



程式設計概論

Programming 101

—函式(function)

授課老師：邱淑怡

Date:10/19/2023

Outline

- 函式(Function): 自訂函式
 - 優點
- Global variables(全域變數) and local variables(區域變數)
- lambda function
- 補充資料
 - 函數的可變參數
 - 遞迴函式(Recursive function)

Function

- ▶ function 是一個建構程式時的小區塊，它就像是一台機器，你可以自行指定它的功能，以及所需要的原料（輸入）、產出（輸出）
- ▶ Example: 自動販賣機就像是一個 function，他的 input 是硬幣和商品的選擇，output 則是你所選的商品

Function 優點

1. 程式的重複利用性
2. 程式的易讀性
3. 程式的易除錯性
4. 程式的一致性
5. 程式的模組化：模組化是一個在撰寫程式時的概念
 - ◆ 將一個完整的程式視為蓋一棟房子，那麼函式就像是房子的鋼筋、水泥、磚塊這樣的東西
 - ◆ 程式是由許多個函式以及其他東西所建構出來的，而且函式與函式之間的分工十分明確，
 - ◆ 每一個函式有它自己所負責的東西，並且可以獨立於這個程式

如何發展出function

- 當發現到兩個程式片段極為類似，只有當中幾個計算用到的數值或變數不同時

```
# get max. value
x=0
if a>b:
    x=a
else:
    x=b
```

可以使用函式來封裝程式片段，
將流程中引用不同數值或變數
的部份設計為參數

```
def max1(a, b):
    x=0
    if a>b:
        x=a
    else:
        x=b
    return x
```

區域變數
(local variable)

Function 定義及呼叫(call)

```
def 函式名稱(參數1,參數2,...):  
    程式碼(statements)  
    [return value]
```



如何呼叫function?

```
函式名稱(引數1,引數2)
```

1. def 是定義我們的function，包含函式名稱及這個函式所需要的參數(parameter)
2. 函式名稱與變數名稱規定一致
3. Return value 是指function傳回值，可傳回0, 1,或多個值，也可以省略不寫
4. 函式呼叫中的數叫引數(argument)

實作題

情境一：

1. 撰寫一個function(CtoF1)，它可以將攝氏溫度轉換於華氏溫度($F=C*1.8+32$)，並把華氏溫度印出來
2. 呼叫CtoF1 function並傳遞攝氏溫度作為參數後印出華氏溫度

情境二：

1. 撰寫一個function(CtoF2)，它可以將攝氏溫度轉換於華氏溫度($F=C*1.8+32$)
2. 呼叫CtoF2 function並傳遞攝氏溫度作為參數，並將華氏溫度當作回傳值
3. 最後，把華氏溫度印出來

Function 實例3(有傳回值): 計算x的y次方

Define function

```
def power(x, y):  
    r=1  
    while y>0:  
        r=r*x  
        y=y-1  
    return r
```

x,y 稱作參數；依據位置對應

Call function

```
Res= power(3,5)
```

 Res =power(3)

Function實例3: 參數有預設值

```
def function_name(param1, param2=value2, param3=value3, ...):
```

cannot do
something
like this

```
def function_name(param1=value1, param2, param3):
```

Step1

```
# define function  
def greet(name="John", message='Hi'):  
    return "%s: %s!"%(name,message)
```

Step2

```
# call function  
greeting = greet('Hello')  
print(greeting)  
greeting = greet(message='Hello')  
print(greeting)  
greeting = greet()  
print(greeting)
```

Function實例3: 參數有預設值

Define function

```
def power(x, y=2):  
    r=1  
    while y>0:  
        r=r*x  
        y=y-1  
    return r
```

沒有預設值的參數一
定要擺在前面

Call function

```
Res= power(3,5)
```

```
Res= power(y=5,x=3)
```

```
Res =power(3)
```

練習題

- 將串列當作參數傳遞給函數`product_msg()`，然後列出所有會員的產品發表會信件

```
def product_msg(users):  
    str1="親愛的"  
    str2="本公司將於2022/12/30於台北舉辦產品發表會"  
    str3="總經理敬上"  
    for person in users:  
        msg=....  
        print(msg)  
members=["小明", "小花", "小白"]  
product_msg(members)
```

Global variable (全域變數) vs.
local variable (區域變數)

Global variable(全域變數) vs. local variable(區域變數)

