Python 基礎入門

黃芷榆、邱秉辰

January 30, 2024

1 什麼是程式語言？

簡單來說，程式語言是人類與電腦溝通的媒介。透過程式語言，編譯器會將人類輸入的程式指令轉換成電腦讀得懂的機器語言，讓電腦執行我們期望的操作。

廣義上，程式語言是一種用來定義和控制電腦程式的語言類型，依照與人類語言的相近程度可大致分爲**低階語言**和**高階語言**。

1.1 低階語言

低階語言 (low-level programming languages) 是一種**更接近電腦硬體的程式語言**，它們的語法相對於更高階的語言，更接近二進制的機器語言。這種語言對人類來說不那麼易讀，但它們允許開發者以更精確且有效率的方式控制電腦。常見的低階語言包括機器語言 (machine code) 和組合語言 (assembly language)，這些語言直接與電腦的硬體操作相關聯，提供對硬體資源的直接和全面的控制。

1.2 高階語言

高階語言 (high-level programming languages) 指的是更接近人類語言的程式語言，其語法和結構**類似於人類日常溝通**的方式。這種語言的特點是**易讀性高**，與低階語言或機器語言相比，它們使得程式開發變得更爲簡潔和方便。高階語言通常提供較多的抽象層次，這使得開發者可以更加專注於解決問題，而非關注硬體細節。一些著名的高階程式語言包括 C/C++、Python、Java 和 JavaScript。這些語言廣泛應用於軟體開發中，從網頁應用到複雜的系統軟體，都依賴於這些更爲直觀和易於學習的語言。

1.3 Python 小檔案

Python 是一個**高階程式語言**，由 Guido van Rossum 於 1991 年首度釋出，目前已經發展至 Python 3.0，擁有簡潔、易學的語法與廣泛的第三方函式庫。

2 基本輸入輸出

2.1 print() 函數

在 Python 中，基本的輸出是透過 print() 函數來實現。

```
print("Hello World") # 輸出 Hello World 後換行
print("Hello Python")
```

程式碼 1: 基本輸出、預設換行

執行程式碼 1 後，即會將字串 “Hello World” 印在螢幕上。值得注意的是，字串印出後，print() **預設會在字串結尾換行**，若我們不想在印出的字串結尾換行，可以透過括號中的 end 參數指定結尾所要放置的字元或字串，如程式碼 2 所示。

```
print("Hello World", end='~~~~!') # 輸出：Hello World~~~~!
print("Hello Python") # 會換行後再輸出嗎？
```

程式碼 2: 基本輸出、自訂結尾字串

我們也可以同時在 print() 中同時放入多個變數或字串，在括號中以逗點分隔，而輸出結果則是**預設以空格字元做分隔**，如程式碼 3 所示。

```
print('1','2','3') # 輸出：1 2 3
```

程式碼 3: 基本輸出、多個變數或字串

若我們想要指定分隔字元，則是可以運用 print() 函數中的 sep 參數。舉例來說，在程式碼 4 中，我們將 sep 參數指定爲 “|” 字元，此時，三個輸出字元間就不再會以空格，而是以我們指定的 “|” 字元作爲分隔字元。

```
print('1','2','3', sep='|') # 輸出：1|2|3
```

程式碼 4: 基本輸出、多個變數或字串、自訂分隔字元

2.2 input() 函數

在 Python 中，基本的輸入則是透過 input() 函數來實現。程式碼 5 運用 input() 函數讀取使用者所輸入的字串，將其指派給名爲 name 的變數，並進行隨後的輸出。

```python
name = input() # 程式會請使用者輸入一個字串
print("Hello,", name) # 若使用者輸入 Benson，則會輸出 Hello, Benson
```

程式碼 5: 基本輸入

3 變數宣告

在電腦科學中，變數表示的是**某個物件**的**名稱**，而某物件包含著我們需要的資訊；我們可以將變數理解成**儲存資料的容器**，而變數名稱便是我們爲容器所取的名字，也就是在容器上貼的標籤！

3.1 變數名稱的取名規則

在寫程式時，我們都會想遵循一些規則，雖然有些規則並不會影響到程式運作與否，但爲了讓我們的變數更易懂，也較不會與其他資料型別混淆，因此取變數名稱時我們有幾個規則要遵守：

1. 名稱只能包含**字母**、**數字**、**底線**
2. **不以數字**作爲變數開頭
3. **不能**和**關鍵字**重複
4. **不能**與**函數名字**重複
5. **大小寫**是**區分**開來的 (case-sensitive)

3.2 變數的定義並初始化

當我們要第一次宣告一個變數以及它的值，我們會以「變數 = 值」的式子表示，並稱之爲「定義並初始化變數」。

```python
a = 5
print(a) # 5
```

程式碼 6: 定義並初始化變數

我們可以將**等號**可以理解爲**代入**，**等號的左邊**放置我們的**變數名稱**，**等號的右邊**放置要讓此變數**代入的資訊**（數值、英文字母等等）。在程式碼 6 中，我們將「5」這個資料儲存到容器中，再將容器貼上「a」的標籤；意旨我們將「數值爲 5」這個資訊儲存到「名稱爲 a 的變數中」。

除了數字之外，我們也可以儲存不同資料型別的值給一個變數，如程式碼 7 所示。

而你可能會想問，資料型別是甚麼？它又有哪些種類呢？讓我們在下一節繼續探索吧！

```
a = "hello!"
b = " 你好！"
print(a) # hello!
print(b) # 你好！
```

程式碼 7: 儲存不同資料型別的值給一個變數

4 資料型別

程式語言最主要的資料型別有非常多，我們可以將常用的資料型別大致分類成以下幾類：

- **數字型別：**整數 (integer)、浮點數 (floating-point number)
- **布林型別：**布林值 (boolean)
- **字串型別：**字串 (string)
- **容器型別：**串列 (list)、字典 (dictionary)、元組 (tuple)

這個章節我們著重在介紹數值型別，其他型別我們會在後面的章節做詳細解說。那就讓我們繼續看下去這些資料型別有甚麼不同吧！

4.1 整數

電腦科學中縮寫成 `int`，是序列 $\{\ldots, -4, -3, -2, -1, 0, 1, 2, 3, 4, \ldots\}$ 中所有的數的統稱，包括負整數、零 (0) 與正整數。

這邊我們先來學一個簡單的方法，讓我們可以知道變數資料型態：`type`。

當我們想要知道一個變數的型態時，我們只要將我們的變數放到 `type` 後的小括號裡，像是 `type(a)`，程式就會印出變數 a 的資料型態。

```
a = 5
print(a) # 5
print(type(a)) # <class 'int'>
```

程式碼 8: 查看變數的資料型別：整數

這邊我們看到程式碼 8 印出 `<class 'int'>`，如此就代表我們放在 `type` 裡面的變數 `a` 是整數的資料型別。大家也可以試看看在小括號裡放其他變數或是數值，觀察會印出什麼結果來！

4.2 浮點數

浮點指的是帶有小數的數值，浮點運算即是小數的四則運算，常用來測量電腦運算速度。

簡單來說浮點數我們可以理解成有代小數的數值，像是 1.5、2.54、1.0 等等。

```
a = 5.6
print(a) # 5.6
print(type(a)) # <class 'float'>
```

程式碼 9: 查看變數的資料型別：浮點數

程式碼 9 印出的結果是 `<class 'float'>` ，代表變數 a 的資料型別是浮點數。

4.3 布林值

布林值是電腦用來判斷眞 (true) 與假 (false) 的資料型別。有時我們需要一個用來判斷的變數，讓我們的知道當此條件爲眞，需要執行甚麼任務。記得，True 跟 False 首字母要大寫！

```
a = True
print(a) # True
print(type(a)) # <class 'bool'>
```

程式碼 10: 查看變數的資料型別：布林值

接著我們來看看資料型別爲布林值的變數可以如何應用吧！如果今天我們要看一個人成年了沒有，可以透過程式碼 11 用他的歲數是否大於 18 歲來判斷：

```
adult = False
a = 20
if a >= 18:
    adult = True
else:
    adult = False

print(adult) # True
```

程式碼 11: 判斷某人的歲數是否大於 18 歲

4.4 不同資料型別的運算

和其他高階的程式語言不同，Python 是可以讓不同資料型別的數字直接進行運算 (C 和 C++ 都會需要寫程式的人親自做資料型別轉換，但是 Python 會自動幫你做轉換)。

如果進行運算的其中一個數值是浮點數，那我們得到的答案也會是浮點數；而且如果我們將兩個整數進行不整除的除法運算，得到的答案也會是浮點數。

```
a = 5.0
b = 2
c = 5
print("5.0 + 2:", type(a + b)) # 5.0 + 2: <class 'float'>
print("5.0 * 2:", type(a * b)) # 5.0 * 2: <class 'float'>
print("5 + 2:", type(c + b)) # 5 + 2: <class 'int'>
print("5 * 2:", type(c * b)) # 5 * 2: <class 'int'>
print("5 / 2:", type(c / b)) # 5 / 2: <class 'float'>
```

