

# Python程式設計

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

# Outline

- 函數與資料結構

# 函數與資料結構

- 函數 (function) 物件 (object) 可以執行一些工作，或是進行計算。Python 中定義函數使用關鍵字 (keyword) **def**，其後空一格接函數的識別字 (identifier) 名稱加小括弧，然後冒號，如
- `def function_name():`  
    `#do something`

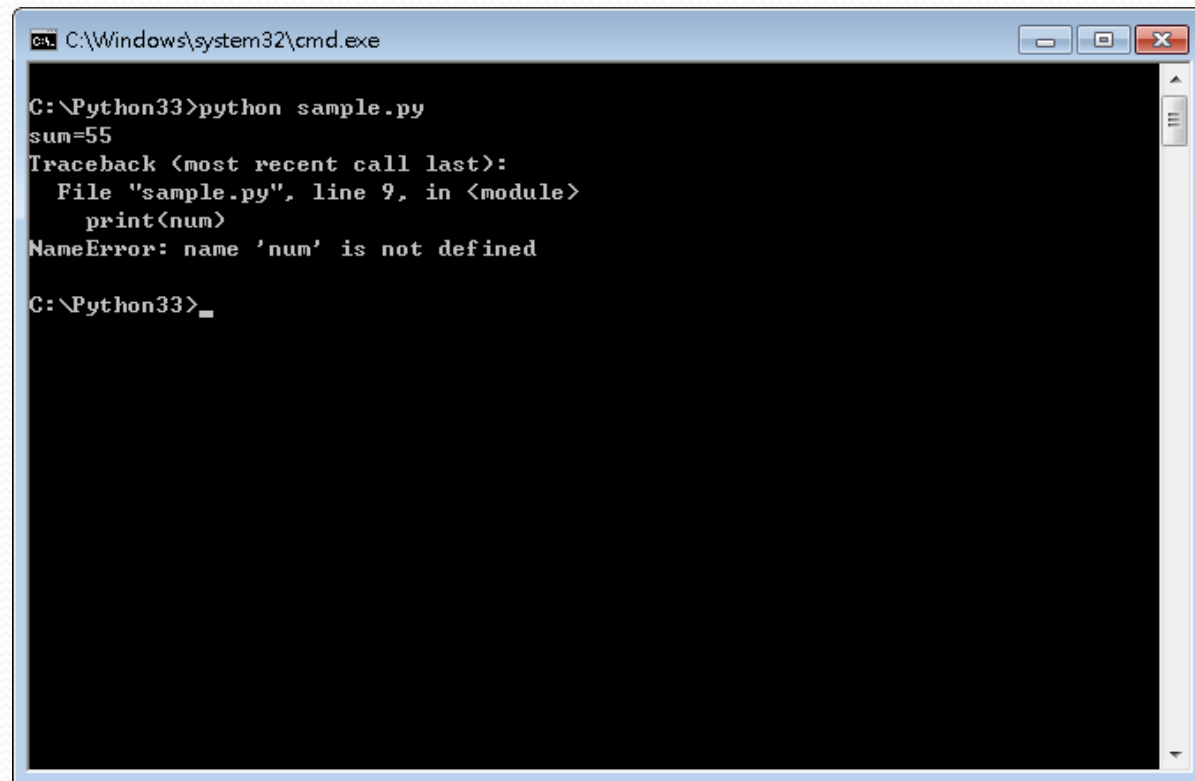
# 函數範例

```
def sum():  
    i = 0  
    num=0  
    while i <= 10:  
        num+=i  
        i += 1  
    print('sum=%d'%num)  
sum()
```

# 函數範例

```
def sum():  
    i = 0  
    num=0  
    while i <= 10:  
        num+=i  
        i += 1  
    print('sum=%d'%num)  
sum()  
print(num)
```

- `i` , `num`為 `fun()` 函數內的區域變數 (local variable) , `sum()` 之外的地方無法存取`i` , `num`的值
- 因為直譯器 (interpreter) 在 `sum()` 函數的區塊內才認得變數 `i` , `num` , 離開函數的地方 , 直譯器便不認識這個名稱。

