

商管程式設計 109-1

TA Lab 6

2020/10/26-30
影片助教 彭晨

Agenda

- Functions
 - Functions and Parameters
 - Variable Scope
 - Return Values
- Recursion
- Coursera 該怎麼找錢（進階版）解答

Function

Function 函數

- 函數：將重複、常被使用的程式碼寫成一包 (**模組化**)
- 要使用 function：
 - (定義函數)
 - 呼叫函數
 - (輸入變數 (variables))



Function 執行

- 當我們呼叫 function 時
 1. 原本的程式暫停，跳到 function
 2. 執行 function 的內容
 3. 跳回到原本程式，繼續執行

如何使用別人寫的 function

1. 不須 `import` 的 function (`print`, `type`, `abs`)

- 直接使用

```
n = -9  
print(abs(-9)) # 9
```

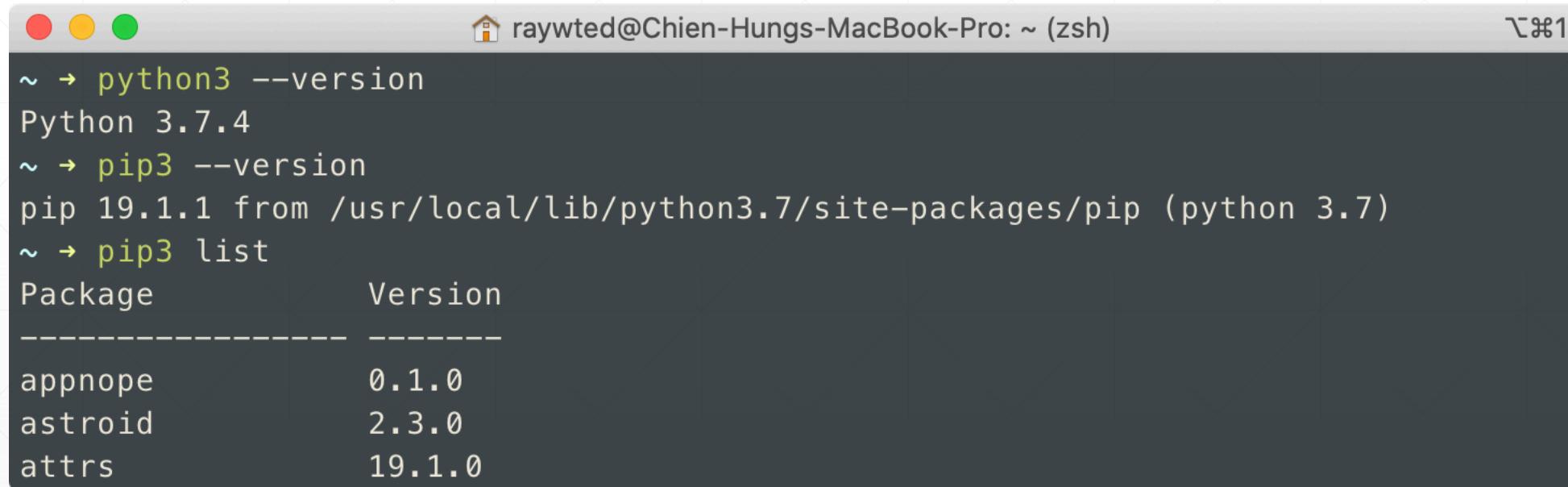
2. 需先 `import` 的 function (`math.sqrt` ...)

- 有些需要另外安裝，有些包含在 python 裡面
- 先 `import library`
- 呼叫時用 `library.function`

```
import math  
n = 9  
a = math.sqrt(n)  
print(a) # 3
```

如何安裝不在 python 裡的 library (1)

1. 打開 cmd / terminal
2. 確認 pip / pip3 版本 (如果需要安裝，請先依照指示安裝)
 - Windows 輸入 : `pip --version`
 - macOS 輸入 : `pip3 --version`



```
raywted@Chien-Hungs-MacBook-Pro: ~ (zsh)
~ → python3 --version
Python 3.7.4
~ → pip3 --version
pip 19.1.1 from /usr/local/lib/python3.7/site-packages/pip (python 3.7)
~ → pip3 list
Package           Version
-----
appnope          0.1.0
astroid          2.3.0
attrs            19.1.0
```

如何安裝不在 python 裡的 library (2)