1. 全域變數和區域變數的定義
2. 在函式中讀取全域變數的值
3. 在函式中改變全域變數的值

全域變數與區域變數的情況及問題？

- ▶ 有個全域變數(x1)希望透過函式(myfun() function)將該數值進行修改

```
x1=10
def myfun():
    x2=10//3
    x1=x2+10
    print(x1, x2)
```

```
myfun()
print(x1)
```



<https://pythontutor.com/>

1. Global variable and local variable 定義

- 在函式中建立的變數都是區域變數
- 在函式以外的區域建立的變數就是全域變數

```
x1=10
def myfun():
    x2=20
    print(x1, x2)
print(x1,x2)
```

2.在函式中讀取全域變數/區域變數的值

- fun()函式中使用x1全域變數，這裡的「使用」是指取得x1的值，不包含改變x1的內容。

```
x1=10
def myfun():
    x2=x1+5 #讀取全域變數(x1)
    print(x1, x2)
```

```
myfun()
print(x1)
```

≠

```
x1=10
def myfun():
    x1=20
    x2=x1+5 #讀取區域變數(x1)
    print(x1, x2)
```

```
myfun()
print(x1)
```


3. 在函式中改變全域變數的值

```
x1=10
def fun():
    x2 = 20
    x2 = x1 + 5 # 讀取全域變數x1的值
    x1 = x1+ 5 # 改變全域變數x1的值

fun()
print(x1)
```

修改錯誤

```
x1=10
def fun():
    x2=20
    global x1
    x2=x1+5
    x1=x1+5 # 改變全域變數x1的值
    print(x1, x2)

fun()
Print(x1)
```

如何修改全域變數的數值？

```
x1=10
def myfun():
    x2=10//3
    x1=x2+10
    print(x1, x2)
```

```
myfun()
print(x1)
```

修改錯誤

```
x1=10
def myfun():
    x2=10//3
    global x1
    x1=x2+10
    print(x1, x2)
```

```
myfun()
print(x1)
```

lambda function

Function 另一種方式:用**lambda**函式來定義函式，執行運算式時將會產生函式物件

- ▶ lambda用來建立小的匿名函式(沒有函式名稱)
- ▶ lambda不是函式名稱，而是指令
- ▶ lambda的語法是(只有一行)：
lambda arg1, arg2, ... : expression
 - ▶ lambda中arg1、arg2等就相當於定義函式時的參數(arguments)，之後你可以在expression中使用這些參數。
 - ▶ lambda是運算式，不是陳述句，在:之後的也必須是運算式，lambda中也不能有區塊
- ▶ 這表示一些小的運算任務你可以使用lambda function，而較複雜的邏輯你可以使用一般的function來定義。

lambda函式實例: find_max()函式

- Define lambda function

- `f = lambda num1, num2: num1 if num1 > num2 else num2`

Python code

- Call lambda function

- `f(5,9)`

- `f(-5, -9)`

- `f(num2=5, num1=-9)`

lambda函式練習題

- 將lambda運算式所產生的匿名函式指派給變數Add，
這個匿名函式會傳回參數x和參數y相加的結果
- 接著，呼叫該lambda函式計算
 1. $1+2$
 2. $50+(-100)$
 3. "abc"+"de"

lambda函式練習題_參考程式

- 將lambda函式所產生的匿名函式指派給變數Add，這個匿名函式會傳回參數x和參數y相加的結果

Python code

```
Add=lambda x, y: x+y
```

透過變數 Add呼叫這個匿名函式

```
print(Add(1,2))  
print(Add(50,-100))  
print(Add("abc","de"))
```

lambda函式實例: 讓lambda函式傳回list資料, 參數有給定預設值

題目: 產生一個資料串列(list)起始點從5開始, 結束點預設值10

Python code

- `start = 5 #global variable`
- `f = lambda stop = 10: [i for i in range(start, stop)]`
 - 建立一個lambda函式, 幫參數stop設定預設值
 - 這個lambda函式會傳回一個list資料組
- 利用f呼叫lambda函式:
 - `f()` # [5, 6, 7, 8, 9]
 - `f(12)` # [5, 6, 7, 8, 9, 10, 11]

lambda函式特性

- ▶ lambda 參數1, 參數2, ... : 運算式A if 關係運算式 else 運算式B
 - ▶ lambda函式的語法中沒有函式名稱
 - ▶ 語法最前面的lambda不是函式名稱，而是指令
 - ▶ lambda函式的特性就是「用過即丟」
 - ▶ lambda函式最典型的用法就是搭配filter()和map()這二個內建函式，對資料組進行處理。需要的話，我們也可以把lambda函式設定給一個物件，如此一來，lambda函式就可以當成一般函式使用。

