

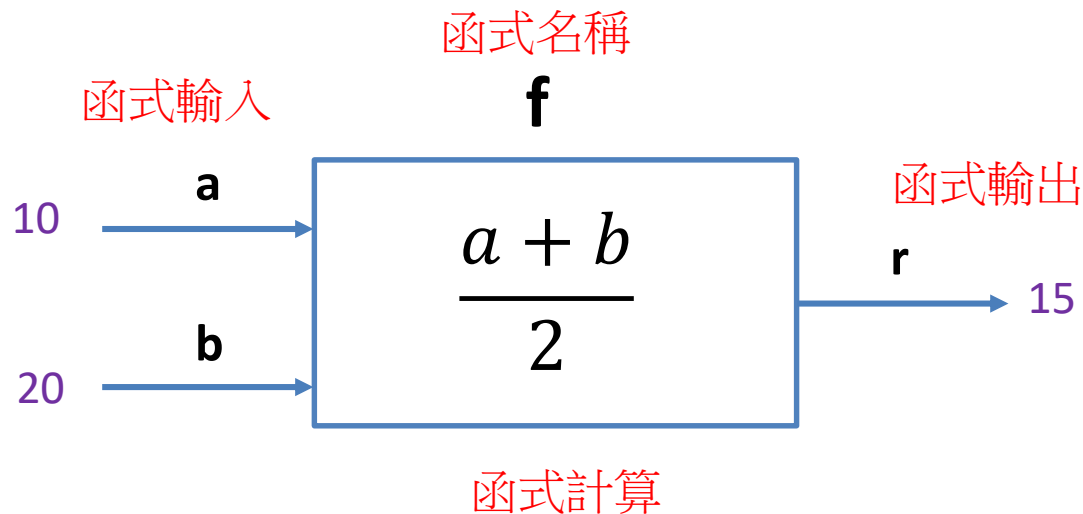
Python_07

自訂函式與內建函式

函式(fuction)

- 函式(**fuction**)或稱為函數，如同數學函數，透過函式名稱，可呼叫函式進行運算，並可輸入資料進行函式運算，也可將結果輸出，函式內部包含特定計算功能的程式碼。
- 函式的輸出可作為令一個函式的輸入
- 函式可讓程式碼依照功能模組化，有利於程式碼的再利用。

$$f(a, b) = \frac{a+b}{2}$$



Python函式

- Python的函式分為兩種
 - 使用者自訂函式 (User-defined functions)
 - 內建函式 (Built-in functions)
- 使用者自訂函式
 - 由使用者撰寫，需要先宣告函式名稱，並自行撰寫函式內部運算程式碼。
 - 宣告與撰寫內部程式碼之後，函式還需要被呼叫才能執行。
- 內建函式
 - 由Python標準函式庫提供函式宣告與內部實作，可直接根據Python手冊提供的內建函式名稱，呼叫內建函式執行。
- 函式在物件導向扮演的腳色是物件的行為或稱為方法
- 函式本身也被視為一種物件，稱為函式物件

使用者自訂函式

- 使用者自訂的Python函式

```
    函數名稱      參數1  參數2    參數N  
def function_name( para1, para2,...paraN ) :
```

```
#函式內部
```

```
#內部程式碼
```

```
return something
```

縮排



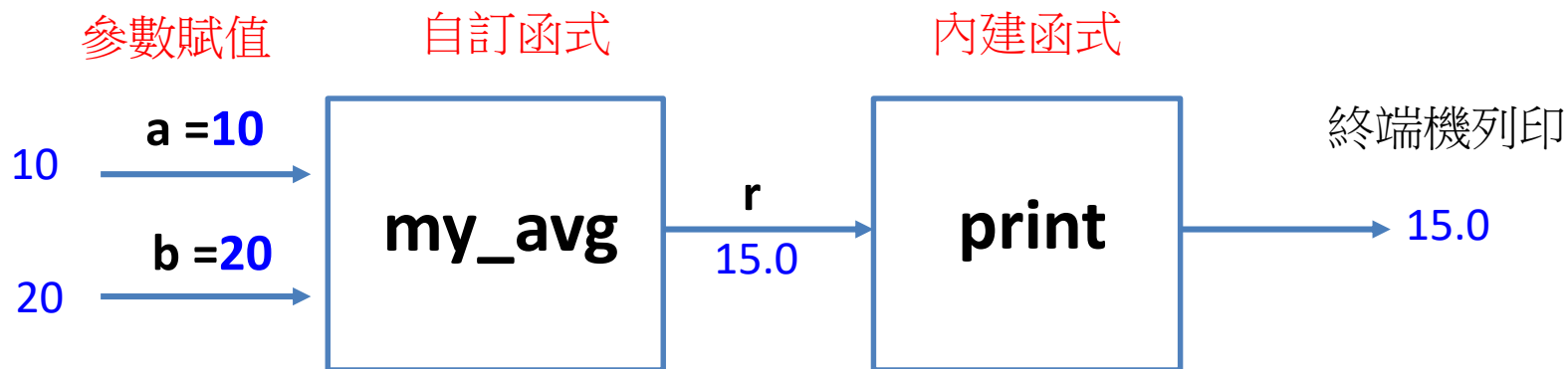
Python 函式呼叫範例

```
def my_avg(a, b):  
    r = (a + b)/2  
    return r
```