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. The text shows a user running a Python script: "C:\Python33>python sample.py". The output is "sum=55", followed by a traceback: "Traceback (most recent call last):", "File 'sample.py', line 9, in <module>", "print(num)", and finally the error message "NameError: name 'num' is not defined". The prompt "C:\Python33>\_" is visible at the bottom.

```
C:\Windows\system32\cmd.exe

C:\Python33>python sample.py
sum=55
Traceback (most recent call last):
  File "sample.py", line 9, in <module>
    print(num)
NameError: name 'num' is not defined

C:\Python33>_
```

- 模組 (module) 也就是 .py 檔案中定義函數，需注意先有函數定義，才可進行函數呼叫。

```
sum()
def sum():
    i = 0
    num=0
    while i <= 10:
        num+=i
        i += 1
    print('sum=%d'%num)
```

- 因為 Python 直譯器從頭一行一行的解譯程式原始碼 (source code)，sum() 既非 Python 的內建名稱，也還沒解譯到底下定義的部份，因此直譯器直接發生 NameError，終止程式的進行。

# 函數回傳值

函數 (function) 可以有回傳值 (return value)，回傳值可以是函數最後的計算結果。沒有回傳值的函數，呼叫 (call) 該函數會自動回傳 **None** 物件 (object)，例如

```
def sum():
```

```
    i = 0
```

```
    num=0
```

```
    while i <= 10:
```

```
        num+=i
```

```
        i += 1
```

```
a=sum()
```

```
print(type(a))
```



# 函數回傳值

- 有回傳值的函數是在函數定義中加入 **return** 陳述 (return statement)，通常是放在函數的最後面，形式如下

```
def function_name():  
    #dosomething  
    return something
```

# 改寫sum()使其有回傳值

```
def sum():
```

```
    i = 0
```

```
    num=0
```

```
    while i <= 10:
```

```
        num+=i
```

```
        i += 1
```

```
    return num
```

```
total=sum()
```

```
print(type(total))
```

```
print('sum=%d'%total)
```

# 函數回傳值

- 函數定義中沒有限制 `return` 陳述的數量，可依需要自行設置。

```
import random
def test():
    rand_num=random.randint(0,10)
    if rand_num<5:
        return 'rand_num < 5'
    else:
        return 'rand_num >=5'

a = test()
print(a)
```

# 函數回傳值

- 函數定義中也沒有限制回傳值的數量，可在 **return** 陳述中以逗號分隔回傳值

```
import random
```

```
def test():
```

```
    rand_num=random.randint(0,10)
```

```
    if rand_num<5:
```

```
        return 'rand_num < 5',rand_num
```

```
    else:
```

```
        return 'rand_num >=5',rand_num
```

```
a,num = test()
```

```
print(a)
```

```
print('rand_num=%d'%num)
```

# 函數參數

- 函數 (function) 所用的小括弧可以定義參數 (parameter)，所謂的參數也就是提供給函數計算的數值 (value)，或是物件 (object)。

```
def function_name(arg1, arg2):  
    #dosomething  
    return something
```

# 改寫sum()函數加入參數

```
def sum(n):
```

```
    i = 0
```

```
    num=0
```

```
    while i <= n:
```

```
        num+=i
```

```
        i += 1
```

```
    return num
```

```
total=sum(10)
```

```
print('sum=%d'%total)
```

# 函數(多個參數)

- 參數的數量並沒有限制，但每個參數需要用逗號分隔

```
def add(num1,num2):  
    return num1+num2  
total=add(40,20)  
print('sum=%d'%total)
```

# 課堂練習

- 將計算 $n*n$ 的程式用函數的形式改寫為 $n*m$ 
  - 包含了兩個傳入值 $n, m$
  - 不需回傳值，直接在函數內將相乘的結果印出



# 函數與資料結構

- 區域變數與全域變數
  - global敘述句修改區域變數
  - Ex:

```
name = "John"
def change(n):
    global name
    name = n
    print ("change name", name)

print ("outside function", name)
change("Jack")
print ("outside function", name)
```

