

# UNIT 7



python<sup>TM</sup>

Function and Data Structure

張傑帆 Chang, Jie-Fan



# OUTLINE

- 內建函數
- 自訂函數
  - 函數定義
  - 函數傳遞參數與回傳值
  - 區域變數與全域變數



# 函式

## ■ 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 )

內建函數 ( Build-in Function )

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



函數	描述
abs(x)	回傳 x 的絕對值
all(iterable)	判斷 iterable 中所有元素是否為迭代器
any(iterable)	判斷 iterable 中是否有任一元素為迭代器
ascii(object)	回傳參數物件的字串表達形式，如果該字串含有非 ASCII 字元，所有非 ASCII 字元會以 Unicode 跳脫字元的方式呈現
bin(x)	回傳 x 的二進位形式
bool([x])	將 x 轉換為布林形式
bytearray([source[, encoding[, errors]]])	將 source 轉換為 bytearray，若 source 為字串，需提供 encoding，也就是字串的編碼格式
bytes([source[, encoding[, errors]]])	將 source 轉換為 bytes，若 source 為字串，需提供 encoding，也就是字串的編碼格式
chr(i)	回傳整數 i 的字串編碼
classmethod(function)	回傳 function 為類別方法
compile(source, filename, mode, flags=0, dont_inherit=False)	將 source 字串編譯為可執行物件
complex([real[, imag]])	將 real 轉換成複數的實部，imag 轉換成複數的虛部
delattr(object, name)	刪除 object 的屬性 name
dict([arg])	將 arg 轉換為字典的配對資料型態
dir([object])	回傳所有 object 的屬性及方法名稱的串列
divmod(a, b)	回傳 a 除以 b 的商及餘數的序對
enumerate(iterable, start=0)	回傳以 iterable 與連續整數配對的 enumerate 物件，start 為整數的起始值，預設為 0
eval(expression, globals=None, locals=None)	expression 為具有可執行運算式的字串，此函數執行 expression 中的運算式
exec(object[, globals[, locals]])	object 為具有可執行運算式的陳述，此函數執行 object 中的陳述
filter(function, iterable)	function 為判斷的布林函數，iterable 則為迭代器，此函數取得依 function 篩選後 iterable 中的元素



<code>float(x)</code>	將 x 轉換成浮點數，x 可以是整數、浮點數或字串
<code>format(value[, format_spec])</code>	將 value 轉換成 format_spec 格式化字串表示法
<code>frozenset([iterable])</code>	將 iterable 轉換成 frozenset 物件
<code>getattr(object, name[, default])</code>	取得 object 的屬性值
<code>globals()</code>	回傳儲存在字典的全域符號表
<code>hasattr(object, name)</code>	判斷 name 是否為 object 的屬性名稱
<code>hash(object)</code>	回傳 object 的雜湊值
<code>help([object])</code>	內建的文件系統，若無提供參數，就會進入互動模式
<code>hex(x)</code>	將 x 轉換成十六進位數字的字串
<code>id(object)</code>	回傳 object 的 id 號碼
<code>input([prompt])</code>	接受使用者的輸入，prompt 為提示字串
<code>int([number   string[, base]])</code>	轉換數字或字串為整數，如果是字串，需提供進位的基數
<code>isinstance(object, classinfo)</code>	判斷 object 是否為 classinfo 的實體
<code>issubclass(class, classinfo)</code>	判斷 class 是否為 classinfo 的子類別
<code>iter(object[, sentinel])</code>	回傳迭代器物件
<code>len(s)</code>	回傳複合資料型態的元素個數
<code>list([iterable])</code>	將 iterable 轉換成串列
<code>locals()</code>	回傳儲存在字典的區域符號表
<code>map(function, iterable, ...)</code>	function 為函數，iterable 則為迭代器，此函數取得依 function 計算 iterable 中每個元素的結果
<code>max(iterable[, args...], *[, key])</code>	回傳參數中的最大值
<code>memoryview(obj)</code>	回傳 obj 的 memoryview 型態物件
<code>min(iterable[, args...], *[, key])</code>	回傳參數中的最小值
<code>next(iterator[, default])</code>	回傳迭代器中下一個數值

