

# Python 程式設計

林奇賦 [daky1983@gmail.com](mailto:daky1983@gmail.com)

# Outline

- ▶ 內建函數
- ▶ 自訂函數

# 呼叫函式的基本方法

`Function_name ( Arg1, Arg2, ... )`

命名的方法如同變數一般

給予引數

- 範例：`print ( "Hello, world" )`

函式名稱後面跟隨著小括號

# 內建函數 (Build-in Function)

- ▶ 內建函式可以直接使用
- ▶ 內建函式有哪些：
  - ▶ `dir( __builtins__ )`
  - ▶ `iterator: range()`
- ▶ Length: `len()`
- ▶ Sort: `sorted()` v.s. `object.sort()`
- ▶ 轉換型態或建立物件
  - ▶ `int()`
  - ▶ `str()`
  - ▶ `float( )`
  - ▶ `list()`
  - ▶ `set()`
  - ▶ ...

# 內建函數 (Build-in Function)

## Built-in Functions

<a href="#">abs()</a>	<a href="#">dict()</a>	<a href="#">help()</a>	<a href="#">min()</a>	<a href="#">setattr()</a>
<a href="#">all()</a>	<a href="#">dir()</a>	<a href="#">hex()</a>	<a href="#">next()</a>	<a href="#">slice()</a>
<a href="#">any()</a>	<a href="#">divmod()</a>	<a href="#">id()</a>	<a href="#">object()</a>	<a href="#">sorted()</a>
<a href="#">ascii()</a>	<a href="#">enumerate()</a>	<a href="#">input()</a>	<a href="#">oct()</a>	<a href="#">staticmethod()</a>
<a href="#">bin()</a>	<a href="#">eval()</a>	<a href="#">int()</a>	<a href="#">open()</a>	<a href="#">str()</a>
<a href="#">bool()</a>	<a href="#">exec()</a>	<a href="#">isinstance()</a>	<a href="#">ord()</a>	<a href="#">sum()</a>
<a href="#">bytearray()</a>	<a href="#">filter()</a>	<a href="#">issubclass()</a>	<a href="#">pow()</a>	<a href="#">super()</a>
<a href="#">bytes()</a>	<a href="#">float()</a>	<a href="#">iter()</a>	<a href="#">print()</a>	<a href="#">tuple()</a>
<a href="#">callable()</a>	<a href="#">format()</a>	<a href="#">len()</a>	<a href="#">property()</a>	<a href="#">type()</a>
<a href="#">chr()</a>	<a href="#">frozenset()</a>	<a href="#">list()</a>	<a href="#">range()</a>	<a href="#">vars()</a>
<a href="#">classmethod()</a>	<a href="#">getattr()</a>	<a href="#">locals()</a>	<a href="#">repr()</a>	<a href="#">zip()</a>
<a href="#">compile()</a>	<a href="#">globals()</a>	<a href="#">map()</a>	<a href="#">reversed()</a>	<a href="#">__import__()</a>
<a href="#">complex()</a>	<a href="#">hasattr()</a>	<a href="#">max()</a>	<a href="#">round()</a>	
<a href="#">delattr()</a>	<a href="#">hash()</a>	<a href="#">memoryview()</a>	<a href="#">set()</a>	

# 自訂函數

- Programmer可以自行設計與定義函式
- 自定義函式的方法 → 使用關鍵字def進行函式定義

```
def Function_name (Pra1, Pra2, ... ):  
    suite
```

- 必先定義而後使用 → 任何 suite 必定縮排!

# 函數的命名

- 函數之命名慣例:
  - GUI函式習慣以駝峰式(Camel-Case)命名
  - 一般函式習慣以小寫或輔以底線命名
  - 避免非慣用縮寫
- 函數之命名應有意義, 且須指出函數之目的或回傳之資料, 而非達成目的所使用的方法

# 參數與引數

- 參數與引數的差別 (很容易混淆)
- **Parameter** / 參數
  - 函式運行時之**參**考
  - 放在函式的標記式, 用來說明這個函式, 當它被呼叫時必須接收到什麼樣的資料
- **Argument** / 引數
  - 用以**引**發函式
  - 當你呼叫函式的時候, 你可以放在括號內的東西



# 傳遞引數給參數 1/2

- 以引數呼叫函式而將其值賦予參數以供函式 (副程式) 運行參考

```
Function_name ( Arg1, Arg2, ... )
```



```
def Function_name ( Pra1, Pra2, ... )
```

- 依序一個對一個 (只是一個丟接遊戲!), 這種依參數順序給定的引數稱為 **positional argument**

# 傳遞引數給參數 2/2

- 傳遞的動作可以想像成若干個賦值運算

```
Function_name ( Arg1, Arg2, ... )
```



```
def Function_name ( Pra1, Pra2, ... )
```

- Pra1 = Arg1
- Pra2 = Arg2
- ...
- PraN = ArgN

# 函式之回傳值

- ▶ 一個函式於運行完畢之後可以回傳結果(也可以忽略不回傳)
- ▶ 當程式運行到return陳述句時將會無條件直接離開(結束)函式並回傳結果
- ▶ Return Statement:

```
return something
```

- ▶ 回傳值可以為:
  - ▶ 單一的值或物件
  - ▶ 多個值或物件所構成的 tuple
  - ▶ 當未使用return陳述句時,預設將會回傳None

# 參數預設值 1/2

- 參數預設值可減少 **function call** 撰寫上面的麻煩

```
def Function ( Pra1=value1, Pra2=value2, ... ):
```

- 即便呼叫函式時沒有給定引數, 函式依然有參考值 (預設參數值)
- 問題思考:
  - 如果有的有給引數有的沒有呢?
  - 如果有的有參數預設值有的沒有呢?