a與b稱為
參數(Parameter)

```
print(my_avg(10, 20))
```

輸入的10與20，稱為引數(Argument)
呼叫函式會進行參數賦值



$$f(a, b) = \frac{a+b}{2}$$

函式名稱

- 函式命名規則

- 所謂內部(**Internal**)表示僅模組內可用, 或者, 在類別內是保護的.
- 單下劃線(_)開頭表示模組變數或函式是protected的(使用import * from時不會包含).
- 雙下劃線(__)開頭的實例變數或方法表示類內私有.
- 將相關的類別和頂級函式放在同一個模組裡., 沒必要像Java限制一個類一個模組.
- 類別名稱使用大寫字母開頭的單詞(如CapWords), 但是模組名應該用小寫加下劃線的方式(如lower_with_under.py).

- 應該避免

- 單字元名稱, 除計數器和迭代器.
- 套件/模組名中的連字符(-)
- 雙下劃線開頭並結尾的名稱 (Python保留字, 例如 __init__)

Guido推薦的命名規範

Type	Public	Internal
Modules	lower_with_under	_lower_with_under
Packages	lower_with_under	
Classes	CapWords	_CapWords
Exceptions	CapWords	
Functions	lower_with_under()	_lower_with_under()
Global/Class Constants	CAPS_WITH_UNDER	_CAPS_WITH_UNDER
Global/Class Variables	lower_with_under	_lower_with_under
Instance Variables	lower_with_under	_lower_with_under (protected) or __lower_with_under (private)
Method Names	lower_with_under()	_lower_with_under() (protected) or __lower_with_under() (private)
Function/Method Parameters	lower_with_under	
Local Variables	lower_with_under	

函式回傳

- `return` 陳述句用於將函式計算結果回傳
- 可將回傳值使用 `=` 賦值給函數外部的變數
- 當執行到函式的 `return` 陳述句時，將會離開(結束)函式並回傳結果。
- 函式不一定要寫 `return`，敘述函式於執行之後可以回傳計算結果，也可以不回傳。
- 回傳值可以為
 - 單一的值或物件
 - 多個值或物件所構成的 tuple
 - 當函式沒寫 `return` 陳述句時, 預設將會回傳 `None`

Python函式回傳範例

宣告

```
def ret_test1(m,n):  
    a = m + n  
    return a
```

```
def ret_test2(m,n):  
    a = m + n
```

```
def ret_test3(m,n):  
    a = m + n  
    b = m - n  
    c = 'Hello'  
    return a, b, c
```

