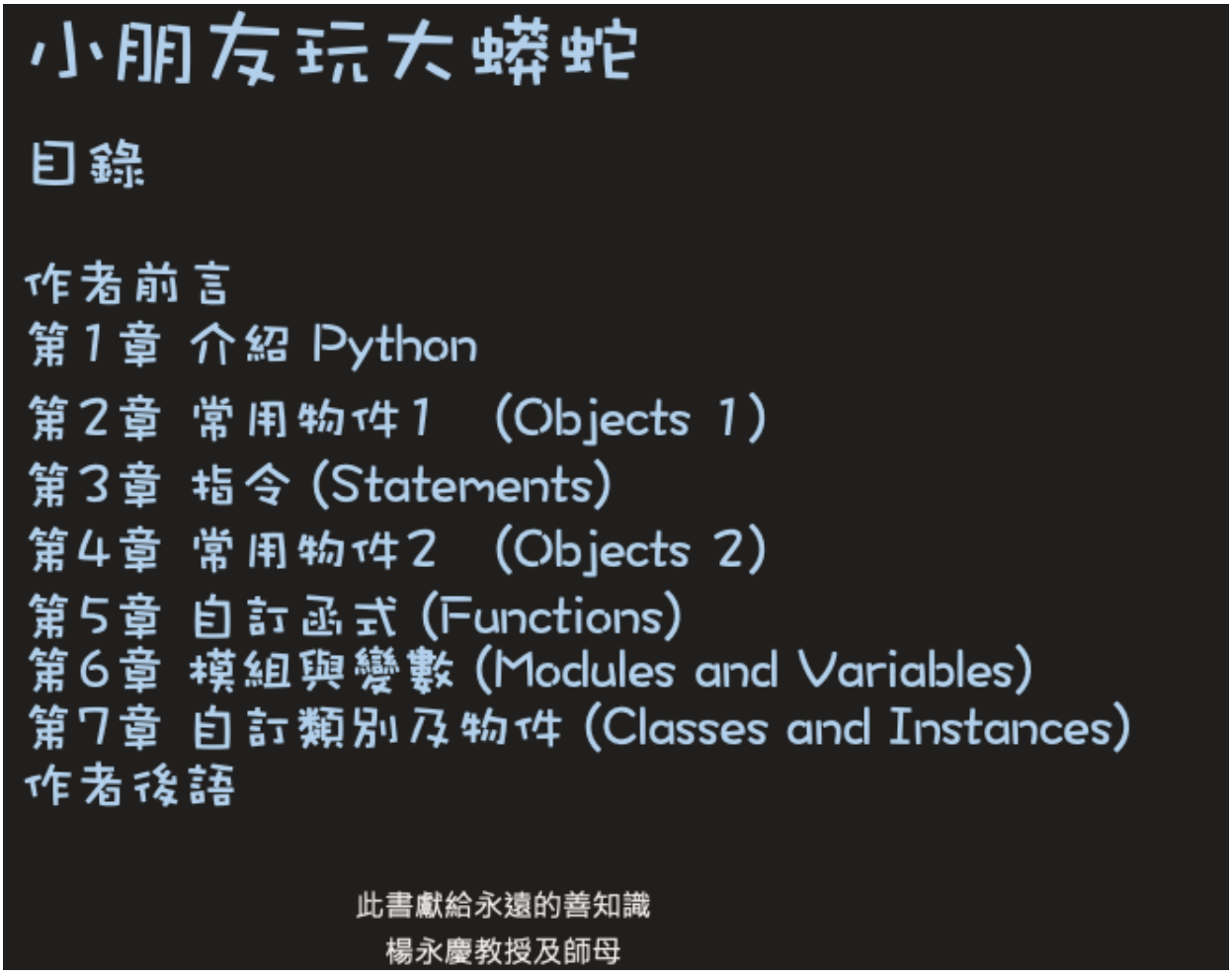

小朋友玩大蟒蛇
目錄
作者前言
第1章 介紹 Python
第2章 常用物件1 (Objects 1)
第3章 指令 (Statements)
第4章 常用物件2 (Objects 2)
第5章 自訂函式 (Functions)
第6章 模組與變數 (Modules and Variables)
第7章 自訂類別及物件 (Classes and Instances)
作者後語
此書獻給永遠的善知識
楊永慶教授及師母

作者前言

對於從沒寫書經驗，但受困於新冠疫情緣故，閒來在家沒事就思考到2022年有沒有可以動手做的計劃。於是乎寫書的念頭就成了選項之一。

出身軟體工程師，玩程式一直是我的愛好之一。自己也正在 YouTube 學 Python 的緣故，不如邊學時做的筆記，邊把它翻譯成繁體中文，以便嘉惠後進，尤其是偏鄉小學生們乃成為我寫這本書主要目標。

又考量到沒有實際操作電腦環境，所以用很多程式片段當例子來作參考。雖然有筆上談兵之嫌，然一旦有實作演練的環境下，這本書也可當入門工具書的參考。

本書共分7章。

開頭第1章 Python 簡介對剛入門的小朋友可以從此章入手。依循 Dennis Ritchie 介紹 C 語言的傳統，從 'Hello World' 開始。

第2章進入常用的基本資料介紹，像如何定義、訪問及運算等。另外第4章也是常用的基本資料介紹，但較複雜所以另闢一章討論。

中間插入第3章的是介紹常用的程式指令，讓小朋友可以動手寫一些小程式。

接着第5章讓小朋友開始思考把有用的方法或邏輯變成可重複使用的函式。

第6章進入模塊化觀念及善用變數，打好架構軟體工程的基礎。

最後第7章介紹物件導向的核心，如何應用類別模型解決問題。

這本書先以電子書免費流通，歡迎拷貝。當然書中若有瑕疵在所難免，隨時電郵指教。小朋友有任何疑問，也可以來電郵討論。

高嘉祥

2022.1.31

mkaoy2k@gmail.com (mailto:mkaoy2k@gmail.com)

第1章 介紹 Python

Python 是人類用來與電腦溝通的程式語言之一。就像早期的人學程式語言就要學 C 程式語言與電腦溝通一樣，現在人們又多了一個學習選擇是 Python。

就像人類語言為便於溝通會一再演化一樣，Python 也從前一代語言程式演化成較易懂又好學的。所以 Python 成為時下最夯的程式語言之一。

原來 Python 英文是「蟒蛇」的意思。發明者 Guido van Rossum 當初在一邊設計這套程式語言系統的時候，也正在看一部 1970 年代有關蟒蛇的連續劇，於是乎突發其想就命名它為 「Python」。

好了，現在開始學習。讓我們試著寫第一個 Python 程式看看吧！

問：如何請 Python 列印：哈囉， 您好！

答：很簡單。

Python 只要用一行指令就可以了，這個指令叫列印指令如下：

In [1]:
```
print('哈囉，您好!')
```

```
哈囉，您好!
```

問：列印指令括號裡，前後都有個單引號，怪怪的，是啥東西啊？

答： 單引號是告訴 Python 參數列只有一個。

Python 好像是一部遊戲，有很多寶物用保護盒裝著。這些引號就是一個寶物盒之一。

Python 用這些引號來包裝我們要的訊息。

讓我們進一步打開 Python 的藏寶箱，來看一下 Python 常用的幾種寶物盒子吧。

Python 把它的寶物盒取了一個專有名詞叫「物件 (Object)」，有點文謅謅的，英文翻譯過來的，沒辦法啦！而要學好 Python 就必須記住一些專有名詞及其代表的意義。

現在我們學習先了解下面五類常用的物件吧。

我們將會陸續介紹給大家，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別， 在這一章我們就要學習的。

而後面两類（4–5）物件就觀念比較複雜一點，屬於進階的資料物件類別，我們在後面另外別立一章來學習。

第2章 常用物件1 （Objects 1）

2.1 字串（String）

第一個常用的物件就是我們程式中用的「字串」。

字串的定義是用**單引號（‘）**或**雙引號（”）**一前一後來包裝一段字元訊息。

語法

' <字元列> '
" <字元列> "

上面學習過第一個哈囉程式中的（哈囉，您好！），就是一串<字元列>，也就是我們用單引號，一前一後包裝起來的字串物件的例子。再比如：

‘我叫志明’
“我叫春嬌”

以上皆是字串物件的例子。

In [2]:
```
name1 = '我叫志明'
name2 = "我叫春嬌"
```

介紹「指派」指令 (assignment statement)

語法

A = <敘述>

Python 設計指派指令讓我們可以將某<敘述>指派一個名稱或者說是將某<敘述>貼上一個標記A。其特色如下：

1) 指派指令不用英文字 assign 而是用大家耳熟能詳的數學符號：**等號（=）**來表示。
2) 指派的目標寫在**等號（=）**的左邊
3) 等號右邊的<敘述>可以是任何資料運算操作。

舉上面的例子，我們就是把右邊一串招呼訊息（**'哈囉，您好！'**）包裝成字串物件，然後指派一個名稱叫做 message。這個名稱 Python 叫它是「變數」。

變數的命名，Python 有以下命名規則：

1) 變數名稱可以是任何大小寫的英文字母或下底線（_）或數字所組成。
2) 變數名稱開頭不可以是數字

詳細變數的介紹有專篇容後詳細再加以說明。

現在回來小結一下

如果把前面我們學過的列印指令，加上剛學的指派指令，於是哈囉程式也可以變成二行，如下：

In [3]:
```
message = '哈囉，您好!'
print(message)
```

```
哈囉，您好!
```

太簡單了！

哈囉程式二版，我們加了一個然後列印指令的參數列用這個字串物件的變數完成了。

接下來再印兩個字串物件看看，例子如下：

In [4]:
```
name1 = '我叫志明'
print(name1)

name2 = "我叫春嬌"
print(name2)
```

```
我叫志明
我叫春嬌
```

問：字串物件可以裝多長的訊息呢？只能一行訊息，不能很多行嗎？

答：凡事都有它的侷限，字串物件亦然。

字串長度決定的因素有二：

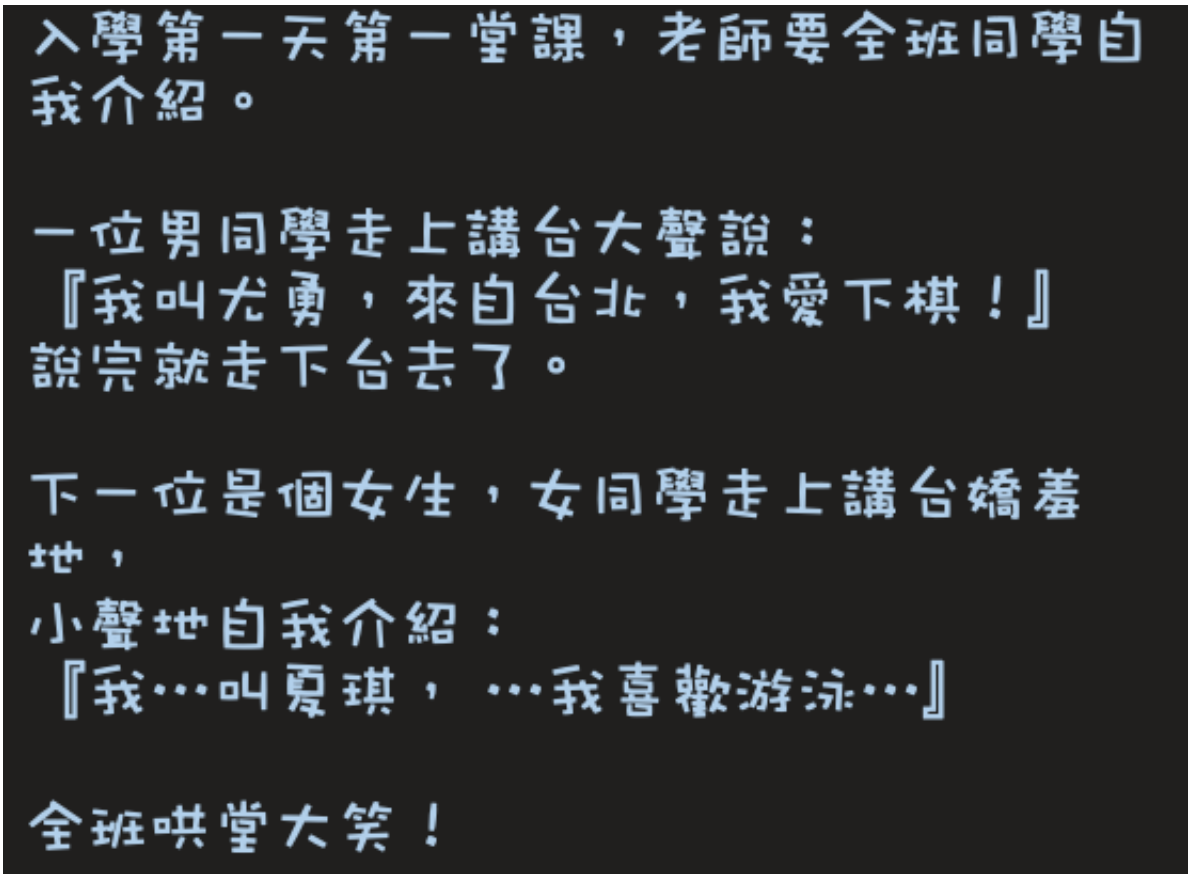
1) 不同操作系統 (Operating System)的架構
2) 不同國家地區使用的字集 (Character Set)

