mission 單元的用法

mission 單元設計成直接在紙上作答的問題集。答題方法包括填寫編號、或從選項中選擇正確解答，每個 mission 上方會顯示作答方法。

mission 解答請見 P.227。除了確認答案是否正確外，當你答錯時，也要仔細思考原因。

這裡說明了答題方法。請按照紅框內顯示的內容作答。

「達成目標」代表有經驗的人可以在這些秒數內解決問題。請盡量多挑戰幾次，盡力達成目標。

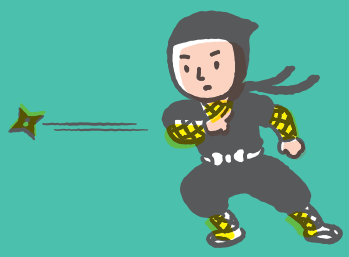
請將答案寫在空白處。

※ 本書同時提供了範例檔（請參考 P.2）及 mission 部分的 PDF 檔。假如你想反覆練習，可以列印出來使用。

1 章

練習前的準備工作

開始練習 Python 之前，請先準備安裝 Python 的程式設計環境。這一章的後半部分也會介紹正式開發程式時使用的工具，熟悉之後請試著挑戰看看。



SECTION 01

Python 的學習重點

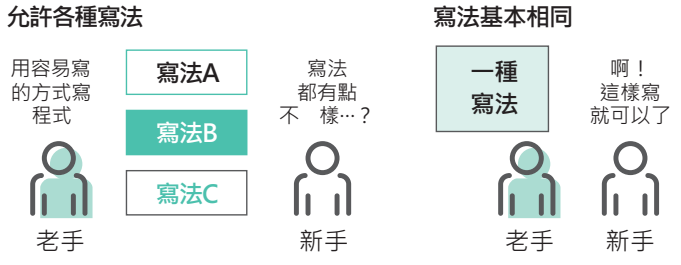
隨著機器學習與資料科學的蓬勃發展，使得 Python 受到矚目，並迅速普及成適合初學者使用的程式設計語言。

重視「易讀性」的程式設計語言

Python 是 1990 年初出現的程式設計語言，卻因為機器學習與大數據的發展，而在 2010 年後半開始受到矚目。有趣的是，Python 不僅在機器學習與大數據等領域受到重視，也常用來當作**初學者的程式設計入門語言**，而且還出現了與 Python 有關的資格認證，愈來愈多學校的教科書也採用了 Python。

Python 獲選成為入門用程式設計語言的原因之　，就是 Python **很重視「程式的易讀性」**。換句話說，Python 的語法簡潔俐落，不論是誰都能看懂，而且**每個人的寫法都一樣**。

其實在程式設計的世界裡，這種特色並非理所當然。有些程式語言的特色是語法十分靈活，允許開發人員依照個人喜好設計程式。愈靈活有彈性，開發效率愈好，可是站在初學者的立場，五花八門的寫法會阻礙學習。基於這　點，Python 成為最適合初學者的程式設計語言。



POINT

Python 並非只有種寫法，而是與其他程式設計語言相比的相對說法。

「重視易讀性」的另　種表現方式是鼓勵**文件化**（製作解說程式的文件）。本書第 9 章介紹的 Python 官方文件就是程式設計語言附屬資料中，最豐富的內容之　。

SECTION 01

使用數值與運算子執行運算

程式是由命令與資料構成的。以下將說明最基本的數值資料及使用該數值執行運算的方法。

資料與命令

程式是下達命令給電腦的命令集合，少了資料就無法執行任何動作。沒有數值無法進行運算，如果想在畫面上顯示任何東西，也需要文字與影像。

程式處理的資料稱作**值**。值包括各式各樣的種類，以下要說明的是最基本的**數值**與**字串**。

忍者只能聽命行事，希望可以多點彈性

數值與字串

數值包含沒有小數點的**整數（int 型的值）**以及包含小數點的**浮點數（float 型的值）**。字串是由文字構成的資料，前後會加上**單引號（'）**或**雙引號（"）**。

以下將試著編寫並執行只有數值與字串的程式。

POINT

值的種類稱作型（資料型），可以分成 int、float、str 等型名。

c2_1_1.py

```
001  64        整數
002  3.25      浮點數
003  'Hello'   字串
```

執行之後沒有出現錯誤，由此可知程式沒有問題，不過畫面上不會顯示任何結果。如果要在畫面上顯示結果，必須使用**print 函數**。請在數值及字串前面輸入「print(」，後面再輸入「)」，接著執行程式。

