第4章 常用物件之2 （Objects 2/2）

我們在前面有提到五種常用的資料物件，依序如下：

1. 字串 (String)
2. 數字 (Number)
3. 布林值（Boolean）
4. 列表 (List), 元組 (Tuple), 集合 (Set)
5. 字典 (Dictionary)

前面三類 (1-3) 屬於初级的基本資料物件類型， 我們學習過了。

這裡我們將學習後兩類 (4-5) 的物件類型。

一旦常用的五種資料物件能熟練，我們對Python 程式的讀寫目標應該更近 一步了。

4.1 列表 (List)

列表 (List) 是一個有排列順序的元素列。

語法

[<元素列>]

列表有以下特性：

1. 列表用方括號 "[", "]" 一前一後包裝元素列
2. <元素列>中每個元素 (item) 之間以逗號 "," 分隔
3. 列表中每個元素不需要是相同資料類型，可重複
4. 可以直接訪問列表中元素，依元素在列表中的順序位置（或稱索引） 第一個索引是0， 第二個索引是1， 依此類推。
5. 最後一個元素索引也可用-1來表示。
6. 空白列表以 [ ] 或 list( ) 表示。

訪問列表，例子如下：

In [24]:
```
courses = ['歷史', '物理', '數學', '電腦']
print(f'courses 物件類型: {type(courses)}\n')

print(f'課程列表: {courses}')

# 用索引訪問列表物件
print(f'--->列表第一個元素 (索引是0):\t {courses[0]}')
print(f'--->列表最後一個元素 (索引-1):\t {courses[-1]}')

```

```
courses 物件類型: <class 'list'>

課程列表: ['歷史', '物理', '數學', '電腦']
--->列表第一個元素 (索引是0):	 歷史
--->列表最後一個元素 (索引-1):	 電腦
```

截取列表物件

列表物件和字串物件的截取語法相似。

語法

1) 列表 [開始索引：截止索引：間隔]
2) 字串 [開始索引：截止索引：間隔]

其中：

1. 含開始索引
2. 不含截止索引
3. 若省略開始索引，預設為第一個元素
4. 若省略截止索引，預設最後一個元素
5. 若省略間隔，預設為1

截取列表，例子如下：

In [6]:
```
# 截取列表物件例子1

courses = ['歷史', '物理', '數學', '電腦']
print(f'課程列表: {courses}')

print(f'---> 列表範圍截取前面二個元素:\t {courses[:2]}')
print(f'---> 列表範圍截取後面二個元素:\t {courses[-2:]}')

```

```
課程列表: ['歷史', '物理', '數學', '電腦']
---> 列表範圍截取前面二個元素:        ['歷史', '物理']
---> 列表範圍截取後面二個元素:        ['數學', '電腦']
```

In [5]:
```
# 截取列表物件例子2

my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
# 元素索引   0, 1, 2, 3, 4, 5, 6, 7, 8, 9
# 元素索引 -10,-9,-8,-7,-6,-5,-4,-3,-2,-1
print(f'my_list: {my_list}\n')

print(f'---> 列表範圍截取 0-5: {my_list[0:6]}')
print(f'---> 列表範圍截取 5-9: {my_list[5:]}')

```

```
my_list: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

---> 列表範圍截取 0-5: [0, 1, 2, 3, 4, 5]
---> 列表範圍截取 5-9: [5, 6, 7, 8, 9]
```

In [3]:
```
# 列表截取例子3
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
# 元素索引   0, 1, 2, 3, 4, 5, 6, 7, 8, 9
# 元素索引 -10,-9,-8,-7,-6,-5,-4,-3,-2,-1
print(f'my_list: {my_list}\n')

# 列表[開始索引:截止索引:間隔]
print(f'---> 列表截取偶數(由前到後): {my_list[2:-1:2]}\n')
print(f'---> 列表截取偶數(由後到前): {my_list[-2:1:-2]}\n')
print(f'---> 列表截取(由後到前): {my_list[::-1]}\n')

```

```
my_list: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

---> 列表截取偶數(由前到後): [2, 4, 6, 8]

---> 列表截取偶數(由後到前): [8, 6, 4, 2]

---> 列表截取(由後到前): [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

In [2]:
```
# 列表截取例子4
sample_url = 'https://github.com/mkaoy2k'
print(f'我的網址: {sample_url}\n')

# 網址反列
print(f'---> 網址反列: {sample_url[::-1]}\n')

# 截取網域全稱，沒有 http://
print(f'---> 截取網域全稱: {sample_url[8:]}\n')

# 截取第一層網域名稱
print(f'---> 截取第一層網域名稱: {sample_url[8:-8]}\n')

# 截取第二層網域名稱
print(f'---> 截取第二層網域名稱: {sample_url[-7:]}\n')

```

```
我的網址: https://github.com/mkaoy2k (https://github.com/mkaoy2k)

---> 網址反列: k2yoakm/moc.buhtig//:sptth

---> 截取網域全稱: github.com/mkaoy2k

---> 截取第一層網域名稱: github.com

---> 截取第二層網域名稱: mkaoy2k
```

刪除列表中的元素

Python 設計「刪除」指令 (del) 來讓我們刪除列表中的元素。

語法

```
del < 列表[索引] >
```

其中:

1. 索引是列表的元素相對位置
2. 從前面的相對位置，0開始
3. 從後面的相對位置，-1開始

如下實例：

In [7]:
```
# 列表元素可重複
courses = ['歷史', '物理', '電腦', '物理']
print(f'課程列表: {courses}\n')

# 刪除第一個課程
del courses[0]
print(f'---> 課程列表刪除第一個課程:\t {courses}')

# 刪除最後一個課程
del courses[-1]
print(f'---> 課程列表刪除最後一個課程:\t {courses}')

```

```
課程列表: ['歷史', '物理', '電腦', '物理']

---> 課程列表刪除第一個課程:          ['物理', '電腦', '物理']
---> 課程列表刪除最後一個課程:        ['物理', '電腦']
```

列表運算

對於列表物件 Python 設計有兩個 "+" 和 "*" 的運算子與字串物件操作相似。

1. "+" 號用於組合多個列表物件成一個列表物件
2. "*" 號用於重複列表中元素

列表運算例子如下所示：

In [8]:
```
list1 = [1, 2, 3]
list2 = [4, 5, 6]

print(f'數字列表 list1 = {list1}')
print(f'數字列表 list2 = {list2}\n')

# 組合多個列表物件成一個列表
list3 = list1 + list2
print(f'list1 + list2 = {list3}')

# 重複列表中元素
list_hello = ['哈囉'] * 4
print(f'列表 [\'哈囉\'] * 4 = {list_hello}')
```

```
數字列表 list1 = [1, 2, 3]
數字列表 list2 = [4, 5, 6]

list1 + list2 = [1, 2, 3, 4, 5, 6]
列表 ['哈囉'] * 4 = ['哈囉', '哈囉', '哈囉', '哈囉']
```

列表比較運算

列表物件的比較運算通常可以是兩種情況：

1. 最常用且簡單比較運算就是判斷元素是否在列表中
2. 比較兩個列表的內容是否相同，有兩種可能：
   A. 順序可能不同，但內容相同 例如: [1, 2, 3] 與 [3, 2, 1]
   B. 順序相同，內容也相同

列表物件的比較運算，例子如下：

In [16]:
```python
# 列表物件的比較運算，例子1
list1 = [4, 2, 3]
print(f'數字列表 list1 = {list1}\n')

# 元素是否存在於列表中
answer = 3 in list1
print(f'問：3 在 list1 數字列表 list1 中嗎?')
print(f'答：{answer}\n')

# 元素是否存在於列表中(也可以這樣寫)
if 3 in list1:
    print(f'3 於數字列表 {list1}中')

```

```
數字列表 list1 = [4, 2, 3]