詳細情形，有興趣可 Google 一下電腦操作系統及字集國際標準的課程有更透徹的原理說明。

對剛入門的我們可以簡單這樣了解：若在 64-bit 操作系統下，只用英文字集的話，最長约900萬 Tera Bytes (1 TB = 9x1018 Bytes)。這個數字超大的。若用中文字集 (2個 Bytes 代表一個中文字) 就要減半，變成可容納 450萬 TB也夠大了。我們一般應用程式不會受到此限制所影響。

再者，字串物件是否可存放多行訊息？答案是可以。

舉例，以下一則很多行的笑話如下：



這一種情況，Python 設計讓我們用三個雙引號（"）或單引號（'），前後把整篇多行的笑話包裝進一個字串物件中。

多行字串物件的程式如下：

In [5]:
```
# 二行以上落落長的笑話用三個雙引號前後包裝成一個字串
# 然後指派到一個變數，名叫 joke
joke = """
    入學第一天第一堂課，老師要全班同學自我介紹。

    一位男同學走上講台大聲說：
    『我叫尤勇，來自台北，我愛下棋！』
    說完就走下台去了。

    下一位是個女生，女同學走上講台嬌羞地，
    小聲地自我介紹：
    『我…叫夏琪， …我喜歡游泳…』

    全班哄堂大笑！

"""

print(joke)
```

```
    入學第一天第一堂課，老師要全班同學自我介紹。

    一位男同學走上講台大聲說：
    『我叫尤勇，來自台北，我愛下棋！』
    說完就走下台去了。

    下一位是個女生，女同學走上講台嬌羞地，
    小聲地自我介紹：
    『我…叫夏琪， …我喜歡游泳…』

    全班哄堂大笑！

```

註解指令 (Comments)

上面這一段程式除了執行指令（给 Python 執行的）之外，不知各位是否注意到加入二條註解指令（"#" 開頭， 專為人類看的敍述）Python 不會當指令執行 。 如上面第一行和第二行都是註解指令。

小結一下：

任何一個Python 程式都可以有下列三種指令類型所组成。

1. Python 執行的指令 (statements)
2. 人們看的註解指令 (comments)
3. 人們留白的空白行指令 (blank lines)

Python 只懂得第一種執行指令而依序執行。

其中後面兩種指令：註解和空白行是專為程式讀者做註解及留白所設的。一個好的程式設計師必須養成在程式中寫註解指令的好習慣。

詳細的執行指令我們陸續會學習， 現在有一個簡單的概念就行。

問：字串物件中的訊息輸出簡單，那麼如何輸入訊息給程式呢？

答：輸入訊息一樣簡單。

接著下面就來介紹如何呼叫輸入函式 input(<提示字串>)，Python 能印出一段提示字串，然後等使用者輸入資訊，按 enter 鍵後，Python 再返回程式生成字串物件。這個物件包裝有使用者輸入的資訊。

至於函式的詳細內容後面會介紹的。

現在我們看程式如下：

In [6]:
```
# 從鍵盤輸入，呼叫 input(<提示字串>)函式
name = input("請問你的名字叫: ")

# 螢幕上輸出
print(name)
```

```
請問你的名字叫: 尤勇
尤勇
```

字串物件的運算

字串物件的常用運算操作，有下列五種：

1）字串物件的粘接（Concatenation）
2）字串物件的方法（Method）
3）字串物件的格式化（Format）
4）字串物件可套用的函数（Function）
5）字串物件的截取（Slicing）

字串物件的粘接 (Concatenation)

多個字串物件可以粘接成一個字串物件。其運算子（Operator）Python 設計用 "+" 來敘述。

字串物件粘接的例子，如下：

In [7]:
```
greeting = '哈囉'
name = '尤勇'

# 兩個字串物件之間，用 '+' 運算子粘接
print('多個字串物件粘接成一個字串物件，列印出來如下：')
message = greeting + ', ' + name + '. 歡迎!'
print(message)
```

```
多個字串物件粘接成一個字串物件，列印出來如下：
哈囉, 尤勇. 歡迎!
```

字串物件的方法（Method）

物件的方法（Method）是附屬於物件而提供跟該物件相關的而且可以重覆呼叫的程式。詳細的物件方法我們後面會再更詳細介绍。

目前先介绍字串物件幾個常用的方法，例子如下：

In [8]:
```
message = 'Hello, 尤勇. 歡迎!'
# 字元位置  0123456 78 9

# lower case method (適用於英文轉換小寫的方法)
print(message.lower())
```

```
# upper case method (適用於英文轉換大寫的方法)
print(message.upper())

# count method (字串中算字元出現次數的方法)
print('message 字串中有幾個 "h"?')
print(message.count('h'))
print()

print('message 字串中有幾個 "H"?')
print(message.count('H'))
print()

print('message 字串中有幾個 "l"?')
print(message.count('l'))
print()

# find method (字串中搜尋字元或子字串的方法)
print("字串中有'world'嗎？")
print(message.find('world'))
print()

print("字串中有'World'嗎？")
print(message.find('World'))
print()

print("字串中有'勇'嗎？")
print(message.find('勇'))
print()

print("字串中有'夏'嗎？")
print(message.find('夏'))
print()

print("字串中有' '(空格) 嗎？")
print(message.find(' '))
print()

# replace method (字串中取代字元或子字串的方法)
print("用'哈囉' 取代 'Hello'")
message = message.replace('Hello', '哈囉')
print(message)
```

```
hello, 尤勇. 歡迎!
HELLO, 尤勇. 歡迎!
message 字串中有幾個 "h"?
0

message 字串中有幾個 "H"?
1
```

```
message 字串中有幾個 "l"?
2

字串中有'world'嗎？
-1

字串中有'World'嗎？
-1

字串中有'勇'嗎？
8

字串中有'夏'嗎？
-1

字串中有' '(空格) 嗎？
6

用'哈囉' 取代 'Hello'
哈囉, 尤勇. 歡迎!
```

字串物件可套用的函式（Function）

Python 系統也提供一些內建通用程式，讓我們寫程式可方便呼叫而且可套用到多個物件，叫做「函式」（Function），這與上面剛學習的物件方法（Method）雖然都是可以重覆運用的程式，但不同的是物件方法配屬在某特定的物件之下的程式，而函式是獨立於物件之外的程式。換言之，函式可以套用到不同的物件類型。

Python 方法與函式呼叫的文法也不同：

1) 物件方法的呼叫：
   用物件名稱加小數點來呼叫方法，如上。
2) 函式的呼叫：
   是直接呼叫函式名稱加前後括號，把物件放入括號中當參數（Arguments）。
   比如說 'len ()' 就是一個函式用來測量物件的長度，把量測物件放入括號裡，當參數來呼叫。
   函式的呼叫例子如下：

In [9]:
```
message = '哈囉，您好!'

print("len()函式功能是量測物件的長度:")
print(len(message))

print("type()函式功能是顯示物件的類型:")
print(type(message))
```

```
len()函式功能是量測物件的長度:
6
type()函式功能是顯示物件的類型:
<class 'str'>
```

字串物件的格式化 (Format)

字串最常看到的運算就是放在 print指令的參數中，運行格式化。

其目的主要是因應人們不同顯示需求而給予字串物件重整，輸出表格化的訊息。

下面我們來學習兩種格式化字串物件的例子：

1) 字串物件 format 方法
2) Python 提供的 f-字串指令

In [10]:
```
greeting = '哈囉'
name = '尤勇'

# using format method (用字串物件的 'format' 方法)
print("用字串物件的 'format' 方法...")
message1 = '{}, {}. 歡迎!'.format(greeting, name)
print(message1)
print()

# using f-string in Python 3 (用 Python 版本3以後才有的 'f-字串指令')
print("用 Python 版本3以後才有的 'f-字串指令'...")
message2 = f'{greeting}, {name}. 歡迎!'
print(message2)
```

```
用字串物件的 'format' 方法...
哈囉, 尤勇. 歡迎!

用 Python 版本3以後才有的 'f-字串指令'...
哈囉, 尤勇. 歡迎!
```

字串格式化的應用

字串物件有很多常用格式化功能，尤其應用在列印上。

下面來介紹列印指令如何配合字串格式化帶給我們不同的報表效果。

1） 列印指令括號中，若無參數或只有一個空字串，就列印一條空白行
2） 列印指令括號中，若有二個字串以上作為參數，之間逗號“，”分開
也代表二個字串粘接運算，但中間加印一個空格。

另外 Python 設計字串物件中，可以有反斜線“\”開頭，視為特別的格式化字元符號。

其代表不同格式如下：

特別字串	代表格式
\t	移到下一個製表點
\n	移到下一行
\r	移到此行第一個位置
\b	移回上一個位置
\f	移到下一頁
\'	列印單引號
\"	列印雙引號
\\	列印反斜線

用上面的例子，我們加上一些反斜線特別字元試試，看看格式化後帶來什麼不同效果：

In [11]:
```
greeting = '哈囉'
name = '尤勇'

# using format method (用字串物件的 'format' 方法)
print("用字串物件的 'format' 方法...")
message1 = '{}, {}. 歡迎!\n'.format(greeting, name)
print('\t' + message1)

# using f-string in Python 3 (用 Python 版本3以後才有的 'f-字串指令')
print('用 Python 版本3以後才有的 \'f-字串指令\'...')
message2 = f'{greeting}, {name}. 歡迎!'
print(f'\t{message2}\n')

# 1. 列印指令括號中若無參數，就印一條空白行
print('下一條是空白行...')
print()
print('上一條是空白行!\n')

# 2. 列印指令括號中若有二個字串以上作為參數，之間可以用逗號 “,” 代表二字串粘接中間
print(greeting, name)
```

```
用字串物件的 'format' 方法...
        哈囉, 尤勇. 歡迎!

用 Python 版本3以後才有的 'f-字串指令'...
        哈囉, 尤勇. 歡迎!

下一條是空白行...

上一條是空白行!

哈囉 尤勇
```

字串物件的截取（Slicing）

太棒了！字串物件格式化也算簡單吧！

除了訊息可格式化外，有時候我們要的訊息必須從字串中切開截取出來。

下面就來學習字串截取，例子如下：

In [12]:
```
name = '哈囉 我的名字叫尤勇'
#       0-1 3 4 5 6  8

# 字串可視為一條字元的排列式，位置從0開始算起
print("字串中，切割出打招呼'哈囉'兩個字元：從第1個字元到第2個字但不包含第2個字元:"
message1 = name[0:2]
print(f'{message1}\n')

print("字串中，切割出'我的名字'四個字元：從第3個字元到第7個字但不包含第7個字元:")
message2 = name[3:7]
print(f'{message2}\n')

print("字串中，切割出名字：從第8個字元到最後一個字元:")
message3 = name[8:]
print(f'{message3}\n')
```

```
字串中，切割出打招呼'哈囉'兩個字元：從第1個字元到第2個字但不包含第2個字元:
哈囉

字串中，切割出'我的名字'四個字元：從第3個字元到第7個字但不包含第7個字元:
我的名字

字串中，切割出名字：從第8個字元到最後一個字元:
尤勇
```