進入點



```
r = ret_test1(10,20)  
print(r)
```

30

```
r = ret_test2(10,20)  
print(r)
```

None

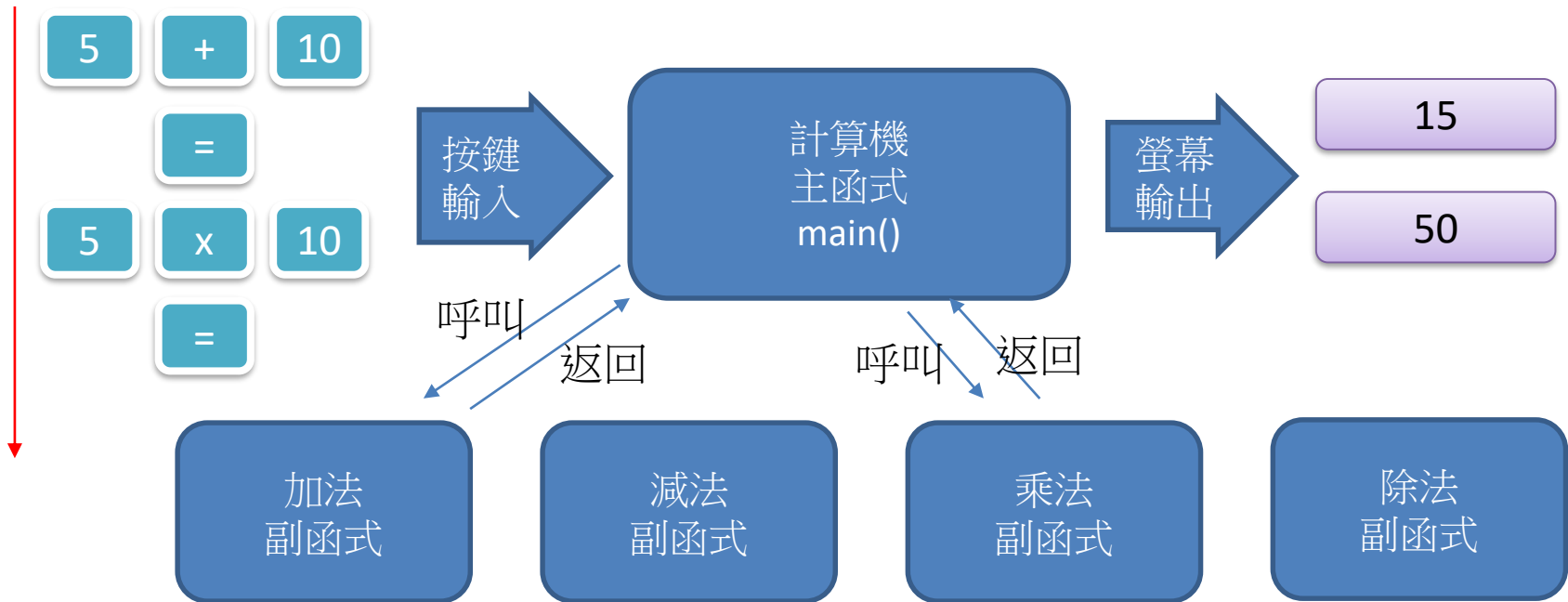
```
r1, r2, r3 = ret_test3(10,20)  
print(r1, r2, r3)
```

30 -10 Hello

程序導向程式設計

- 以上而下的方式來分析問題
- 採用模組化程式設計概念來解決問題，將每個函式視為一個功能模組。
- 函式區分為主函式(主模組)與副函式(子模組)
- 具備完整流程控制(循序、選擇、重複)，以主函式為處理問題的核心與程式進入點，掌控流程順序，主函式可呼叫副函式處理問題再返回主函式。

按鍵順序



主函式與副函式

宣告

```
def fun1(a,b):  
    print('進入fun1')  
    c = (a + b) * 2  
    return c  
  
def fun2(a,b):  
    print('進入fun2')  
    c = (a + b) * 3  
    return c  
  
def main():  
    print('主程序開始')  
    r =fun1(10,20)  
    print('fun1 回傳:',r)  
    r =fun2(5,6)  
    print('fun2 回傳:', r)  
    print('主程序結束')
```

進入點

```
if __name__ == "__main__":  
    main()
```

`__name__` 是 python內建變數。

當 `__name__` 的值是 `__main__` 可確定目前的.py檔(模組)，是主要執行模組，因此由此處開始執行main()。

若這個模組被匯入使用時，不再是主要執行模組，可確保這個模組的main()，不會被執行。



可用偵錯模式觀察執行順序

參數與資料型別

- 參數本質是函式的內部變數，Python的變數是動態型別，型的別是根據函示呼叫所輸入的資料
- 參數為可更改(`mutable`)型別
 - 參數的值若改變，參數依然指向原本的物件，會改變原本物件的值。
- 參數為不可更改(`immutable`)型別
 - 參數的值若改變，參數不再指向舊值的物件，會改指向新值的物件。
 - 必須回傳新物件