程式碼 12: 不同資料型別的運算

從程式碼 12 的結果可以再次確認，當其中一個數值是浮點數，我們得到的答案會是浮點數；如果將兩個整數進行不整除的除法運算，得到的答案也會是浮點數。

5 運算子

運算子 (operator) 是**運算式**中的元素，我們稱運算式中要進行計算的數值爲**運算元** (operand)，而運算子則可以告訴我們要**執行何種規則的運算**。

今天我們要介紹以下的運算子：賦值運算子、算術運算子、比較運算子與邏輯運算子。

5.1 賦值運算子

賦值運算子的符號是「 = 」，如程式碼 13 所示，它將等號右邊的結果賦值給等號左邊的變數。

```
a = 256
b = "good job"
c = True
```

程式碼 13: 賦值運算子

5.2 算術運算子

算術運算子就是我們平時數學運算學到的 **加、減、乘、除、商數、餘數以及次方數**。程式碼 14 提供了算術運算子的實作範例。

```
# 加
a = 3 + 2 # 5
# 減
b = 3 - 2 # 1
# 乘
c = 3 * 2 # 6
# 除
d = 3 / 2 # 1.5
# 取商數
e = 3 // 2 # 1
# 取餘數
f = 3 % 2 # 1
# 次方數
g = 3 ** 2 # 9
```

程式碼 14: 算術運算子

此外，當我們將算數運算子和賦值運算子結合，那就相當於我們將變數做運算之後，再把得到的結果賦值給同一個變數。我們就可以同時對變數做運算後再賦值。例如：

`a += 1` 相當於 `a = a + 1`；
`a -= 2` 相當於 `a = a - 2`；

5.3 比較運算子

比較運算子只會回傳兩種結果：True 或是 False (此爲布林值，在之後章節將詳細介紹)。而這些運算子的意思分別在以下表列，程式碼 15 則提供了比較運算子的實作範例。

- `>`：大於
- `<`：小於
- `>=`：大於等於
- `<=`：小於等於

- ==：等於
- !=：不等於

```
a = (5 > 3) # True
b = (5 < 3) # False
c = (5 >= 3) # True
d = (5 <= 3) # False
e = (5 == 3) # False
f = (5 != 3) # True
```

程式碼 15: 比較運算子

5.4 邏輯運算子

最後一個運算子就是邏輯運算子！我們在邏輯運算子兩邊放布林值，圖 1 整理了兩個布林值進行邏輯運算後的結果，而程式碼 16 提供了邏輯運算子的實作範例。

	and	or
True / True	True	True
True / False	False	True
False / True	False	True
False / False	False	False

Figure 1: 邏輯運算子

```
a = (True and True) # True
b = (True and False) # False
c = (False and False) # False

d = (True or True) # True
e = (True or False) # True
f = (False or False) # False
```

程式碼 16: 邏輯運算子

6 條件判斷式

這章我們會用到上一章所提到的「運算子」，大家記得把上一章內容看熟悉喔！

小情境 1. 今天，如果我們想將一群人依據是否大於 *18*（包含）歲分成 a 組和 b 組 *2* 組，那麼我們首先要做的就是判斷這個人是否大於等於 *18* 歲，再將他分到對應的組別。這時候我們該如何用程式幫我們達成目標呢？

在撰寫程式時，我們常常會用到：「若這個狀況發生時，我要執行…；若非這個狀況發生，我則要執行…」。這時 `if-elif-else` 的判斷條件句便能幫助我們完成。

6.1 條件判斷 1：if-else

`if` 指的是如果這個條件成立，那麼我們要執行甚麼；`else` 是如果不是 `if` 所包含的任何條件，我們則要執行甚麼。

```
a = 20
if a >= 18: # 如果大於等於 18 歲
    print("Group a!")

else: # 如果小於 18 歲
    print("Group b!")
```

程式碼 17: 條件判斷：`if-else`

程式碼 17 印出的結果是：`Group a!`。

我們會是被分配到 Group a，這是因爲 `a` 大於 18，所以在第三行跑進 `if` 的條件中去印出 `Group a!`，故也不理會 `else` 裡面的敍述。

由此，我們看到 Python 會根據不同條件去執行相對應區塊中的敍述。

6.2 條件判斷 2：if-elif-else

第二個狀況 `if-elif-else` 可以應用在當我們想分類成三組以上。

小情境 2. 今天我們想要依照年紀分類成兒童（*0* 至 *17* 歲）、青壯年（*18* 至 *44* 歲）、中老年（*45* 歲以上）*3* 個組別呢？我們該如何做到呢？

這個時候，`elif` 便登場了！`elif` 使用於非「`if` 或是前面的 `elif`」所包含的條件中的部分條件。也可以說是我們需要兩個以上的區塊去做分類。

```
a = 40

if a <= 17:
    print(" 兒童")

elif a <= 44:
    print(" 青壯年")

else:
    print(" 中老年")
```

程式碼 18: 條件判斷：if-elif-else

程式碼 18 印出的結果是：青壯年。

我們可以看到因爲 40 歲大於兒童的 17 歲，所以會第一個 if 不會成立，但是小於青壯年的 44 歲，所以會進入第二個 elif 的條件，並且印出青壯年。

因爲程式只會進入這些區塊中的其中一個，所以就會以 if-elif-else 的條件判斷式來寫程式。像是如果在前面的範例，我們用兩個 if 而不是用 if-elif-else 來包裝的話，我們就需要用以下的條件假設：

```
a = 40

if a <= 17:
    print(" 兒童")

if a > 17 and a <= 44:
    print(" 青壯年")

if a > 44:
    print(" 中老年")
```

程式碼 19: 條件判斷：連續使用三個 if

程式碼 19 印出的結果是：青壯年。

這樣會需要寫更多的條件，出錯的機率也會比較高，因此很多時候能夠用 if-elif-else 會比用 if 來得方便許多，寫錯的機率也會減少！

但是如果我們的條件是相互不相斥，就可以用不同 if 來寫，像是同時要判斷這個人的「年紀」、「性別」等等，就可以分開成兩個條件判斷，所以寫程式時要注意條件間是否有相斥性喔。

7 串列

串列 (list) 是將一連串相同或是不同資料型別的單位放在一起的序列。我們可以將它想成一個裝著很多不同東西（可能是整數、字母、字串、布林值或甚至另一個串列等等）的容器，如程式碼 20 所示。

```
['a', 1, 2, 'hello', [1, 2, 3]]
```

程式碼 20: 串列——一連串相同或是不同資料型別的單位放在一起的序列

建立串列

我們可以用中括號 `[]` 去創建串列，或是直接宣告 `list()`，如程式碼 21 所示：

```
list_a = []
list_b = list()
```

程式碼 21: 建立空白串列

此外，我們可以在中括號裡放入我們想要取出的元素的索引值；這邊要注意的是，程式語言大多是從 0 開始計數，所以當我們要取串列中的第 1 個元素，要在中括號中要放索引值 0，要取第 5 個元素時要放索引值爲 4，並以此類推。程式碼 22 提供了串列索引值的實作範例。

```
a = [1, 2, "hi", True]
print(a) # [1, 2, 'hi', True]
print(type(a)) # <class 'list'>
print(a[2]) # hi   （取出索引值爲 2 的元素「hi」）
```

程式碼 22: 串列的索引值

至於爲什麼我們會需要串列來儲存資料，而不要用不同變數來儲存呢？那是因爲串列有一些很方便使用的性質，讓我們可以同時操作一組相關的資料。我們以一個情境來理解吧！

小情境 3. 今天有 *30* 個同班同學，他們的數學成績登記下來後儲存在一個串列中，我們想要計算這個班級的數學成績平均。

在程式碼 *23* 中，我們以 *math_scores* 爲儲存成績的串列變數，取出這個串列的所有元素並相加，再除以全班人數以得到我們的答案。

```python
math_scores = [76, 51, 71, 69, 98, 62, 82,
               87, 90, 62, 50, 85, 75, 63, 93,
               92, 72, 73, 66, 100, 53, 83, 63,
               95, 58, 97, 84, 91, 81, 88]

total = 0
for i in range(30):
    total += math_scores[i]

average = total / 30
print(average) # 77.0
```

程式碼 23: 計算全班平均分數

印出的結果是 77.0 便是全班的數學平均成績。

由此可見，我們可以用一個串列變數去儲存所有人的成績，再用一個迴圈就將成績加總，而不需要去尋找零散的各個變數，讓我們的程式架構更加系統化！

7.1 取出串列的方法

取出部分串列的方法主要遵循一個語法：

[首：尾：間隔數]

舉例來說，如果我們要從第一個數開始到第十個每兩個取一個的時候，我們可以這樣寫：

[0：10：2]

從頭取

回歸到我們上面舉的例子，要取出在串列中每個學生的成績，我們可以用中括號來得到我們需要的值，如程式碼 24 所示：

```python
math_scores = [76, 51, 71, 69, 98, 62, 82,
               87, 90, 62, 50, 85, 75, 63, 93,
               92, 72, 73, 66, 100, 53, 83, 63,
               95, 58, 97, 84, 91, 81, 88]

# 第一個學生的成績
print(math_scores[0]) # 76

# 第五個學生的成績
print(math_scores[4]) # 98
```