lambda 運算式 v.s. function

- 在 Python 中，可以使用 lambda 表示式來定義一個函式

該函式稱為 λ 函式或是匿名函式 (Anonymous function)

```
lambda a, b : a if a>b else b
```

lambda 運算式不能有程式區塊

一般的函式

```
def max1(a, b):  
    return a if a>b else b
```

補充資料

函數的可變參數

Function: 參數有預設值

```
def fun(a,b=2,c=3):  
    print("a={}, b={}, c={}".format(a,b,c))  
  
# call function  
fun(1)  
fun(1,22,33)
```

Function:

*args是可變的positional arguments列表

```
# define function
def fun1 (a, *args):
    print("a={}".format(a))
    for arg in args:
        print('Optional argument: {}'.format( arg ) )

#call function
fun1 (1,22,33)
fun1 (1,22,33,44,55)
fun1 (1,22,33,44,55,66)
```

遞迴函式 (recursive function)

遞迴函式(recursive function)

- 在解決問題時，我們經常會遇到較難使用 loop 或是 if statement 就能處理的問題，像是走迷宮問題遇到死路時需要回到上一層計算、或是在解決河內塔問題時儘管操作相同，卻因為每次要進行操作的參數不同而需要寫重複的程式碼等等
- 遞迴的基本概念
 - 解決一個問題的方法是將其拆解，再各自將小問題解決以後得到答案，這樣的概念我們稱之為 “Divide and Conquer” (分治法)
 - 遞迴(recursion)這個方法就是依據此概念形成
- 當一個函式會在執行當中，會不斷地自己呼叫自己時，我們便認為這個函式具有遞迴的性質
- 為避免函式永無止盡地自我呼叫 (self-calling)，我們也需要設計一個明確的**終止條件**

遞迴函式 (recursive function)

- ▶ 設計一個遞迴函式的兩個重點
 - ▶ 遞迴自我呼叫的方式
 - ▶ 結束呼叫的終止條件
- ▶ 遞迴函式通常可以被for或while迴圈取代，但由於遞迴函式邏輯性、可讀性及彈性均比迴圈來得好

遞迴函式_實例: 計算階乘(4!)

- ▶ 當 $n=0$ 時， $F(n)=n!=0!=1$
- ▶ 當 $n>0$ 時， $F(n)=n!=n*(n-1)!=n*F(n-1)$
- ▶ 當 $n<0$ 時， $F(n)=-1$ 表示無法計算階乘

define recursive function

```
def factorial(n):  
    if n==0 or n==1:  
        return 1  
    else:  
        return n* factorial(n-1)
```

Call function

```
print("0!=" , factorial(0))  
print("4!=" , factorial(4))
```

```
Factorial(4)
```

```
= 4 * Factorial(3)
```

```
= 3 * Factorial(2)
```

```
= 2 * Factorial(1)
```

```
= 1
```

一般函式_實例: 計算階乘

define function

```
def factorial_loop (n):  
    factor = 1  
    for l in range(1,n+1):  
        factor *= l  
    return factor
```

Call function

```
print("4!=", factorial_loop(4))
```

課堂練習題

練習題_1

- 撰寫一個Python程式，讓使用者輸入一個整數，進行判斷是奇數或偶數，回傳值為"奇數"或"偶數"？
 1. 利用自行定義函式
 2. 利用lambda function

練習題_2

- 運用函式一個Python程式，讓使用者輸入起始值和結束值，將這兩筆資料傳給cal函式進行累加及累乘運算，函式結束後回傳累加值及累乘積值。
- 若輸入起始值:3 和結束值:6
- 累加(3+4+5+6):18
- 累乘(3*4*5*6): 360

3. 猜數字的小程式直到猜到為止

- Q1：程式設計人員先給定一個兩位數字以內的正整數(真值)，接著讓使用者猜這個數字("請使用者輸入一個兩位數字以內的正整數:")，然後，程式進行比較大小，若猜測數字>真值，請印出"你猜的數字太大"；若猜測數字<真值，請印出"你猜的數字太小"；若剛好相等，請印出"恭喜你猜對了"
- Q2：請用random module下的randint() function
- Hint: random.randint(a,b) ➔ Return a random integer N such that $a \leq N \leq b$.