POINT

執行程式時，請參考 P.19 說明的方式，先建立新檔案，輸入程式並存檔後再執行。

SECTION 04

資料有不同型態

「字串」、「數值」等資料種類稱作「型」。注意資料的型態，可以在程式內處理不同種類的資料。

何謂資料的「型」

程式內的值都有各自的**型**。例如前面處理過的字串是 **str 型**，數值是 **int 型**。Python 還有其他各種型態的資料。

str 是 string 的縮寫，int 是 integer 的縮寫喔

int 型（整數型）	float 型（浮點數型）
沒有小數點的數值 1　202　-5	包含小數點的數值 0.5　-1.5　1.41421

str 型（字串型）	bool 型（真假值型）
以「'」或「"」包圍的字串 'Hello'　'忍者'　"文字"	正確或錯誤 True　False

每種型都有慣用的運算子。例如在運算子 + 的左右輸入 int 型的數值與 float 型，兩者會相加，在運算子 + 左右輸入 str 型的字串，會把字串連接起來。

可是如果在運算子 + 的左右輸入 int 型與 str 型的值，會發生錯誤。

POINT

右側執行結果第 2 行的「line 1」是指發生錯誤的行數。錯誤訊息通常會包含解決問題的線索，出現時請仔細檢視。

數值之間可以進行計算

```
print(1 + 10)
```

執行結果

```
11
```

數值型與字串型無法進行計算

```
print('1' + 10)
```

執行結果

```
Traceback (most recent call last):
 File "<pyshell#0>", line 1, in <module>
  '1' + 10
TypeError: can only concatenate
                  str (not "int") to str
```

這是說明 str 型的值只能與 str 型連結的錯誤訊息

mission 2-05

達成目標 40 秒

填寫增量賦值陳述式的結果

請檢視程式，寫出最後會顯示的計算結果。

```
year = 2000
year += 21
print(year)  2021
```

1
```
i = 0
i += 1
print(i)
```

2
```
num = 10
num -= 5
print(num)
```

3
```
num = 10
num /= 5
print(num)
```

4
```
num = 10
num += 5 * 2
print(num)
```

5
```
text = '山'
text += '川'
print(text)
```

6
```
price = 1000
discount = 100
price -= discount
print(price)
```

※從「#」開始到換行為止是註解陳述式（請參考下一頁的問題）。註解陳述式是寫在程式內的筆記，Python 直譯器會忽略這個部分，不會影響執行結果。

達成目標 50 秒

選擇必須轉換型態的變數

請在必須用 int 函數轉換成數值型，或以 str 函數轉換成字串型的變數下方做標記。

```
year_text = '2021'
wareki = year_text - 2018
```

1
```
# 連結數值與字串
num = 10
text = num + '個'
```

2
```
# 連結數值與字串
text = '10'
num = text + 20
```

3
```
# 連結數值與字串
price = 1500
text = '價格'
text += price + '元'
```

4
```
# 價格與稅金加總
# 與字串連結
price = 1500
tax = 150
price += tax
print(price + '元')
```

5
```
# 根據價格與折扣計算折扣後的價格
price = 1080
discount_rate_text = '2'
price -= price * discount_rate_text / 10
print(price)
```

SECTION 01

呼叫函數與方法

學會如何使用整合一連串處理的「函數」及針對值進行處理的「方法」，就能對電腦下達各種命令。

呼叫函數

前面出現過 print 函數及 int 函數等名詞，這些都是整合　連串處理的**「函數」**，你也可以單純想成「這是執行某項處理的命令」。

在程式中執行函數稱作「呼叫」函數

int 函數是把輸入括弧內的字串或數值轉換成 int 型的函數，括弧內指定的值稱作函數的**參數**，執行結果稱作**傳回值（或回傳值）**。

```
num = int('100')
```

將傳回值賦值給變數　函數　參數

上面的程式是把參數（'100'）傳遞給 int 函數，並將執行後的傳回值（100）代入變數 num。如此　來，可以把傳值的函數當作程式中的值處理。

呼叫方法

方法和函數　樣，是取得參數，執行　連串處理的機制。

方法是針對某個值進行處理，寫法為**「值 . 方法名稱 ()」**。

以下程式是對賦值 str 型值的變數 text，執行 find 方法，以數值傳回特定字串位置。

```
text = 'ninja'
print(text.find('j'))
```