2.2 數字 (Number)

第二個常用的資料物件類型：數字。

數字物件由以下規則來定義：

1) 是一串由0-9十進位數字所組成，例如：12345
2) 數字可以有一位小數點，叫浮點數 (float)，例如：123.45
3) 數字沒有小數點的叫整數 (integer)，例如：123
4) 整數又可分為正整數（如：123）、零（即：0）和負整數（如：-123）

Python 又再细分數字物件為：

1) 整數物件 (int)
2) 浮點數物件 (float)

In [13]:
```
# 字串物件轉換成整數物件
num_1 = int('100')
print(f'num_1 物件類型： {type(num_1)}\n')

print(f'{num_1} 是整數物件')
print(f'\t{num_1 + num_1} 也是整數物件!\n')

# 整數物件轉換成字串物件
str_1 = str(num_1)
print(f'str_1 物件類型： {type(str_1)}\n')

print(f'{str_1} 是字串物件')
print(f'\t{str_1 + str_1} 也是字串物件!')
```

```
num_1 物件類型： <class 'int'>

100 是整數物件
        200 也是整數物件!

str_1 物件類型： <class 'str'>

100 是字串物件
        100100 也是字串物件!
```

In [14]:
```
# 浮點數物件轉換成整數物件
float_1 = 3.45
float_2 = 5.56
print(f'float_1 物件類型： {type(float_1)}\n')
print(f'float_1 浮點數物件轉換成整數物件 = {round(float_1)}\n')

# 浮點數物件互換
print(float_1, '四捨五入到小數點第一位 =',
      round(float_1,1))
print(float_2, '四捨五入到小數點第一位 =',
      round(float_2,1))
```

```
float_1 物件類型： <class 'float'>

float_1 浮點數物件轉換成整數物件 = 3

3.45 四捨五入到小數點第一位 = 3.5
5.56 四捨五入到小數點第一位 = 5.6
```

數字物件的運算

常用的數字物件的算術運算有七種，如下：

1）加（如：x + y）
2）減（如：x − y）
3）乘（如：x * y）
4）除（如：x / y）
5）餘（如：x % y）
6）商（如：x // y）
7）乘冪（如：x ** y）

數字物件的算術運算的例子如下：

In [15]:
```
# 1. 加 (addition):
print(f'3 + 2 = {3 + 2}')

# 2.減 (Subtraction):
print(f'3 - 2 = {3 - 2}')

# 3. 乘 (Multiplication):
print(f'3 * 2 = {3 * 2}')

# 4. 除 (Division):
print(f'3 / 2 = {3 / 2}')

# 5. 餘 (Modulus):
print(f'3 % 2 = {3 % 2}')

# 6. 商 (Floor Division):
print(f'3 // 2 = {3 // 2}')

# 7. 乘冪 (Exponent):
print(f'3 ** 2 = {3 ** 2}')

# 還有其他的運算，如下：
# 加法簡寫
num = 1
print(f'num = {num}')

# num = num + 10 也可以寫成如下：
num += 10
print(f'num += 10 是 num = num + 10 加法簡寫\nnum = {num}\n')

# 絕對值函式 (abs):
num = -4
print(f'絕對值函式 (abs): {num} 絕對值 = {abs(num)}')
```

```
3 + 2 = 5
3 - 2 = 1
3 * 2 = 6
3 / 2 = 1.5
3 % 2 = 1
3 // 2 = 1
3 ** 2 = 9
num = 1
num += 10 是 num = num + 10 加法簡寫
num = 11

絕對值函式 (abs): -4 絕對值 = 4
```

2.2 布林值 (Boolean)

2.3 布林值（Boolean）

目前為止我們已經學習了字串、數字這兩種資料物件的類型。第三個資料物件類型：布林值，我們繼續來學習。

布林值物件有以下特色：

1）不像字串、數字，布林值是比較簡單但特殊的物件，它只有兩個固定值："真"和"偽"。
2）布林真值 Python 保留字是：True
3）布林偽值 Python 保留字是：False

注意：True 和 False 第一個字母是大寫。

布林值的用法

有了布林值，我們可以設定條件讓 Python 作邏輯判斷其真偽而執行不同的任務。與布林值相關的控制指令中，最常用的就是假設指令（if statement）。

假設指令的功能就是根據一個條件敘述（conditional expression，或 condition）的真偽判斷，比如說兩個整數： x 和 y 比較大小，條件敘述就是 "x > y"。經過這個條件敘述運算最後，Python 得出一個或真或偽的布林值。再配合下面將介紹的假設指令，我們的程式就有邏輯判斷的功能了。

介紹「假設」指令：if statement

語法

1. if <條件敘述1成真>：
        <程式區塊 A>

2. elif <條件敘述2成真>：
        <程式區塊 B>

3. else ：
        <程式區塊 C>

其中：

1）假設指令由三種保留字組成：if, elif, 及else
2） if 敘述一定要有
3）其他兩種 elif 和 else 敘述看需要條件而定
4）elif 可以多個而 else 最多只能出現一次
5）條件敍述（conditional expression）
不管是簡單或複合的條件，只要 Python 能判斷出布林值即可
6）每一個程式區塊都要同一排縮格（identation）
Python 用縮格來分別執行不同的程式區塊

Python 執行假設指令時：

1）首先判斷<條件敘述1>的真偽
2）若<條件敘述1>為真，則執行<程式區塊 A>
3）若<條件敘述1>為偽，則執行 elif 判斷<條件敘述2>的真偽
4）若<條件敘述2>為真，則執行<程式區塊 B>
5）若<條件敘述2>為偽，則執行 else 的<程式區塊 C>

注意：

1）三個程式區塊 A, B, C中，假設指令只有擇一區塊執行。
2）有保留字: if , elif, 和 else 的那一行最後的字元必須是「：」。

以下用假設指令例子，如下：

In [16]:
```
x = 0
y = 5
print(f'x = {x}\ny = {y}')

bool_exp = x > y
print(f'bool_exp 物件類型: {type(bool_exp)}\n')

if x > y:
    # <程式區塊 A>
    print('x 大於 y')

elif x < y:
    # <程式區塊 B>
    print('x 小於 y')

else:
    # <程式區塊 C>
    print('x 等於 y')

# 更多例子...簡單的條件敘述
if x:
    print('x 不是零')
else:
    print('x 是零')

if y:
    print('y 不是零')
else:
    print('y 是零')

# 例子...用 or 或 and 組成的複合條件敘述
if x or y:
    print('x 或 y 至少有一個不是零')

if x and y:
    print('x 和 y 兩個皆不是零')
```

```
x = 0
y = 5
bool_exp 物件類型: <class 'bool'>

x 小於 y
x 是零
y 不是零
x 或 y 至少有一個不是零
```

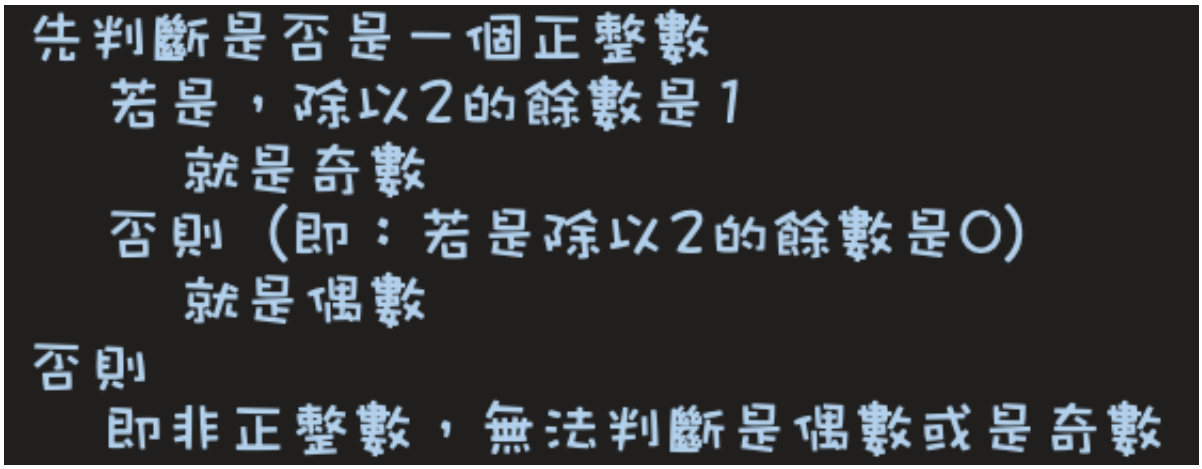
問：上面學的餘數運算"%"在程式上如何應用呢？

答：餘數運算在分類應用很廣。

一個最常見的應用，如下：

當我們要判斷一個整數是偶數或是奇數。寫程式時，配合假設指令就可以完成。

所以我們的程式邏輯，流程如下：



接著我們試著以下程式來說明餘數的應用：

In [17]:
```
num_str = input('請輸入一個正整數: ')

# 字串轉換成整數
num = int(num_str)

if num > 0:
    if (num % 2):
        print(f'{num} 是奇數!')
    else:
        print(f'{num} 是偶數!')
else:
    print(f'{num} 不是正整數，無法判斷是偶數或是奇數!')
```

```
請輸入一個正整數: 142857
142857 是奇數!
```

問：條件敘述中，數字物件的比較真偽，應該很常用吧？

答：是的。

有六個常用的數字比較運算：

1）等於（如：x == y）
2）不等於（如：x != y）
3）大於（如：x > y）
4）小於（如：x < y）
5）大於或等於（如：x >= y）
6）小於或等於（如：x <= y）

數字比較運算

數字比較運算，例子如下：

In [18]:
```
num_1 = 3
num_2 = 2
print(f'num_1 = {num_1}')
print(f'num_2 = {num_2}\n')

# 1. 等於（如：x == y）
print(f'問：{num_1} == {num_2} 嗎?')
print (f'答：{num_1 == num_2}\n')

# 2. 不等於（如：x != y）
print(f'問：{num_1} != {num_2} 嗎?')
print (f'答：{num_1 != num_2}\n')

# 3. 大於（如：x > y）
print(f'問：{num_1} > {num_2} 嗎?')
print (f'答：{num_1 > num_2}\n')

# 4. 小於（如：x < y）
print(f'問：{num_1} < {num_2} 嗎?')
print (f'答：{num_1 < num_2}\n')

# 5. 大於或等於（如：x >= y）
print(f'問：{num_1} >= {num_2} 嗎?')
print (f'答：{num_1 >= num_2}\n')

# 6. 小於或等於（如：x <= y）
print(f'問：{num_1} <= {num_2} 嗎?')
print (f'答：{num_1 <= num_2}\n')
```

```
num_1 = 3
num_2 = 2

問：3 == 2 嗎?
答：False

問：3 != 2 嗎?
答：True

問：3 > 2 嗎?
答：True

問：3 < 2 嗎?
答：False

問：3 >= 2 嗎?
答：True

問：3 <= 2 嗎?
答：False
```

問：條件敘述中，字串物件也能比較真偽嗎？

答：是的。

字串物件的比較條件通常可以是兩種情況：

1）比較字串的內容條件是否相同。
2）呼叫字串物件方法（method）後，比較返回的回值（returned value）與期望值是否相同。

字串的比較運算

字串物件的條件敘述比較運算，例子如下：

In [19]:
```
# 比較字串的內容條件是否相同
language = 'Python'

if language == 'Python':
    print('程式語言是 Python\n')
elif language == 'Java':
    print('程式語言是 Java\n')
elif language == 'JavaScript':
    print('程式語言是 JavaScript\n')
else:
    print('程式語言未知\n')

message = '哈囉, 尤勇. 歡迎!'

# 呼叫字串物件方法 (method) 後，比較返回的回值 (returned value) 與期望值是否相
target = '勇'
print(f'問：字串中有\'{target}\'嗎？')

# 字串中，搜尋字元或子字串的方法 'find' method
index = message.find(target)
if  index == -1:
    print(f'答：\'{message}\'字串中，沒有:\'{target}\'')
else:
    print(f'答：\'{message}\'字串中，在第{index}個索引有:\'{target}\'\n')

# 條件敘述也是可以用： 'and', 'or', 'not' 組合成複合式條件敘述
user = 'Admin'
logged_in = True

if user == 'Admin' and logged_in:   #複合式條件敘述
    print('管理員頁面，登入成功！\n')
else:
    print('使用權限不足\n')
```

```
程式語言是 Python

問：字串中有'勇'嗎？
答：'哈囉, 尤勇. 歡迎!'字串中，在第5個索引有:'勇'

管理員頁面，登入成功！
```

