruits = ('cherry', 'apple', 'banana', 'apple', 'apple')
print(fruits[2])                    # 'banana'
print(fruits[1:3])                  # ('apple', 'banana', 'apple')
print(fruits[-4:-2])                # ('apple', 'banana', 'apple')
```

資料組(Tuple) – 更動物品

- 因為Tuple 創建後是**不可以更動**的 (unchangeable)，所以在 List 時介紹的各種新增、移除、更新的方法都是不可以使用的!
- 但我們仍然可以用一些偷吃步的方式去改變 Tuple 的值，簡單的概念就是: **先變成 List 再變回 Tuple!**
  - 先把 Tuple 變成 List 就可以利用前面所學的 List 修改方法去更動物品，最後再把 List 變回 Tuple即可

```
a_tuple = ('cherry', 'apple', 'banana')  # 這是一個不能更動的 Tuple
temp_list = list(a_tuple)                # 這是一個可以更動的 List
temp_list.append('durian')
a_tuple = tuple(temp_list)               # 這是一個被更動過的 Tuple
```

資料組(Tuple) – 分裝 (unpacking)

- 一般我們把創建Tuple看成打包(packing)一堆物品，因此當我們想要把其中的物品拆分給不同的變數時，我們就稱他為分裝(unpacking)

```
a_tuple = ('cherry', 'apple', 'banana')
(small, middle, big) = a_tuple          # 將a_tuple分裝給三個變數
```

- 分裝時可以使用星號 * 來把多個物品分給同一個變數，而這些物品會被裝在 List 中

```
a_tuple = ('cherry', 'apple', 'banana', 'durian', 'egg fruit')
(small, *middle , big) = a_tuple     # 將a_tuple分裝給三個變數
print(small)                         # 'cherry'
print(middle)                        # [ 'apple', 'banana', 'durian' ]
print(big)                           # 'egg fruit'
```

集合(Set)

Set方法列表 https://www.w3schools.com/python/python_sets_methods.asp

- Python的集合是沒有順序、沒有編號、不可更改的、不可有重複物品的!
  - 不過雖然不能更改原有的物品，但是**可以新增和刪除**物品唷!
- Set 使用大括號 **{ }** 來創建，因為他的物品沒有編號也沒有取名，因此不能用編號 (index) 或名稱 (key) 來指定，不過仍然可以用 **for** 迴圈去遍歷，也可以用 **in/not in** 去確認是否包含特定物品

```
fruits = {'cherry', 'apple', 'banana', 'apple'} #會自動去除重複的物品
print(fruits)                                   # {'cherry', 'apple', 'banana'}

# 可使用 for 迴圈來遍歷
for fruit in fruits :
    print(fruit)

# 可使用 in/not in 來檢視特定物品是否在集合內
print('apple' in fruits)
```

集合(Set) – 新增物品

- 雖然不能更改原有的物品，但是**可以新增**物品唷!
- 新增一個物品 **add()**

```
fruits = { 'cherry', 'apple', 'banana' }
fruits.add('durian')
print(fruits) # {'cherry', 'apple', 'banana', 'durian'}
```

- 新增任意可疊代容器內的物品 **update()**

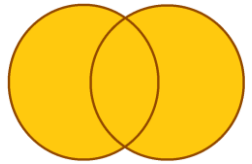
```
fruits = { 'cherry', 'apple' }
another_fruits = [ 'durian', 'banana' ]
the_other_fruits = ( 'egg fruit', 'fig' )
fruits.update(another_fruits)
print(fruits) # {'cherry', 'apple', 'banana', 'durian'}
fruits.update(the_other_fruits)
print(fruits) # {'cherry', 'apple', 'banana', 'durian', 'egg fruit', 'fig'}
```

集合(Set) – 集合的運算

```
fruits_a = { 'cherry', 'apple', 'banana' }
fruits_b = { 'cherry', 'durian', 'banana', 'egg fruit' }
```

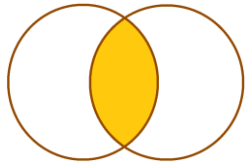
- 分成兩種系列:
  - **有update字尾**: 不會產生新的集合，而是更新原集合，如: **update()**, **intersection_update()**, **symmetric_difference_upate()**
  - **沒有update字尾**: 產生新的集合，原集合不變，如: **union()**, **intersection()**, **symmetric_difference()**
- 聯集

```
fruits_c = fruits_a.union(fruits_b)
print(fruits_c) # {'cherry', 'apple', 'banana', 'durian', 'egg fruit'}
fruits_a.update(fruits_b)
print(fruits_a) # {'cherry', 'apple', 'banana', 'durian', 'egg fruit'}
```



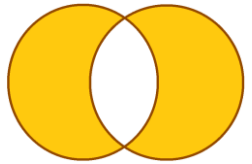
- 交集

```
fruits_c = fruits_a.intersection(fruits_b)
print(fruits_c) # {'cherry', 'banana'}
```



- 對稱差集

```
fruits_c = fruits_a.symmetric_difference (fruits_b)
print(fruits_c) # {'apple', 'durian', 'egg fruit'}
```



Dictionary方法列表 https://www.w3schools.com/python/python_dictionaries_methods.asp

字典(Dictionary)

- 字典是用來儲存 鍵值對 (key-value pairs) 的，之後也是用名字來取值
  - 鍵就是名字，值就是資料值
- 從 Python 版本 3.7 以後，字典是有序的，可以更動的，不允許重複物品
- 和集合 Set 一樣，是使用大括號 **{ }** 來創建，但是不是直接給資料值，而是要給鍵值對
  - 鍵 (key) 會是一個字串，接下來會用一個冒號 **:** 來分隔鍵和資料值
- 使用中括號 **[鍵 key]** 取值