程式碼 24: 從頭取串列值

從尾取

那麼如果我們是想要拿到倒數第三個學生的成績呢？Python 有一個從尾開始計數的方式，只要像程式碼 25 一樣加上一個負號即可，這邊要注意，從尾開始第一個數是 -1 喔。

```
# 倒數第一個學生
print(math_scores[-1]) # 88

# 倒數第五個學生
print(math_scores[-5]) # 97
```

程式碼 25: 從尾取串列值

取一個範圍

如果要取某一個範圍內的子串列，我們可以在中括號的中間放「起點：終點」的語法來告訴 Python 我們要從哪裡開始擷取我們的串列，並在哪裡結束。

這邊要特別注意「起點」是被包括在我們的範圍中，而「終點」則是不被包含在我們的範圍中。取一個範圍的範例可參見程式碼 26：

```
# 第 1 個到第 5 個
print(math_scores[0:5]) # [76, 51, 71, 69, 98]
# 第 15 個到倒數第 3 個
print(math_scores[14:-2]) # [93, 92, 72, 73, 66, 100, 53, 83, 63, 95, 58, 97, 84, 91]
```

程式碼 26: 從串列中取一個範圍

那麼如果我們想要擷取某個索引後的全部元素，我們可以直接在該索引後加上一個冒號，語法會是 `[index:]`；如果是要該索引之前的全部元素，冒號則是加在前面，語法會是 `[:index]`。以上用法如程式碼 27 所示：

```
# 第 26 個之後
print(math_scores[25:]) # [97, 84, 91, 81, 88]
# 第 10 個之前
print(math_scores[:10]) # [76, 51, 71, 69, 98, 62, 82, 87, 90, 62]
```

程式碼 27: 從串列取出某個索引前後的全部元素

7.2 串列的操作

合併

當我們想要將兩個串列合併，我們可以使用「 + 」將它們合併，如程式碼 28 所示：

```
a = [1, 2, 3, 4]
b = [5, 6, 7, 8, 9, 10]

c = a + b
print(c) # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

程式碼 28: 合併兩串列

複製

我們可以使用 [:] 的語法，相當於將這個串列裡的資料全部複製一份新的給另一個串列變數，所以當更動原串列時，並不會更動到新的串列。

這和直接使用 b = a 將兩個變數指向同一個串列有所不同，因爲當兩個變數的標籤都是貼在同一個串列上，只要任何一個變數去做操作，便會同時更改到兩者的內容物。

如果直接使用 b=a，如程式碼 29 所示，a 與 b 兩者會一起被更改。

```
a = [1, 2, 3]
b = a
del(a[1])
print(a)  # [1, 3]
print(b)  # [1, 3]
```

程式碼 29: 直接使用 b = a 將兩個變數指向同一個串列

如果用複製 [:]，如程式碼 30 所示，更動 a 並不會影響 b 的內容物。

```
a = [1, 2, 3]
b = a[:]
del(a[1])
print(a)  # [1, 3]
print(b)  # [1, 2, 3]
```

程式碼 30: 使用 [:] 複製串列

7.3 串列中常見的函數

現在就讓我們來看看 Python 幫我們寫好用來操作串列的功能吧！在後面的章節會再詳細介紹函數的定義，現在我們可以把函數想成是一些小功能：

7.3.1 len(list)

首先，如果我們想要知道某個串列的長度，把串列名稱放進 len 的括號中就可以辦到，如程式碼 31 所示：

```
my_list = [1, 5, 2, 7, 4]
print(len(my_list)) # 5
```

程式碼 31: len(list)

7.3.2 append(element)、extend(list)

如果我們要增加元素（數字、字串、串列）到串列，可以使用 append()；如果是要將一個串列合併到原本的串列，則可以使用 extend()，如程式碼 32 所示：

```
my_list = [1, 2, 3]
other_list = [10, 9, 8]
my_list.append(other_list)
print(my_list) # [1, 2, 3, [10, 9, 8]]
my_list.extend(other_list)
print(my_list) # [1, 2, 3, [10, 9, 8], 10, 9, 8]
```

程式碼 32: append(element)、extend(list)

7.3.3 insert(index, element)

將某值加到我們想要的位置，則可以用 insert()，第一個參數是我們想將元素加到的位置 (index)，第二個參數則是我們想加入的元素 (element)，如程式碼 33 所示：

```
my_list = ['good evening', 'good afternoon', 'good morning', 'good night']
my_list.insert(2, 'hello')
# ['good evening', 'good afternoon', 'hello', 'good morning', 'good night']
```

程式碼 33: insert(index, element)

7.3.4 pop(index)

我們可以將不要的元素用 pop() 刪除，這個函式會回傳被刪除的值，且預設是刪除串列中的最後一個元素，如程式碼 34 所示：

```
my_list = [2, 3, 4, 5, 6]
my_list.pop(1) # [2, 4, 5, 6]
my_list.pop() # [2, 4, 5]
print(my_list.pop()) # 5
```

程式碼 34: pop(index)

7.3.5 del list[index]

del 是用我們想要刪除的元素的索引值，語法是在 del 後面寫上串列以及要刪除的索引值，如程式碼 35 所示：

```
my_list = [1, 2, 3, 4, 5]
del my_list[0] # [2, 3, 4, 5]
```

程式碼 35: del list[index]

7.3.6 remove(element)

remove 則是用值來刪除「第一個」是該值的元素，如程式碼 36 所示：

```
my_list = [1, 2, 3, 4, 5]
my_list.remove(3) # [1, 2, 4, 5]
```

程式碼 36: remove(element)

7.3.7 split("delimeter")

我們可以用 split() 將字串依照其分隔符 (delimeter) 分隔成數個串列，如程式碼 37 所示。

7.3.8 sep.join(list)

我們將串列合併成一個字串，並且在 join 前面寫上合併後串列元素間的連接符號，如程式碼 38 所示。

```
my_string = "1@anismke@slje@sirjg"
print(my_string.split("@")) # ['1', 'anismke', 'slje', 'sirjg']
```

程式碼 37: split("delimeter")

```
my_list = ["2024", "01", "01"]
my_list = '-'.join(my_list)
my_list # '2024-01-01'
```

程式碼 38: sep.join(list)

7.3.9 count

count 可以計算串列裡某個值總共出現的次數，如程式碼 39 所示：

```
happy_list = ['h', 'a', 'p', 'p', 'y']

print(happy_list.count('p')) # 2
print(happy_list.count('b')) # 0
```

程式碼 39: count

7.3.10 sort

將串列依照某個順序去做排列，通常是數字由小到大或是由大到小的順序。

sort 有許多參數可以去做調整，首先是 reverse，因爲 sort 的預設是升冪排列，如果我們想要數列以降冪排列，我們則可以把 reverse 設爲 True，便可以達到效果。程式碼 40 是 sort 的基本使用範例：

```
list.sort(cmp=None, key=None, reverse=False)
my_list = [3, 7, 2, 6, 1]
my_list.sort()
my_list # [1, 2, 3, 6, 7]

my_list.sort(reverse = True)
my_list # [7, 6, 3, 2, 1]
```

程式碼 40: sort、指定 reverse 參數

另外一個可以調整的參數是 `key`，也就是排序的規則。我們通常會使用一個叫 `lambda` 的語法，舉例來說 `lambda x:x[index]` 可以取出每一個我們要比較的元素其索引爲 `index` 的值，再進行比較排序。

如果我們想要將串列裡的字串按照它們的第二個字母去做排序，那麼我們便可以使用程式碼 41 的寫法：