<code>object()</code>	回傳最基本的 <code>object</code> 型態的物件
<code>oct(x)</code>	將 <code>x</code> 轉換成八進位數字的字串
<code>open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True)</code>	讀取檔案並回傳檔案串流物件
<code>ord(c)</code>	回傳字元 <code>c</code> 的 Unicode 編碼值
<code>pow(x, y[, z])</code>	回傳 $x^y$ 之值，或是 $x^y \% z$ 之值
<code>print([object, ...], *, sep=' ', end='\n', file=sys.stdout)</code>	印出 <code>object</code> 的內容
<code>property(fget=None, fset=None, fdel=None, doc=None)</code>	回傳類別的屬性值
<code>range([start], stop[, step])</code>	建立整數序列
<code>repr(object)</code>	回傳物件的字串表達形式
<code>reversed(seq)</code>	反轉 <code>seq</code> 中元素的順序
<code>round(x[, n])</code>	回傳 <code>x</code> 的最接近數字，預設回傳整數， <code>n</code> 代表小數點位數
<code>set([iterable])</code>	將迭代器建立為 <code>set</code> 型態的物件
<code>setattr(object, name, value)</code>	設定物件 <code>object</code> 的新屬性
<code>slice([start], stop[, step])</code>	回傳 <code>slice</code> 型態的物件
<code>sorted(iterable[, key][, reverse])</code>	回傳將 <code>iterable</code> 排序過的串列
<code>staticmethod(function)</code>	回傳 <code>function</code> 為 <code>static</code> 方法
<code>str([object[, encoding[, errors]]])</code>	回傳物件 <code>object</code> 的字串版本
<code>sum(iterable[, start])</code>	回傳迭代器 <code>iterable</code> 的總和，若有提供 <code>start</code> ， <code>start</code> 會被加入總和
<code>super([type[, object-or-type]])</code>	用為呼叫父類別定義的方法
<code>tuple([iterable])</code>	將 <code>iterable</code> 轉換為序對
<code>type(object)</code>	回傳 <code>object</code> 的型態名稱
<code>vars([object])</code>	回傳 <code>object</code> 的屬性與屬性值的字典
<code>zip(*iterables)</code>	將 <code>*iterables</code> 的元素進行配對，儲存在回傳的序對中



# 自訂函數

- 包函許多程式碼的一行程式 (用來代表某種功能)
- 當程式碼太多且會重覆出現時，可以將部份程式碼抽離主程式，寫成一段函式，有需要用到時再去呼叫它
- 函數是經過組織且可重複使用的程式碼，是能用來實現單一或是相關聯的程式碼
- 巧妙的運用函數可以提高程式碼的重複利用率，避免一樣的事情卻需要重複寫好幾次的程式碼來執行。
- 這跟迴圈的概念有點像，都是在重複利用程式碼；不同的地方在於函數是在需要時才呼叫使用