問：3 在 list1 數字列表 list1 中嗎?
答：True

3 於數字列表 [4, 2, 3]中
```

In [14]:
```python
# 列表物件的比較運算，例子2
list1 = [1, 2, 3]
list2 = [3, 2, 1]
print(f'數字列表 list1 = {list1}')
print(f'數字列表 list2 = {list2}\n')

# sorted() 排序函式後面介紹
s_list1 = sorted(list1)
s_list2 = sorted(list2)

if list1 == list2:
    print(f'{list1} 與 {list2} 順序相同，內容也相同\n')
elif s_list1 == s_list2:
    print(f'{list1} 與 {list2} 順序不同，但內容相同\n')
else:
    print(f'{list1} 不同於 {list2}\n')

```

```
數字列表 list1 = [1, 2, 3]
數字列表 list2 = [3, 2, 1]

[1, 2, 3] 與 [3, 2, 1] 順序不同，但內容相同
```

問：列表中對每一個元素用迴圈指令處理，應該常用吧？

答：是的。

Python 讓我們用 for 迴圈指令，將列表中每一元素，逐個比較並且執行相應的處理。

來看看列表迴圈的例子，如下：

In [21]:
```
# 列表迴圈的例子1
courses = ['歷史', '物理', '電腦', '物理']
print(f'課程列表: {courses}\n')

# 列表迴圈
print(f'列表迴圈1: 從第一個元素開始印元素...')
for course in courses:
    print(f'---> {course}')
print(f'列表迴圈1 終止!\n')

```

```
課程列表: ['歷史', '物理', '電腦', '物理']

列表迴圈1: 從第一個元素開始印元素...
---> 歷史
---> 物理
---> 電腦
---> 物理
列表迴圈1 終止!
```

In [22]:
```
# 列表迴圈的例子2
courses = ['歷史', '物理', '電腦', '物理']
print(f'課程列表: {courses}\n')

# 列表迴圈: 印索引及元素
print(f'列表迴圈2: 從第一個元素(從索引0)開始印索引及元素...')
for index, course in enumerate(courses): # 預設索引值 0
    print(f'---> {index} {course}')
print(f'列表迴圈2 終止!\n')

```

```
課程列表: ['歷史', '物理', '電腦', '物理']

列表迴圈2: 從第一個元素(索引0)開始印索引及元素...
---> 0 歷史
---> 1 物理
---> 2 電腦
---> 3 物理
列表迴圈2 終止!
```

In [17]:
```
# 列表迴圈的例子3
courses = ['歷史', '物理', '電腦', '物理']
print(f'課程列表: {courses}\n')

print(f'列表迴圈3: 從第一個元素(從索引2)開始...')
for index, course in enumerate(courses, start=2):
    print(f'---> {index} {course}')
print(f'列表迴圈3 終止!\n')

```

```
課程列表: ['歷史', '物理', '電腦', '物理']

列表迴路3: 從第一個元素(索引2)開始...
---> 2 歷史
---> 3 物理
---> 4 電腦
---> 5 物理
列表迴路3 終止!
```

列表相關函式

編號	列表相關常用的函式
1	len (list) 返回列表元素個數（長度）
2	max (list) 返回列表中最大元素
3	min (list) 返回列表中最小元素
4	sum (list) 返回列表總計
5	sorted (list) 新增一個排序的列表

列表物件相關函式，例子如下：

In [23]:
```
numbers = [9, 8, 1, 2, 4, 7]
print(f'數字列表: {numbers}\n')

print(f'1. 列表長度: {len(numbers)}')

print(f'2. 列表中最大元素: {max(numbers)}')

print(f'3. 列表中最小元素: {min(numbers)}')

print(f'4. 列表總計: {sum(numbers)}')

numbers_sorted = sorted(numbers)
print(f'5. 新排序過的列表: {numbers_sorted}')

```

```
數字列表: [9, 8, 1, 2, 4, 7]

1. 列表長度: 6
2. 列表中最大元素: 9
3. 列表中最小元素: 1
4. 列表總計: 31
5. 新排序過的列表: [1, 2, 4, 7, 8, 9]
```

列表相關方法

編號	列表相關常用的方法
1	list.append (obj) 在列表末尾添加新的元素
2	list.count (obj) 返回元素在列表中出現的次數
3	list.extend (seq) 在列表末尾擴充多個元素
4	list.index (obj) 列表中找出第一個相同元素的索引位置
5	list.insert (index, obj) 元素插入列表
6	list.pop () 移除列表中的最後一個元素，並且返回該元素的值
7	list.remove (obj) 移除列表中第一個匹配元素
8	list.reverse () 反向列表
9	list.sort (key=None, reverse=False) 對原列表排序
10	string.join (list) 轉換列表成字串，列表元素之間用 "string" 隔開
11	string.split (<delimiter>) 轉換字串成列表，字串用 <delimiter> 隔開成列表元素

列表物件相關方法，例子如下：

In [27]:
```
# 列表物件相關方法，例子1-4
numbers = [9, 8, 1, 2, 4, 7]
print(f'數字列表: {numbers}\n')

item_new = 1
numbers.append(item_new)
print(f'1. 末尾加上 {item_new} 後列表: {numbers}')

print(f'2. 列表中 {item_new} 出現次數: {numbers.count(item_new)}')

list_new = [10, 20]
numbers.extend(list_new)
print(f'3. 擴充 {list_new} 後列表: {numbers}')

print(f'4. 列表中 {item_new} 第一次出現的索引: {numbers.index(item_new
```

```
數字列表: [9, 8, 1, 2, 4, 7]

1. 末尾加上 1 後列表: [9, 8, 1, 2, 4, 7, 1]
2. 列表中 1 出現次數: 2
3. 擴充 [10, 20] 後列表: [9, 8, 1, 2, 4, 7, 1, 10, 20]
4. 列表中 1 第一次出現的索引: 2
```

In [28]:
```
# 列表物件相關方法，例子5-8
numbers = [9, 8, 1, 2, 4, 7]
print(f'數字列表: {numbers}\n')

item_new = 1

numbers.insert(0, item_new)
print(f'5. 插入 {item_new} 後列表: {numbers}')

print(f'6. 列表中彈出: {numbers.pop()} 後列表: {numbers}')

numbers.remove(item_new)
print(f'7. 移除第一個 {item_new} 後列表: {numbers}')

numbers.reverse()
print(f'8. 反向列表: {numbers}')
```

```
數字列表: [9, 8, 1, 2, 4, 7]

5. 插入 1 後列表: [1, 9, 8, 1, 2, 4, 7]
6. 列表中彈出: 7 後列表: [1, 9, 8, 1, 2, 4]
7. 移除第一個 1 後列表: [9, 8, 1, 2, 4]
8. 反向列表: [4, 2, 1, 8, 9]
```

In [35]:
```
# 列表物件相關方法，例子9-11
numbers = [9, 8, 1, 2, 4, 7]
print(f'數字列表: {numbers}')
courses = ['歷史', '物理', '電腦', '物理']
print(f'課程列表: {courses}\n')

# 列表排序
numbers.sort()
print(f' 9. 數字列表重新排序: {numbers}')

# 列表轉字串
course_str = '-'.join(courses)
print(f'10. 列表轉字串: {course_str}')

# 字串轉列表
new_list = course_str.split('-')
print(f'11. 字串轉列表: {new_list}')

```

```
數字列表: [9, 8, 1, 2, 4, 7]
課程列表: ['歷史', '物理', '電腦', '物理']