第3章 指令 (Statements)

問：複習一下，我們到目前為止學習過的指令有記得多少？

答：試試大家的記憶體有多大：

1) 列印指令
2) 指派指令
3) f-字串指令
4) 註解指令
5) 空白行指令
6) 假設指令

以上指令都有學過喔！忘了回頭複習一下吧。

其中除了註解指令及空白行指令外，全都是 Python 的執行指令。接下來我們要學習迴路指令。

這裡我們將介紹更多指令，叫做迴路或迴圈指令。

對於重複的任務我們可以用 Python 迴路指令來設定在某種條件下，由一個程式區塊來執行重複的任務。

Python 的迴路指令，只有兩種：

1) for 迴路指令
2) while 迴路指令

3.1 介紹 for 迴路指令

語法

for <條件敘述> 若為真：

<程式區塊>

其中：

1) <條件敘述> 最常用的型態是 <物件變數> in <序列物件>
2) 序列物件的特性就是有可迴圈（iterable）的迭代性
   目前我們學習過的字串物件，就是序列物件的一個例子
   下一章我們還會學習更多的序列物件，包含有：
   列表、元組、集合、字典等序列物件
3) 當序列物件從頭到尾的迴圈中，
   Python 逐個元素載入物件變數，執行此程式區塊
4) 在程式區塊中，Python 又設計兩個控制指令，如下：
   (1) 破圈指令（break statement）：
   讓我們終止執行 for 迴路指令破圈而出
   接著執行 for 迴路指令下面的指令
   (2) 跳圈指令（continue statement）：
   讓我們終止執行此程式區塊並執行下一輪迴路的<條件敘述>

注意：

保留字: for 那一行最後的字元必須是「：」
保留字: in <序列物件> 就是要求 Python 把此序列中元素建成一個迴圈 (iterable)

我們來看一下 for 迴路指令的程式，例子如下：

In [20]:
```
message = '哈囉, 尤勇. 歡迎!'

print('for 迴路開始...')

for char in message:
    print(f'\t 字元 = {char}')

print('for 迴路終止')
```

```
for 迴路開始...
	 字元 = 哈
	 字元 = 囉
	 字元 = ,
	 字元 =
	 字元 = 尤
	 字元 = 勇
	 字元 = .
	 字元 =
	 字元 = 歡
	 字元 = 迎
	 字元 = !
for 迴路終止
```

In [21]:
```
message = '哈囉, 尤勇. 歡迎!'

print('for 迴路開始...')

for char in message:

    if char == ".":
        print(f'碰到 "{char}" 破圈而出!')

        # 破圈指令 break
        break

    print(f'\t 字元 = {char}')

print('for 迴路終止\n')


print('for 迴路開始...')

for char in message:

    if char == ".":
        print(f'碰到 "{char}" 跳下一迴圈!')

        # 跳圈指令 continue
```

```
        # 跳圈指令 continue
        continue

    print(f'\t 字元 = {char}')

print('for 迴路終止')
```

```
for 迴路開始...
        字元 = 哈
        字元 = 囉
        字元 = ,
        字元 =
        字元 = 尤
        字元 = 勇
碰到 "." 破圈而出!
for 迴路終止

for 迴路開始...
        字元 = 哈
        字元 = 囉
        字元 = ,
        字元 =
        字元 = 尤
        字元 = 勇
碰到 "." 跳下一迴圈!
        字元 =
        字元 = 歡
        字元 = 迎
        字元 = !
for 迴路終止
```

問：迴路指令中，有內建計數器的功能嗎？

答：是的。

一般較古老計數器程式是這樣寫：

```
counter = 0                 # 計數器設初值
counter_end = 5    # 計數器設終值

for counter < counter_end:
    <程式區塊>
    counter += 1  # 計數器加一
```

Python 簡化成用一個函式 range(範圍) 來當計數器。

以下程式說明：

In [22]:
```
# range() 函式例子
print('for 計數器開始...')

for num in range(3):
    print(f'num = {num}')
print('for 計數器終止\n')

# 兩層迴路的例子
print('兩層迴路開始...')

for num in range(3):

    print(f'第一層 for 迴路 {num}')
    for char in 'abc':

        print(f'\t 第二層 for 迴路 {char}...')
        print(f'\t {num} {char}')
```

```
for 計數器開始...
num = 0
num = 1
num = 2
for 計數器終止

兩層迴路開始...
第一層 for 迴路 0
         第二層 for 迴路 a...
         0 a
         第二層 for 迴路 b...
         0 b
         第二層 for 迴路 c...
         0 c
第一層 for 迴路 1
         第二層 for 迴路 a...
         1 a
         第二層 for 迴路 b...
         1 b
         第二層 for 迴路 c...
         1 c
第一層 for 迴路 2
         第二層 for 迴路 a...
         2 a
         第二層 for 迴路 b...
         2 b
         第二層 for 迴路 c...
         2 c
```

3.2 介紹 while 迴路指令

語法

while <條件敘述>為真：
<程式區塊>

其中：

1) 條件敘述若為真，就執行迴圈內程式區塊。
2) while迴路指令也可在程式區塊中，
呼叫破圈指令（break）和跳圈指令（continue）。

注意：

保留字 while 那一行最後的字元必須是「：」。

我們來看一下 while 迴路指令的程式例子如下：

In [23]:
```
print('while 迴路開始...')
counter = 0   # counter 設初值

while counter < 10:
    print(f'\t counter = {counter}')

    counter += 1   # counter 加一

print('while 迴路終止\n')

# break 例子
print('while 迴路開始...')

counter = 0   # counter 設初值

while counter < 10:
    print(f'\t counter = {counter}')

    if counter == 5:
        print(f'\t while 迴路中碰到 {counter} 破圈而出...')
        break

    counter += 1   # counter 加一

print('while 迴路終止\n')
```

```
while 迴路開始...
          counter = 0
          counter = 1
          counter = 2
          counter = 3
          counter = 4
          counter = 5
          counter = 6
          counter = 7
          counter = 8
          counter = 9
while 迴路終止

while 迴路開始...
          counter = 0
          counter = 1
          counter = 2
          counter = 3
          counter = 4
          counter = 5
          while 迴路中碰到 5 破圈而出...
while 迴路終止
```

第4章 常用物件2 （Objects 2）

我們在前面有提到五種常用的物件，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。

一旦常用的五種物件能熟練，我們對Python 程式的讀寫目標應該更近了。

4.1 列表 (List)

列表 （List） 是一個有排列順序的元素列。

列表有以下特性：

1） 列表用方括號 "[", "]" 前後包裝一元素列
2） 列表中每個元素 （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。

訪問列表物件

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 [25]:
```
courses = ['歷史', '物理', '數學', '電腦']
print(f'課程列表: {courses}')

# 截取列表物件
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 [26]:
```
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]}\n')
print(f'---> 列表範圍截取 5-9: {my_list[5:]}\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')


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')
```

```
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]

---> 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]

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。

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 [27]:
```
# 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}\n')

# 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}\n')
```

```
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 [28]:
```
list1 = [1, 2, 3]
list2 = [4, 5, 6]

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

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

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

```
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看看元素是否在列表中
2) 比較兩個列表的內容是否相同，有兩種可能：
   (1) 順序可能不同，但內容相同
       例如: [1, 2, 3] 與 [3, 2, 1]
   (2) 順序相同，內容也相同

列表物件的比較運算，例子如下：

In [29]:
```
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'3 在 list1 數字列表 list1 中嗎? {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: {list1}\n')

list2 = [3, 2, 1]
print(f'數字列表 list2 = {list2}\n')

# sorted() 函式下面解說
list1_copy = sorted(list1)
list2_copy = sorted(list2)

if list1 == list2:
    print(f'{list1} 與 {list2} 順序相同，內容也相同\n')
elif list1_copy == list2_copy:
    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 = [4, 2, 3]

3 在 list1 數字列表 list1 中嗎? True

3 於數字列表 list1 中: [4, 2, 3]

數字列表 list2 = [3, 2, 1]

[4, 2, 3] 不同於 [3, 2, 1]
```

問：列表中對每一個元素用迴路指令處理，應該常用吧？

答：是的。

> Python 讓我們用 for 迴路指令，將列表中每一元素，逐個比較並且執行相應的處理。
>
> 來看看列表迴路的例子，如下：

In [30]:
```
courses = ['歷史', '物理', '電腦', '物理']
print(f'課程列表: {courses}\n')

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

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

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

```
課程列表: ['歷史', '物理', '電腦', '物理']

列表迴路1: 從第一個元素開始...
---> 歷史
---> 物理
---> 電腦
---> 物理
列表迴路1 終止!

列表迴路2: 從第一個元素(索引0)開始...
---> 0 歷史
---> 1 物理
---> 2 電腦
---> 3 物理
列表迴路2 終止!

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

列表物件相關函式，例子如下：

In [31]:
```
numbers = [9, 8, 1, 2, 4, 7]
print(f'數字列表: {numbers}\n')

print(f'1. 列表長度: {len(numbers)}\n')

print(f'2. 列表中最大元素: {max(numbers)}\n')

print(f'3. 列表中最小元素: {min(numbers)}\n')

print(f'4. 列表總計: {sum(numbers)}\n')

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

```
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 [32]:
```
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}\n')

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

list_new = [10, 20]
```

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

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

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

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

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

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

numbers.sort()
print(f'9. 列表重新排序: {numbers}\n')

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

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

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

```
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

5. 插入 1 後列表: [1, 9, 8, 1, 2, 4, 7, 1, 10, 20]

6. 列表中彈出: 20 後列表: [1, 9, 8, 1, 2, 4, 7, 1, 10]

7. 移除第一個 1 後列表: [9, 8, 1, 2, 4, 7, 1, 10]

8. 反向列表: [10, 1, 7, 4, 2, 1, 8, 9]

9. 列表重新排序: [1, 1, 2, 4, 7, 8, 9, 10]
```

```
課程列表: ['歷史', '物理', '電腦', '物理']

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) 及列表物件類型， 我們學習過了。

這裡我們將學習第4類類型中的元組物件。

元組 (tuple) 是一個沒有排列順序的元素列。元組有以下特性：

1) 元組用小括號 "(", ")" 前後包裝元素列，元素之間以逗號分隔。
2) 元組中每個元素不需要具有相同類型。
3) 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。
4) 元組中只包含一個元素時，需要在元素後面添加逗號。
5) 元組不能新增、修改、或刪除元素，但是可以進行截取，組合。
6) 空白元組以 () 或 tuple() 來表示。

訪問元組

元組可以使用索引來訪問元組中的元素，注意索引都要用方括號，如下實例：

In [33]:
```
tuple1 = (1, 2, 3, 4, 5, 4)
tuple2 = (1, 2, 5)

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

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

```
tuple1 物件類型: <class 'tuple'>

tuple1 = (1, 2, 3, 4, 5, 4)

tuple2 = (1, 2, 5)

tuple1 元組第一個元素 = 1

tuple1 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 [34]:
```
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]}\n')
print(f'tuple2 元組第一個到第三個元素 = {tuple2[:3]}\n')
```

```
tuple1 元組奇數索引的元素 = (2, 4)

tuple2 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. "+" 號用於組合多個元組物件成一個元組物件
2. "*" 號用於重複元組中元素

元組運算例子如下所示：

In [35]:
```
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)

