Python
技術者們 實踐!

第 4 章函式 Function

4-0 設計自己的函式

- 定義函式要使用 def, 語法如下：

```
def hello():          ← 定義沒有參數的函式 hello, 最後別忘了加 ":"
    print('Hello!') ← 函式的內容

def sayHi(name, title): ← 此函式有 2 個參數 (代表姓名和頭銜)
    print(name + title + ' 你好！') ← 函式的內容

hello()                  輸出 Hello!
sayHi('王小明', '同學')   輸出 王小明同學 你好！
```

4-1 參數的傳遞與傳回值

- 位置參數法與指名參數法

呼叫函式時的參數值傳遞方式

```
sayHi('王小明', title='同學')          ← 第 2 個參數值用名稱指定
sayHi(title='同學', name='王小明')     ← 全部參數值都用名稱指定，可以不照順序
sayHi(title='同學', '王小明')          ← 錯誤了！因為'王小明'並未指定參數名，所以
                                         不是指名參數而是位置參數，要放在前面才行
```


我們怎麼知道參數名?

除了 help(函式名) 之外，也可使用第 0-3 節介紹的 3 種方法來得知函式的參數資訊：使用輔助說明窗格、在 IPython 窗格中執行「?函式名」、或使用 Spyder 的智慧輔助輸入功能。

一步一腳印

```
In [1]: help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.
stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

In [2]: help(len)
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

就如之前學過的 sep 和 end 都必須是指名參數

這樣表示可以傳入多個 value(至少一個)

/ 表示不接受指名參數

指定參數的預設值

```
def calc(w, h, d=1):  ← 深度若省略, 預設為 1
    return w * h * d

print(calc(3, 4))       # 算面積  輸出  12
print(calc(3, 4, 5))    # 算體積  輸出  60
```


- 有預設值的參數必須定義在最後，這樣才不會影響到其他參數的位置。

```
def sayHi(name, title='先生', hi='你好'): ← 最後 2 個參數有預設值
    print(name + title, hi)

print(sayHi('王小明', hi='好久不見')) # 傳入第 1 個位置參數、然後指名 hi 參數
                                     # title 參數沒傳入，所以使用預設值
                                     # ('先生')
```

位置參數　指名參數

輸出

```
王小明先生 好久不見
```

使用 return 來傳回物件

- 用 return xxx 來結束函式並傳回 xxx 物件, 若 xxx 省略則傳回 None

```
def calc(w, h):        # w,h 代表寬、高
    if(w<=0 or h<=0):
        return ← 結束函式, 無傳回值
    return w*h ← 結束函式並傳回面積

print(calc(3, 4)) 輸出 12
print(calc(3,-4)) 輸出 None ← 函式無傳回值時, 會傳回 None
```

```
def calc(w, h):
    return ((w+h)*2, w*h)      # 傳回 (周長, 面積) 的 tuple

print(calc(3,4))    # (14, 12) ← 傳回值是一個 tuple
```

- 在第 2 章末的補充學習中介紹的「變數多重指定」及「容器自動解包與打包」，也都適用於函式的傳回值上

```
def calc(w, h):
    return [((w+h)*2, w*h), '正方形' if w==h else '長方形']

a = calc(4,5) ←— 用一個變數名來接收傳回值
print(a)                  ➡ [(18, 20), '長方形']
((a,b),c) = calc(4,5) ←— 用 2 維 tuple 的變數來接收傳回值
print(a, b, c)            ➡ 18 20 長方形
```


函式和變數、容器一樣都是物件

```
def calc(w, h):       # 定義計算面積的函式
    return w * h

a = calc              # 將 a 綁定到函式物件
print(a(2, 3))        # 用 a() 一樣可以呼叫到 calc() 函式   輸出 6
```

把函式當成參數來傳遞

```
s = [(3, 4), (2, 4), (5, 3)]    # s 為包含多組（寬，高）的容器

def calc(w, h):                 # calc()將被當成參數傳入 calcAll()
    return w * h

def calcAll(conta, func):  ← 參數為容器(例如 s)及函式物件(例如 calc)
    for r in conta:                          # 走訪容器
        print(func(r[0], r[1]), end=' ')     # 以元素的寬、高為參數呼叫
                                               func 函式
calcAll(s, calc) ← 把容器 s 及函式 calc 傳入 calcAll()
```

輸出

```
12 8 15
```

這個程式的概念就是把一個容器 _A 和一個函式 _B 傳入另一個函式，然後在函式中用 for 迴圈一一取出容器 _A 中的元素（為一個 tuple）交給函式 _B 來處理。

4-2 不定數目的參數 *args 和 **kwargs

- 用 *args 接收不定數目的位置參數
  - 在定義函式時，可以用「*args」來接收不定數目的位置參數