 9. 數字列表重新排序: [1, 2, 4, 7, 8, 9]
10. 列表轉字串: 歷史-物理-電腦-物理
11. 字串轉列表: ['歷史', '物理', '電腦', '物理']
```

4.2 元組 (Tuple)

我們在前面有提到五種常用的資料物件，依序如下：

1. 字串 (String)
2. 數字 (Number)
3. 布林值（Boolean）
4. 列表 (List), 元組 (Tuple), 集合 (Set)
5. 字典 (Dictionary)

前面三類 (1-3) 及列表物件資料類型， 我們學習過了。這裡我們將學習第4種資料類型中的元組物件。

元組 (tuple) 是一個沒有排列順序的元素列。

語法

(<元素列>)

元組有以下特性：

1. 元組用小括號 "(", ")" 一前一後包裝元素列
2. <元素列>中每個元素 (item) 之間以逗號 "," 分隔
3. 元組中每個元素不需要具有相同類型，可重複
4. 可以直接訪問元組中每個元素，依元素在中的位置（或稱索引） 第一個索引是0， 第二個索引是1， 依此類推。 最後一個元素索引也可用 "-1" 來表示。
5. 元組中只包含一個元素時，需要在元素後面添加逗號，與敍述(expression) 有所區別。
6. 元組中元素不可修改或增減，但是可以進行截取，組合。
7. 空白元組以 () 或 tuple() 來表示。

訪問元组

元組可以使用索引來訪問元組中的元素，注意索引也跟列表的索引一樣都要用方括號。

如下實例：

In [2]:
```
tuple1 = (1, 2, 3, 4, 5, 4)
tuple2 = (1, 2, 5)

print(f'{tuple1} 物件類型: {type(tuple1)}')
print(f'tuple1 = {tuple1}')
print(f'tuple2 = {tuple2}\n')

print(f'{tuple1} 元組第一個元素 = {tuple1[0]}')
print(f'{tuple1} 元組最後一個元素 = {tuple1[-1]}')

```

```
(1, 2, 3, 4, 5, 4) 物件類型: <class 'tuple'>
tuple1 = (1, 2, 3, 4, 5, 4)
tuple2 = (1, 2, 5)

(1, 2, 3, 4, 5, 4) 元組第一個元素 = 1
(1, 2, 3, 4, 5, 4) 元組最後一個元素 = 4
```

截取元組物件

元組物件的截取與上面我們學習過的列表和字串相似。

語法

元組 [開始索引:截止索引:間隔]

其中：

1. 含開始索引
2. 不含截止索引
3. 若省略開始索引，預設為第一個元素
4. 若省略截止索引，預設最後一個元素
5. 若省略間隔，預設為1

截取元組，例子如下：

In [3]:
```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = (1, 2, 5, 7)

print(f'{tuple1} 元組奇數索引的元素 = {tuple1[1::2]}')
print(f'{tuple2} 元組第一個到第三個元素 = {tuple2[:3]}')

```

```
(1, 2, 3, 4, 5) 元組奇數索引的元素 = (2, 4)
(1, 2, 5, 7) 元組第一個到第三個元素 = (1, 2, 5)
```

元組運算

對於元組物件 Python 設計有兩個 "+" 和 "*" 的運算子與列表物件操作相似。

1. + 號運算子用於組合多個元組物件成一個元組物件
2. * 號運算子用於重複元組中元素

元組運算例子，如下所示：

In [4]:
```
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)

print(f'元組1 = {tuple1}')
print(f'元組2 = {tuple2}\n')

# 組合多個元組物件成一個元組
tuple3 = tuple1 + tuple2
print(f'{tuple1} + {tuple2} = {tuple3}')

# 重複元組中元素
# 注意: 元組中只包含一個元素時，需要在元素後面添加逗號
tuple_hello = ('哈囉',) * 4
print(f'元組 (\'哈囉\',) * 4 = {tuple_hello}')
```

```
元組1 = (1, 2, 3)
元組2 = (4, 5, 6)

(1, 2, 3) + (4, 5, 6) = (1, 2, 3, 4, 5, 6)
元組 ('哈囉',) * 4 = ('哈囉', '哈囉', '哈囉', '哈囉')
```

元組比較運算

元組物件的比較運算通常可以是兩種情況：

1. 相同: 最常用且簡單比較運算就是用 "**==**" 比較兩個元組的元素內容及排序都是相同：
   A. 兩個元組中第一個元素先比較大小
   B. 若相等，繼續比較兩個元組中第二個元素
   C. 依此類推
   D. 比較到最後都相等 ，也就是
      a. 兩個元組長度相等
      b. 兩個元組中每個元素的值相等

   上述兩者條件都相等的話，則兩個元組內容相同
2. 相似: 判斷每個元素是否在兩個元組中，不管元素次序

元組物件的比較運算，例子如下：

In [10]:
```
# 1. 相同: 用 "==" 比較兩個元組的元素內容及排序都是相同
tuple1 = (1, 2, 3)
tuple2 = (1, 2, 4)

print(f'元組1 = {tuple1}')
print(f'元組2 = {tuple2}\n')

if tuple1 == tuple2:
    print(f'===>{tuple1} 等於 {tuple2}')
elif tuple1 > tuple2:
    print(f'===>{tuple1} 大於 {tuple2}')
else:
    print(f'===>{tuple1} 小於 {tuple2}')

```

```
元組1 = (1, 2, 3)
元組2 = (1, 2, 4)

===>(1, 2, 3) 小於 (1, 2, 4)
```

In [8]:

```
# 2. 相似: 判斷每個元素是否在兩個元組中，不管元素次序
tuple1 = (1, 2, 3)
tuple2 = (3, 2, 1)

print(f'元組1 = {tuple1}')
print(f'元組2 = {tuple2}\n')

if len(tuple1) == len(tuple2):
    matched = True
    for i in tuple1:
        if i not in tuple2:
            matched = False
            break
else:
    matched = False

if matched:
    print(f'===>{tuple1} 與 {tuple2} 相似')
else:
    print(f'===>{tuple1} 與 {tuple2} 不相似')

```

```
元組1 = (1, 2, 3)
元組2 = (3, 2, 1)

===>(1, 2, 3) 與 (3, 2, 1) 相似
```

元组相關函式

編號	元组相關常用的函式
1	len (list) 返回元组元素個數（長度）
2	max (list) 返回元组中最大元素
3	min (list) 返回元组中最小元素
4	sum (list) 返回元组總計
5	tuple (seq) 列表轉換成元組 list (seq) 元組也可轉換成列表

元組物件的相關函式，例子如下：

In [12]:
```
numbers = (9, 8, 1, 2, 4, 7)
print(f'數字元组: {numbers}\n')

print(f'1. 元组長度: {len(numbers)}')
print(f'2. 元组中最大元素: {max(numbers)}')
print(f'3. 元组中最小元素: {min(numbers)}')
print(f'4. 元组總計: {sum(numbers)}\n')

courses_list = ['歷史', '物理', '電腦', '物理']
courses_tuple = tuple(courses_list)
print(f'課程列表: {courses_list}\n')

print(f'5. 列表轉元组: {courses_tuple}')
print(f'   元组轉列表: {list(courses_tuple)}')

```

```
數字元组: (9, 8, 1, 2, 4, 7)

1. 元组長度: 6
2. 元组中最大元素: 9
3. 元组中最小元素: 1
4. 元组總計: 31

課程列表: ['歷史', '物理', '電腦', '物理']

5. 列表轉元组: ('歷史', '物理', '電腦', '物理')
   元组轉列表: ['歷史', '物理', '電腦', '物理']
```

元组相關方法

編號	元组相關常用的方法
1	tuple.count (obj) 返回元素在元组中出現的次數
2	tuple.index (obj) 元组中找出第一個相同元素的索引位置
3	string.join (tuple) 轉換元组成字串，元组元素之間用 "string" 隔開
4	tuple(string.split (<delimiter>)) 轉換字串成元组，字串用 <delimiter> 隔開成元组元素

元組物件的相關方法，例子如下：

In [14]:
```
numbers = (1, 9, 8, 1, 2, 4, 7, 1)
print(f'數字列表: {numbers}\n')
item_new = 1

print(f'1. 列表中 1 出現次數: {numbers.count(item_new)}')
print(f'2. 列表中 1 第一次出現的索引: {numbers.index(item_new)}\n')

courses = ('歷史', '物理', '電腦', '物理')
print(f'課程元组: {courses}\n')