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

# 組合多個元組物件成一個元組
tuple3 = tuple1 + tuple2
print(f'元組1 + 元組2 = {tuple3}\n')

# 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}\n')
```

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

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看看元素是否在元組中
2) 比較兩個元組的內容是否相同：
   (1) 兩個元組中第一個元素先比較大小
   (2) 若相等，繼續比較兩個元組中第二個元素
   (3) 依此類推
   (4) 最後都相等 ，也就是
      (1) 兩個元組長度
      (2) 兩個元組中每個元素的值

   都相等的話，則兩個元組內容相同

元組物件的比較運算，例子如下：

In [36]:
```
tuple1 = (1, 2, 3)
tuple2 = (1, 2, 4)

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

print(f'3 in {tuple1}: {3 in tuple1}\n')
print(f'3 in {tuple2}: {3 in 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)

3 in (1, 2, 3): True

3 in (1, 2, 4): False

(1, 2, 3) 小於 (1, 2, 4)
```

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 [37]:
```
numbers = (9, 8, 1, 2, 4, 7)
print(f'數字元组: {numbers}\n')

print(f'1. 元组長度: {len(numbers)}\n')

print(f'2. 元组中最大元素: {max(numbers)}\n')

print(f'3. 元组中最小元素: {min(numbers)}\n')

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}\n')
print(f'   元组轉列表: {list(courses_tuple)}\n')
```

```
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 (list) 轉換元组成字串，元组元素之間用 "string" 隔開
4	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 [38]:
```
numbers = (1, 9, 8, 1, 2, 4, 7, 1)
print(f'數字列表: {numbers}\n')

print(f'1. 列表中 1 出現次數: {numbers.count(item_new)}\n')

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. 元组轉字串: {course_str}\n')

# 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}\n')
```

```
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) 、列表及元組物件類型， 我們學習過了。

這裡我們將學習第4類類型中的集合物件。

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，重複元素會被忽略
4) 集合中元素不能依索引直接訪問或截取
5) 集合中元素不可修改或增減。但是可以有集合運算。
6) 空集合必須用 set() 而不是 {}，因為 {} 是表示一個空字典（將介紹於後）。

集合物件例子如下：

In [39]:
```
# 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(f'basket 物件類型: {type(basket)}\n')
print(f'集合物件 basket = {basket}\n')
```

```
basket 物件類型: <class 'set'>

集合物件 basket = {'梨子', '蘋果', '香蕉', '橘子'}
```

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 [40]:
```
# 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'cs_courses: {cs_courses}\n')
print(f'art_courses: {art_courses}\n')

# a-b: 集合a中包含而集合b中不包含的元素
print(f'cs-art:\t {cs_courses - art_courses}\n')

# a|b: 集合a或b中包含的所有元素
print(f'ca|art:\t {cs_courses | art_courses}\n')

# a&b: 集合a和b中都包含了的元素
print(f'cs&art:\t {cs_courses & art_courses}\n')

# a^b: 不同時包含於集合a和b的元素
print(f'cs^art:\t {cs_courses ^ art_courses}\n')
```

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

art_courses: {'歷史', '數學', '設計', '藝術'}

cs-art:  {'電腦', '物理'}

ca|art:  {'藝術', '設計', '電腦', '物理', '歷史', '數學'}

cs&art:  {'歷史', '數學'}

cs^art:  {'藝術', '設計', '電腦', '物理'}
```

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判斷元素是否在集合中
2) 比較兩個集合的內容是否相同，常用的判斷法有二：
   (1) 兩個集合a, b，若是 a&b = a|b，
   則兩集合相同
   (2) 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

語法

```
x in s
```

其中：
1) x 是元素或變數，s 是一個集合
2) 判斷元素 x 是否在集合 s 中，
   (1) 若是存在返回 True
   (2) 若是不存在返回 False

集合比較運算，例子如下：

In [41]:
```
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？\n答： {target in c

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

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

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

# 集合交集
intersection = cs_courses & art_courses

# 集合聯集
```

```
union = cs_courses | art_courses

if intersection == union:
    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')

new_courses = {'歷史', '數學', '藝術', '設計'}
# 集合交集
intersection = new_courses & art_courses

# 集合聯集
union = new_courses | art_courses

if intersection == union:
    print(f'new_courses 與 art_courses 內容相同\n')
else:
    print(f'new_courses 與 art_courses 內容不同\n')
```

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

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

cs_courses: {'歷史', '電腦', '數學', '物理'}

art_courses: {'歷史', '數學', '設計', '藝術'}

cs_courses 與 art_courses 內容不同

new_courses 與 art_courses 內容相同
```

集合相關函式

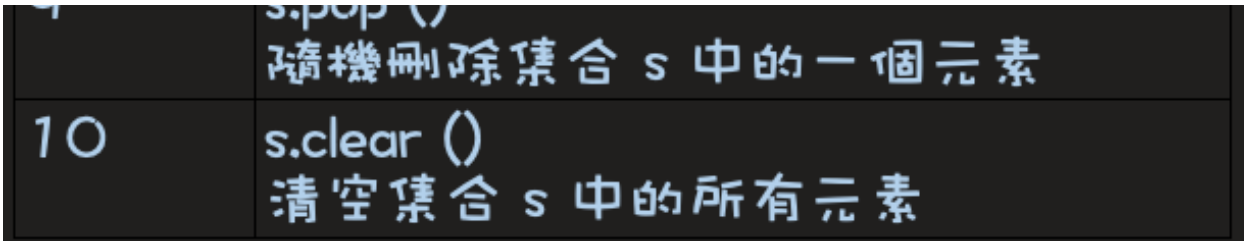
集合物件的相關函式，例子如下：

In [42]:
```
numbers = {9, 8, 1, 2, 4, 7}
print(f'數字集合: {numbers}\n')

print(f'1. 集合長度: {len(numbers)}\n')

print(f'2. 集合中最大元素: {max(numbers)}\n')

print(f'3. 集合中最小元素: {min(numbers)}\n')

print(f'4. 集合總計: {sum(numbers)}\n')


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}\n')
print(f'課程元組: {courses_tuple}\n')

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

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

```
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: {'歷史', '電腦', '物理'}

   集合轉列表: ['歷史', '電腦', '物理']

   元組轉集合: {'歷史', '電腦', '物理'}

   集合轉元組: ('歷史', '電腦', '物理')
```

集合相關方法

編號	集合相關常用的方法
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 ()



集合物件的相關方法，例子如下：

In [43]:
```
# 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}\n')
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 cs_courses: {cs_courses}\n')

art_courses.update(['書法', '繪畫'], (1, 2, 3))
print(f'2. 加數學到藝術課程集合 art_courses: {art_courses}\n')

# 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. cs-art:\t {difference}\n')

# a&b: 集合a和b中都包含了的元素
intersection = cs_courses.intersection(art_courses)
print(f'4. cs&art:\t {intersection}\n')

# a|b: 集合a或b中包含的所有元素
union = cs_courses.union(art_courses)
print(f'5. cs|art:\t {union}\n')

# a^b: 不同時包含於集合a和b的元素
print(f'   cs^art:\t {union - intersection}\n')

basket1 = {'橘子', '蘋果', '梨子', '香蕉'}
basket2 = {'梨子', '橘子'}


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

print(f'---> 問：basket2: {basket2} 是 basket1: {basket1} 的子集嗎？')
```

```
print(f'---> 答：{basket2.issubset(basket1)}\n')

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}\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}\n')

# 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 ()')
# 隨機刪除集合 s 中的一個元素
print(f'挑掉 basket1 中的 {basket1.pop()}: {basket1}\n')

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

```
電腦課程集合 cs_courses: {'歷史', '電腦', '數學', '物理'}

藝術課程集合 art_courses: {'歷史', '數學', '設計', '藝術'}

1. 加數學到電腦課程集合 cs_courses: {'歷史', '電腦', '數學', '物理'}

2. 加數學到藝術課程集合 art_courses: {'書法', 1, 2, 3, '藝術', '繪畫', '設
計', '歷史', '數學'}

3. cs-art:       {'電腦', '物理'}

4. cs&art:       {'歷史', '數學'}

5. cs|art:       {'書法', 1, 2, 3, '藝術', '繪畫', '設計', '電腦', '物理'
, '歷史', '數學'}

   cs^art:       {'書法', 1, 2, 3, '繪畫', '設計', '電腦', '物理', '藝術'
}

6. a.issubset (b)
---> 問：basket1: {'梨子', '蘋果', '香蕉', '橘子'} 是 basket2: {'梨子', '
橘子'} 的子集嗎？
---> 答：False
```

```
---> 問：basket2: {'梨子', '橘子'} 是 basket1: {'梨子', '蘋果', '香蕉', '
橘子'} 的子集嗎？
---> 答：True

7. s.discard (x)
移除 basket2 中的 蘋果: {'梨子', '橘子'}

8. s.remove ( x )
移除 basket2 中的 橘子: {'梨子'}

9. s.pop ()
挑掉 basket1 中的 梨子: {'蘋果', '香蕉', '橘子'}

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：

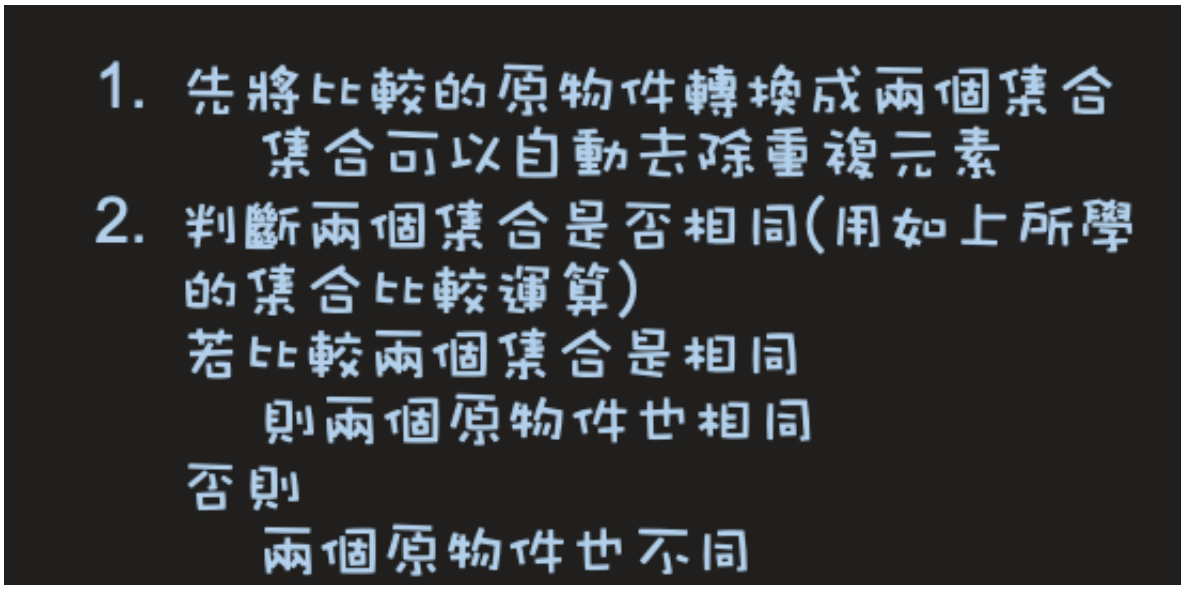
In [44]:
```
cs_courses = {'歷史', '數學', '物理', '電腦'}

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

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

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_course
    print(f'new_courses 與 art_courses 內容相同\n')
else:
    print(f'new_courses 與 art_courses 內容不同\n')
```

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

art_courses: {'歷史', '數學', '設計', '藝術'}

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 [45]:
```
tuple1 = (1, 2, 3)
tuple2 = (1, 2, 3, 1)

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

# 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(用如上所學的集合比較運算)
intersection = set1 & set2
union = set1 | set2

if intersection == union:
    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) 內容相同
```

4.4 字典 (Dictionary)