```
def prnSum(name, *args):    # 不定數目參數通常會以 args 為名
    print(name, args, '=', sum(args))

prnSum('加總', 1, 2, 3, 4)  輸出  加總 (1, 2, 3, 4) = 10
```

這 4 個參數會打包成 tuple 指定給 args

- 在呼叫函式時，所有定義在「*參數」前面的參數都不可省略，而定義在「*參數」後面的參數則只能是指名參數或 **kwargs (見後文)

```
def prnSum(name, pre='>', *args, post='#'):
    print(name, pre, args, '=', sum(args), post)
prnSum('加總', ':', 1, 2, 3, 4, post='元')
```

輸出 加總 : (1, 2, 3, 4) = 10 元

前 2 個參數不可省略

post 要以關鍵字指定(指名)

使用 **kwargs 來接收不定數目的指名參數

- 除了用「*args」接收不定數目的位置參數之外，還可用「**kwargs」來接收不定數目的指名參數

```
def prnPrice(name, **kwargs):      ← 不定數目的關鍵字(指名)參數
    print(name, kwargs)                通常會以 kwargs 為名

prnPrice('飲料', 紅茶=40, 咖啡=70, 果汁=85)   ← 呼叫時使用指名參數法

輸出                                  所有關鍵字參數被打包成dict

飲料 {'紅茶': 40, '咖啡': 70, '果汁': 85}   ← 以 dict 的形式印出來了
```


在呼叫函式時, 可用 * 、** 將容器解包

- 除了定義函式時使用 * 和 ** 參數, 如果在呼叫函式時使用 * 及 **, 則作用完全相反, 會由打包 (packing) 變成解包 (unpacking)
- 「*容器」會解包成「元素1, 元素2, …」。
- 「**字典」會解包成「鍵1 = 值1, 鍵2 = 值2, …」。

前面無關鍵字的位置參數會打包成 tuple 給 args

後面有關鍵字的指名參數打包成 dict 給 kwargs

```
def prnPrice(name, *args, **kwargs):
    print(name, args, ':', kwargs, sep='')

dscnt = ('早餐 8 折', '消夜 9 折')
drink = {'紅茶': 40, '咖啡': 70, '果汁': 85}
prnPrice('飲料', *dscnt, **drink)
```

② 會將 drink 的 dict 解包成紅茶=40, 咖啡=70, 果汁=85 3 個指名參數

輸出

呼叫函式時, ① 會將 dscnt 的 tuple 解包成 '早餐8折', '消夜9折' 2 個位置參數

```
飲料('早餐 8 折', '消夜 9 折'):{'紅茶': 40, '咖啡': 70, '果汁': 85}
```

4-3 lambda 匿名函式

- 函式如果很簡短, 只需要參數列和傳回值 (例如之前的 calc), 那麼就可以簡寫成一個 lamdba 運算式, lambda 的語法如下

```
lambda 參數1, 參數2, ... : 傳回值
```

```
s = [(3, 4), (2, 4), (5, 3)] # s 為包含多組 (寬, 高) 的容器

def calcAll(rects, func):     # 參數為：容器(例如 s) 及函式物件
    for r in rects:                          # 走訪容器
        print(func(r[0], r[1]), end=' ') # 以元素的寬、高為參數呼叫 func 函式

calcAll(s, lambda w, h: w*h)   執行 12 8 15
```


- lambda 運算式由於不需要替函式取名，因此也稱為匿名函式。不過在必要時還是可以給它綁定一個名字

```
calc = lambda w, h: w*h    # 等同於使用傳統方式定義 calc
print(calc(3, 4))          # 同樣可以當成函式來呼叫    12
```


- lambda 決定了每個元素的計算方式, 例如底下增加計算每個矩形的周長

```
calcAll(s, lambda w, h: w*h)       # 計算面積
calcAll(s, lambda w, h: (w+h)*2)   # 計算周長
```

```
14 12 16
```

- 另外例如內建函式 sorted() 也可多加一個「key=函式物件」參數，那麼 sorted() 就會將容器的元素一一傳給此函式，而此函式則須傳回該元素用來排序的值