# 元组轉字串
course_str = '-'.join(courses)
print(f'3. 元組轉字串: {course_str}')

# 字串轉列表，再轉元组
new_list = course_str.split('-')
new_tuple = tuple(new_list)
print(f'4. 元组轉元组: {new_tuple}')

```

```
數字列表: (1, 9, 8, 1, 2, 4, 7, 1)

1. 列表中 1 出現次數: 3
2. 列表中 1 第一次出現的索引: 0

課程元组: ('歷史', '物理', '電腦', '物理')

3. 元组轉字串: 歷史-物理-電腦-物理
4. 元组轉元组: ('歷史', '物理', '電腦', '物理')
```

4.3 集合 (Set)

我們在前面有提到五種常用的資料物件，依序如下：

1. 字串 (String)
2. 數字 (Number)
3. 布林值（Boolean）
4. 列表 (List), 元組 (Tuple), 集合 (Set)
5. 字典 (Dictionary)

前面三類 (1-3) 、列表及元組物件資料類型， 我們學習過了。這裡我們將學習第4類資料類型中的集合物件。

集合（set）是一個沒有排列順序且不可重複的元素列。

語法

```
{ <元素列> }
```

集合有以下特性：

1. 集合物件用小括號 "{", "}" 一前一後包裝元素列
2. <元素列>中每個元素 (item) 之間以逗號 "," 分隔
3. 集合中每個元素不需要具有相同類型，重複元素會被忽略
4. 集合中元素列沒有排列順序
5. 集合中元素不能依索引直接訪問或截取
6. 集合中元素可增減或修改
7. 集合之間可以運算

注意

空集合的語法較特殊必須用 set() 而不是 { }，常會搞混。 因为 { } 是表示一個空字典（將介紹於後）。

集合物件例子如下：

In [19]:
```
# 集合物件有去重複元素的功能
basket = {'蘋果', '橘子', '蘋果', '梨子', '橘子', '香蕉'}
print("{'蘋果', '橘子', '蘋果', '梨子', '橘子', '香蕉'}\n")

print(f'集合物件 basket = {basket}')

print(f'basket 物件類型: {type(basket)}')

```

```
{'蘋果', '橘子', '蘋果', '梨子', '橘子', '香蕉'}

集合物件 basket = {'香蕉', '橘子', '蘋果', '梨子'}
basket 物件類型: <class 'set'>
```

集合運算

兩個集合間的運算	結果
a - b	集合a中包含而集合b中不包含的元素
a | b	集合a或b中包含的所有元素
a & b	集合a和b中都包含了的元素
a ^ b	不同時包含於集合a和b的元素

集合運算例子，如下所示：

In [27]:
```

```

```python
# 集合 a
cs = {'歷史', '數學', '物理', '電腦'}

# 集合 b
art = {'歷史', '數學', '藝術', '設計'}

print(f'cs: {cs} 物件類型: {type(cs)}')
print(f'art: {art} 物件類型: {type(art)}\n')

# a-b: 集合a中包含而集合b中不包含的元素
print(f'1. 集合 cs 中包含而集合 art 中不包含的元素')
print(f'cs - art ===> {cs - art}\n')

# a|b: 集合a或b中包含的所有元素
print(f'2. 集合 cs 或 art 中包含的所有元素')
print(f'cs | art ===> {cs | art}\n')

# a&b: 集合a和b中都有的元素
print(f'3. 集合 cs 和 art 中都有的元素')
print(f'cs & art ===> {cs & art}\n')

# a^b: 不同時包含於集合a和b的元素
print(f'4. 不同時包含於集合 cs 和 art 的元素')
print(f'cs ^ art ===> {cs ^ art}')

```

```
cs: {'物理', '歷史', '電腦', '數學'} 物件類型: <class 'set'>
art: {'藝術', '歷史', '數學', '設計'} 物件類型: <class 'set'>

1. 集合 cs 中包含而集合 art 中不包含的元素
cs - art ===> {'物理', '電腦'}

2. 集合 cs 或 art 中包含的所有元素
cs | art ===> {'藝術', '物理', '電腦', '設計', '歷史', '數學'}

3. 集合 cs 和 art 中都有的元素
cs & art ===> {'歷史', '數學'}

4. 不同時包含於集合 cs 和 art 的元素
cs ^ art ===> {'藝術', '物理', '電腦', '設計'}
```

集合比較運算

集合物件的比較運算是 Python 物件比較中最有效率的運算。

集合物件 a 和 b 的比較通常可以是兩種情況：

1. 最常用且簡單比較運算就是用 "==" 運算子判斷元素是否在集合中
2. 比較兩個集合的內容是否相同，用數學理論的判斷法有二：
   A. 兩個集合a, b，若是 a & b == a | b，則兩集合相同
   B. 兩個集合互為子集合則兩集合相同。下面將會介紹集合方法 issubset()時，再詳說

集合比較運算，例子如下：

In [32]:
```
cs_courses = {'歷史', '數學', '物理', '電腦'}
art_courses = {'歷史', '數學', '藝術', '設計'}
new =  {'數學', '歷史', '物理', '電腦'}

print(f'cs_courses: {cs_courses}')
print(f'art_courses: {art_courses}')
print(f'new: {new}\n')

# 1. 兩集合的比較
if cs_courses == art_courses:
    print(f'cs_courses 與 art_courses 內容相同\n')
else:
    print(f'cs_courses 與 art_courses 內容不同\n')

if cs_courses == new:
    print(f'cs_courses 與 new 內容相同\n')
else:
    print(f'cs_courses 與 new 內容不同\n')

```

```
cs_courses: {'物理', '歷史', '電腦', '數學'}
art_courses: {'藝術', '歷史', '數學', '設計'}
new: {'物理', '歷史', '電腦', '數學'}

cs_courses 與 art_courses 內容不同

cs_courses 與 new 內容相同
```

In [33]:
```
cs_courses = {'歷史', '數學', '物理', '電腦'}
art_courses = {'歷史', '數學', '藝術', '設計'}
new =  {'數學', '歷史', '物理', '電腦'}

print(f'cs_courses: {cs_courses}')
print(f'art_courses: {art_courses}')
print(f'new: {new}\n')

# 兩集合的交集
cs_and_art = cs_courses & art_courses
cs_and_new = cs_courses & new

# 兩集合的聯集
cs_or_art = cs_courses | art_courses
cs_or_new = cs_courses | new

# 2. 交集 = 聯集?
if cs_and_art == cs_or_art:
    print(f'cs_courses 與 art_courses 內容相同\n')
else:
    print(f'cs_courses 與 art_courses 內容不同\n')

if cs_and_new == cs_or_new:
    print(f'cs_courses 與 new 內容相同')
else:
    print(f'cs_courses 與 new 內容不同')

```

```
cs_courses: {'物理', '歷史', '電腦', '數學'}
art_courses: {'藝術', '歷史', '數學', '設計'}
new: {'物理', '歷史', '電腦', '數學'}

cs_courses 與 art_courses 內容不同

cs_courses 與 new 內容相同
```

問：如何檢視元素是否在集合中?

答： Python 設計 "**in**" 指令讓我們檢視元素是否在集合中。 同理列表、元組資料物件亦適用。

語法

```
x in s
```

其中：

1. x 是元素或變數，s 是一個集合物件 (或列表、元組亦同)
2. 判斷元素 x 是否在集合 s中，
   A. 若是存在返回 True
   B. 若是不存在返回 False

In [34]:

```
cs_courses = {'歷史', '數學', '物理', '電腦'}

# 判斷'數學'是否在集合中
target = '數學'

print(f'問：\'{target}\' 在課程 {cs_courses} 集合中嗎？')
print(f'答： {target in cs_courses}\n')

target = '藝術'
print(f'問：\'{target}\' 在課程 {cs_courses} 集合中嗎？')
print(f'答： {target in cs_courses}')

```