```
another_list = ["hello", "okay", "bye bye", "lah"]
another_list.sort(key = lambda x : x[1])
print(another_list) # ['lah', 'hello', 'okay', 'bye bye']
```

程式碼 41: `sort`、指定 `key` 參數

課堂練習 1 (ZeroJudge d074). 蝸牛老師在一所優質高中擔任電腦老師，在學校裡有一間他專用的電腦教室。最近學校有一筆經費要幫這個電腦教室更新電腦。學校的原則是，每個上課的學生都要有自己的電腦，但是不希望購買多餘的電腦。給你蝸牛老師的任教班級數及每班人數，請你幫他算出要買幾部新電腦給學生使用。

輸入只有兩行。第一行有一個正整數 $n(n \leq 10)$，代表蝸牛老師的任教班級數。第二行有 n 個由空白隔開的正整數，代表各班人數。請輸出需購買的電腦數量。

課堂練習 2 (ZeroJudge j605). 給定 s_i 個提交紀錄，第 i 個提交紀錄有兩個整數 t_i 和 s_i 代表上傳時間和該次上傳的分數，若第 i 次的提交結果爲嚴重錯誤，則 s_i 爲 -1。計算總分的公式爲

$$\text{提交紀錄中的最高分} - \text{總提交次數} - \text{總嚴重錯誤次數} \times 2$$

若計算出來的分數爲負則計爲 *0*。請計算並輸出總分和第一次獲得最高分的時間點。

課堂練習 3 (ZeroJudge c067). *3* 歲的小明喜歡玩他的方塊積木，他總是把方塊疊在一起形成高度不一的方塊堆。然後他說：這是一面牆。*5* 歲的姊姊小美聽到了就跟小明說：眞正的牆高度應該要一樣才行。小明聽了覺得有道理於是決定要搬動一些方塊使所有方塊堆的高度一樣。由於小明是個懶惰的小孩，他想要搬動最小數目的方塊以達成這個目的，你能幫助他嗎？

課堂練習 4 (ZeroJudge h081). 小明最近想要用程式做股票交易，給定一個股票的歷史價格 $a[1], a[2], ..., a[n]$，他的投資策略如下：

1. 同一個時間最多只會持有一張股票，並會在時間點 *1* 花 $a[1]$ 買進。
2. 若當下持有股票且買進價格爲 x，當遇到價格 $y \geq x + D$ 時即賣出，並賺得利潤 $y - x$。
3. 若當下未持有股票且上一次賣出價格爲 x，當遇到價格 $y \leq x - D$ 時則會買進股票。

輸出依照上述規則買賣後所得到的利潤和，若交易結束仍持有股票，則不考慮該股票買進的成本，直接無視該股票即可。

8 迴圈

在前面的章節中，我們學習了判斷的用法。透過 `if-elif-else` ，我們得以在 Python 中根據不同條件而執行相對應區塊中的敍述。在本章，我們繼續討論 Python 的控制流程中另一個常會遇到且重要的主題——迴圈 (loops)。

根據維基百科的定義，迴圈是一段在程式中只出現一次，但可能會**連續執行多次**的程式碼。在 Python 中，迴圈中的程式碼會**執行特定的次數**，或者是**執行到特定條件成立**時結束迴圈，或者是針對**某一容器中的所有項目**都執行一次。

在 Python 中，主要的迴圈語法包含 `while` 與 `for` 迴圈：

- 在 `while` 迴圈中，程式碼會重複執行直到特定條件成立時結束迴圈。
- 在 `for` 迴圈中，程式碼會執行特定的次數，或者其針對某容器中的所有項目都執行一次。

8.1 `while` 迴圈

`while` 迴圈是一個以條件判斷爲重點的迴圈：

1. 進入迴圈時，程式會執行第一次條件判斷，若該判斷式的結果爲 `False`，程式會直接跳過迴圈並執行接下來的敍述。
2. 若該判斷式的結果爲 `True`，程式會執行迴圈內的敍述，執行完畢後回到判斷式執行判斷。
3. 重複執行**上一步驟**，直到判斷式的結果爲 `False`，程式離開迴圈並執行接下來的敍述。

圖 2 提供了 `while` 迴圈執行步驟的圖解。

Figure 2: `while` 迴圈圖解

程式碼 42 提供了 `while` 迴圈的實作範例，其主要目的是重複列印 “Hello World” 十次。初始時，變量 `n` 被賦值爲 1。接著，進入一個 `while` 迴圈，條件爲 `n <= 10`。在迴圈內部，每次迴圈都會執行兩個操作：首先列印出 “Hello World”，然後將 `n` 的值增加 1。當 `n` 增加到 11 時，條件 `n <= 10` 不再成立，迴圈結束。最後，程式列印出變數 `n` 的最終值，此時 `n` 爲 11。

```
n = 1
while n <= 10:
    print("Hello World")
    n += 1
print(n)
```

程式碼 42: while 迴圈

8.2 for 迴圈

8.2.1 計數迴圈——使用 range()

透過 for 迴圈與 range() 函數的搭配，我們得以讓迴圈中的程式碼執行特定的次數，使用方式簡述於圖 3，當中的 range(start, end, step) 會依照我們給定的起始值 (start)、終點值 (end) 與公差 (step) 生成相對應的數列，並在產生數列後，透過 for i in range(...) 讓 i 走訪該數列中的每一個元素。

要特別注意的是，數列的右界並**不包含終點值 (end)**，換句話說，我們皆是討論 $[start, end)$ 的左閉右開區間範圍。

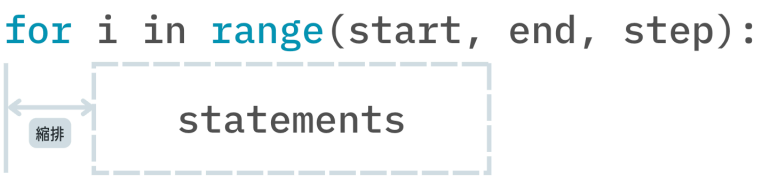

Figure 3: for 迴圈與 range() 函數

程式碼 43 包含兩個分開的 for 迴圈。第一個迴圈使用 range(10)，意味著它從 0 開始到 9 結束，迴圈內的 print(i) 指令會在每次迭代[1] 時列印出當前的 i 值，因此它將依序列印出數字 0 到 9。第二個迴圈使用 range(1, 10, 2)，意味著它從 1 開始到 9 結束，**並且每次迭代增加 2**。因此，這個迴圈將列印出 1, 3, 5, 7, 和 9 這些奇數。

```
for i in range(10): # loop 1
    print(i)
for i in range(1, 10, 2): # loop 2
    print(i)
```

程式碼 43: for 迴圈與 range() 函數

[1]在電腦科學中，「迭代」是程式中對一組指令（或一定步驟）的重複。

8.2.2 走訪容器——以串列爲例

在迴圈的定義中，迴圈中的程式碼也可以針對某一容器中的所有項目都執行一次，使用方式簡述於圖 4，我們讓當中的 `i` 依序走訪 `container_name` 容器中每一項元素，並且每次走訪皆執行迴圈內的敍述。

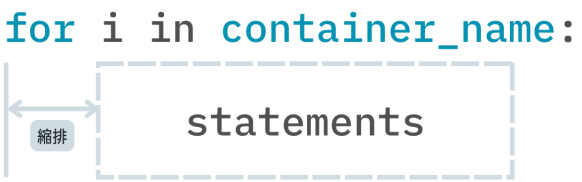

Figure 4: 使用 `for` 迴圈走訪容器

程式碼 44 展示了如何使用 `for` 迴圈走訪串列中的每個元素並逐一列印出來。首先我們創建了名爲 `l` 的串列 [2]，其中包含五個元素：1, 2, 3, 4, 和 5。接著，進入 `for` 迴圈，迴圈中的變數 `i` 在每次迭代中被賦予串列 `l` 中的下一個元素的值。在迴圈的每次迭代中，都會執行 `print(i)`，列印出當前的 `i` 值。因此，這段程式碼將會按順序列印出串列中的數字：1, 2, 3, 4, 5。

```
l = [1, 2, 3, 4, 5]
for i in l:
    print(i)
```

程式碼 44: 使用 `for` 迴圈走訪容器

8.3 break 與 continue 敍述

8.3.1 break

當迴圈執行時遇到 `break` 敍述，程式會**直接中止該迴圈的運作**並執行接續的程式碼。

程式碼 45 使用了一個無窮迴圈來不斷要求使用者輸入。在迴圈內部，程式碼透過 `if` 語句檢查使用者輸入的 `name` 是否等於 “Benson”。如果輸入的名字是 “Benson”，則**迴圈會遇到 `break` 敍述並立即終止**，不再執行任何後續的 `print` 命令。如果輸入的名字不是 “Benson”，則會執行 `print("Hello, ", name)`，向該名字的使用者問好，然後迴圈繼續，再次等待新的輸入。

8.3.2 continue

當迴圈執行時遇到 `continue` 敍述，程式會**跳過接下來要執行的敍述，直接執行下一輪**。

[2] 我們將會在下一節詳細介紹

```python
while True:
    name = input()
    if name == "Benson":
        break
    print("Hello, ", name)
```

程式碼 45: `break` 敍述

```python
while True:
    name = input()
    if name == "Benson":
        continue
    print("Hello, ", name)
```

程式碼 46: `continue` 敍述

程式碼 46 同樣地使用了一個無窮迴圈來不斷要求使用者輸入。迴圈內，如果使用者輸入的名字是 “Benson”，則 `continue` 會被執行，**迴圈將跳過其餘部分並立即開始下一次迭代**。若名字不是 “Benson”，則會執行 `print("Hello, ", name)`，向用戶問好後繼續迴圈。

課堂練習 5 (ZeroJudge d070). 已知除了不是 *400* 的倍數的 *100* 的倍數以外，*4* 的倍數的年份均爲閏年。現在請你編寫一個程式，使用一個無窮迴圈來不斷要求使用者輸入西元年份 y，並且對於每一個輸入的 y 輸出其是屬於閏年還是平年。使用者若輸入 *0*，代表程式結束，這個數字請勿做任何處理。

課堂練習 6 (ZeroJudge a038). 輸入任意數字，請使用 *`while`* 迴圈將其數字全部倒轉。

課堂練習 7 (ZeroJudge a024). 給定兩個整數，請使用輾轉相除法求出它們的最大公因數。

課堂練習 8 (ZeroJudge d649). 輸入一數字 N，輸出一個高爲 N，底也爲 N 的三角形，每組輸出請用空行隔開。

課堂練習 9 (ZeroJudge c013). 根據所給的振幅 *(Amplitude)* 及頻率 *(Frequency)*，產生對應的 *Triangle Wave*。

9 字串

9.1 字串的基本架構

字元 (character) 是人類語文的最基本單位，舉凡英文字母、標點符號、中文漢字、阿拉伯數字、空格等等都是字元的其中一個種類。若我們將多個字元組合起來，這些被組合在一起字元就變成了所謂的**字串 (string)**，像是一個英文單字 (E.g. “Hello”) 抑或是一句話都可以 (E.g. “It’s a good day!”) 被視爲一個字串。

字串是 Python 最常見的資料型態之一。在 Python 中，爲表示某一串字元是字串，我們使用**單引號**或**雙引號**將其包覆。字串的初始化非常簡單，如程式碼 47 所示。