```
s = [(3, 4), (2, 2), (5, 3)]
print(sorted(s, key=lambda e: e[1])) ← 用每個元素的第 1 個(由 0 算起)
↓                                      子元素來排序
(2, 2), (5, 3), (3, 4) ← 傳回的結果是依 2、3、4 的順序排列
```

4-4 變數的有效範圍 Scope Rule

- Python 的變數分為全域變數 (Global riable) 及區域變數 (Local variable) 二種。

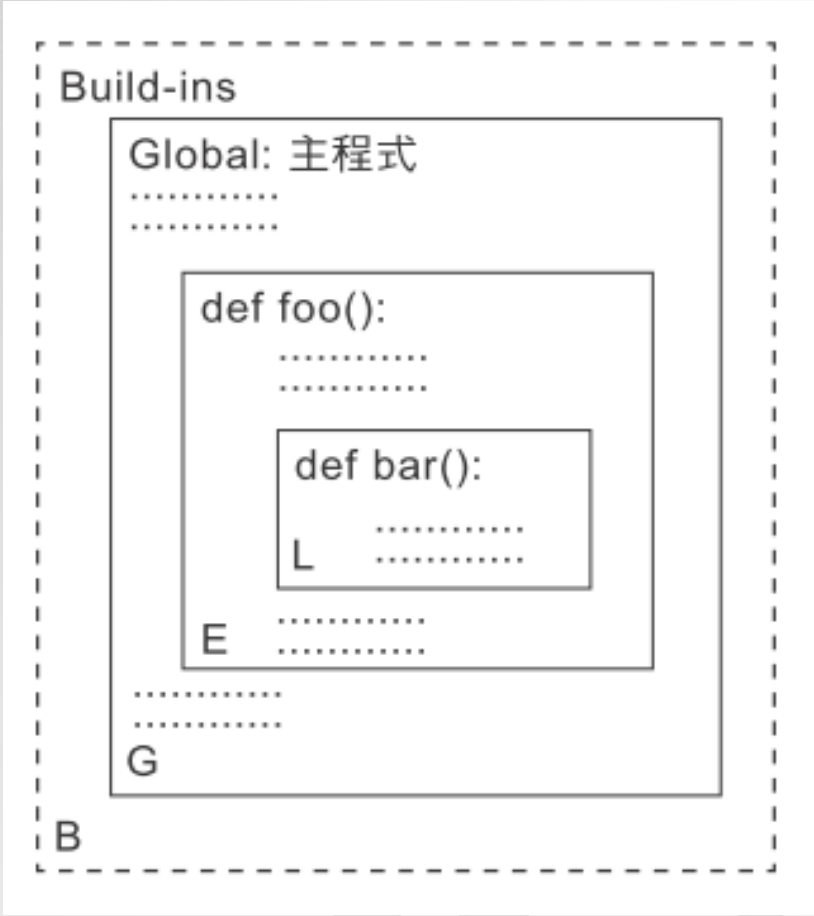
```
a = b = c = 1          ←— 建立 a、b、c 三個全域變數

def test(b):           ←— 參數 b 為區域變數
    a = 2              ←— 建立區域變數 a
    print(a, b, c)     ←— 輸出區域變數 a、b 及全域變數 c

test(3)    輸出 2 3 1 ← 呼叫 test(), 在函式中會輸出區域變數 a、b 及全域變數 c
print(a, b, c) 輸出 1 1 1 ← 輸出全域變數 a、b、c
```

LEGB 規則

- Python 的 scope rule 是依照 LEGB 的順序來尋找名稱 (name) 的 (包含變數和函式名)


Build-ins
Global: 主程式
.....
.....
def foo():
.....
.....
def bar():
.....
L .....
.....
E .....
.....
.....
G
B

global 宣告:
在函式中變更全域變數值

```
a = b = c = 1

def test(b):
    global c          ← 在定義函式時,指明要使用全域變數 c
    a = 2             ← 這時的 a 並非上層的全域變數, 而是新建立的區域變數
    c = 33            ← 因為已經宣告了 global c, 所以這個 c 是指上層的
                        全域變數, 而非新建一個區域變數
    print(a, b, c)    ← 輸出區域變數 a、b 及全域變數 c

test(3)           輸出 2 3 33 ← 呼叫 test(3)會輸出區域變數 a、b 及全域變數 c
print(a, b, c)    輸出 1 1 33 ← 全域變數 c 的值被改為 33 了
```

容器物件的 scope rule

- 如果變數名是綁定到像 list 這種「元素可更改」的容器, 那麼無論在一般程式或函式中, 都可以透過變數名 (容器名) 來更改容器的元素

```
s = [1, 2, 3]          # 在最上層程式中建立串列 s
t = [4, 5, 6]          # 在最上層程式中建立串列 t

def test(a):
    a[0] = 'aaa'
    t[0] = 'ttt'
    s = [7, 8, 9]
    s[0] = 'sss'
    print(a, t)

test(s)      # 呼叫 test 並傳入串列 s
print(s, t) # 輸出串列 s、t 的內容
```

全域串列名 s → [1][2][3]
區域串列名 a →

← 若以 test(s) 呼叫時，參數 a 會綁定到傳入的串列 s
← 透過區域變數(參數) a 更改串列 s 的元素
← 在函式內透過全域變數 t 直接更改串列 t 的元素
← 建立一個同名區域變數 s 並綁定到一個新串列
← 更改區域變數 s 所綁的串列元素，不影響全域串列 s、t
← 輸出全域串列 s、t 的內容

輸出 ['aaa', 2, 3] ['ttt', 5, 6]
輸出 ['aaa', 2, 3] ['ttt', 5, 6]