```
問：'數學' 在課程 {'物理', '歷史', '電腦', '數學'} 集合中嗎？
答： True

問：'藝術' 在課程 {'物理', '歷史', '電腦', '數學'} 集合中嗎？
答： False
```

集合相關函式

編號	集合相關函式
1	len (set) 返回集合元素個數（長度）
2	max (set) 返回集合中最大元素
3	min (set) 返回集合中最小元素
4	sum (set) 返回集合總計
5	set (seq) 轉換成集合

集合物件的相關函式，例子如下：

In [35]:
```
numbers = {9, 8, 1, 2, 4, 7}
print(f'數字集合: {numbers}\n')

print(f'1. 集合長度: {len(numbers)}')
print(f'2. 集合中最大元素: {max(numbers)}')
print(f'3. 集合中最小元素: {min(numbers)}')
print(f'4. 集合總計: {sum(numbers)}')

courses_list = ['歷史', '物理', '電腦', '物理']
courses_tuple = ('歷史', '物理', '電腦', '物理')

print(f'===>課程列表: {courses_list}')
print(f'===>課程元組: {courses_tuple}')

courses_set = set(courses_list)
print(f'5. 列表轉集合: {courses_set}')
print(f'   集合轉列表: {list(courses_set)}')

courses_set = set(courses_tuple)
print(f'   元組轉集合: {courses_set}')
print(f'   集合轉元組: {tuple(courses_set)}')

```

```
數字集合: {1, 2, 4, 7, 8, 9}

1. 集合長度: 6
2. 集合中最大元素: 9
3. 集合中最小元素: 1
4. 集合總計: 31
===>課程列表: ['歷史', '物理', '電腦', '物理']
===>課程元組: ('歷史', '物理', '電腦', '物理')
5. 列表轉集合: {'物理', '歷史', '電腦'}
   集合轉列表: ['物理', '歷史', '電腦']
   元組轉集合: {'物理', '歷史', '電腦'}
   集合轉元組: ('物理', '歷史', '電腦')
```

集合相關方法

編號	集合相關方法
1	s.add (x) 將元素 x 加到集合 s 中，如果元素已存在，則忽略操作。
2	s.update (Seq) 添加參數列所有元素，且參數列 Seq 可以是多個列表，元組等
3	a.difference(b) 返回一個集合其元素只在集合 a，但不在集合 b中
4	a.intersection (b) 返回一個集合其元素在集合 a，同時也在集合 b中
5	a.union (b) 返回一個集合其元素在集合 a，或在集合 b中
6	a.issubset (b) 判斷集合 a 是否是集合 b 的子集合
7	s.discard (x) 移除集合 s 中的元素 x，且如果元素不存在，Python 不會產生錯誤訊息
8	s.remove (x) 將元素 x 從集合 s 中移除，若元素不存在，Python 則會產生錯誤訊息
9	s.pop () 隨機刪除集合 s 中的一個元素
10	s.clear () 清空集合 s 中的所有元素

集合物件的相關方法，例子如下：

In [50]:
```
# 集合 a
cs_courses = {'歷史', '數學', '物理', '電腦'}

# 集合 b
art_courses = {'歷史', '數學', '藝術', '設計'}

print(f'電腦課程集合 cs_courses: {cs_courses}')
print(f'藝術課程集合 art_courses: {art_courses}\n')

cs_courses.add("數學")
print(f'1. 加數學到電腦課程集合:')
print(f'===>{cs_courses}')

art_courses.update(['書法', '繪畫1'], (1, '繪畫2'))
print(f"2. 把['書法', '繪畫1'], (1, '繪畫2')更新到藝術課程集合:")
print(f'===>{art_courses}')

```

```
電腦課程集合 cs_courses: {'物理', '歷史', '電腦', '數學'}
藝術課程集合 art_courses: {'藝術', '歷史', '數學', '設計'}

1. 加數學到電腦課程集合:
===>{'物理', '歷史', '電腦', '數學'}
2. 把['書法', '繪畫1'], (1, '繪畫2')更新到藝術課程集合:
===>{1, '繪畫1', '繪畫2', '藝術', '書法', '設計', '歷史', '數學'}
```

In [52]:
```
# 集合 a
cs_courses = {'歷史', '數學', '物理', '電腦'}

# 集合 b
art_courses = {'歷史', '數學', '藝術', '設計'}

print(f'電腦課程集合 cs_courses: {cs_courses}')
print(f'藝術課程集合 art_courses: {art_courses}\n')

# difference() 同 a-b: 集合a中包含而集合b中不包含的元素
difference = cs_courses.difference(art_courses)
print(f'3. difference(): {difference}')

# intersection() 同 a&b: 集合a和b中都包含了的元素
intersection = cs_courses.intersection(art_courses)
print(f'4. intersection():{intersection}')

# union() 同 a|b: 集合a或b中包含的所有元素
union = cs_courses.union(art_courses)
print(f'5. union():{union}')

```

```
電腦課程集合 cs_courses: {'物理', '歷史', '電腦', '數學'}
藝術課程集合 art_courses: {'藝術', '歷史', '數學', '設計'}

3. difference(): {'物理', '電腦'}
4. intersection():{'歷史', '數學'}
5. union():{'藝術', '物理', '電腦', '設計', '歷史', '數學'}
```

In [53]:
```
basket1 = {'橘子', '蘋果', '梨子', '香蕉'}
basket2 = {'梨子', '橘子'}
print(f'集合 basket1: {basket1}')
print(f'集合 basket2: {basket2}\n')

print('6. a.issubset(b)')
# 判斷集合 a 是否是集合 b 的子集合
print(f'===>問：basket1 是 basket2 的子集嗎？')
print(f'===>答：{basket1.issubset(basket2)}')

print(f'===>問：basket2 是 basket1 的子集嗎？')
print(f'===>答：{basket2.issubset(basket1)}')

print('7. s.discard(x)')
# 移除集合 s 中的元素 x，且如果元素不存在，Python 不會產生錯誤訊息
basket2.discard('蘋果')  # 蘋果不存在，Python 不會產生錯誤訊息
print(f'===>移除 basket2 中的 蘋果: {basket2}')

print('8. s.remove(x)')
# 將元素 x 從集合 s 中移除，如果元素不存在，Python 則會產生錯誤訊息
basket2.remove("橘子")
print(f'===>移除 basket2 中的 橘子: {basket2}')

```

```
集合 basket1: {'香蕉', '橘子', '蘋果', '梨子'}
集合 basket2: {'橘子', '梨子'}

6. a.issubset(b)
===>問：basket1 是 basket2 的子集嗎？
===>答：False
===>問：basket2 是 basket1 的子集嗎？
===>答：True
7. s.discard(x)
===>移除 basket2 中的 蘋果: {'橘子', '梨子'}
8. s.remove(x)
===>移除 basket2 中的 橘子: {'梨子'}
```

In [54]:
```
basket1 = {'橘子', '蘋果', '梨子', '香蕉'}
basket2 = {'梨子', '橘子'}
print(f'集合 basket1: {basket1}')
print(f'集合 basket2: {basket2}\n')

print('8. s.remove(x)')
# 將元素 x 從集合 s 中移除，如果元素不存在，Python 則會產生錯誤訊息
basket2.remove("橘子")
print(f'===>移除 basket2 中的 橘子: {basket2}')

# 再一次呼叫 basket2.remove("橘子")，會產生錯誤訊息
# KeyError: '橘子'
# basket2.remove("橘子")

print('9. s.pop()')
# 隨機出集合 s 中的一個元素
print(f'===>隨機挑出 basket1 中的 {basket1.pop()}: {basket1}')

print('10. s.clear()')
# 清空集合 s 中的所有元素
basket1.clear()
print(f'===>清空 basket1: {basket1}')

```

```
集合 basket1: {'香蕉', '橘子', '蘋果', '梨子'}
集合 basket2: {'橘子', '梨子'}

8. s.remove(x)
===>移除 basket2 中的 橘子: {'梨子'}
9. s.pop()
===>隨機挑出 basket1 中的 香蕉: {'橘子', '蘋果', '梨子'}
10. s.clear()
===>清空 basket1: set()
```

問：比較集合是否相同時，如何用 issubset () 方法來判斷？

答：好的。就是判斷兩個集合是否是互為子集合。

集合方法 issubset() 詳細舉例說明如下：

In [55]:
```
cs_courses = {'歷史', '數學', '物理', '電腦'}

art_courses = {'歷史', '數學', '藝術', '設計'}

print(f'cs_courses: {cs_courses}')
print(f'art_courses: {art_courses}\n')