```
s1 = "Hello"
s2 = 'This is a string'
```

程式碼 47: 字串的初始化

和串列類似，我們也可以對字串使用索引取值 (程式碼 48 前兩行)，但因字串在 Python 中是一個**不可變 (immutable)** 的容器，我們不可以對字串的某一項進行改值 (程式碼 48 末行)。

```
print(s1[0]) #H
print(s2[-1]) #g
s1[0] = 'B' #Error!
```

程式碼 48: 字串的取值與改值

字串在 Python 中被視爲容器的其中一種，因此，我們可以搭配 `for` 迴圈對字串的每一項進行走訪，如程式碼 49 所示。

```
s = "Hello World"
for i in s:
    print(i)
```

程式碼 49: 字串的走訪

在字串中，有一個特別的運算子 `in`。透過 `a in b` 與 `a not in b`，我們可以判斷字串 `a` 是否是字串 `b` 的子字串，如程式碼 50 所示。

```python
s = "Welcome to NTUIM"
print("NTUIM" in s) #True
print("NTUIM" not in s) #False
print("NTUCSIE" not in s) #True
```

程式碼 50: 使用 in 運算子判斷是否爲子字串

9.2 字串的切片

和串列的複製類似，我們可以使用 [:] 對字串的指定範圍進行切片 (slicing) 的動作，並將其指派給新的變數。

Figure 5: 字串切片的公式

於此，我們指定字串切割的起始點 (圖 5 的 start) 、終點 (圖 5 的 end) 與公差 (圖 5 的 step)，並且在 $[start, end)$ 區間以 $step$ 爲公差對字串 s 進行取值。

課堂練習 10. 程式碼 *51* 使用 *[:]* 對字串的的 *7* 個不同指定範圍進行切片並輸出，請先用「人腦」判斷各行的輸出結果，再將其與實際的程式執行結果對照。

```python
s = 'abcdefg'
print(s[:])
print(s[1:7])
print(s[1:7:2])
print(s[::-1])
print(s[6:1:-1])
print(s[-1:-8:-2])
print(s[1:7:-2])
```

程式碼 51: 練習：各行的輸出結果爲何

9.3 字串的運算

9.3.1 串接

在 Python 中，+ 被用來串接 (concatenate) 左右兩個字串，而 * 被用來串接多個相同的字串，如程式碼 52 所示。

```python
s1 = "Hello"
s2 = "World"
print(s1 + s2) # HelloWorld
print(s1 * 3) # HelloHelloHello
```

程式碼 52: 字串的串接

9.3.2 比較

在 Python 中，透過 == 與 !=，我們可以判斷兩字串相等與否，如程式碼 53 所示。

```python
s1 = "abc"
s2 = "def"
s3 = "abc"
print(s1 == s2) #0
print(s1 != s3) #0
```

程式碼 53: 字串的比較 (== 與 !=)

此外，透過 > 和 <，我們可以比較兩字串的字典序 [3]。如程式碼 54 所示。

```python
print(s1 < s2)
print(s2 > s3)
```

程式碼 54: 字串的比較 (> 與 <)

9.4 字串的常用方法

9.4.1 大小寫轉換

`s.upper()` 可將字串 `s` 的內容全部轉成大寫，而 `s.lower()` 可以將字串 `s` 的內容全部轉成小寫，如程式碼 55 所示。

9.4.2 判斷

- `s.isalpha()`：判斷字串每項元素是否都是英文字母
- `s.isnumeric()`：判斷字串每項元素是否都是數字

[3]簡單來說，字典序是一種將單詞或字符串按照字典中的順序來排列的方法。就像查字典時，單詞是按照字母表的順序排列的，先看第一個字母，相同的再看第二個字母，以此類推。

```
s = "hEllOntUiM"
print(s.upper()) #HELLONTUIM
print(s.lower()) #hellontuim
```

程式碼 55: 字串的大小寫轉換

- `s.isalnum()`：判斷字串每項元素是否都是字母或數字
- `s.isupper()`：判斷字串每項元素是否都是大寫

以上方法的實作範例，如程式碼 56 所示。

```
s1 = "123456abc"
print(s1.isalpha()) #False
print(s1.isnumeric()) #False
print(s1.isalnum()) #True

s2 = "NTUIM"
print(s2.isupper()) #True
```

程式碼 56: 字串的判斷

9.4.3 `s.split(delimiter, max)`

使用 `s.split()`，我們可以用給定的分割符作爲中斷點將字串「大卸八塊」，並將每一塊存進一條新的串列中。

值得注意的是，若我們沒有指定分割符，Python 預設以空格進行分割；另外，我們亦可以指定最多分割的次數，而 Python 預設會處理字串中所有的分割符。

`s.split()` 的實作範例如程式碼 57 所示。

```
print('A B C'.split(' ')) # ['A', 'B', 'C']
print('A B C'.split(',')) # ['A B C']
print('A,B,C'.split(',', 1)) # ['A', 'B,C']
```

程式碼 57: `s.split()`

9.4.4 s.join()

我們可以將 s.join() 視爲 s.split() 的反敘述，以 s 作爲分隔符將給定序列 (型態可以是串列或元組) 的各項用分割符串接起來。

需特別注意的是：給定序列之內的各元素皆需要是字串型態，才能使用 s.join() 的方法。

s.join() 的實作範例如程式碼 58 所示。

```
print('!'.join(['A', 'B', 'C'])) # A!B!C
print('!'.join(('A', 'B', 'C'))) # A!B!C
```

程式碼 58: s.join()

9.4.5 s.strip(characters)

使用 s.strip()，我們可以從字串的開頭與結尾移除指定的符號 (characters)。若我們沒有指定任何符號，Python 預設會移除字串頭尾的所有空格。

s.strip(characters) 的實作範例如程式碼 59 所示。

```
print("   @@@@   ".strip()) # "@@@@"
print("!!!??!!!@@@@!!!!!!!!!".strip("!?")) # "@@@@"
```

程式碼 59: s.strip(characters)

9.4.6 s.find(substr, start, end)

使用 s.find()，我們可以從字串 s 尋找指定的子字串 substr 並回傳位置。另外，我們亦可以使用額外的參數 (start 與 end) 去限制對字串搜尋的範圍，而 Python 預設會搜尋整個字串。

s.find(substr, start, end) 的實作範例如程式碼 60 所示。

```
print("NTUIM".find("IM")) # 3
print("NTUIM".find("IM", 0, 3)) # -1
```

程式碼 60: s.find(substr, start, end)

9.4.7 s.replace(oldstr, newstr, num)

使用 s.replace()，我們可以將字串中的 oldstr 全部 (或部分) 替換成 newstr。另外，我們使用額外的參數 num 去限制取代的總次數，而 Python 預設會用 newstr 取代所有 oldstr。

`s.replace(oldstr, newstr, num)` 的實作範例如程式碼 61 所示。

```
print("ILOVETARO".replace("LOVE", "HATE")) # IHATETARO
print("ABABABABA".replace("A", "C")) # CBCBCBCBC
print("ABABABABA".replace("A", "C", 3)) # CBCBABABA
```

程式碼 61: `s.replace(oldstr, newstr, num)`

課堂練習 11 (ZeroJudge a022). 迴文的定義爲正向，反向讀到的字串均相同。舉例來說：*abba, abcba ...* 等字串就是迴文。請編寫一程式，判斷一個字串是否是一個迴文？

課堂練習 12 (ZeroJudge b969). 輸入共二行。第一行含有要問候的人名，人名之間以空白隔開，人名中不含空白。第二行含有問候語。請對於每一個所提供的人名，請輸出 **問候語** $+$ **“, ”** $+$ **人名** 於一行。

課堂練習 13 (ZeroJudge c659). 輸入只有一行，第一個字爲連接詞，第二個以後爲單字。連接詞與單字之間均以空白隔開。連接詞與單字包含字母及/或符號，但不含空白。請輸出所有的單字，且**單字與單字間均有一個連接詞**。

課堂練習 14 (ZeroJudge e051). 輸入只有一行，爲一個英文單字。請輸出把輸入單字頭尾字母留住、中間全填成底線後的單字長相。

課堂練習 15 (ZeroJudge d139). 有一種字串壓縮的方法是將重覆出現的字母，以「數字 $+$ 字母」的方式表示。例如：$AAABBC$ 即以 $3ABBC$ 表示，這樣就可以節省一個字元的空間。而其中的 BB，若以 $2B$ 表示，一樣是兩個字元，因此，仍以 BB 表示。

以大寫英文字母組成的字串作爲輸入，請輸出經過壓縮後的字串。

課堂練習 16 (ZeroJudge c007). 你現在必須要寫一個程式，將普通的雙引號 *(")*，轉成有方向性的雙引號，而其它文字則不變。而在把普通的雙引號換掉的時候，要特別注意，當要開始引述一句話時要用 ``` `` ```，而結束引述時要用 ```` '' ````。不用擔心會有多層巢狀引號的情形，也就是第一個引號一定是用 ``` `` ```來代替，再來用 ```` '' ````，然後用 ``` `` ```，接著用 ```` '' ```` ..... 依此類推。

課堂練習 17 (ZeroJudge b524). *yee* 由一個 y 和兩個 e 組成，實在是太 *yee* 了。你有一個只包含 *“y”* 和 *“e”* 的字串，而且 *“e”* 的數量是 *“y”* 的兩倍，你可以交換相鄰的兩個字母，請問要交換多少次才能把該字串變成 *yeeyeeyee.....* 的形式？

10 元組

tuple 和 list 非常相似，主要的不同是 tuple 建立之後便無法更改裡面的資料。

10.1 建立元組

首先就讓我們來看看要如何建立／宣告元組吧！

值得注意的是，元組大多是用小括號 () 來做表示，而 list 則是用中括號 [] 表示。