我們在前面有提到五種常用的物件，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 [46]:
```
student = {'name': '尤勇',
           'age': 12,
           'courses': ['數學', '電腦']}

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

# 訪問字典物件
print(f'---> 鍵=\'name\': {student["name"]}\n')
print(f'---> 所有鍵列表: {student.keys()}\n')
print(f'---> 所有值列表: {student.values()}\n')
print(f'---> 所有鍵值對列表: {student.items()}\n')
print(f'---> 項目個數 = {len(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，列印如下：')
for key, value in student.items():
    print(f'---> {key}:\t {value}')
```

```
student 物件類型: <class 'dict'>

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

---> 鍵='name': 尤勇

---> 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), ('course
s', ['數學', '電腦'])])

---> 項目個數 = 3

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

In [47]:
```
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}\n')
```

```
# 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.\n')
else:
    print(f'---> 鍵={key} 值={value}\n')

key = 'phone'
value = student.get(key)
if value is None:
    print(f'---> 鍵={key} Not Found.\n')
else:
    print(f'---> 鍵={key} 值={value}\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'---> 鍵={key} 更改值={value}\n')
print(f'字典物件: {student}\n')

# 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'字典物件: {student}\n')
```

```
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

---> 鍵=name 更改值=夏琪

字典物件: {'name': '夏琪', 'age': 12, 'courses': ['數學', '電腦'], 'phon
e': '0911-697-369'}

字典物件: {'name': '小叮噹', 'age': 6, 'courses': ['數學', '電腦'], 'pho
ne': '888-8888'}
```

In [48]:
```
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
del student['courses']
print(f'移除字典中的一個項目: key=courses\n---> {student}\n')

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

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

移除字典中的一個項目: key=courses
---> {'name': '尤勇', 'age': 12}

移除字典中的一個項目: 'age':12
---> {'name': '尤勇'}
```

問：複雜的物件如字典物件，在比較條件敍述中如何判斷其真偽？

答：Python 基於不同的物件的特性，對於判斷比較物件的偽值也有異。

以下條件敍述皆判斷為布林偽值：

1. False 關鍵字
2. None 關鍵字
3. 數字運算結果為零
4. 任何空的序列物件
   如空字串、空的列表、空的集合、空的字典等

條件敍述皆判斷為布林偽值，例子如下：

In [49]:
```
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"\n')
    else:
        print(f'---> "{cond}" : Python 判為偽 "Faslse"\n')
```

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

---> "False" : Python 判為偽 "Faslse"

---> "None" : Python 判為偽 "Faslse"

---> "0" : Python 判為偽 "Faslse"

---> "10" : Python 判為真 "True"

---> "" : Python 判為偽 "Faslse"

---> "Test" : Python 判為真 "True"

---> "()" : Python 判為偽 "Faslse"

---> "[]" : Python 判為偽 "Faslse"

---> "{}" : Python 判為偽 "Faslse"
```

第5章 自訂函式 (Functions)

函式（Function）的應用非常廣泛，前面章節中我們已經學習過多個函式，比如：input() 、range()、len() 等函式，這些都是 Python 的內建函式，可以直接使用。

除了可以直接使用的內建函式外，Python 還讓我們自訂函式，也就是說讓我們自己把一段可重複使用的程式定義成函式，以便一次編寫，而後可以多次調用，也可以供他人使用。自利又利他，很棒吧！

下面就來學習如何自訂函式。

介绍「定義函式」指令：def statement

語法

```
def <函式名稱> (<參數列>):

        <程式區塊>

        return <回值>
```

其中:

1) def 意思是 define（定義）宣告一個函式的開始
2) 函式名稱，命名規則與變數命名規則相同
3) 參數列中每一個參數都是變數，兩個參數間須以逗號 ‘，’ 分開
4) 參數種類又分：位置參數和關鍵字參數兩種
5) 位置參數只有變數名稱，依參數列的位置順序排列
6) 關鍵字（keyword）參數就是 '<變數>=<預設值>' 的型式定義
7) 程式區塊必須至少一個縮格開始定義
8) 函式結束前，Python 讓我們用 return 返回指令，返回到呼叫程式
9) 返回時，也可以視需求帶回一個回值物件（returned object）

注意

1) 若參數列含有位置參數和關鍵字參數兩種時，所有位置參數必須放在關鍵字參數之前
2) 注意保留字 def 那一行最後的字元必須是「：」

自訂函式例子如下：

In [50]:
```
# 例子1：自訂函式一個求兩整數的最大公約數

def gcd (a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

print(f'gcd(3,5) = {gcd(3, 5)}')
print(f'gcd(30,50) = {gcd(30, 50)}')
print(f'gcd(5,50) = {gcd(5, 50)}')
```

```
gcd(3,5) = 1
gcd(30,50) = 10
gcd(5,50) = 5
```

In [51]:
```
# 例子2：自訂函式

# 位置參數必須放在關鍵字參數之前
def hello_func(greeting, name = 'You'):
    return f'{greeting}, {name}'

# location of function
print(type(hello_func))
print(hello_func) # 沒有小括號，Python 只印函式定義，不會執行函式

# 呼叫並執行函式，省掉一個有預設值的參數
print(hello_func('\t 哈囉'))

# 呼叫並執行函式，二個參數
print(hello_func('\t 哈囉', name='尤勇'))
```

```
<class 'function'>
<function hello_func at 0x7fc0aeee3dc0>
        哈囉, You
        哈囉, 尤勇
```

In [52]:
```
# 例子3：自訂函式列印學員資料

def student_info(*args, **kwargs):
    print('Student info listed below:')
    print('\t', args)
    print('\t', kwargs)

# Notice that "=" b/t keyword and value when calling the function
#    but the function returns key-value paors with ":" as a dictionary
student_info('Math', 'Art', name='John', age=22)

courses = ['Math', 'Art']
info = {'name': 'John', 'age': 22}

student_info(*courses, **info)
```

```
Student info listed below:
        ('Math', 'Art')
        {'name': 'John', 'age': 22}
Student info listed below:
        ('Math', 'Art')
        {'name': 'John', 'age': 22}
```

In [53]:
```
# 例子4：自訂函式判斷閏年月與否

# 每個月天數
```

```
month_days = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30 ,31]
#                1   2   3   4   5   6   7   8   9  10  11  12

def is_leap(year):
    """ 閏年回覆為真，否則為偽
    Return True for leap years,
    False for non-leap year."""

    return year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)

def days_in_month(year, month):
    """由年月份判斷該月份有幾天
    Return number of days, given a year and a month."""

    # checking valid month
    if not 1 <= month <= 12:
        return '非正常月份'

    # checking if it is a leap year
    if month == 2 and is_leap(year):
        return 29

    return month_days[month]

year = 2017
month = 8
print(f'問：Year {year} 閏年嗎?', is_leap(year))
print(f'答：{year} 年 {month} 月有: {days_in_month(year, month)} days\n

year = 2000
month = 2
print(f'問：Year {year} 閏年嗎?', is_leap(year))
print(f'答：{year} 年 {month} 月有: {days_in_month(year, month)} days\n

year = 2022
month = 2
print(f'問：Year {year} 閏年嗎?', is_leap(year))
print(f'答：{year} 年 {month} 月有: {days_in_month(year, month)} days\n
```

```
問：Year 2017 閏年嗎? False
答：2017 年 8 月有: 31 days

問：Year 2000 閏年嗎? True
答：2000 年 2 月有: 29 days

問：Year 2022 閏年嗎? False
答：2022 年 2 月有: 28 days
```

第6章 模塊與變數 (Modules and Variables)

介紹模塊（Module）

為了可以呼叫別人寫好的程式，Python 設計模塊的觀念，定義所有副檔名為 (.py) 的檔案都是可以被搜索的模塊並可以被載入到我們的程式裡來呼叫。

Python 模塊的架構讓人們可以方便疊床架屋，使得程式可以像蓋房子砌牆一樣從簡單的一磚一瓦一棟樑地建起一棟高樓，理論上可以完成一套複雜系統。

首先我們來學習如何載入模塊。

介绍「進口」指令：import statement

語法

```
import <模塊名稱>
```

Python 進口指令會依以下順序規則搜索檔案，名字叫'<模塊名稱>.py'：

1. 目前執行我們程式的資料夾（目錄）中搜索
2. 若還找不到，從環境變數 PYTHONPATH 定義的目錄列中搜索
   PYTHONPATH
   原理跟一般電腦 PATH 環境變數用來搜索執行檔 (.exe)類似
3. 若還找不到，從 Python 系統內建目錄列中搜索
4. 若還找不到，Python 列印出錯誤信息並停止執行

最後若 Python 搜索模塊一旦成功，Python 會幫忙載入。我們就可以呼叫模塊裡面的所有函式功能。

進口指令例子如下：

In [54]:
```
# 輸入系統模組 'sys'
import sys

# 系統模組中有變數名稱, version, 提供 Python 版本資訊
print(f'Python sys.version:\n{sys.version}\n')


# 系統模組中有變數名稱, executable, 提供 Python 執行程式所在的資料夾位置
print(f'Python sys.executable:\n{sys.executable}\n')
```

```
Python sys.version:
3.9.7 (default, Sep 16 2021, 08:50:36)
[Clang 10.0.0 ]

Python sys.executable:
/Users/michaelkao/opt/anaconda3/bin/python
```

問：Python 搜索模塊好像很複雜！可以簡單看到搜索的目錄列嗎？

答：可以。

Python 提供系統模组中，把搜索目錄列存到 sys.path 列表物件（list）裡。

檢視搜索目錄列，例子如下：

In [55]:
```
import sys

print(f'sys.path 物件類型: {type(sys.path)}\n')

print('Python 搜索的目錄列:')
print(sys.path)
```

```
sys.path 物件類型: <class 'list'>

Python 搜索的目錄列:
['/Users/michaelkao/Library/CloudStorage/OneDrive-Personal/My Projects/Python', '/Users/michaelkao/opt/anaconda3/lib/python39.zip', '/Users/michaelkao/opt/anaconda3/lib/python3.9', '/Users/michaelkao/opt/anaconda3/lib/python3.9/lib-dynload', '', '/Users/michaelkao/opt/anaconda3/lib/python3.9/site-packages', '/Users/michaelkao/opt/anaconda3/lib/python3.9/site-packages/aeosa', '/Users/michaelkao/opt/anaconda3/lib/python3.9/site-packages/locket-0.2.1-py3.9.egg', '/Users/michaelkao/opt/anaconda3/lib/python3.9/site-packages/IPython/extensions', '/Users/michaelkao/.ipython']
```

複習變數命名規則：

前面學習過的指派指令中，知道變數命名規則，現在複習一下：

1. 變數（variable）的名稱可以是任何大小寫的英文字母或下底線（_）或者數字的組成。
2. 變數名稱開頭不可以是數字。

除名稱外，變數在程式中也有範圍限制。

變數的有效範圍

Python 程式中，當一個變數名稱第一次出現時，我們可以想像成 Python 必須有一張變數名稱表上登記該變數及其相關物件資料，包括記憶體儲存地址等。這一張變數名稱表 Python 叫它為 Variable Scope （變數範圍）。而且一個模組或函式都至少有一張變數範圍。

若在不同的模組或函式中有同名的變數被重覆定義時，Python 就得依底下四種變數範圍規則來認定變數的有效範圍。這四種範圍規則依序如下：

1) 局域變數 (Local variable) 範圍
局域變數只存在某一段程式區域中有效。例如上面學習過的定義函式指令中，參數列中的變數都是局域變數。
2) 外圍變數 (Enclosing variable) 範圍
外圍變數定義在外面縮格層的程式區塊中而內層程式區塊仍能有效訪問。內層變數必須用 'nonlocal' 來宣告。
3) 全域變數 (Global variable) 範圍
全域變數相對於上面的局域變數在整個程式中都有效。變數必須用 'global' 來宣告。
4) 內建變數 (Built-in variable) 範圍
Python 系統內建在 builtin 模組內的變數。

當一個變數須要被訪問時，Python 依上面 LEGB 規則順序搜索變數的定義。若還找不到變數的定義範圍的話，Python 列印出錯誤信息並停止執行。

問：可以檢查變數是屬那一種範圍嗎？

答：可以。

Python 提供兩個函式 locals () 和 globals ()