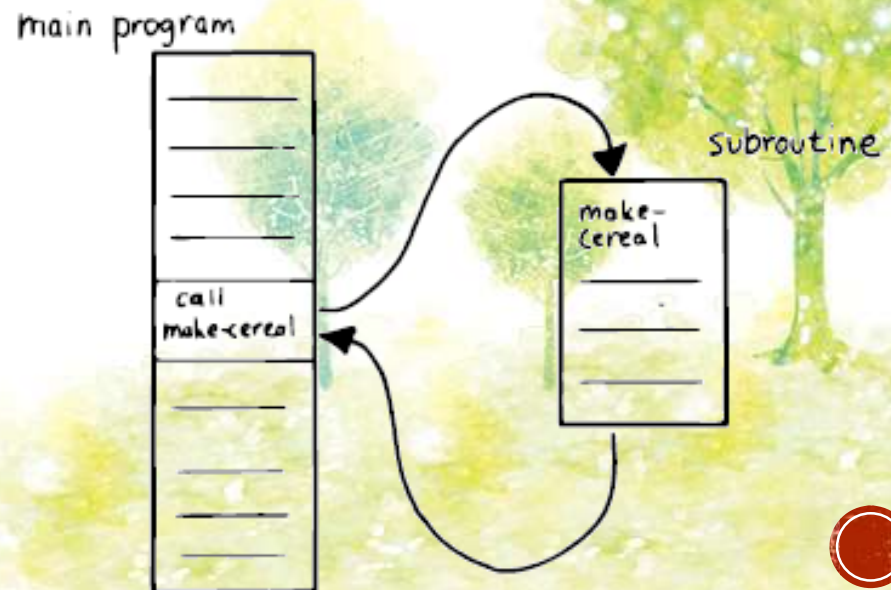


# 函數 (FUNCTION) 物件 (OBJECT)

- Python 已經內建許多好用的函數，例如 `input()`、`print()` 等等
- Python 內建函數是常用到的功能，如果想要客製化的功能，還是可以 **自己動手做**，執行一些工作，或是進行計算
- **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**：
- 其從屬內容必定要縮排如：

**def function\_name():**

**縮排** **#do something**



# 定義函式 DEFINING FUNCTIONS

函式定義從“def”開始

函式名稱 傳入參數

```
def get_final_answer(filename):
```

```
    """說明文字"""
```

縮排

```
    line1
```

```
    line2
```

```
    total_counter = 0
```

```
    return total_counter
```

```
filename = "abc.txt"
```

冒號

縮排很重要

若是沒有縮排的話

會被認為是不屬於此函式的敘述

關鍵字‘return’代表要回傳給呼叫者的值  
同時也是結束函式

- 函式內的變數是獨立的，不會受外在變數的影響。
- 不需標頭檔，也不需宣告函式回傳值與參數型態

呼叫執行: ans => 0

```
ans = get_final_answer("abc.txt")
```





# 參數與引數

- 參數與引數的差別(很容易混淆)
- **Parameter / 參數 (虛引數)**
  - 函式運行時之參考
  - 放在函式的標記式，用來說明這個函式被呼叫時必須**接收**到什麼樣的資料
- **Argument / 引數 (實引數)**
  - 用以呼叫引發函式
  - 呼叫函式時，放在括號內**傳入**被呼叫的函式





# 函式使用練習

## Hello\_func.py

```
1 def hello():  
2     print('hello',1)  
3     print('hello',2)  
4     print('hello',3)  
5  
6     print("before hello")  
7     hello()  
8     print("after hello")  
9     print(hello())  
10    print("last hello")  
11    print(hello)
```

>>>

before hello

hello 1

hello 2

hello 3

after hello

hello 1

hello 2

hello 3

None

last hello

<function hello at 0x02D7F108>

>>>

沒有回傳值的函數，呼叫 (call) 該函數會  
自動回傳 **None** 物件 (object)



# 傳入參數與回傳值

- 函數的小括弧可以定義 **參數** (parameter)  
是提供給函數計算的 **數值** (value) ，或是 **物件** (object)
- 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** 。

## Function.py

```
1 def F(x, y):  
2     ans = 2*x+y  
3     return ans  
4  
5  
6  
7  
8 num1 = eval(input())  
9 num2 = eval(input())  
10 result = F(num1, num2)  
11 print("F(%d, %d) = %d" % (num1, num2, result))
```

>>>  
20  
50  
F(20, 50) = 90  
>>>



- 注意先有函數定義，才可進行函數呼叫

### Call\_error.py

```
1 F(10, 20)
2
3 def F(x, y):
4     ans = 2*x+y
5     return ans
```

```
>>>
Traceback (most recent call last):
  File "D:/Dropbox/NTU/系統訓練班/Python
odule>
    F(10, 20)
NameError: name 'F' is not defined
>>>
```

- 因為 Python 直譯器從頭一行一行的解譯程式原始碼 (source code)，F() 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



