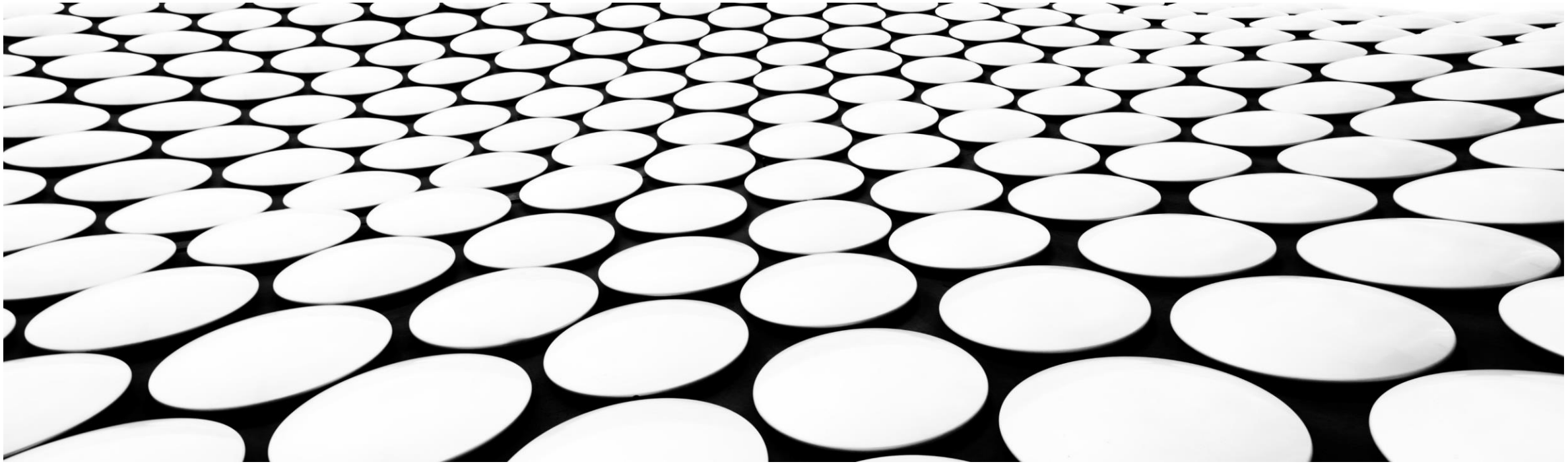
# Python 基本語法 II

MARs教育團隊 – 孫汶琳 交大電機博士候選人



# 目錄

# 函式(Functions)

- 一個只有當被呼叫(call)時才會運行的程式區段
- 需要先被定義(definition)才能被使用
- 我們可以傳入資料到函式做處理，這類資料我們稱作「參數」
  - 在函式定義時的傳入資料英文是: parameters
  - 在函數呼叫時的傳入資料英文是: arguments
  - 在Python的文件中，常用 **arg** 來做傳入的參數簡寫
- 亦可以將在函式內處理過後的資料回傳到主程式，透過「回傳」(return) 這個指令

```
# 這是一個函式的定義
def my_function():
  print("Hello from a function")

# 這是一個函式的呼叫
my_function()
```

# 自定義函式(Self-Defined Functions)

- 自行定義的函式，包含函式名稱、內容、參數、回傳值
- 函式的基本語法:

1. 沒有參數

```
def 函式名稱() :
    # 欲執行之程式片段
```

2. 有固定參數

```
def 函式名稱(arg1, arg2) :
    # 欲執行之程式片段
```

```
# 沒有參數
def my_function():
  print("Hello from a function")

# 這是對應的函式呼叫
my_function()
```

```
# 有固定參數
def my_function(a, b):
  print(a + b)

# 這是對應的函式呼叫
my_function(10, 12)
```