1) 'locals()' 返回一個局域字典供查詢變數是否在局域範圍
2) 'global()' 返回一個全域字典供查詢變數是否在全域範圍

變數有效範圍例子如下：

In [56]:
```
# 局域和全域變數範圍的例子
def test():

    # 宣告全域變數 y
    print('\t 宣告全域變數 y')
    global y

    # 指派局域變數 x
    x = 'test()指派的局域變數 x'
    print(f'\t x = \'{x}\'')
```

```
    print(f' \t x = \'{x}\'')

    y = 'test()指派的全域變數 y'
    print(f'\t y = \'{y}\'')

    # 檢查變數有效範圍
    if 'x' in locals():
        print('\t test()中的 x 是局域變數')

    if 'y' in globals():
        print('\t test()中的 y 是全域變數\n')


print('主程式開始...')

# 指派全域變數 x
x = '主程式指派的全域變數 x'
print(f'x = \'{x}\'\n')

print(f'呼叫 test()中...')
test()

print('返回主程式...')
print(f'y = \'{y}\'')

if 'y' in globals():
    print(f'主程式中的 y 是全域變數\n')
```

```
主程式開始...
x = '主程式指派的全域變數 x'

呼叫 test()中...
         宣告全域變數 y
         x = 'test()指派的局域變數 x'
         y = 'test()指派的全域變數 y'
         test()中的 x 是局域變數
         test()中的 y 是全域變數

返回主程式...
y = 'test()指派的全域變數 y'
主程式中的 y 是全域變數
```

In [57]:
```
# 外圍變數範圍的例子

# 外層函式
def outer():
    print('\t 在 outer() 外圍範圍...')
    x = '指派外圍變數 x'
    y = '指派外圍變數 y'

    print(f'\t x = \'{x}\'')
    print(f'\t y = \'{y}\'\n')

    # 內層函式
    def inner():
        print('\t\t 在 inner() 局域範圍...')

        x = 'inner() 指派局域變數 x'
        print(f'\t\t x = \'{x}\'')

        # 宣告外圍變數 y
        print('\t\t 宣告外圍變數 y')
        nonlocal y
        y = 'inner() 指派外圍變數 y'
        print(f'\t\t y = \'{y}\'\n')

    print('\t 呼叫 inner()...')
    inner()
    print('\t 返回 outer() 外圍範圍...')
    print(f'\t x = \'{x}\'')
    print(f'\t y = \'{y}\'')

# 主程式
print('在主程式模組範圍...')
outer()
```

```
在主程式模組範圍...
         在 outer() 外圍範圍...
         x = '指派外圍變數 x'
         y = '指派外圍變數 y'

         呼叫 inner()...
                 在 inner() 局域範圍...
                 x = 'inner() 指派局域變數 x'
                 宣告外圍變數 y
                 y = 'inner() 指派外圍變數 y'

         返回 outer() 外圍範圍...
         x = '指派外圍變數 x'
         y = 'inner() 指派外圍變數 y'
```

In [58]:
```
# 內建變數範圍的例子
import builtins

print(f'\'builtins\' 模組中內建變數 (屬性和方法)如下:\n{dir(builtins)}\n')

list_1 = [5, 1, 4, -8, 9]

num_min = min(list_1)
print(f'例子1: min({list_1}) =\t {num_min}')
print(f'例子2: abs({num_min}) =\t {abs(num_min)}')
```

```
'builtins' 模組中內建變數 (屬性和方法)如下:
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseExceptio
n', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarnin
g', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning
', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'Future
Warning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning',
'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryEr
ror', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError',
'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'No
tImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'Pe
ndingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'R
ecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError',
'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError
', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'Timeout
Error', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError
', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'Un
icodeWarning', 'UserWarning', 'ValueError', 'Warning', 'ZeroDivisionE
rror', '__IPYTHON__', '__build_class__', '__debug__', '__doc__', '__i
mport__', '__loader__', '__name__', '__package__', '__spec__', 'abs',
'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray', 'byt
es', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyrig
ht', 'credits', 'delattr', 'dict', 'dir', 'display', 'divmod', 'enume
rate', 'eval', 'exec', 'execfile', 'filter', 'float', 'format', 'froz
enset', 'get_ipython', 'getattr', 'globals', 'hasattr', 'hash', 'help
', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', '
len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min',
'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', '
range', 'repr', 'reversed', 'round', 'runfile', 'set', 'setattr', 'sl
ice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type
', 'vars', 'zip']

例子1: min([5, 1, 4, -8, 9]) =    -8
例子2: abs(-8) =  8
```

第7章 自訂類別及自訂物件 (Classes and Instances)

Python 也是基於物件導向（Object Oriented）的程式語言之一。

所有能描述能處理的物件（Object）Python 設計下就是某一個類別（Class）的實例（Instance）。

前面我們學習過 Python 的每一個常用物件都是 Python 系統內建物件類別。

同理 Python 也讓我們也可以在程式中量身打造自己的類別和適合自己應用的物件。並依照自訂的類別建立實體化（instantiate）的實例（instance)物件。

這個時候建立起來的實例物件 Python 會載入到電腦記憶體中某一個位址，及佔有某一塊資料物件空間。讓我們簡單在程式中用變數來訪問這個物件或來運算處理。

當我們設計一套程式，除輸入（import）一些我們需要且Python 內建或別人設定好的類別（class）以外，自己還可依自己的應用自訂一些類別，來建構自己的模型。

譬如我們試做一個學校模型：

1）學校有三種人員：學員、教員、職員
2）每一個人員都有姓名、生日及電郵
3）每一個學員還有不同班別、學習不同的課程及其成績
4）每一個教員有不同的教育課程及指導班級
5）每一個職員有不同的職務及職稱

那麼我們該如何用 Python 建立一個物件導向的模型呢？

1）每一個人員都有姓名、生日及電郵
　　建議：可以訂一個 Person 類別（人員類別）

2）每一個學員還有不同班別、學習不同的課程及其成績
　　建議：可以訂一個 Person 類別下的 Student 次類別（學員類別）
　　而每一位學生都是屬於學員類別的一個物件（實例）。

3）每一個教員有不同的教育課程及指導班級
　　建議：可以訂一個 Person 類別下的 Teacher 次類別（教員類別）

4）每一個職員有不同的職務及職稱
　　建議：可以訂一個 Person 類別下的 Staff 次類別（職員類別）

自訂類別模型如下：

主類別	屬性	次類別	屬性
Person 人員類別			
	last_name 姓		
	first_name 名		
	birth_date 出生年-月-日		
	email 電郵		
		Student 學員類別	
			class_id 學生所屬班級
			courses {course : score} 學生所選的課程及評分
		Teacher 教員類別	
			class_id 老師所輔導的班級
			class_members 老師所輔導的學生
			teaching_courses 老師所教的課程
		Saff 職員類別	
			title 職稱
			jobs 工作

自訂類別程式如下：

In [59]:
```
import datetime

# 定義人員類別
class Person:

    # 人員物件實體化 函式
    def __init__(self, last, first, dob):
        self.last_name = last
        self.first_name = first
```

```
        dob_yyyy, dob_mm, dob_dd = dob.split('-')
        self.birth_date = datetime.date(int(dob_yyyy),
                                        int(dob_mm),
                                        int(dob_dd))
    # 定義 email 屬性
    @property
    def email(self):
        return f'{self.last_name}{self.first_name}@school.edu.tw'

    # 定義 fullname 全名 屬性
    @property
    def fullname(self):
        return f'{self.last_name} {self.first_name}'

    # 定義 email 屬性
    @fullname.setter
    def fullname(self, name):
        last, first = name.split(' ')
        self.last_name = last
        self.first_name = first

# 定義學員類別
class Student(Person):

    # 學員物件實體化 函式
    def __init__(self, last, first, dob, class_id, courses=None):
        super().__init__(last, first, dob)
        self.class_id = class_id
        if courses is None:
            self.courses = {}
        else:
            self.courses = courses

    # 加入班級別 函式
    def add_classId(self, class_id):
        self.class_id = class_id

    # 加入課程和分數 函式，課程不在就新增
    def add_course(self, course, score=0):
        self.courses.update({course : score})

    # 退選課程 函式
    def remove_course(self, course):
        if course in self.courses:
            self.courses.pop(course)

    # 列印所選的課程及分數 函式
    def print_courses(self):
        print(self.fullname)
        for course, score in self.courses.item():
            print(f' -> {course}: {score}')
```

```
            print(f' ---> {course}: {score}')

# 建立兩個學生資料
student_1 = Student('尤', '勇', '2010-3-15', '6甲', {'數學':0, '電腦':0}
student_2 = Student('夏', '琪', '2011-12-20', '6乙', {'數學':0, '電腦':0

print(f'{student_1.fullname}')
print(f'---> 生日 = {student_1.birth_date}')
print(f'---> 電郵 = {student_1.email}')
print(f'---> 班級 = {student_1.class_id}')
print(f'---> 課程表 = {student_1.courses}\n')


print(f'{student_2.fullname}')
print(f'---> 生日 = {student_2.birth_date}')
print(f'---> 電郵 = {student_2.email}')
print(f'---> 班級 = {student_2.class_id}')
print(f'---> 課程表 = {student_2.courses}\n')
```

```
尤 勇
---> 生日 = 2010-03-15
---> 電郵 = 尤勇@school.edu.tw
---> 班級 = 6甲
---> 課程表 = {'數學': 0, '電腦': 0}

夏 琪
---> 生日 = 2011-12-20
---> 電郵 = 夏琪@school.edu.tw
---> 班級 = 6乙
---> 課程表 = {'數學': 0, '電腦': 0, '藝術': 0}
```

問：如何檢視類別從屬關係，尤其是別人先前建立的模型中類別從屬關係及定義內容？

答：的確，要是有原始碼就可以一目了然。

除此之外，Python 也提供一些內建函式可以查詢，列表如下：

編號	類別相關常用的函式
1	help(class) 列印出類別 class 的詳細內容，包括類別相關的方法及屬性
2	isinstance(obj, class) 判斷 obj 是否是類別 class 的實例
3	issubclass(class1, class2) 判斷類別 class1 是否是類別 class2 的次類別

我們用上面學習的學校模型（存放在 "schoolModel_v1.py" 模塊檔案中）以便用進口 import 指令載入。

檢視類別的例子如下：

In [60]:
```
import schoolModel_v1

person_1 = Person('高', '大雄', '2019-2-19')

print(f'{person_1.fullname}')
print(f'---> 生日 = {person_1.birth_date}')
print(f'---> 電郵 = {person_1.email}')

# help(class)
# 列印出類別 class 的詳細內容，包括類別相關的方法及屬性
```

```
print(help(Person))
print(help(Student))

# isinstance(obj, class)
# 判斷 obj 是否是類別 class 的實例
print(f'問：person_1 是類別 Person 的實例嗎？')
print(f'答：{isinstance(person_1, Person)}\n')

print(f'問：person_1 是類別 Student 的實例嗎？')
print(f'答：{isinstance(person_1, Student)}\n')


# issubclass(class1, class2)
# 判斷類別 class1 是否是類別 class2 的次類別
print(f'問：類別 Student 是類別 Person 的次類別嗎？')
print(f'答：{issubclass(Student, Person)}\n')

print(f'問：類別 Person 是類別 Student 的次類別嗎？')
print(f'答：{issubclass(Person, Student)}\n')
```

```
高 大雄
---> 生日 = 2019-02-19
---> 電郵 = 高大雄@school.edu.tw
Help on class Person in module __main__:

class Person(builtins.object)
 |  Person(last, first, dob)
 |
 |  Methods defined here:
 |
 |  __init__(self, last, first, dob)
 |      Initialize self.  See help(type(self)) for accurate signature
.
 |
 |  ----------------------------------------------------------------------
-----
 |  Readonly properties defined here:
 |
 |  email
 |
 |  ----------------------------------------------------------------------
-----
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