# 回傳值-示範

- 用函數撰寫一函式，令其計算 $n*m$
- 包含了兩個傳入值 $n, m$
- 練習一
  - `f1()` 不需回傳值，直接在函數內將相乘的結果印出
- 練習二
  - `f2()` 有回傳值，將相乘的結果回傳給呼叫函式後印出

```
>>>
```

```
5
```

```
10
```

```
f1() n*m = 50
```

```
f2() n*m = 50
```

```
相乘的結果： 50
```

```
f2() + 50 = 100
```

```
>>>
```





# 小練習

- 請試寫一函式可回傳 1 加到  $n$  的結果
- 請試寫另一函式可回傳 1 乘到  $n$  ( $n!$ ) 的結果
- 若呼叫需放參數的 **function** 時不放引數會發生什麼事？

```
>>> sum()
```



# 函數 預設參數

- 函數的參數可以提供預設值 (default argument)
- 可以在參數列 (parameter list) 直接將參數指派數值

## Def\_arg.py

```
1 def PrintData(uid, name='No name', age=0):  
2     print('The user id:', uid)  
3     print('The username:', name)  
4     print('The user age:', age)  
5  
6 PrintData(1, 'John', 20)  
7 PrintData(2, 'May')  
8 PrintData(3)
```

>>>

The user id: 1

The username: John

The user age: 20

The user id: 2

The username: May

The user age: 0

The user id: 3

The username: No name

The user age: 0

>>>

- 如果只執行 PrintData() 呢？



# 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

## Multi\_return.py

```
1 import random
2
3 def rand2int():
4     rand1 = random.randint(1,10)
5     rand2 = random.randint(1,10)
6     return rand1, rand2
7
8
9 a,b = rand2int()
10 print(a)
11 print(b)
12 print(rand2int())
```

```
>>>
4
9
(3, 1)
>>>
```



# 小練習

- 請試寫一function 可同時回傳  $n!$  與  $\sum 1 \sim n$  的function，並改成即使呼叫時不加入引數也不會錯誤





# 函數範例

## Call\_error2.py

```
1 def F(x, y):  
2     ans = 2*x+y  
3     return ans  
4 print(F(10,20))  
5 print(x, y) #這行可以執行嗎？會發生什麼事？
```



- $x, y$  為  $F()$  函數內的區域變數 (local variable)
- $F()$  之外的地方無法存取  $x, y$  的值
- 直譯器在  $F()$  函數的區塊內才認得變數  $x, y$
- 離開函數直譯器便不認識這個名稱。

```
Traceback (most recent call last):  
  File "D:/Dropbox/NTU/系統訓練班/Python/  
odule>  
    print(x,y)  
NameError: name 'x' is not defined  
>>>
```



# 小練習

- 排列組合應用

- 從  $n$  個相異物中不重覆取出  $m$  個之排列數

$${}_nP_m = \frac{n!}{(n-m)!} = n(n-1)(n-2)\dots(n-m+1)$$

- 從  $n$  個相異物中不重覆取出  $m$  個之組合數

$${}_nC_m = \binom{n}{m} = \frac{{}_nP_m}{m!} = \frac{n!}{m!(n-m)!}$$

- 重複組合 從  $n$  類相異物中任意取出  $m$  個（可重複）之組合數

$${}_nH_m = {}_{n+m-1}C_m = \frac{(n+m-1)!}{m!(n-1)!}$$



# 進階思考

- 設計函式時如何讓使用的人方便使用？
- 設計函式時應如何避免使用函式的錯誤？
- 參數與引數的合理性





- 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，
- 沒有給預設值的參數必須給值

### Def\_arg.py

```
1 def PrintData(uid, name='No name', age=0):  
2     print('The user id:',uid)  
3     print('The username:',name)  
4     print('The user age:',age)  
5  
6 PrintData(age=20,name='John',uid=0)  
7 PrintData(name='May',uid=1)
```

```
>>>
```

```
The user id: 0
```

```
The username: John
```

```
The user age: 20
```

```
The user id: 1
```

```
The username: May
```

```
The user age: 0
```

```
>>>
```