# 自定義函式(Self-Defined Functions)

3. 呼叫函式時用參數名稱來給值
   - 因為是用具名的方式給參數，所以傳參數時不需照定義時的順序

```
函式名稱 (arg1 = ..., arg2 = ...)
```

```
# 有固定參數
def my_function(a, b):
 print(a + b)

# 用具名的方式給參數
my_function(b = 12, a = 10)
```

4. 預設參數值
   - 因為有預設參數值了，所以呼叫時可以不用給值
   - 沒有預設參數值的參數，呼叫時還是要給值唷!

```
def 函式名稱(arg1 = ...) :
    # 欲執行之程式片段
```

```
# 有預設參數值
def my_function(a = 10):
  print(a)

# 這是對應的函式呼叫
my_function(12)
my_function()
```

# 自定義函式(Self-Defined Functions)

5. 有任意數量之參數
   - 因為是用具名的方式給參數，所以傳參數時不需照定義時的順序

```
def 函式名稱 (*args) :
 # 欲執行之程式片段
```

```
# 有任意數量參數
def my_function(*nums):
 print(nums[1])

# 可傳入任意數量的參數
my_function(10, 11, 12)
my_function(10, 11)
```

6. 有任意數量且具名之參數
   - 因為有預設參數值了，所以呼叫時可以不用給值
   - 沒有預設參數值的參數，呼叫時還是要給值唷!

```
def 函式名稱(**kwargs) :
    # 欲執行之程式片段
```

```
# 有預設參數值
def my_function(**nums):
  print(nums["a"])

# 可傳入任意數量的具名參數
my_function(a = 10, b = 11)
my_function(a = 10)
```

# 自定義函式(Self-Defined Functions)

- 當函式運算完後，常會需要傳回運算結果到呼叫的地方，這時我們使用 **return** 這個指令

```
# 寫個函式計算 a+b
def my_add(a, b):
  return a + b

# 可以用變數去接
c = my_add(10, 12)
print(c)

# 也可以直接當作print的參數使用
print(my_add(10, 12))
```

# 內建函式(Built-In Functions)

- 預設的函式，不須額外定義或導入(import)模組，即可使用
- 一般為常用的函式，像是 input() 和 print()
- 官方文件中有列出所有的內建函式: 連結，以下整理出其中較為常用的函式:
  - 轉型用的 int(), float() 等，以及檢查型態用的 type() 就不特別列出了

| 函式 | 說明 | 範例 | 範例結果<br>L = [1, 9, 5, 7, 3] |
|---|---|---|---|
| **abs()** | 計算絕對值 | abs(-1) | 1 |
| **pow()** | 計算指數 | pow(2, 3) | 8 |
| **len()** | 回傳序列長度 | len(L) | 5 |
| **max()** | 回傳序列最大值 | max(L) | 9 |
| **min()** | 回傳序列最小值 | min(L) | 1 |
| **sum()** | 回傳序列總和 | sum(L) | 25 |
| **sorted()** | 回傳排序好的序列 | sorted(L) | [1, 3, 5, 7, 9] |

# 模組(Modules)

- 若想要使用別人定義的函式，也就是外部函式(External Functions)，則需導入外部模組
- Python的模組也就是其他語言函式庫的概念
- 所謂的函式庫就是一個包含一些你想要在你的程式內使用的函式之檔案
- 想要將模組導入自己的程式內，則需要用到導入指令 **import**
- 除了函式外，亦可定義變數
- 基本使用方式:

**my_module.py**

```
# 寫個函式計算 a+b
def my_add(a, b):
  return a + b
# 定義變數 pi
pi = 3.1415926
```

**my_program.py**

```
# 導入 my_module.py
import my_module

# 使用模組內定義的函式和變數
my_module.my_add(10, 12)
print(my_module.pi)
```