# 不同例子：判斷兩個集合是否是互為子集合嗎?
if cs_courses.issubset(art_courses) & art_courses.issubset(cs_cour
    print(f'cs_courses 與 art_courses 內容相同')
else:
    print(f'cs_courses 與 art_courses 內容不同')

new_courses = {'歷史', '數學', '藝術', '設計'}

# 相同例子：判斷兩個集合是否是互為子集合嗎?
if new_courses.issubset(art_courses) & art_courses.issubset(new_co
    print(f'new_courses 與 art_courses 內容相同')
else:
    print(f'new_courses 與 art_courses 內容不同')

```

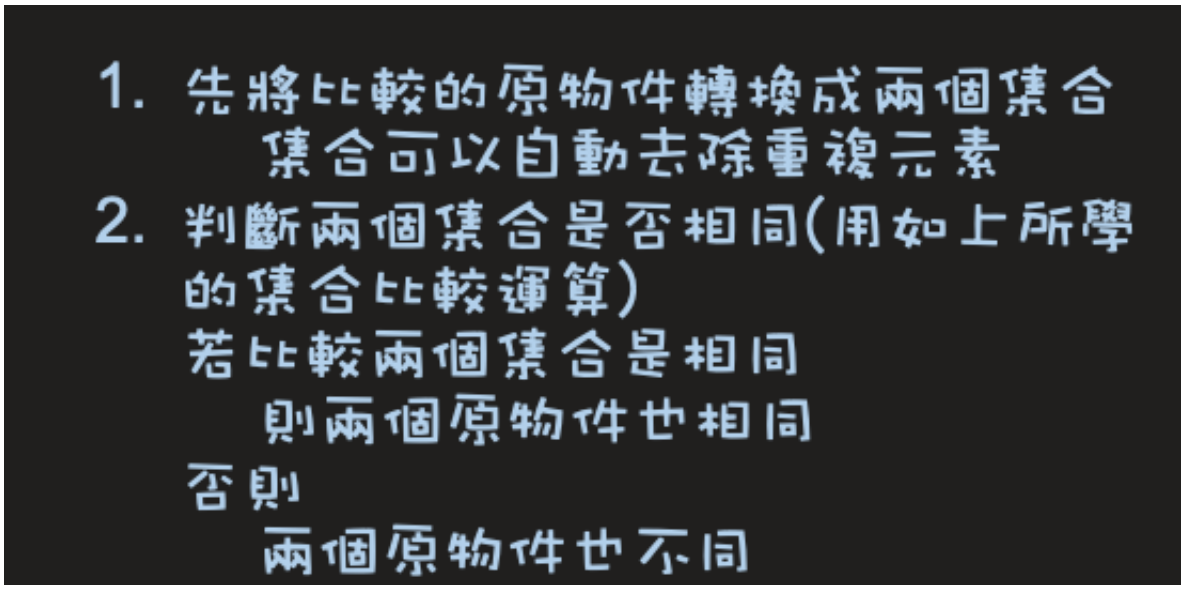
```
cs_courses: {'物理', '歷史', '電腦', '數學'}
art_courses: {'藝術', '歷史', '數學', '設計'}

cs_courses 與 art_courses 內容不同
new_courses 與 art_courses 內容相同
```

問：回顧元素在列表或元組中都可重複，比較時，如何簡單地不計重複元素而視為相同內容？

答：簡單的辦法是利用集合能捨棄重複的功能。

我們的程式邏輯，流程如下：



不計重複而視為相同內容，程式如下：

In [59]:

```
tuple1 = (1, 2, 3)
tuple2 = (1, 2, 3, 1)

print(f'元組1 = {tuple1}')
print(f'元組2 = {tuple2}\n')

if tuple1 == tuple2:
    print(f'{tuple1} 與 {tuple2} 內容相同')
else:
    print(f'{tuple1} 與 {tuple2} 內容不同')

# 1. 先將比較的原物件轉換成兩個集合
set1 = set(tuple1)
set2 = set(tuple2)

# 2. 判斷兩個集合是否相同
print(f'若重複不計的話...')
if set1 == set2:
    print(f'{tuple1} 與 {tuple2} 內容相同\n')
else:
    print(f'{tuple1} 與 {tuple2} 內容不同\n')

```

```
元組1 = (1, 2, 3)
元組2 = (1, 2, 3, 1)

(1, 2, 3) 與 (1, 2, 3, 1) 內容不同
若重複不計的話...
(1, 2, 3) 與 (1, 2, 3, 1) 內容相同
```

4.4 字典 (Dictionary)

我們在前面有提到五種常用的資料物件，依序如下：

1. 字串 (String)
2. 數字 (Number)
3. 布林值（Boolean）
4. 列表 (List), 元組 (Tuple), 集合 (Set)
5. 字典 (Dictionary)

前面四類物件資料類型， 我們學習過了。這裡我們將學習第五類的字典物件。

語法

```
{ <鍵> : <值> }
```

Python 中的字典物件是一個項目 (item) 的集合，每個項目由一個鍵值對組成，中間用 "：" 隔開。

其特性：

1. Python字典被優化以鍵 (key) 來檢索其值
2. 字典是有序，可更改且不允許重複鍵的集合
3. 創建字典時，如果同一個鍵被儲值兩次，後一個值會蓋掉前一個
4. 鍵必須不可變的 (immutable) 物件，所以可以用字串，數字或元組等物件當作鍵，但列表就不行，因為列表可變 (mutable)
5. 字典值可以沒有限制地取任何 python 物件，既可以是標準的物件，也可以是用戶定義的

字典物件例子如下：

In [71]:
```
student = {'name': '尤勇',
           'age': 12,
           'courses': ['數學', '電腦']}

print(f'字典物件 student:')
print(f'{student}\n')

# 訪問字典物件
print(f'===> name:\t{student["name"]}')
print(f'===> age:\t{student["age"]}')
print(f'===> courses:\t{student["courses"]}')

```

```
字典物件 student:
{'name': '尤勇', 'age': 12, 'courses': ['數學', '電腦']}

===> name:      尤勇
===> age:       12
===> courses:   ['數學', '電腦']
```

字典相關函式

編號	字典相關函式
1	len (dict) 返回字典元素個數（長度）
2	type (dict) 返回字典物件類型 <class 'dict'>
3	dict (kv-pairs) 轉換 kv對{<鍵> =<值>, ...} 成字典
4	del (dict[key]) 移除字典中鍵 key 的項目

字典物件的相關函式，例子如下：

In [68]:
```
student = dict(student_id='142857', grade=5,
    info={'name': '尤勇', 'age': 12, 'courses': ['數學', '電腦']})
print(f'新建的字典 student:')
print(f'{student}\n')

print(f'===> student 物件類型: {type(student)}')
print(f'===> 字典中項目個數 = {len(student)}')

```

```
新建的字典 student:
{'student_id': '142857', 'grade': 5, 'info': {'name': '尤勇', 'age':
12, 'courses': ['數學', '電腦']}}