- 如果執行 PrintData(name='Dany') 呢？



# 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

## Multi\_arg.py

```
1 def function_name(*arguments, **keywords):  
2     #dosomething  
3     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)



# 不定個數參數 範例

## Multi\_arg\_tuple.py

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

```
>>>
Hello, Tom.
Hello, Peter.
Hello, Bob.
Hello, Rain.
```

```
Hello Bob, you are 33 years old
Hello John, you are 25 years old
Hello Tom, you are 20 years old
>>>
```

## Multi\_arg\_dict.py

```
1 def hello(**names):
2     for n in names:
3         print("Hello",n,end=', ')
4         print("you are",names[n],"years old")
5     hello(John=25, Tom=20, Bob=33)
```

就像宣告變數一般，不得為數字開頭，不能有標點符號



# 小練習

- 請試寫一函數 `add()` 可以接收不定個數字的傳入，並最後將所有數值加總後回傳
- 請試寫一函數 `PrintData()` 可以接收不定個關鍵字參數，若傳入的關鍵字參數中有
  - 'id' 這個key就印出 `The user id: id`
  - 'name' 這個key就印出 `The username: name`
  - 'age' 這個key就印出 `The user age: age`
- 若沒有以上這些key則不用印出

```
PrintData (age=20, name='John', uid=0)  
PrintData (name='May', uid=1)
```





# 傳遞數值參數

- 傳遞的參數內容會被copy到函式用來接收參數的變數中
- 事實上是兩個不同的變數
- 所以函式中改變參數數值時，原來呼叫處的數值並不會改變

## PassValue.py

```
1 def passValue(args):  
2     print(' before change:', args)  
3     args = 999  
4     print(' after change:', args)  
5  
6 num = 10  
7 print('before call:', num)  
8 passValue(num)  
9 print('after call:', num)
```

args = num = 10

```
>>>  
before call: 10  
before change: 10  
after change: 999  
after call: 10  
>>>
```



# 傳遞容器參數

- 函式在做引數傳遞呼叫函式的引數是容器時
- 會將記憶體位址傳給被呼叫函式內的引數
- 函式中對容器所做的改變會影響引數容器  
(除非引數用 `list.copy()` 複製一份進去)
- 函式中對容器=數值  
也不會影響原容器

```
>>>
before call: [44, 63, 7, 22, 5, 1]
before sort: [44, 63, 7, 22, 5, 1]
after sort: [1, 5, 7, 22, 44, 63]
after call: [1, 5, 7, 22, 44, 63]
>>>
```

## PassList.py

```
1 def passList(lst):
2     print(' before sort:',lst)
3     lst.sort()
4     print(' after sort:',lst)
5
6 list1 = [44,63,7,22,5,1]
7 print('before call:',list1)
8 passList(list1)
9 print('after call:',list1)
```

*pass by reference*



fillCup(        )

*pass by value*



fillCup(        )

# 傳遞容器參數

- 把容器當參數來傳遞時，如果傳遞的只是容器的**某欄位**中的一個變數，那和傳普通變數沒有什麼分別

**Ex:**

```
list1 = [1,2,3,4,5]  
func(list1[0])
```

- 如果傳的是**整個容器**，傳遞的東西會是容器的位址

**Ex:**

```
list1 = [1,2,3,4,5]  
func(list1)
```



# 課堂練習

- 寫一函式 `inputList(lst)` 令使用者輸入一 `N` 個數字的數列以 `-1` 結束並存入 `list` (不包含 `-1`)
- 再試寫一個函數 `Find3rdMax(lst)` 將此 `list` 傳入在此函數中對 `list` 排序後印出並且找出此 `list` 中第三大的數 (`sort`) 後回傳印出
- 印出此未排序的 `list` (令容器在函式中不會改變)