# 其他模組使用方法

1. 另取名稱
   - 若原本的模組名稱太長，亦可以在導入時幫他取別名

**my_program.py**

```
# 導入 my_module.py 並取別名
import my_module as mm

# 使用模組內定義的函式和變數
mm.my_add(10, 12)
print(mm.pi)
```

2. 選擇導入
   - 若不想導入整個模組，亦可選擇部分導入
   - 若用這種方式導入，則使用時不需要再加上模組名稱

**my_program.py**

```
# 選擇導入 my_module.py
from my_module import pi

# 使用模組內定義的變數
print(pi)
```

# 內建模組(Built-In Modules)

- Python 提供多元的內建模組，開發者可以隨時導入來使用
- 官方模組列表: https://docs.python.org/3/py-modindex.html

# 外部模組(External Modules)

- 為了方便開發者安裝並使用其他開發者所開發的模組，Python使用了pip 這個套件(package)安裝工具，並預設於安裝Python時安裝
- Python Package Index (PyPI) 為一開放的套件資料庫，開發者可以上架自己所編寫之模組供其他開發者使用，並可以透過 pip 工具來下載並安裝所選之套件，套件資料庫官方網站: https://pypi.org/

# 使用 pip 安裝外部模組

- 開啟命令提示字元(cmd)或者PowerShell
- 確認是否有安裝 pip

```
python -m pip --version
```

- 更新pip相關工具

```
python -m pip install –upgrade pip setuptools wheel
```

- 從 PyPI 下載安裝
  - 安裝最新版本

```
python -m pip install 欲安裝之套件名稱
```

  - 安裝特定版本

```
python -m pip install 欲安裝之套件名稱==版本號
```

# 常用套件列表

- os
  - 用於進行作業系統級別的操作，像是檔案和資料夾的移動、刪除、複製等
- time & datetime
  - 時間的運算、轉換、格式輸出等
- random
  - 隨機產生整數、小數，打亂串列、從串列中隨機挑選一個物件等
- NumPy
  - 資料處理常用的重要模組
- Matplotlib
  - 用於資料視覺化的重要模組

# 變數範圍 (Scope)

- 一個變數只能在他被創建時的位置以內的範圍使用，常見會造成變數範圍的有函式、if-else、迴圈
- 以Python來說，基本上看到有冒號 **:** 的語法時，就會有新的變數範圍出現
- 要存取變數時會找離自己最近的
- 所謂的變數範圍可以分為兩種:
  - 區域變數 (Local Variables) : 只能在局部區域被存取
  - 全域變數 (Global Variables) : 可以在所有區域被存取

```
def my_func():
    a = 100         # 這是個區域變數
    def my_inner_func():
        print(a)    # 可以! 這在 a 被創建時的範圍以內
    print(a)        # 可以! 這在 a 被創建時的範圍

print(a)            # 錯誤! 這在 a 被創建時的範圍以外

b = 200             # 這是個全域變數
```

# 變數範圍 (Scope) – global 關鍵字的使用 I

- global 關鍵字可以使變數強制變成全域變數

```
def my_func():
    a = 100         # 這是個區域變數
    def my_inner_func():
        print(a)    # 可以! 這在 a 被創建時的範圍以內
    print(a)        # 可以! 這在 a 被創建時的範圍

print(a)            # 錯誤! 這在 a 被創建時的範圍以外
```

```
def my_func():
    global a = 100   # 變成全域變數了
    def my_inner_func():
        print(a)        # 可以! a 是全域變數
    print(a)            # 可以! a 是全域變數

print(a)                # 可以! a 是全域變數
```

# 變數範圍 (Scope) – global 關鍵字的使用 II

- 讓內層範圍也可以使用全域變數

```
a = 100          # 這是個全域變數
def my_func():
    a = 200      # 這是個區域變數，不會更動到全域變數的值
    print(a)     # 200

my_func()
print(a)         # 100
```

```
a = 100          # 這是個全域變數
def my_func():
    global a     # 讓函式內的可以使用全域變數 a
    a = 200      # 這是全域變數
    print(a)     # 200

my_func()
print(a)         # 200
```