===> student 物件類型: <class 'dict'>
===> 字典中項目個數 = 3
```

字典相關方法

編號	字典相關方法
1	d.clear () 清除字典 d 中所有項目
2	d.copy () 返回一個 d 的複製字典
3	d.fromkeys(keys, v) 返回一個新字典，其中含特定 keys 所含鍵並設其值為 v (可以多個鍵)
4	d.get(k) 返回鍵 k 的值
5	d.items() 返回一個列表，其元素為字典 d 中 (鍵, 值)元組
6	d.keys() 返回一個列表，其元素為字典 d 中

	所有鍵
7	d.pop(k) 返回鍵 k 的值，並從字典 d 中移除此項鍵鍵 k 及值 (與 del 函式相似)
8	d.popitem() 返回最近加入鍵的值，並從字典 d 中移除此項鍵值
9	d.update(key, value) 插入字典 d 中鍵 key 的值 value
10	d.values() 返回一個列表，其元素為字典 d 中所有值

字典物件的相關方法，例子如下：

- keys()
- values()
- items()

In [72]:
```
student = {'name': '尤勇',
           'age': 12,
           'courses': ['數學', '電腦']}
print(f'{student}\n')

print(f'===> 所有鍵列表: {student.keys()}')
print(f'===> 所有值列表: {student.values()}')
print(f'===> 所有鍵值對列表:')
print(f'{student.items()}')

```

```
{'name': '尤勇', 'age': 12, 'courses': ['數學', '電腦']}

===> 所有鍵列表: dict_keys(['name', 'age', 'courses'])
===> 所有值列表: dict_values(['尤勇', 12, ['數學', '電腦']])
===> 所有鍵值對列表:
dict_items([('name', '尤勇'), ('age', 12), ('courses', ['數學', '電腦']
)])
```

字典物件的相關方法，例子如下：

- items()
- popitem()
- pop()

In [75]:
```
student = {'name': '尤勇',
           'age': 12,
           'courses': ['數學', '電腦']}
print(f'{student}\n')

# 列印字典物件中所有的鍵值對 (key-value pair)
print('字典物件中所有的鍵值對，逐項列印如下：')
for key, value in student.items():
    print(f'===> {key}:\t {value}')
print()

print(f'===> 移除字典中最近一項鍵值: {student.popitem()}')

key = "age"
print(f'===> 移除字典中鍵="{key}" 及值="{student.pop(key)}"')
print(f'===> 僅存的字典中鍵值對列表:')
print(f'{student.items()}')

```

```
{'name': '尤勇', 'age': 12, 'courses': ['數學', '電腦']}

字典物件中所有的鍵值對，逐項列印如下：
===> name:       尤勇
===> age:        12
===> courses:    ['數學', '電腦']

===> 移除字典中最近一項鍵值: ('courses', ['數學', '電腦'])
===> 移除字典中鍵="age" 及值="12"
===> 僅存的字典中鍵值對列表:
dict_items([('name', '尤勇')])
```

字典物件的相關方法，例子如下：

- get()

In [77]:
```
student = {'name': '尤勇',
           'age': 12,
           'courses': ['數學', '電腦']}

print(f'字典物件: {student}\n')

# 字典物件中，新增項目
student['phone'] = '0911-697-369'
print(f'字典物件新增項目:\n===> {student}')

# 字典物件中，查詢項目
# 若鍵不存在，返回 'None'
key = 'phon'
value = student.get(key)
if value is None:
    print(f'===> 鍵="{key}" Not Found.')
else:
    print(f'===> 鍵="{key}" 值="{value}"')

key = 'phone'
value = student.get(key)
if value is None:
    print(f'===> 鍵="{key}" Not Found.')
else:
    print(f'===> 鍵="{key}" 值="{value}"')

```

```
字典物件: {'name': '尤勇', 'age': 12, 'courses': ['數學', '電腦']}

字典物件新增項目:
===> {'name': '尤勇', 'age': 12, 'courses': ['數學', '電腦'], 'phone':
'0911-697-369'}
===> 鍵="phon" Not Found.
===> 鍵="phone" 值="0911-697-369"
```

字典物件的相關方法，例子如下：

- update()
- clear()

In [81]:
```
student = {'name': '尤勇',
           'age': 12,
           'courses': ['數學', '電腦']}

print(f'字典物件: {student}\n')

# 字典物件中，以鍵更改值
# 若鍵不存在，新增新項目
key = 'name'
value = '夏琪'
student[key] = value
print(f'以鍵="{key}" 更改值="{value}"')
print(f'更改過一項鍵值的字典物件:')
print(f'{student}')

# 字典物件中，以鍵更改多個值
student.update({'name': '小叮噹', 'age': 6, 'phone': '888-8888'})
print(f'更改過多項鍵值的字典物件:')
print(f'{student}')

student.clear()
print(f'清除字典中所有項目:')
print(f'{student}')
```

```
字典物件: {'name': '尤勇', 'age': 12, 'courses': ['數學', '電腦']}

以鍵="name" 更改值="夏琪"
更改過一項鍵值的字典物件:
{'name': '夏琪', 'age': 12, 'courses': ['數學', '電腦']}
更改過多項鍵值的字典物件:
{'name': '小叮噹', 'age': 6, 'courses': ['數學', '電腦'], 'phone': '888
-8888'}
清除字典中所有項目:
{}
```

字典物件的相關方法，例子如下：

- pop() 並與 del() 函式比較

In [85]:
```
```

```
student = {'name': '尤勇',
           'age': 12,
           'courses': ['數學', '電腦']}
print(f'字典物件: {student}\n')

# 移除 (del 函式) 字典中的一個項目
# 若鍵不存在，會有錯誤訊息 KeyError
key = 'phone'
try:
    del student[key]
except KeyError:
    print(f'移除(呼叫 del 函式)字典中的一個項目:')
    print(f'===> 鍵="{key}" 不存在')
except:
    print(f'移除(del 函式)字典中的一個項目:')
    print(f'===> 操作錯誤')

key= 'courses'
del student['courses']
print(f'移除(呼叫 del 函式)字典中的一個項目:')
print(f'===> 鍵="{key}"\n{student}')

# 移除 (pop 方法)字典中的一個項目，有返回其值
# 若鍵不存在，也會有錯誤訊息 KeyError
key ='age'
age_value = student.pop(key)
print(f'移除(呼叫 pop 方法)字典中的一個項目:')
print(f'===>"{key}":{age_value}\n{student}')

```

```
字典物件: {'name': '尤勇', 'age': 12, 'courses': ['數學', '電腦']}

移除(呼叫 del 函式)字典中的一個項目:
===> 鍵="phone" 不存在
移除(呼叫 del 函式)字典中的一個項目:
===> 鍵="courses"
{'name': '尤勇', 'age': 12}
移除(呼叫 pop 方法)字典中的一個項目:
===>"age":12
{'name': '尤勇'}
```

問： 上面的例子 try-ecept 是什麼東東？

答： 那是 Python 設計的一種指令讓我們處理錯誤狀況

「例外處理」指令 (try-except Statement)

語法

```
try:
    <程式區塊 a>

except:
    <程式區塊 b>
```

執行 Python <程式區塊 a>的時候，往往會遇到「錯誤」的狀況，如果沒有處理錯誤，Python 就會造成整個程式停止。 因此，透過「例外處理」的機制，能夠在發生錯誤時進行對應的動作 <程式區塊 b>，不僅能繼續程式的流程，也能夠掌握問題出現的位置，馬上進行修正。

except 的錯誤資訊

只要程式發生錯誤，控制台中都會出現對應的錯誤資訊，下方列出常見的幾種錯誤資訊 (詳細錯誤資訊參考：Built-in Exceptions (https://docs.python.org/3/library/exceptions.html)

問：複雜的物件如字典物件，在比較條件敘述中如何判斷其真偽？

答：Python 基於不同的物件的特性，對於判斷物件的偽值也有異。

以下評估條件敘述時，皆判斷為偽：

1. 條件敘述結果是：False 關鍵字
2. 條件敘述結果是：None 關鍵字
3. 條件敘述結果是：數字運算結果為零
4. 條件敘述結果是：任何空的序列物件，例如：空字串、空的列表、空的集合、空的字典等

以上條件皆判斷為偽，例子如下：

In [7]:
```
conditions = [False, None, 0, 10, '', 'Test', (), [], {}]
print(f'條件敘述: {conditions} 判斷如下：\n')

for cond in conditions:

    if cond:
        print(f'===> "{cond}" : Python 判為真 "True"')
    else:
        print(f'===> "{cond}" : Python 判為偽 "False"')

```