```
>>>
```

```
11
```

```
55
```

```
77
```

```
88
```

```
99
```

```
66
```

```
33
```

```
-1
```

```
input = [11, 55, 77, 88, 99, 66, 33]
```

```
sorted = [11, 33, 55, 66, 77, 88, 99]
```

```
max 3rd = 77
```

```
list = [11, 55, 77, 88, 99, 66, 33]
```

```
>>>
```





# 回家作業

- 假定某班有5位學生，每位學生各修3門科目
- 請利用二維List的方式儲存學生的各科成績
- 並設計一函式可計算一List之總分與平均
- 最後將每位學生的各科成績、總分及平均列印出來，並找出班上最平均高分的學生

座號	科目A	科目B	科目C
1	76	73	85
2	88	84	76
3	92	82	92
4	82	91	85
5	72	74	73



# 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)

```
>>>  
<generator object gen at 0x02CB0698>  
In func 1  
In func 2  
In func 3  
In func 4  
In func 5  
In func 6  
In func 7  
In func 8  
In func 9  
In func 10  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
>>>
```

## Yield.py

```
1 def gen(n):  
2     for i in range(1,n+1):  
3         print('In func',i)  
4         #return i  
5         yield i  
6 print(gen(10))  
7 print(list(gen(10)))
```



# 全域變數與區域變數

- 函數內的變數包括參數 (parameter) 都稱為區域變數 (local variable)，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由記憶體中釋放掉，便無法繼續使用原本在區塊內的變數值，下次呼叫該函式時再重新配置記憶體給該函式使用。
- 另外提供一種變數可供多個函式共同使用，變數的有效時間一直到程式結束為止，我們將此類的變數稱為「全域變數」 (Global variable)。



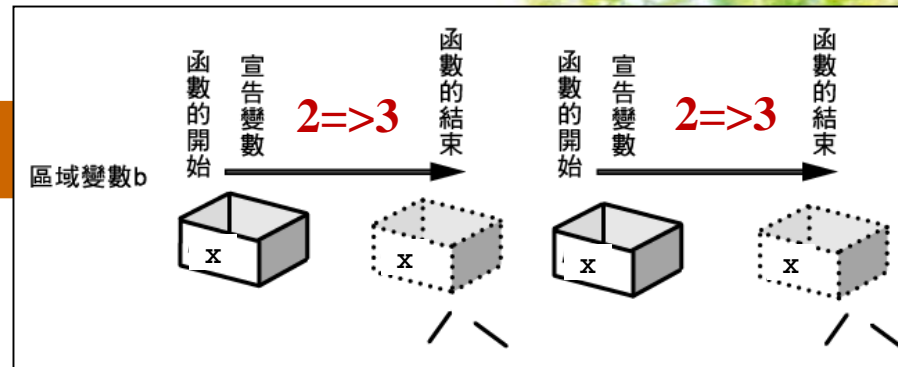


# 局部/區域變數 (LOCAL VARIABLE)

- 自動變數只在它所定義的**區塊內有效**。只要在變數所屬的區塊結構內執行，該變數的資料是有效而正確的。
- 當程式執行**離開了該區塊**，所有於區塊內定義的自動變數就**不存在了**。

## Local.py

```
1 def func():  
2     x = 2  
3     x = x + 1  
4     print('In func():', x)  
5  
6 x = 1  
7 func()  
8 func()  
9 print(x) https://goo.gl/8k3Mxx
```



```
>>>  
In func(): 3  
In func(): 3  
1  
>>>
```



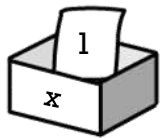
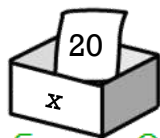
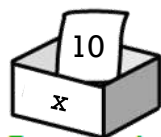


- 程式中區域變數與全域變數的**名稱相同**，當存取函式內的變數時會以**區域變數為優先**，使用時最好注意，建議全域變數**最好不要**和區域變數的**名稱重複**，以免參用時造成混淆。

## Local2.py

```
1  #x = 1 #試試x的位置有何不同
2  def func1():
3      x=10
4      print("In func1:", x)
5  def func2():
6      x=20
7      print("In func2:", x)
8
9
10 x = 1
11 func1()
12 func2()
13 print(x)
```

<https://goo.gl/Jny6RG>



區域變數的名稱**可以重複使用**

在**不同函數**內被宣告的**區域變數**就代表是**3個不同的變數**