```
a_tuple = tuple()
b_tuple = ()
c_tuple = (1, 'd', 5.6, True)
```

那麼我們也想像串列一樣去增加元組裡的值，這樣會可行的嗎？

```
a_tuple.add(1)
AttributeError                            Traceback (most recent call last)
----> 1 a_tuple.add(1)

AttributeError: 'tuple' object has no attribute 'add'
```

tuple 是不能去增加的！因爲 tuple 的值不可做更改。同樣的問題也會出現在刪除和修改，tuple 只能取出值來喔！

11 集合

set 也是一個裝資料的大容器，它的特色是同樣的值只出現一次，且他們是沒有位置關係的。

11.1 建立集合

請注意，set 我們會用大括號 {} 表示：

```
a_set = set()
b_set = {1, 2, 3, 4}
c_set = {2, '2'}
```

11.2 更新集合

再來我們來看看要如何對 set 的資料做增減：

add：加上一筆資料

```
a_set = set() # {}
a_set.add(6)
a_set # {6}
```

update：加上另一個 set

```
b_set = {1, 2, 3, 4}
c_set = {2, '2'}
b_set.update(c_set)
b_set # {1, 2, '2', 3, 4}
```

注意！重複的部分僅會留下一個喔

remove：移除某個值，找不到該值會產生錯誤

```
b_set # {1, 2, '2', 3, 4}
b_set.remove(3) # {1, 2, '2', 4}
b_set.remove(5) # error
```

discard：也是移除某個值，但是找不到該值時不會產生錯誤

```
b_set # {1, 2, '2', 4}
b_set.discard(4) # {1, 2, '2'}
b_set.discard(5) # {1, 2, '2'} 不會出現 error
```

set 僅僅是儲存不重複的資料，沒有對應關係，因此只能做增加和移除，無法做更改（把某位置改成某個值）!

12 字典

字典 dict() 是有鍵 key 以及值 value 對應的容器。要注意的是我們的 key 不能重複，value 則允許重複。

12.1 建立字典

我們建立字典的語法是使用大括號和冒號 {:} 去做 key 和 value 的對應。
如果今天我們分別有以下物品及它們的數量：三把鑰匙、六顆蘋果、一臺火車，那麼我們的 key 是 apple、newspaper 以及 train，而它們分別對應到的是 value 是 3, 6, 以及 1：

```
a_dict = dict()
b_dict = {}
c_dict = {'apple': 3, 'newspaper': 6, 'train': 1}
```

當我們想要取出某個 key 所對應到的 value，我們可以用這樣的語法：dict[key]
舉例來說，當我們想知道有幾顆蘋果，也就是想求出 apple 這個 key 所對應的 value，那麼就可以寫：

```
c_dict['apple'] # 3
```

我們的程式便會回傳給我們它的 value，也就是 3。

12.2 更新字典

更新字典有以下的方法：「增加」、「刪除」以及「更改」，讓我們來認識一下它們的語法：
增加：我們有兩種方法可以增加字典的內容：
dict[key] = value；
update(key:value)

```
c_dict = {'apple': 3, 'newspaper': 6, 'train': 1}
c_dict['boat'] = 8 # {'apple': 3, 'newspaper': 6, 'train': 1, 'boat': 8}
c_dict.update({'bus': 7}) # {'apple': 3, 'newspaper': 6, 'train': 1, 'boat': 8, 'bus': 7}
```

刪除：我們也有兩種方法去刪除字典的內容：
pop(key)：會回傳被 pop 掉的 value；
del dict[key]

```
c_dict # {'apple': 3, 'newspaper': 6, 'train': 1, 'boat': 8, 'bus': 7}
c_dict.pop('apple') # {'newspaper': 6, 'train': 1, 'boat': 8, 'bus': 7}
del c_dict['bus'] # {'newspaper': 6, 'train': 1, 'boat': 8}
```

更改：字典更改和增加的語法一模一樣！
dict[key] = value;
update(key)

```
c_dict # {'newspaper': 6, 'train': 1, 'boat': 8}
c_dict['newspaper'] = 5 # {'newspaper': 5, 'train': 1, 'boat': 8}
c_dict.update({'boat': 6}) # {'newspaper': 5, 'train': 1, 'boat': 6}
```

12.3 分別取出鍵、值

有時候我們想知道單只有 key 的集合，或是單只有 value 的集合，或是我們想在不知道 key 和 value 的內容物的情況下取出它們的值。

針對以上的狀況，我們分別可以用 keys()、values() 以及 items() 來完成，就讓我們來看看該如何使用：

keys()：當我們想要取得字典裡的所有 key 所形成的集合，我們便可以使用 keys() 這個函式：

```
score_dict = {'Math': 88, 'English': 97, 'Science': 90}

for key in score_dict.keys():
    print(key, end = ' ')
```

程式會印出：

```
Math English Science
```

values()：而當我們想要取得所有 value 所形成的集合，則是使用 values()：

```
for value in score_dict.values():
    print(value, end = ' ')
```

程式會印出：

```
88 97 90
```

items()：如果我們想要同時取得所有 key 以及 value，則可以用 items()：

使用 a, b = dict.items() 時，Python 會依序分配 key 以及 value 給兩個變數：

```
for my_key, my_value in score_dict.items():
    print(my_key, ":", my_value)
```

程式會印出：

```
Math : 88
English : 97
Science : 90
```

我們可以看到每一層的迴圈都會取到字典裡一組的 key 以及對應的 value，並且將它們分配給 my_key 和 my_value 這兩個變數。

課堂練習 18 (ZeroJudge b523). 有多行輸入，每一行都是可能包含大小寫英文字母、數字、空白的字串。對於每一行輸入的字串，判斷其是否爲第一次出現，若該字串是第一次出現，就印出 *"NO"*。若該字串曾經出現過，則印出 *"YES"*。

課堂練習 19 (ZeroJudge b515). 寫一程式，把摩斯電碼轉成摩斯電碼對應到的英文字。

13 函數

在 Python 中，**函數 (functions)** 是一段組織好的、可以重複使用的程式碼，用於執行特定的任務。你可以把函數想像成一個小型的機器，當你向它提供一些資料（稱爲**參數 [parameters]**[4]）時，它會執行一系列操作，然後可能會給你一些結果（稱爲**回傳值 [return values]**）。

13.1 函數的定義與呼叫

若要定義一個函數，你需要使用 `def` 關鍵字，然後是你的函數名稱和一對括號，最後以冒號作結。在這對括號裡，你可以放入一些**參數**，這些是你執行任務時可能需要的資訊。在接下來的各行，你會寫下一些程式碼，這些程式碼描述了函數要執行的具體操作。最後，函數可以有一個**回傳值**，代表函數執行完畢後的結果。你可以使用 `return` 關鍵字來指定回傳值。如果你的函數不需要回傳任何東西，那麼可以不寫 `return` 或者讓它返回 `None`（Python 中表示「無」的值）。

```
def intro(name, country, age):
    print("My name is " + name + ". I'm " + str(age) + " years old. I'm from " + country + ".")

intro("John",  "Taiwan (R.O.C.)", 18)
```

程式碼 62: 函數的定義與宣告：有參數、無回傳值、依照順序傳參數

程式碼 62 中定義了一個名爲 `intro` 的函數，用於印出一個人的自我介紹。這個函數接受三個參數：`name`（名字）、`country`（國家）和 `age`（年齡）。在函數內部，它使用 `print` 來輸出一段包含這些參數的訊息。隨後，這個函數被呼叫並傳入了實際的參數值：“John” 作爲名字、“Taiwan (R.O.C.)” 作爲國家，和 18 作爲年齡。這樣，當程式運行時，它會輸出一段 John 的自我介紹，告訴聽者他的名字、年齡和他來自哪個國家。

```
def intro(name, country, age):
    print("My name is " + name + ". I'm " + str(age) + " years old. I'm from " + country + ".")