```
person = {
    'name' : 'Winnie Sun',
    'age' : 18,
    'married' : False,
    'fruits' : [ 'grape', 'watermelon', 'dragon fruit' ]
}

print(person['age'])  # 18
```

字典(Dictionary) – 存取 I

```
person = {
    'name' : 'Winnie Sun',
    'age' : 18,
    'married' : False,
}
```

- 可以使用 **[鍵 key]** 或者 **get()** 方法來取值

```
print(person['age'])          # 18
print(person.get('name'))  # 'Winnie Sun'
```

- 取得鍵 key 的列表 **keys()** 方法
  - 並非直接給予一個 List ，因此若字典有更動，也不用重新取得

```
x = person.keys()
print(x)  # dict_keys(['name', 'age', 'married'])
person['height'] = 160.9    # 新增一個項目
print(x)  # dict_keys(['name', 'age', 'married', 'height' ])
```

- 取得值 value 得列表 **values()** 方法
  - 並非直接給予一個 List，因此若資料值有更動，也不用重新取得

```
x = person.keys()
print(x)  # dict_values(['Winnie Sun', 18, False])
person['age'] = 20    # 更新一個項目
print(x)  # dict_values(['Winnie Sun', 20, False])
```

字典(Dictionary) – 存取 II

```
person = {
    'name' : 'Winnie Sun',
    'age' : 18,
    'married' : False,
}
```

- 取得物品的列表 items() 方法，物品(鍵值對)會被包成 Tuple
  - 並非直接給予一個 List ，因此若字典有更動，也不用重新取得

```
x = person.items()
print(x)  # dict_items([ ('name', 'Winnie Sun'), ('age', 18), ('married', False) ])
person['married'] = True    # 更新一個項目
print(x)  # dict_items([ ('name', 'Winnie Sun'), ('age', 18), ('married', True) ])
```

- 檢查鍵 key 是否存在 **in/not in**

```
if 'age' in person :
    print('Yes, it exists!' )
```

字典(Dictionary) – 修改字典

```
person = {
    'name' : 'Winnie Sun',
    'age' : 18,
    'married' : False,
}
```

- 可以使用 **[鍵 key]** 直接賦值或更新值，若鍵 key 不存在，則新增到字典中

```
person['age'] = 20
person['height'] = 160.9
```

- 亦可使用 **update()** 方法傳入一個字典來新增或修改原本的字典的鍵值對

```
print(person)  # { 'name': 'Winnie Sun', 'age' : 18, 'married': False }
person.update( { 'age' : 20, 'height' : 160.9 } )
print(person)
# { 'name': 'Winnie Sun', 'age' : 20, 'married': False, 'height': 160.9 }
```

字典(Dictionary) – 刪除物品

```
person = {
    'name' : 'Winnie Sun',
    'age' : 18,
    'married' : False,
}
```

- 使用 **pop()** 移除指定的鍵 key

```
print(person)  # { 'name': 'Winnie Sun', 'age' : 18, 'married': False }
person.pop('age')
print(person)  # { 'name': 'Winnie Sun', 'married': False }
```

- 使用 **popitem()** 移除最後加入的鍵值對

```
print(person)  # { 'name': 'Winnie Sun', 'age' : 18, 'married': False }
person.popitem()
print(person)  # { 'name': 'Winnie Sun', 'age' : 20  }
```

- 使用 **clear()** 清空字典

```
print(person)  # { 'name': 'Winnie Sun', 'age' : 18, 'married': False }
person.clear()
print(person)  # { }
```

字典(Dictionary) – 遍歷字典

```
person = {
    'name' : 'Winnie Sun',
    'age' : 18,
    'married' : False,
}
```

- 單純遍歷字典，會拿到 鍵 key

```
for x in person :
    print(x)            # 鍵 key
    print(person[x])    # 值 value
```

- 使用 keys() 遍歷字典的鍵 key

```
for x in person.keys() :
    print(x)            # 鍵 key
```

- 使用 values() 遍歷字典的值 value

```
for x in person.values() :
    print(x)            # 鍵 key
```

- 使用 **items()** 將鍵和值分別放置不同變數中

```
for x, y in person.items() :
    print(x)            # 鍵 key
    print(y)            # 值 value
```

字典(Dictionary) – 複製字典

```
person = {
    'name' : 'Winnie Sun',
    'age' : 18,
    'married' : False,
}
```

- 和 List 一樣，字典不能直接使用賦值來複製一個新的出來，可以使用以下方法:
  - 方法 **copy()**

```
new_person = person.copy()
```

  - 內建函式 **dict()**

```
new_person = dict(person)
```

字典(Dictionary) – 巢狀字典 (Nested Dictionaries)

- 巢狀字典就是字典裡面包著字典的意思

一起創建

```
my_class = {
    'student_1' : {
        'name' : 'Winnie Sun',
        'age' : 27
    },
    'student_2' : {
        'name' : 'Maru Sun',
        'age' : 30
    },
    'student_3' : {
        'name' : 'Alen Wu',
        'age' : 22
    },
}
```

分開創建

```
stu1 = {
    'name' : 'Winnie Sun',
    'age' : 27
}
stu2 = {
    'name' : 'Maru Sun',
    'age' : 30
}
stu3 = {
    'name' : 'Alen Wu',
    'age' : 22
}
my_class = {
    'student_1' : stu1,
    'student_2' : stu2,
    'student_3' : stu3
}
```