```
 |
 |  fullname

None
Help on class Student in module __main__:

class Student(Person)
 |  Student(last, first, dob, class_id, courses=None)
 |
 |  Method resolution order:
 |      Student
 |      Person
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, last, first, dob, class_id, courses=None)
 |      Initialize self.  See help(type(self)) for accurate signature
.
 |
 |  add_classId(self, class_id)
 |      # 加入班級別 函式
 |
 |  add_course(self, course, score=0)
 |      # 加入課程和分數 函式，課程不在就新增
 |
 |  print_courses(self)
 |      # 列印所選的課程及分數 函式
 |
 |  remove_course(self, course)
 |      # 退選課程 函式
 |
 |  ----------------------------------------------------------------------
-----
 |  Readonly properties inherited from Person:
 |
 |  email
 |
 |  ----------------------------------------------------------------------
-----
 |  Data descriptors inherited from Person:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  fullname
```

```
None
問：person_1 是類別 Person 的實例嗎？
答：True

問：person_1 是類別 Student 的實例嗎？
答：False

問：類別 Student 是類別 Person 的次類別嗎？
答：True

問：類別 Person 是類別 Student 的次類別嗎？
答：False
```

學校模型續篇1

上面學校模型實作剩下教職員工的類別，尚未完成。

這裡我們繼續定義教員次類別，並且存到 "schoolModel_v2.py"檔案，程式定義如下：

In [61]:
```
import schoolModel_v1

# 定義教員類別
class Teacher(Person):

    # 教員物件實體化 函式
    def __init__(self, last, first, dob, class_id, members=None, cours
        super().__init__(last, first, dob)
        self.class_id = class_id

        # 若不是班導師，學生列表為空
        if members is None:
            self.class_members = []
        else:
            self.class_members = members

        if courses is None:
            self.teaching_courses = []
        else:
            self.teaching_courses = courses

    # 加入班上學生 函式
    def add_member(self, member):
        if member not in self.class_members:
            self.class_members.append(member)
```

```
    # 移除班上學生 函式
    def remove_member(self, member):
        if member in self.class_members:
            self.employees.remove(member)

    # 列印班上學生 函式
    def print_members(self):
        print(f'{self.class_id}班導師 {self.fullname} 的學生名單:')
        for member in self.class_members:
            print('-->', member.fullname)
        print()

    # 加入授課課程 函式
    def add_course(self, course):
        if course not in self.teaching_courses:
            self.teaching_courses.append(course)

    # 移除授課課程 函式
    def remove_course(self, course):
        if course in self.teaching_courses:
            self.teaching_courses.remove(member)

    # 列印授課課程 函式
    def print_courses(self):
        print(f'{self.fullname} 授課列表: {teaching_courses}\n')

# 建立兩個教員資料
teacher_1 = Teacher('高', '大雄', '1992-2-19', '6甲', courses=['數學', '
teacher_2 = Teacher('小', '叮噹', '1983-7-22', '6乙')
teacher_2.add_course('藝術')
teacher_2.add_course('設計')

# 建立五個學生資料及班導師
student_1 = Student('尤', '勇', '2010-3-15', '6甲', {'數學':0, '電腦':0}
teacher_1.add_member(student_1)

student_2 = Student('夏', '琪', '2011-12-20', '6乙', {'數學':0, '電腦':0
teacher_2.add_member(student_2)

student_3 = Student('王', '志明', '2010-3-15', '6甲', {'數學':0})
teacher_1.add_member(student_3)

student_4 = Student('李', '春嬌', '2010-2-5', '6甲', {'電腦':0})
teacher_1.add_member(student_4)


student_5 = Student('蔡', '小英', '2010-8-8', '6乙', {'設計':0, '藝術':0]
teacher_2.add_member(student_5)

# 列印兩個班導師資料
print(f'{teacher_1.fullname}')
```

```
print(f'{teacher_1.fullname}')
print(f'---> 生日 = {teacher_1.birth_date}')
print(f'---> 電郵 = {teacher_1.email}')
print(f'---> 班級 = {teacher_1.class_id}')
print(f'---> 教授課程 = {teacher_1.teaching_courses}\n')
teacher_1.print_members()

print(f'{teacher_2.fullname}')
print(f'---> 生日 = {teacher_2.birth_date}')
print(f'---> 電郵 = {teacher_2.email}')
print(f'---> 班級 = {teacher_2.class_id}')
print(f'---> 教授課程表 = {teacher_2.teaching_courses}\n')
teacher_2.print_members()
```

```
高 大雄
---> 生日 = 1992-02-19
---> 電郵 = 高大雄@school.edu.tw
---> 班級 = 6甲
---> 教授課程 = ['數學', '電腦']

6甲班導師 高 大雄 的學生名單:
--> 尤 勇
--> 王 志明
--> 李 春嬌

小 叮噹
---> 生日 = 1983-07-22
---> 電郵 = 小叮噹@school.edu.tw
---> 班級 = 6乙
---> 教授課程表 = ['藝術', '設計']

6乙班導師 小 叮噹 的學生名單:
--> 夏 琪
--> 蔡 小英
```

問：期中考後，學生成績如何處理？

答：問得好。

> 有同學知道學員類別那個方法可以用嗎？
>
> 記得學員類別中的 courses 課程，我們特別設計成字典物件。
> 也就是說，每一學生的課程有鍵值對 (key-value pair)，如下：

In [62]:
```
import datetime
```

```
# 定義人員類別
class Person:

    # 人員物件實體化 函式
    def __init__(self, last, first, dob):
        self.last_name = last
        self.first_name = first
        dob_yyyy, dob_mm, dob_dd = dob.split('-')
        self.birth_date = datetime.date(int(dob_yyyy),
                                        int(dob_mm),
                                        int(dob_dd))
    # 定義 email 屬性

    @property
    def email(self):
        return f'{self.last_name}{self.first_name}@school.edu.tw'

    # 定義 fullname 全名 屬性
    @property
    def fullname(self):
        return f'{self.last_name} {self.first_name}'

    # 定義 email 屬性
    @fullname.setter
    def fullname(self, name):
        last, first = name.split(' ')
        self.last_name = last
        self.first_name = first

# 定義學員類別
class Student(Person):

    # 學員物件實體化 函式
    def __init__(self, last, first, dob, class_id, courses=None):
        super().__init__(last, first, dob)
        self.class_id = class_id
        if courses is None:
            self.courses = {}
        else:
            self.courses = courses

    # 加入班級別 函式
    def add_classId(self, class_id):
        self.class_id = class_id

    # 加入課程和分數 函式，課程不在就新增
    def add_course(self, course, score=0):
        self.courses.update({course: score})

    # 退選課程 函式
```

```
    def remove_course(self, course):
        if course in self.courses:
            self.courses.pop(course)

    # 列印所選的課程及分數 函式
    def print_courses(self):
        print(self.fullname)
        for course, score in self.courses.items():
            print(f'--> {course}: {score}')

# 定義教員類別
class Teacher(Person):

    # 教員物件實體化 函式
    def __init__(self, last, first, dob, class_id, members=None, cours
        super().__init__(last, first, dob)
        self.class_id = class_id

        # 若不是班導師，學生列表為空
        if members is None:
            self.class_members = []
        else:
            self.class_members = members

        if courses is None:
            self.teaching_courses = []
        else:
            self.teaching_courses = courses

    # 加入班上學生 函式
    def add_member(self, member):
        if member not in self.class_members:
            self.class_members.append(member)

    # 移除班上學生 函式
    def remove_member(self, member):
        if member in self.class_members:
            self.employees.remove(member)

    # 列印班上學生 函式
    def print_members(self):
        print(f'{self.class_id}班導師 {self.fullname} 的學生名單:')
        for member in self.class_members:
            print('-->', member.fullname)
        print()

    # 加入授課課程 函式
    def add_course(self, course):
        if course not in self.teaching_courses:
            self.teaching_courses.append(course)
```

```
    # 移除授課課程 函式
    def remove_course(self, course):
        if course in self.teaching_courses:
            self.employees.remove(course)

    # 列印授課課程 函式
    def print_courses(self):
        print(f'{self.fullname} 授課列表: {self.teaching_courses}\n')

# 建立6甲班上學生資料
student_1 = Student('尤', '勇', '2010-3-15', '6甲', {'數學':0, '電腦':0}
student_2 = Student('王', '志明', '2010-3-15', '6甲', {'數學':0})
student_3 = Student('李', '春嬌', '2010-2-5', '6甲', {'電腦':0})

# 建立6甲班上學生考試成績
student_1.add_course('數學', 89)
student_1.add_course('電腦', 100)

student_2.add_course('數學', 99)

student_3.add_course('電腦', 95)

print('6甲班上學生考試成績')
student_1.print_courses()
print()

student_2.print_courses()
print()

student_3.print_courses()
print()
```

```
6甲班上學生考試成績
尤 勇
--> 數學: 89
--> 電腦: 100

王 志明
--> 數學: 99

李 春嬌
--> 電腦: 95
```

學校模型續篇2

上面學校模型實作剩下職員的類別，尚未完成。

最後我們繼續定義職員次類別，並且存到 "schoolModel_v3.py"檔案。程式定義如下。

另外還要介紹類別一個重要觀念，叫類別及實例變數。

介紹類別及實例變數 (Class and Instance Variables)

類別變數（class variable）在同一類別的所有實例中共享相同的值。

實例變數（instance variable）的值在同一類別的每個實例中可能含有不同值。

類別變數只有在定義類別時才能設定初值。 另一方面，實例變數可以隨時指派或更改其值。

語法

1. 類別變數
   <變數名稱> = <值或值的運算>
2. 實例變數
   self<.變數名稱> = <值或值的運算>

注意

保留字 self 後面有加句點“．”

程式實作例子中，我們加一個類別變數 head _count (人數統計)在 Person 類別上，如下：

In [63]:
```
import datetime

# 定義人員類別
class Person:

    # 類別變數初值
    head_count = 0

    # 人員物件實體化 函式
    def   init  (self, last, first, dob):
```

```
        self.last_name = last
        self.first_name = first
        dob_yyyy, dob_mm, dob_dd = dob.split('-')
        self.birth_date = datetime.date(int(dob_yyyy),
                                        int(dob_mm),
                                        int(dob_dd))
        # 實體化一次加一個人
        Person.head_count += 1

    # 定義 email 屬性

    @property
    def email(self):
        return f'{self.last_name}{self.first_name}@school.edu.tw'

    # 定義 fullname 全名 屬性
    @property
    def fullname(self):
        return f'{self.last_name} {self.first_name}'

    # 定義 email 屬性
    @fullname.setter
    def fullname(self, name):
        last, first = name.split(' ')
        self.last_name = last
        self.first_name = first

# 定義職員類別
class Staff(Person):

    # 職員物件實體化 函式
    def __init__(self, last, first, dob, title, jobs=None):
        super().__init__(last, first, dob)
        self.title = title

        if jobs is None:
            self.jobs = []
        else:
            self.jobs = jobs


# 建立兩個職員資料
staff_1 = Staff('秦', '煙投', '1958-6-7', '校長', ['學務', '沒人做的雜務'])

print(f'{staff_1.fullname}')
print(f'---> 職稱：{staff_1.title}')
print(f'---> 職務：{staff_1.jobs}\n')

# 類別變數可以繼承 (inheritance)
print(f'全校共有: {staff_1.head_count} 人\n')
```

```
staff_2 = Staff('郝', '佳再', '1988-9-9', '教務長', ['學生教務', '教師人事'

print(f'{staff_2.fullname}')
print(f'---> 職稱：{staff_2.title}')
print(f'---> 職務：{staff_2.jobs}\n')

print(f'全校共有: {Person.head_count} 人\n')
```

```
秦 煙投
---> 職稱：校長
---> 職務：['學務', '沒人做的雜務']

全校共有: 1 人

郝 佳再
---> 職稱：教務長
---> 職務：['學生教務', '教師人事']

全校共有: 2 人
```

作者後語

朋友捎來一則 Line：「Happy Valentine's Day!! (今天開學) 」方驚覺編寫已經半月有餘了。電子書改版不是難事，就此打住，版本 1.0 的『小朋友玩大蟒蛇』分享給大家。先給小朋友試試水溫吧！

請小朋友幫個忙，指出難懂的地方或改善之道，歡迎電郵分享。讓下次改版更完善，甘溫喔！

最後迴向：

願以此功德
普及於世界
疫情早遠離
人類上火星

高嘉祥

2022.2.14

mkaoy2k@gmail.com (mailto:mkaoy2k@gmail.com)