參數為mutable物件

```
def modify(L):  
    T = [5]  
    L += T
```

不用回傳，可修改原本L 串列的值

```
def main():  
    L = [1,2,3,4] ← L 串列是 mutable type  
    modify(L)  
    print(L) ← 印出 [1, 2, 3, 4, 5]
```

```
if __name__ == "__main__":  
    main()
```

參數為不可改型別

```
def modify(s):  
    s = s + "!"
```

```
def modify_with_return(s):  
    s = s + "  
    return s
```

 ← 修改後的，s 已指向新的字串物件，需要回傳

```
def main():  
    s = 'Hello Python'  
    modify(s)  
    print(s) ← 無法改變s字串  
    #沒有加上驚嘆號
```

```
    s = modify_with_return(s) ← 而且要回傳設定給某個變數  
    print(s)  
    # 成功加上驚嘆號
```

```
if __name__ == "__main__":  
    main()
```

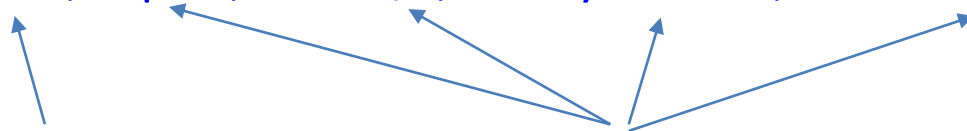
參數的使用

- 傳入引數要與參數宣告的數量與順序一致
- 傳入引數可用參數名稱賦值(para=value)的方式
- 參數宣告可設定「參數預設值」
- 可變數量參數的宣告方式
 - 參數宣告 *args, 傳入引數使用值組
 - 參數宣告 **kwargs, 傳入引數使用字典
- 參數名稱為星號(*)

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

不定長度的參數

參數預設值



傳入引數要符合參數宣告

```
def para_test(a, b, c):  
    print(a,b,c)
```

```
def main():  
    para_test(10, 20, 30)  
    para_test(10, 20)  #有宣告參數，引數數目不一致
```


傳入引數可用參數名稱賦值

- 使用參數名稱賦值，引數傳入順序可以改變
- 也稱為 keyword argument，參數名稱為 keyword

```
def para_assign(a, b, c):  
    print(a,b,c)
```

```
def main():  
    para_assign(c=30, b=20, a=10)
```

```
if __name__ == "__main__":  
    main()
```

參數預設值

- 參數預設值，呼叫函式時沒有給引數,函式依然有預設值
- 無預設值之參數寫在前，有預設值之參數寫在後，避免被預設值覆蓋。

```
def para_dft1(a=0, b=0, c=0):  
    print(a,b,c)  
  
def para_dft2(a, b, c=0):  
    print(a, b, c)  
  
def main():  
    para_dft1()  
    para_dft1(b=20,c=30)  
    para_dft1(c=30, b=20, a=10)  
    para_dft2(10,20)  
  
if __name__ == "__main__":  
    main()
```

*args 可變數量參數

- 讓函式傳入引數能更有彈性。
- 以值組(tuple)存放所有未命名的引數。

```
def para_tuple(p, *v):  
    print('p =', p)  
    print('v =', v)  
    return sum(v)-p  
  
def main():  
    r = para_tuple(10,20,30,40)  
    print(r)  
    print()  
  
    t = (20,30,40,50)  
    r = para_tuple(10, *t)  
    print(r)  
  
if __name__ == "__main__":  
    main()
```

****kwargs**可變數量參數

- 以字典的形式存放關鍵字引數。

```
def para_kw(p, **d):  
    print(p)  
    print(d)  
  
def para_tup_kw(p, *t, **d):  
    print(p)  
    print(t)  
    print(d)  
  
def main():  
    para_kw(10, i=20, j=30, k=40)  
  
    k = {'m': 22, 'n': 33}  
    para_kw(11, **k)    # m=22 n=33  
  
    para_tup_kw(5, 'a', 'b', 'c', i=20, j=30, k=40)  
  
if __name__ == "__main__":  
    main()
```

參數名稱為星號(*)

- 出現星號*後的參數必須用關鍵字引數傳入。

```
def para_star(a, *, b, c):  
    print(a,b,c)
```

```
def main():  
    para_star(10, b=3, c='AAA')
```

```
if __name__ == "__main__":  
    main()
```

巢狀函式(nested function)