```
>>>
In func1: 10
In func2: 20
1
>>>
```

# 全域變數 (GLOBAL VARIABLE)

- 全域變數的**有效範圍**不是區域性，而是**整體**
- 變數定義在任何函數的外面，可被其他函數所共用
- 僅限取值，不得放在=號左邊，除非宣告成**global**

## Global\_v1.py

```
1 def func1():
2     print(' In func1():',x)
3     #x = x + 1 #這行不能執行
4 def func2():
5     print(' In func2():',x)
6
7 x = 10
8 print('1 x:',x)
9 func1()
10 func2()
11 print('2 x:',x)
```

```
1 x: 10
  In func1(): 10
  In func2(): 10
2 x: 10
```



# 全域變數

- global 修改區域變數

- Ex:

## Global\_v2.py

```
1 name = "John"
2 def change(n):
3     #global name #測試有無此行會有何不同
4     name = n
5     print ("change name", name)
6 print ("outside function", name)
7 change("Jack")
8 print ("outside function", name)
```

```
>>>
```

```
outside function John
change name Jack
outside function John
```

```
>>>
```

```
outside function John
change name Jack
outside function Jack
```

# 一般參數傳遞

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

<https://goo.gl/QqQj3f>

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

```
>>>  
global a = 10  
inner a = 12  
outer a = 11  
global a = 10
```





# 全域 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)
```

<https://goo.gl/Qe4Wxx>

```
global a = 10
inner a = 11
outer a = 11
global a = 11
>>>
```



# NONLOCAL

**def outer(a):** #inner() 裡的 **a** 是上一層的 **a**，就是 **outer()** 的 **a**

**def inner():**

**nonlocal 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)**

<https://goo.gl/9ZpnFz>

>>>

global a = 10

inner a = 12

outer a = 12

global a = 10

>>>



# LAMBDA 函式

- 簡易的function define：lambda，用完即丟，不著痕跡。

## lamb.py

```
1 def func1(x, y, z):  
2     return x + y + z  
3 func1(1, 2, 3)    #6  
4  
5 func2 = lambda x,y,z : x+y+z  
6 func2(1, 2, 3)    #6  
7  
8 #也可以應用在map上 # map(func, *iterables)  
9 my_list = [1, 2, 3]  
10 map( lambda i: i * i, my_list )  
11 # (1, 4, 9)
```

<https://goo.gl/A3mj3T>



# SORT 補充

- <https://docs.python.org/3.7/howto/sorting.html>
- <https://marco79423.net/articles/%E6%B7%BA%E8%AB%87-python-%E7%9A%84%E6%8E%92%E5%BA%8F/>





# SORT KEY

- 設定一個function給key這個參數，在排序之前會自動對每個element call一次key所指定的function，透過這個方法可以做一些比較複雜一點的排序

```
sorted("aa BB cc ZZ".split())
```

```
#['BB', 'ZZ', 'aa', 'cc'] (case sensitive)
```

```
sorted("aa BB cc ZZ".split(), key=str.lower)
```

```
#['aa', 'BB', 'cc', 'ZZ'] (case-insensitive)
```

- 如果要根據某些欄位來做sort，也可以用key這個參數key只接受function當作參數

下列範例使用lambda來傳入一個匿名函式

```
students = [ ('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]
```

```
sorted(students, key = lambda x : x[2]) # sort by age
```

```
# [('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```



# 小練習

- 請列出 長恨歌中 每一個字/詞的出現次數
- 並依出現次數排序 (由高到低)
- 如果出現次數相同的則依字元/詞的**unicode**排序(由小排到大)
- (Python 排序 屬穩定排序)
- Hint:
  - `[[天,9],[不,14].....]`
  - 先對字元的**unicode**做排序
  - 再對出現的次數排序



# 延申閱讀與思考

- 遞迴函式
- 預設參數，如果是容器的話會如何？
- 在不宣告**global**的情況下，使用容器(**list**)型態的全域變數，可以修改內容嗎？(不使用'='指定新的數值)