intro(country = "Taiwan (R.O.C.)", age = 18, name ="John")
```

程式碼 63: 函數的定義與宣告：有參數、無回傳值、用關鍵字傳參數

程式碼 62 中，我們需要**依照順序**將參數對應的值傳入函數，這種方式可能會對不清楚函數定義的使用者造成混淆的情況，尤其是當有大量的參數存在於函數定義時。在 Python 中，我們可以**用關鍵字傳參數**，在傳入的值前面「註明」其所對應的參數名稱，如程式碼 63 所示。

[4] 更嚴謹地說，我們稱函數定義括號裡面的變數爲參數 (parameters)，而呼叫函數時傳入的值爲引數 (arguments)。本講義爲了方便，將會一律稱上述兩者爲參數。

```python
def intro(name, country, age = 18):
    print("My name is " + name + ". I'm " + str(age) + " years old. I'm from " + country + ".")

intro(country = "Taiwan (R.O.C.)", name ="John")
intro(country = "Taiwan (R.O.C.)", name ="John", age = 20)
```

程式碼 64: 函數的定義與宣告：有參數、無回傳值、用關鍵字傳參數、有預設值

此外，我們也可以爲函數定義中的參數設定**預設值** (default values)。若呼叫函數時未傳入對應的參數，程式就會將該參數的值設爲預設值，並照常執行函數內容。在程式碼 64 中，我們將預設的 age（年齡）設爲 18，若使用者呼叫函數時並沒有傳入 age 的參數值，age 就會是函數定義內的預設值，也就是 18。

最後，讓我們來看一個**有回傳值**的簡單範例：

```python
def f(x):
    return 2 * x + 5
a = 10
b = f(a)
```

程式碼 65: 函數的定義與宣告：有參數、有回傳值

程式碼 65 首先定義了一個名爲 f 的函數，這個函數接收一個參數 x，並回傳 2 * x + 5 的結果。接下來，程式碼創建了一個變數 a，並將其值設爲 10。然後，它呼叫了 f 函數，將 a 作爲參數傳遞，這意味著將 10 傳入函數 f。在這個函數呼叫中，x 被賦值爲 10，因此函數返回的是 2 * 10 + 5，即 25。這個返回值最後被賦予給另一個變數 b ——顯然地，b 目前的值即爲 25。

課堂練習 20. 學校正在舉辦園遊會，校長請你負責設計一個 *Python* 程式來幫助計算各個攤位的總收入。你需要創建一個名爲 *calculate_booth_income* 的函數，這個函數接受兩個參數：每種產品的單價和每種產品的銷售數量（資料型態爲字典）。函數的任務是計算出每種產品的銷售總收入，並將這些收入相加以得到攤位的總收入。

此外，如果某個產品的銷售數量超過 *50* 個，則該產品可以享受 *5%* 的額外收入獎勵。函數應該返回攤位的最終總收入。

舉例來說，給定輸入的兩字典爲：

```python
prices = {" 蛋糕": 10, " 三明治": 15, " 果汁": 20}
sales = {" 蛋糕": 30, " 三明治": 60, " 果汁": 40}
```

函數就需要回傳對應的總收入金額 *2045* 元。

在這一個練習中，我們假設兩個字典都具有相同的鍵 *(key)*。換句話說，我們保證任何一個產品名稱都會同時出現在兩個字典中。

13.2 變數範疇與命名空間

範疇 (scope)，有時也被稱爲變數的生命週期，是指一個變數在程式碼中存在的範圍。它決定了在何處以及何時你可以訪問這個變數，同時也影響著變數的創建和銷毀時機。**命名空間 (namespace)** 包含了特定範疇中所有變數名其所對應的對象，有助於區分和管理不同範疇下的同名變數。

在 Python 中，我們主要有兩種命名空間並且對應兩種不同的變數類型：

- 全域命名空間 (global namespace) 對應到全域變數 (global variable)：包含了所有函數之外創建的變數，可以在整個程式碼中被訪問。
- 區域命名空間 (local namespace) 對應到區域變數 (local variable)：只能在該函數內部被訪問，它的範疇僅限於該函數。

值得注意的是：一個 Python 程式只會有一個全域命名空間，但可以有多個區域命名空間——因爲每一個被定義的函數都可以代表一個區域命名空間。

```python
def function_1():
    x = 5 #(1)
    print("In function_1 x =", x)

def function_2():
    x = 6 #(2)
    print("In function_2 x =", x)

x = 8 #(3)
function_1()
function_2()
print("In main x =", x)
```

程式碼 66: 範疇與命名空間

課堂練習 21. 觀察程式碼 *66*，回答下列問題：

1. 請問 *(1)* 左側的 x 是區域變數還是全域變數？它屬於哪一個命名空間？
2. 請問 *(2)* 左側的 x 是區域變數還是全域變數？它屬於哪一個命名空間？
3. 請問 *(3)* 左側的 x 是區域變數還是全域變數？它屬於哪一個命名空間？

13.3 匿名函數

在 Python 中，有一種特別方便且簡潔的函數叫做**匿名函數 (lambda function)**。這種函數不像普通函數那樣需要用 `def` 關鍵字定義並命名。相反，它們是快速定義的小函數，不需要名稱，這也是爲什麼稱它們爲「匿名」的原因。

匿名函數的語法非常簡單：你只需要寫 `lambda` 關鍵字，然後列出函數的參數，後面跟上一個冒號和函數主體。舉例來說，一個將兩個數相加的匿名函數可以寫成 `lambda x, y: x + y`。

```
def run_something(func, arg1, arg2):
    func(arg1, arg2)

run_something(lambda x1, x2: print(x1 + x2), 5, 9)
run_something(lambda x1, x2: print(x1 * x2), 5, 9)
```

程式碼 67: 匿名函數

程式碼 67 展示了如何靈活地使用匿名函數與其他函數結合來執行特定的任務。首先，我們有一個名爲 `run_something` 的函數，它設計用來接收三個參數：一個函數 `func` [5] 和兩個數值 `arg1`、`arg2`。`run_something` 的主要功能是執行傳入的 `func`，並將 `arg1` 和 `arg2` 作爲這個函數的參數。

在這段代碼中，我們使用了 `run_something` 兩次，每次都傳入了一個不同的匿名函數以及兩個數字參數 5 和 9。在第一次呼叫中，傳入的匿名函數是 `lambda x1, x2: print(x1 + x2)`，這個函數將兩個參數相加並輸出結果。因此，它輸出的是 14。在第二次呼叫中，傳入的匿名函數是 `lambda x1, x2: print(x1 * x2)`，這個函數將兩個參數相乘並輸出結果。因此，它輸出的是 45。

課堂練習 22. 請將程式碼 *68* 中的函數定義與呼叫改寫爲匿名函數的形式。

```
def add_numbers(a, b):
    return a + b
x, y = 1, 2
result = add_numbers(x, y)
```

程式碼 68: 練習：改寫爲匿名函數

[5] 沒錯！函數本身也可以作爲參數

13.4 遞迴函數

遞迴函數 (recursive function) 是一種在其定義中**調用自身**的函數，可用來解決那些**可以分解爲相同問題的較小版本**的問題。

```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

程式碼 69: 遞迴函數的範例：計算階乘

程式碼 69 是一個經典的遞迴函數範例，用來計算一個數字的階乘。讓我們一步步解釋 `factorial` 函數內部：

1. 在函數內部，首先檢查 `n` 是否爲 0。這是遞迴的基礎情況 (也稱爲中斷條件)，因爲 0 的階乘被定義爲 1。如果 `n` 是 0，函數就直接返回 1。

2. 如果 `n` 不爲 0，函數則進入遞迴的一般情況。在這裡，它返回 `n` 乘以 `factorial(n-1)`。這意味著**函數會再次呼叫自己**，但這次是用 `n-1` 作爲參數。每次這樣的遞迴調用都會將 `n` 減少 1，直到它變成 0，達到基礎情況 (中斷條件) 並結束遞迴。

課堂練習 23. 斐波那契數列是一個著名的數學序列，每個數是前兩個數的和。序列以 *0* 和 *1* 開始，隨後的數就是前兩個數的和。所以，斐波那契數列的開始是 *0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...* 請寫一個名爲 *`fibonacci`* 的 *Python* 遞迴函數，它接受一個整數 *`n`* 作爲參數，並返回斐波那契序列中的第 *`n`* 個數。例如，*`fibonacci(0)`* 應該返回 *0*，*`fibonacci(4)`* 應該返回 *3*。

課堂練習 24 (ZeroJudge a227)**.** 河內塔 *(towers of Hanoi)* 是一個著名的數學遊戲或謎題。它由法國數學家愛德華·盧卡斯 *(Édouard Lucas)* 於 *1883* 年發明，以越南的河內城命名。這個遊戲包括三根柱子和一系列大小不同的盤子，這些盤子開始時按照大小順序疊放在一根柱子上，最大的盤子在最下面。