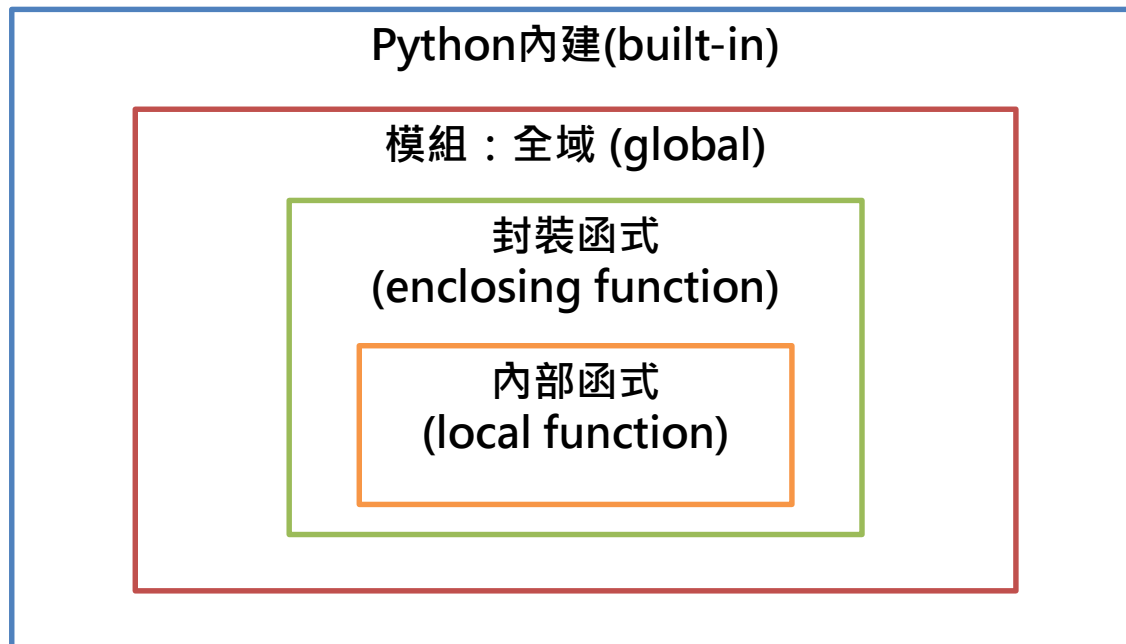
- 當函式內部宣告另一個函式，稱為巢狀函式(nested function)
- 包含巢狀函式的函式謂之外包函式(enclosing function)
- 巢狀函式可以使用外包函式的變數，稱為外部變數

```
def func_outer():  
    print('執行外包函式')  
    a = 10  
  
    def func_inner():  
        b = 20  
        print('執行內部函式')  
        print('func_inner()的外部變數 a =',a)  
        print('func_inner()的內部變數 b =',b)  
  
    func_inner() # func_outer 呼叫內部函式  
  
def main():  
    print('執行主函式')  
    func_outer() # main 呼叫外部函式  
  
if __name__ == "__main__":  
    main()
```

變數的有效範圍

- Python名稱的有效範圍，稱為命名空間(name space)
- Python有四種範圍的變數：內建(built-in)、全域(global)、封裝函式(enclosing function)、函式(local function)
- 函式使用變數時，從內部名稱開始找，若沒找到，則尋找外部變數。
- 函式內部變數又稱區域變數，變數在模組的所有函式的外部稱為全域變數
- `global` 關鍵字可讓函式內部指定使用全域變數

命名空間 name space



非區域變數(nonlocal Variable)

- Python 3新增**nonlocal**來指明所操作的變數是外部變數
- 非區域變數從區域內往外來找，找最接近的，而且不會找全域變數。

```
flag = 10  #global

def func_outer():
    flag = 20

    def set_flag():
        #nonlocal flag
        flag = 30

    set_flag()
    print(flag)

def main():
    func_outer()

    print(flag)  #global

if __name__ == "__main__":
    main()
```

執行
結果
20
10

```
flag = 10  #global

def func_outer():
    flag = 20

    def set_flag():
        nonlocal flag
        flag = 30

    set_flag()
    print(flag)

def main():
    func_outer()

    print(flag)  #global

if __name__ == "__main__":
    main()
```

執行
結果
20
10

匿名函式 (Lambda function)