```
條件敘述: [False, None, 0, 10, '', 'Test', (), [], {}] 判斷如下：

===> "False" : Python 判為偽 "False"
===> "None" : Python 判為偽 "False"
===> "0" : Python 判為偽 "False"
===> "10" : Python 判為真 "True"
===> "" : Python 判為偽 "False"
===> "Test" : Python 判為真 "True"
===> "()" : Python 判為偽 "False"
===> "[]" : Python 判為偽 "False"
===> "{}" : Python 判為偽 "False"
```

問：字典物件中的項目有鍵及值， 可以擇一排序嗎？比起前面學的列表、元組物件中元素的大小排序似乎較簡單！

答：的確字典物件的排序較複雜。首先三者來比較排序的話，差異如下：

1. 列表是可變形的物件 (Mutable Object)，所以排序結果可放回原物件
2. 元組及字典的物件卻是不可變形 (Immutable Objects)，所以要新增一個列表來存放元組或字典物件的排序結果
3. 推而廣之，自訂的物件中若有字母數字型的屬性 (Alphanumeric Attributes)的話，也可以當排序鍵來排序自訂的物件

比較物件排序 (Sorting Objects)

1. 列表排序
2. 元組排序
3. 字典排序

列表排序例子如下：

In [88]:
```
""" Examples of sorting lists
列表排序例子
Two ways of sorting lists:
1. By sorted() function to create another sorted list (建立新的列表)
2. By sort() method to sort on the same list (原列表重新排序)
"""

#
li = [9, 1, 8, 2, 7, 3, 6, 4, 5]
print(f'列表排序例子...')
print(f'===>原列表:\n{li}')
print(f'===>Type: {type(li)}\n')

# 1. sorted() 函式建立新的列表 (正向排序)
s_li = sorted(li)
print(
    f'1. 用 sorted() 函式建立新的列表(遞增排序):\n{s_li}\n===>Type: {t

# 2. sort() 方法於原列表重新排序
li.sort()
print(f'2. 用 sort() 方法於原列表重新排序(遞增排序):\n{li}\n')

```

```
列表排序例子...
===>原列表:
[9, 1, 8, 2, 7, 3, 6, 4, 5]
===>Type: <class 'list'>

1. 用 sorted() 函式建立新的列表(遞增排序):
[1, 2, 3, 4, 5, 6, 7, 8, 9]
===>Type: <class 'list'>

2. 用 sort() 方法於原列表重新排序(遞增排序):
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

列表反向排序以及絕對值排序例子如下：

In [87]:
```
li = [9, 1, 8, 2, 7, 3, 6, 4, 5]
print(f'列表排序例子...')
print(f'===>原列表:\n{li}')

# Sort in a reverse order
s_li = sorted(li, reverse=True)
print(f'3. 用 sorted() 函式建立新的列表(遞減排序):\n{s_li}')

li.sort(reverse=True)
print(f'4. 用 sort() 方法於原列表重新排序(遞減排序):\n{li}\n')

# 絕對值排序
print(f'絕對值排序例子...')
li = [-6, -5, -4, 1, 2, 3]
print('Original List:\t', li)

s_li = sorted(li, key=abs)
print(f'5. 用 sorted() 函式建立新的列表(絕對值遞增排序):\n{s_li}')

```

```
列表排序例子...
===>原列表:
[9, 1, 8, 2, 7, 3, 6, 4, 5]
3. 用 sorted() 函式建立新的列表(遞減排序):
[9, 8, 7, 6, 5, 4, 3, 2, 1]
4. 用 sort() 方法於原列表重新排序(遞減排序):
[9, 8, 7, 6, 5, 4, 3, 2, 1]

絕對值排序例子...
Original List:   [-6, -5, -4, 1, 2, 3]
5. 用 sorted() 函式建立新的列表(絕對值遞增排序):
[1, 2, 3, -4, -5, -6]
```

元組排序例子如下：

In [90]:
```
"""Examples of Sorting Tuples by sorted() function
元組排序例子
Note: no sort method with Tuples, since tuples are immutable.
i.e. <tuple obj>.sort() resulted in an error
元組排序只能用 sorted() 函式間接排序
"""

tup = (9, 1, 8, 2, 7, 3, 6, 4, 5)
print(f'元組記憶體位置在：{id(tup)}:')
print(f'{tup}')
print(f'===>Type: {type(tup)}\n')

# Sort a tuple
print(f'元組經 sorted() 函式後變成列表:')
slist = sorted(tup)
print(f'列表記憶體位置在：{id(slist)}:')
print(f'{slist}')
print(f'===>Type: {type(slist)}\n')

s_tup = tuple(slist)
print(f'新元組記憶體位置在：{id(s_tup)}:')
print(f'{s_tup}')
print(f'===>Type: {type(s_tup)}\n')

```

```
元組記憶體位置在：140612619103360:
(9, 1, 8, 2, 7, 3, 6, 4, 5)
===>Type: <class 'tuple'>

元組經 sorted() 函式後變成列表:
列表記憶體位置在：140612643616448:
[1, 2, 3, 4, 5, 6, 7, 8, 9]
===>Type: <class 'list'>

新元組記憶體位置在：140612619103248:
(1, 2, 3, 4, 5, 6, 7, 8, 9)
===>Type: <class 'tuple'>
```

字典排序例子如下：

In [10]:
```

```

```
""" Examples of Sorting Dictionaries in two ways:
    字典排序例子
    1. Sorted by key (依鍵排序)
    2. Sorted by value (依值排序， 但前提值要有可排序性，即文字數字類型),
        assuming sortable alphanumeric attribute values
"""

# Original dictionary
di = {'name': 'Mike', 'job': 'programmer', 'age': '20', 'os': 'Mac
print(f'原字典:\n{di}')
print(f'===>類型: {type(di)}\n')

# 1. 依字典用 sorted() 函式建立新的字典(依鍵遞增排序)
s_li = sorted(di)
print(f'1. 依字典用 sorted() 函式建立新的列表(依鍵遞增排序)...')
print(f'===>其元素只含鍵，不含值:\n{s_li}')
print(f'===>類型: {type(s_li)}\n')

# 2. 依字典逐項用 sorted() 函式依鍵遞增排序 ()
s_li = sorted(di.items(), key=lambda kv: (kv[0], kv[1]))  # kv[0]
print(f'2. 依字典逐項用 sorted() 函式建立新的列表(依鍵遞增排序)...')
print(f'===>其元素是含鍵及值元组:\n{s_li}')
print(f'===>類型: {type(s_li)}\n')


# 3. 依字典逐項用 sorted() 函式依值遞增排序
s_li = sorted(di.items(), key=lambda kv: (kv[1], kv[0]))  # kv[0]
print(f'3. 依字典逐項用 sorted() 函式建立新的列表(依值遞增排序)...')
print(f'===>其元素是含鍵及值元组:\n{s_li}')
print(f'===>類型: {type(s_li)}\n')

```

```
原字典:
{'name': 'Mike', 'job': 'programmer', 'age': '20', 'os': 'Mac'}
===>類型: <class 'dict'>

1. 依字典用 sorted() 函式建立新的列表(依鍵遞增排序)...
===>其元素只含鍵，不含值:
['age', 'job', 'name', 'os']
===>類型: <class 'list'>

2. 依字典逐項用 sorted() 函式建立新的列表(依鍵遞增排序)...
===>其元素是含鍵及值元组:
[('age', '20'), ('job', 'programmer'), ('name', 'Mike'), ('os', 'Mac'
)]
===>類型: <class 'list'>

3. 依字典逐項用 sorted() 函式建立新的列表(依值遞增排序)...
===>其元素是含鍵及值元组:
[('age', '20'), ('os', 'Mac'), ('name', 'Mike'), ('job', 'programmer'
```

```
[{'age': 20}, {'sex': 'Male'}, {'name': 'Mike'}, {'job': 'programmer'}
)]
===>類型: <class 'list'>
```