值　方法　參數

POINT

方法可以視為 int 型、str 型等資料型態的函數。

SECTION 04

用 if 陳述式執行不同處理

把前面學到的條件判斷與 if 子句及 else 子句結合，就可以利用條件真假執行不同處理。

if 陳述式的寫法

if 陳述式是只有符合某個條件時，才執行後面處理的語法。

前面說明過輸入條件，取得判斷 True 或 False 的方法，但是**判斷條件搭配 if 陳述式，可以在程式內建立條件式**。

if 陳述式是在關鍵字 if 後面輸入半形空格，接著輸入成為條件的表達式，最後加上冒號。

關鍵字 if　表達式　冒號

```
if␣2019 <= birth_year:
␣␣␣␣執行處理
```

從條件的下　行開始輸入符合條件時要執行的處理。這個部分會**在行頭插入 4 格半形空格**。這種在程式的行頭插入空格的方式稱作**縮排**。在 IDLE 的編輯視窗中，按下 Tab 鍵，會插入 4 格半形空格。

Python 會根據縮排定義處理範圍，因此符合 if 條件時想執行的處理都要統　縮排。

請編寫並執行取得使用者輸入的西元出生年份，數值為 2019 以上就顯示「令和」，否則不執行處理，不論結果為何，最後都會顯示「結束判斷」的程式。

if 陳述式是「如果（條件式）怎樣就執行以下處理」的命令

參考網址

if 陳述式
https://docs.python.org/zh-tw/3.10/reference/compound_stmts.html#the-if-statement

縮排對 Python 而言是非常重要的規則，請先記下來

SECTION 02

用切片取出部分資料

依照順序管理的資料如列表、元組等，可以利用「從幾號開始到幾號為止」的形式指定想要的資料，單獨取出其中一部分。

切片是從列表或元組中，取出部分資料的記法。在 []（角括弧）中，輸入開始切片的索引（start）、結束切片的索引（end）、每隔幾個取出元素（step）等三個部分，並用 :（冒號）分隔。

```
[start : end : step]
```

開始切片的索引　結束切片的索引（※）　每隔幾個取出元素

※…要取出的是到 end 前　個索引為止，因此切片取出的元素是到 end-1 索引為止

值可以省略。省略 start 時，會從開頭的索引開始，省略 end 是到末尾索引為止，省略 step 是每隔　個就取出資料。

以下是建立整合中文字串的列表，切片取出從索引 4 到末尾的程式。

c4_2_1.py

```
001 letters = ['花', '開', '豔', '麗', '終', '須', '落']
002 print(letters[4:])                    只設定 start
```

執行結果

```
['終', '須', '落']
```

請練習調整第 2 行的切片設定。只設定 end，會從頭開始到設定值 -1 的索引為止進行切片。

```
003 print(letters[:4])                    只設定 end
```

列表的切片會變成列表，元組的切片會變成元組喔

POINT

列表及元組屬於後面要說明的序列，可以用切片取出其中　部分。

執行結果

```
    前面省略
['花', '開', '豔', '麗']
```

設定 step，可以改變每隔幾個取出元素。例如把 step 設定成 2，代表每隔 2 個取出元素。

```
004  print(letters[::2])                    只設定 step
```

執行結果

```
    前面省略
['花', '豔', '終', '落']
```

此外，**在索引輸入負數，會從末尾開始依序計數**。如下圖所示，letters 列表的元素索引有兩種表現方法。

'花'	'開'	'豔'	'麗'	'終'	'須'	'落'
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

利用這一點，在 start 輸入負數，可以執行「從末尾開始取出第〇個字」的操作。

```
005  print(letters[-3:])
```

執行結果

```
    前面省略
['終', '須', '落']
```

在 step 設定負數，會從末尾往前倒數元素。將 step 設定成 -1 時，會以反方向顯示元素。

```
006  print(letters[::-1])
```

執行結果

```
    前面省略
['落', '須', '終', '麗', '豔', '開', '花']
```

POINT

在 end 設定索引 4 是指取出到前面的索引 3 為止的元素。

如果需要末尾的元素，只要設定 -1 即可，不用數字數

POINT

step 當然也可以設定成 -1 以外的負數。設定成 -2 時，從末尾開始依相反順序，每隔 2 個取出元素。

mission 4-02

達成目標 300 秒

寫出切片結果

請檢視程式，寫出最後顯示的輸出結果。
假設所有範例都是對 alphabets 列表進行切片。

```
alphabets = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
print(alphabets[1:5])
['b', 'c', 'd', 'e']
```



```
1 print(alphabets[4:])
```

```
2 print(alphabets[2:5])
```

```
3 print(alphabets[:-5])
```

```
4 print(alphabets[2:-2])
```

```
5 print(alphabets[:4:2])
```

```
6 print(alphabets[1::3])
```

```
7 print(alphabets[2::-1])
```

```
8 print(alphabets[4:1:-1])
```