- 匿名函式是指一種不使用def 宣告的函式。
- lambda的主體是一行表達式，擁有自己的命名空間，不能訪問自己參數列表之外或全域命名空間的其它變數。

lambda [arg1 [,arg2,.....argn]] : expression

```
AVG = lambda arg1, arg2: (arg1 + arg2)/2
```

```
def main():  
    print(AVG(10, 20))
```

```
if __name__ == "__main__":  
    main()
```

遞迴函式 (Recursive function)

- 遞迴(Recursion)的意思是，函式進行自我呼叫，讓大問題化成小問題，再各個擊破(Divide-and-Conquer)的策略。
- 以N階層的計算為例

```
def factorial(n):  
    if n == 1 :  
        return n  
    else:  
        return n * factorial(n-1)
```

```
def main():  
    print(factorial(4))
```

```
if __name__ == "__main__":  
    main()
```

4!
= 4 x 3!
= 4 x 3 x 2!
= 4 x 3 x 2 x 1!

factorial(4)

4*factorial(3)

3*factorial(2)

2 * factorial(1)

factorial(4)

4*factorial(3)

3*factorial(2)

2 * factorial(1)

1

生成器函式(Generator Function)

- 生成器函式特徵是使用 **yield**，讓函式**連續回傳數值**，可搭配**next()**函式控制回傳一個數值後中斷，再次執行**next()**會**從上次中斷處繼續**。

```
def main():  
    print(list(my_range(8)))  
    g = my_range(8)  
    next(g)  
    next(g)  
    print(next(g))
```

```
def my_range(n):  
    x = 0  
    while True:  
        yield x  
        x += 1  
        if x == n:  
            break
```

```
if __name__ == "__main__":  
    main()
```

執行結果

```
[0, 1, 2, 3, 4, 5, 6, 7]  
2
```

裝飾器函式(Decorator)

- 使用裝飾器函式是希望在不修改原函式的程式碼，加入其它函式的功能
- @Decorator是一種Python語法糖，方便裝飾器函式整合

```
def my_deco (func):  
    def wrapper_func (*args):  
        print('執行裝飾器的功能')  
        func(*args)  # 執行原本的函式  
  
    return wrapper_func
```

```
@my_deco  
def func_a(s):  
    print(s)
```

```
@my_deco  
def func_b(s):  
    print(s)
```

```
def main():  
    func_a('執行func_a的功能')  
    func_b('執行func_b的功能')
```

```
if __name__ == "__main__":  
    main()
```

內建函式(Build-in Function)

- <https://docs.python.org/3.7/library/functions.html>

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

Python內建函式

- 數學計算相關
- 資料型態相關
- 邏輯判斷相關
- 輸入與輸出相關
- 物件與類別相關
- 模組相關

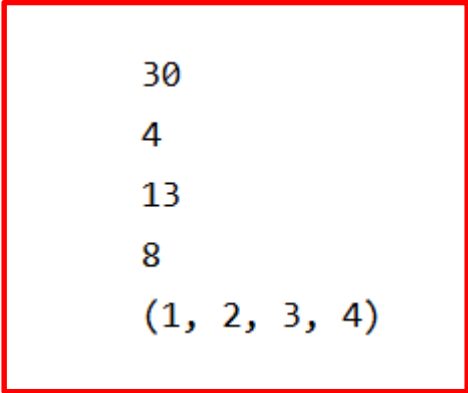
隨堂練習

- 內建函式`eval()`會試著將字串轉換成Python的表達式

```
def main():  
    x = 10  
    s1 = 'x + 3'  
    s2 = 'pow(2,3)'  
    s3 = '1,2,3,4'  
  
    print(eval('3 * x'))  
    print(eval('pow(2,2)'))  
    print(eval(s1))  
    print(eval(s2))  
    print(eval(s3))
```

```
if __name__ == "__main__":  
    main()
```

執行結果



```
30  
4  
13  
8  
(1, 2, 3, 4)
```

Python_07 作業

費布納西數為：0, 1, 1, 2, 3, 5, 8, ...，除了第一個0與第二個1之外，每個費布納西數是由前面兩個費布納西數相加，如：5是由 $2+3$ 得出。

- 使用遞迴函數方式製作一個費布納西數列，產生10個費布納西數。
- 使用yield製作一個費布納西數列生成器，產生10個費布納西數。