# 函數 預設參數

- 函數 (function) 的參數 (parameter) 可以提供預設值 (default argument)，也就是可以在參數列 (parameter list) 直接將參數指派數值 (value)

```
def function_name(arg1 = default1, arg2 = default2):  
    #dosomething  
    return something
```

# 函數 預設參數

```
def hello(name = "John"):
    print("hello %s ." % name)
```

```
hello()
```

```
hello("Tom")
```

- 呼叫 (call) 函數時，參數必須按照順序提供，若沒有按照順序，就需要把參數名稱打出來，同時沒有給預設值的參數必須給值

```
def hello(age, name = "John"):
    print("Hello %s, You are %d years old."%(name,age))
```

```
hello(22,"Mary")
```

```
hello(name = "Bill", age = 33)
```

# 函數 不定個數參數

- 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_name(*arguments, **keywords):  
    #dosomething  
    return something
```

- \*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)

# \*arguments

```
def hello(*names):  
    for n in names:  
        print("Hello, %s."%n)  
hello("Tom","Peter","Bob","Rain")
```

# **\*\*keywords**

```
def hello(**names):  
    for n in names:  
        print("Hello %s, you're %d years old"%(n,names[n]))
```

```
hello(John=25, Tom=20, Bob=33)
```

# 課堂練習

- 試寫一個函數傳入一個10個數字的list，並且回傳傳入list中第三大的數。



# 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，而不會結束函數執行，這樣的函數被稱為產生器函數 (generator function) 。

# 函數 yield 產生器

```
def myrange(n):  
    x = 0  
    while True:  
        yield x  
        x += 1  
        if x == n:  
            break  
print(list(myrange(10)))
```

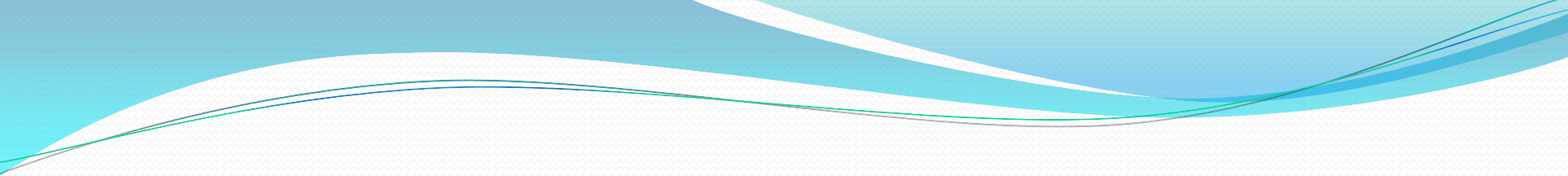
# 輸入字串，使其倒過來印出

```
def reverse(string):  
    for i in range(len(string)-1, -1, -1):  
        yield string[i]
```

```
for i in reverse("Hello python"):  
    print(i,end="")
```

# 函數 範圍規則

- 函數 (function) 內的變數 (variable) 包括參數 (parameter) 都稱為區域變數 (local variable)，這意思是說區域變數的使用範圍 (scope) 限於函數的區塊 (block) 內，一旦離開該區塊範圍，便無法繼續使用原本在區塊內的變數值。



```
def outer(a):  
    def inner(a):  
        a += 1  
        print("inner a =", a)
```

```
    a += 1  
    inner(a)  
    print("outer a =", a)
```

```
a = 10  
print("global a =", a)  
outer(a)  
print("global a =", a)
```

# global

def outer(a):                   這時 inner() 裡的 a global 的 a

    def inner():

        global a

        a += 1

        print("inner a =", a)

    a += 1

    inner()

    print("outer a =", a)

a = 10

print("global a =", a)

outer(a)

print("global a =", a)

# nonlocal

```
def outer(a):  
    def inner():  
        nonlocal a  
        a += 1  
        print("inner a =", a)
```

這時 **inner()** 裡的 **a** 是上一層的 **a**，也就是 **outer()** 的 **a**

```
    a += 1  
    inner()  
    print("outer a =", a)
```

```
a = 10  
print("global a =", a)  
outer(a)  
print("global a =", a)
```