遊戲的目標是將所有盤子從起始柱子移動到另一根柱子上，遵循以下規則：

- 每次只能移動一個盤子。
- 盤子只能從柱頂移動到另一柱的頂端。
- 任何時候，大盤子不能放在小盤子上面。

請編寫一程式，給定圖 *6* 中 A 柱的起始盤數 N，請輸出把 A 上 N 個環移動到 C 的方法。

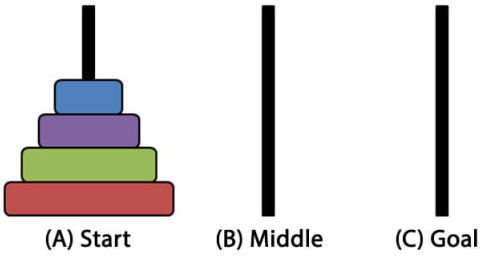

Figure 6: 河內塔示意圖

13.5 裝飾器

在 Python 中，**裝飾器 (decorator)** 是一種非常有用的工具，它允許你以一種優雅且表達性強的方式修改或增強函數的功能。你可以將裝飾器想像成一個包裝紙，將某個函數包裹起來，在不改變該函數內部代碼的情況下，增加一些額外的功能。裝飾器本身是一個函數，它接受一個函數作爲參數，並返回一個新的函數。在 Python 中使用裝飾器非常簡單，只需在你的函數定義之前加上一個 `@` 符號。

```python
def decorator(func):
    def decorated():
        print('start')
        func()
        print('end')
    return decorated

@decorator
def hello():
    print('hello')
```

程式碼 70: 裝飾器

程式碼 70 展示了如何在 Python 中創建和使用裝飾器。首先，我們定義了一個名爲 `decorator` 的裝飾器函數，它接受一個函數 `func` 作爲參數。在這個裝飾器內部，我們定義了另一個名爲 `decorated` 的函數。這個 `decorated` 函數在被調用時會先印出一個 “start” ，然後執行傳入的 `func` 函數，最後印出一個 “end” 。裝飾器 `decorator` 最終返回這個新定義的 `decorated` 函數。

接下來，我們使用 `@decorator` 這個裝飾器來裝飾 `hello` 函數。這意味著當 `hello` 函數被調用時，它實際上會先執行 `decorated` 函數中的代碼。因此，當調用 `hello()` 時，會先印出 “start”，然後執行 `hello` 函數內的 `print('hello')`，最後印出 “end”。

14 物件

14.1 物件的基本概念

在 Python 中，**物件 (objects)** 是非常基本且重要的概念。你可以把物件想像成一個小盒子，裡面裝著一些數據（稱爲**屬性 [attributes]**）和可以對這些數據進行操作的函數（稱爲**方法 [methods]**）。舉一個簡單的例子，假設我們有一個叫做「貓咪」的物件。這個貓咪物件可能有一些屬性，如：名字、體重、毛色。同時，它還有一些方法，如喵喵叫、睡覺、吃東西。

類別 (class) 用以定義物件的藍圖，它規範了物件擁有的屬性和方法。以貓咪爲例，我們可以創建一個「貓咪」類別，這個類別會定義貓咪所共有的特徵，例如名字、花色等屬性，以及睡覺、喵喵叫等方法。

當這個類別被定義後，我們可以根據它來創建具體的貓咪物件，這些具體的物件被稱爲**實例 (instance)**。例如，我們可以根據「貓咪」類別創建三個不同的實例，分別命名爲「大橘」、「小白」和「小花」。儘管它們各自有不同的名字或其他特徵，但它們都是「貓咪」類別的實例，共享該類別定義的基本屬性和方法。這種方式使得我們能夠以模組化和高效的方式重用代碼，同時保持結構的清晰和組織。

14.2 自訂物件

在 Python 中，我們可以使用 `class` 關鍵字自定義物件。程式碼 71 爲上一節「貓咪」物件之具體實作，這個類別包含了三個方法：`meow`、`eat` 和 `sleep`，以及一個初始化方法 `__init__`。

首先，`__init__` 方法是一個特殊的方法，當創建 `Cat` 類別的新實例時，它會被自動調用。這個方法接受三個參數：`name`（名字）、`color`（花色）和 `weight`（體重），並將它們賦值給實例的屬性。這意味著每隻貓咪都有自己的名字、花色和體重。

接下來，`meow` 方法用來讓貓咪自我介紹，當呼叫這個方法時，它會打印出貓咪的名字，並發出喵喵聲。`eat` 方法模擬貓咪吃東西的行爲，它接受一個參數 `food`（食物），然後打印出貓咪的名字和它喜歡吃的食物。最後，`sleep` 方法用於描述貓咪睡覺的行爲，它接受一個參數 `hour`（小時數），並印出貓咪的名字和它想要睡的時長。

程式碼 72 爲宣告並使用 `Cat` 實例的實作。首先，我們創建了三個 `Cat` 類別的實例，接著，程式碼中使用了 `Cat` 類別的方法來與這些貓咪實例互動。

課堂練習 25. 請開發一個包含 *Student* 和 *Course* 兩個類別的簡易成績管理系統：

- *Course* 類別代表一門課程，其中包括課程的名稱、代碼。這個類別提供了設定、獲取成績、以及顯示課程訊息的功能。
- 另一方面，*Student* 類別代表一名學生，包含其姓名、學號與一個紀錄其所修課程與成績的一個字典。在這個類別中，你將實現添加和刪除課程的功能，同時還能列出所有課程及其成績。此外，這個類別還包括一個計算所有課程平均成績的方法。

```python
class Cat:
    def __init__(self, name, color, weight):
        self.name = name
        self.color = color
        self.weight = weight

    def meow(self):
        print(" 你好，我是", self.name, " 喵！")
    def eat(self, food):
        print(" 我是", self.name, "，我喜歡吃", food, " 喵！")
    def sleep(self, hour):
        print(" 我是", self.name, "，我想要睡", hour, " 小時喵！")
```

程式碼 71: 自訂物件：Cat

```python
cat_1 = Cat(" 大橘", "orange", 4.5)
cat_2 = Cat(" 小白", "white, black", 3.5)
cat_3 = Cat(" 小花", "three colors, white", 4.5)

cat_1.meow()
cat_2.meow()
cat_3.sleep(5)
```

程式碼 72: 使用自訂物件 Cat 的實例

15 模組與套件

在 Python 中，我們經常會運用到已經由其他開發者編寫好的程式碼，這不僅提高了開發效率，還讓我們能夠重複運用高品質的代碼。這些程式碼通常包含有用的類別和函數，並被儲存於以.py 爲副檔名的檔案中，我們稱這些檔案爲**模組 (module)**。模組就像是一個包含了一組相關函數和類別的容器，你可以輕鬆地在你自己的程式中引用它們。

當你將多個相關的模組組織在同一個資料夾中時，這個資料夾就被稱爲**套件 (package)**。套件是一種更高層次的組織結構，它允許你將模組分類和封裝，使得代碼結構更加清晰，並且更容易管理和分發。

15.1 基本用法

若要在程式碼中引用模組與套件，我們可以透過程式碼 73 中的三項主要方式。

```
import 模組或套件名稱
import 模組或套件名稱 as 別名
from 模組或套件名稱 import 子模組, 套件或函數名稱
```

程式碼 73: 模組或套件的主要引入方法

15.2 內建模組

Python 的標準函式庫提供裡許多內建模組[6]，一些常見的包括：

1. os：幫助你管理文件和目錄，像是創建、刪除文件夾或獲取文件資訊。
2. sys：讓你可以存取與 Python 解釋器相關的資訊，例如命令行參數或 Python 的版本。
3. math：包含基本的數學運算，例如加減乘除、開平方或三角函數。
4. datetime：用來處理日期和時間，讓你可以輕鬆計算時間差或改變日期格式。
5. json：幫助你讀寫 JSON 格式的數據，這種格式在網路上交換數據時很常用。
6. random：用來生成隨機數，適合用在需要隨機選擇或測試的場合。

15.3 第三方函式庫

除了本身提供的內建模組之外，Python 社群和開發者提供了大量的第三方函式庫 [7]，它們豐富了 Python 的生態系統，使其成爲功能強大且多用途的程式語言。常見的第三方函式庫包含：

1. NumPy 和 Pandas：非常適合資料科學和資料分析。NumPy 提供高效能的陣列操作，而 Pandas 則提供了易用的資料操縱工具。
2. Matplotlib 和 Seaborn：用於資料視覺化，允許創建多種靜態、動態和交互式的圖表。
3. SciPy：用於科學和技術計算的一個函式庫，提供了優化、線性代數、積分等多種功能。
4. Flask 和 Django：用於 Web 開發的強大框架。Flask 是一個輕量級的 Web 框架，適合小型項目，而 Django 則是適合大型、複雜的 Web 應用。
5. TensorFlow 和 PyTorch：深度學習領域中廣泛使用的函式庫，提供了強大的計算能力，用於構建和訓練神經網絡。

一般而言，這些函式庫都可以透過內建的 pip (或 pip3) 下載至電腦中，我們將會在接下來的「數位決策」課程爲大家進一步介紹下載並使用這些函式庫的方法。

[6]關於內建模組，詳細的介紹請參閱 https://docs.python.org/zh-tw/3/tutorial/stdlib.html
[7]關於第三方函式庫，詳細的索引請參閱 https://pypi.org