# 參數預設值 2/2

- Rule :當自定義函式時
  - 無預設值之參數在前
  - 有預設值之參數在後
  - 原因：引數之傳遞是依序的!

```
def Func ( Pra with no default | Pra with default ):
```

- Rule: 當有預設值之參數仍有引數之分配時, 引數值將會取代預設值

# 函數的範例 EX04\_01.py

- ▶ 將 for 迴圈範例 EX02\_05.py 改寫成函式的寫法。
- ▶ 觀察函式的回傳值
- ▶ 改寫成具有回傳值的版本
- ▶ 必定先定義而後呼叫

# 參數預設值

- 參數預設值可減少 **function call** 撰寫上面的麻煩

```
def Function ( Pra1=value1, Pra2=value2, ... ):
```

- 即便呼叫函式時沒有給定引數,函式依然有參考值 (預設參數值)
- Rule :當自定義函式時
  - 無預設值之參數在前
  - 有預設值之參數在後
  - 原因: 引數之傳遞是依序的!

```
def Func ( Pra with no default | Pra with default ):
```

- Rule: 當有預設值之參數仍有引數之分配時, 引數值將會取代預設值

# 利用參數名稱之函式呼叫

- 呼叫函式時不使用參引數依序一對一對應傳遞，換成以參數名稱來進行傳遞與函式之呼叫 (名稱呼叫)
- 這種依名稱傳遞的引數稱為 **keyword argument**

```
Function_name ( b=value1, a=value2, ... )
```

```
def Function_name ( a, b, ... ):
```





# 參數預設值範例

- ▶ 函數的預設值如果是(**mutable**)可變類型的物件，並且在函式執行期間修改了這個可變物件，此預設值的物件也會因此跟著改變，之後再次呼叫同個函式時將會用改變後的值為預設值。
- ▶ 參考範例 `EX04_02.py`

# 變數的有效範圍

- ▶ 每個變數有其有效範圍,也就是被認定(認識)的範圍,一般定義在函式外的變數其範圍是一整個模組,又稱**全域變數**(全域指的是整個模組)
- ▶ 被定義在一個函式中的變數稱為**區域變數**,其範圍是一整個函式
- ▶ 以上兩種範圍都必須**從變數定義被python看到開始算起**
- ▶ 當區域變數與全域變數名稱相同產生衝突時,區域內以區域變數為主,區域外以全域變數為主

# 變數的有效範圍

- ▶ 範例 EX04\_01.py 中，summation函式中的變數 total, i 皆為區域變數(local variable)。
- ▶ 一旦離開區域範圍則無法存取此變數

# 全域變數

- ▶ • 有的時候在函式中我們希望改變或存取的是全域變數時, 為了確認使用到正確的全域變數, 而非定義一個新的區域變數時, 我們需要利用 `global`敘述句來告知 `python` 某變數名稱指稱的是全域變數

`global` variable name

# 全域變數範例

- ▶ 範例 EX04\_06.py

# 不定個數形式的參數

- 函數 (function) 可以有不定個數的參數 (parameter) , 也就是可以在參數列 (parameter list) 提供任意長度的參數個數

```
def Function ( *arguments, **keywords ):
```

- \*arguments 就是參數識別字 (identifier) 前面加上一個星號 , 當成一組序對 (tuple)
- \*\*keywords 參數識別字前面加上兩個星號 , 當成一組字典 (dictionary)

# 不定個數形式的參數範例

- ▶ EX04\_03.py
- ▶ EX04\_04.py
- ▶ 在傳遞引數時，可用\* 將序列型態物件解開傳入
- ▶ 在傳遞引數時，可用\*\* 將字典型態物件解開傳入

# 函數 yield 產生器

- ▶ 函數 (function) 中若使用 **return** ，函數會直接回傳數值 (value) ，也隨之**終止**函數執行。
- ▶ 若使用另一個關鍵字(keyword) **yield** ，可使函數產生數值，隨之**中斷**執行，下次透過next()函數呼叫此函數時會繼續從上次中斷的位置繼續執行，這樣的函數被稱為產生器函數 (generator function)
- ▶ 要使用生成器產生新的元素，有兩種方法：
  - ▶ next( )
  - ▶ for 迴圈



# 使用yield模擬range()函式

- ▶ 範例 EX04\_05.py

# Homework 4

- ▶ 試以 [EX02\\_07.py](#) 的程式碼為基礎，撰寫三個函數
  - ▶ `list_zip(city)`
    - ▶ 傳入值為城市名稱可列出所有某城市裡面所有區域的郵遞區號  
ex. 呼叫 `list_zip("台北市")`，則列出所有台北市內所有區域的郵遞區號
  - ▶ `area_to_zip(area)`
    - ▶ 傳入值為區域名稱回傳此區域的郵遞區號
    - ▶ ex. 呼叫 `area_to_zip("信義區")`，回傳 ('台北市',110)，('基隆市',201)
  - ▶ `zip_to_area(zip)`
    - ▶ 傳入值為郵遞區號回傳區域名稱
    - ▶ ex. 呼叫 `area_to_zip(106)`，回傳 "大安區"