Windows : 輸入 `pip install library`

macOS : 輸入 `pip3 install library`

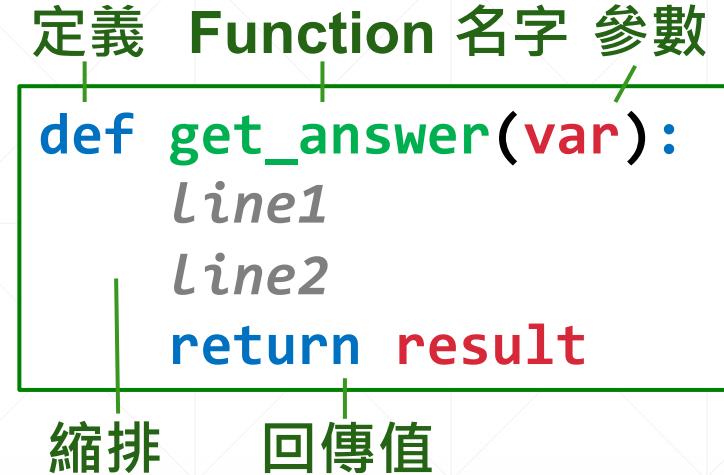
```
pengchen@pengchendeMacBook-Air ~ % pip3 install matplotlib
Collecting matplotlib
  Using cached matplotlib-3.3.2-cp38-cp38-macosx_10_9_x86_64.whl (8.5 MB)
Requirement already satisfied: cycler>=0.10 in /Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages (from matplotlib) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages (from matplotlib) (2.8.1)
Requirement already satisfied: numpy>=1.15 in /Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages (from matplotlib) (1.19.2)
Requirement already satisfied: pyparsing!=2.0.4,!>2.1.2,!>2.1.6,>=2.0.3 in /Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages (from matplotlib) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: pillow>=6.2.0 in /Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages (from matplotlib) (8.0.0)
Requirement already satisfied: certifi>=2020.06.20 in /Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages (from matplotlib) (2020.6.20)
Requirement already satisfied: six in ./Library/Python/3.8/lib/python/site-packages (from cycler>=0.10->matplotlib) (1.15.0)
Installing collected packages: matplotlib
Successfully installed matplotlib-3.3.2
```

以安裝 matplotlib 為例

Functions and Parameters

如何使用自己定義的 function

- `def` 表示開始定義一個 function
 - function name
 - parameters
 - `return` value 將值傳回給呼叫他的人
- 格式：加冒號、固定縮排
- 要先定義完 function 才能呼叫它



預設參數 (default argument)

```
def get_answer(var=3):  
    Line1  
    Line2  
    return result
```

Parameter vs. Argument

- Parameter : 形式參數
 - 定義 function 的時候，需要指定會有哪些參數
- Argument : 實際參數
 - 使用 function 的時候，需要依照當初定義的形式參數，給定正確數量的實際參數
- 實際參數的名字可以不用跟形式參數一樣

```
def my_func(var):  
    return var*2  
num = 4  
print(my_func(num)) # 8
```

形式參數
Parameter

實際參數
Argument

Parameter vs. Argument

- 當我們傳入多個值時，要按照順序傳入 argument，或指定哪一個 argument 對應到哪一個 parameter
 - 有預設值的 parameter 要寫在非預設的後面
 - 有預設值的 parameter，使用時可以省略
 - 有指定的 argument 要寫在沒有指定的後面

```
def borrowing(item, ndays=30):  
    print("Please return %s in %d days." %(item, ndays))  
  
borrowing("cds") # Please return cds in 30 days.  
borrowing("books", ndays=25) # Please return books in 25 days.
```

Function 例子

```
def sing_happybdy():
    print("Happy Birthday to you!")
def sing_to_whom(name):
    print("Happy Birthday to %s!" %name)
sing_happybdy() # Happy Birthday to you!
sing_to_whom("Joy") # Happy Birthday to Joy!
sing_happybdy() # Happy Birthday to you!

def mean(x):
    result = sum(x) / len(x)
    return result
# list comprehension
xlist = [int(j) for j in input().split()]
print(mean(xlist))
# input: 3 4 5
# output: 4.0
```

傳入 Functions 內的 Parameters

`print()`: `sep` & `end` 進行輸出排版

```
>>> print(1, 2, 3) # default sep=' '
1 2 3
>>> print(1, 2, 3, sep=',')
1,2,3
>>> print(1, 2, 3, sep=',', end='!!!!')
# default end='\n'
1,2,3!!! >>>
```

`sorted(list, reverse=bool())`
回傳一個新的 `list`

```
>>> a = [5, 7, 6, 3, 4, 1, 2]
>>> b = sorted(a)
>>> a # 原本的 list a 不變
[5, 7, 6, 3, 4, 1, 2]
>>> b # 由小到大
[1, 2, 3, 4, 5, 6, 7]
```

```
>>> a = [5, 7, 6, 3, 4, 1, 2]
>>> b = sorted(a, reverse=True)
# 由大到小
>>> b
[7, 6, 5, 4, 3, 2, 1]
```

常用 Functions

- `max()` / `min()`
- `len()`
- `abs()`
- `sorted(x, reverse=True)`
- `if item in list`

```
animals = ["PDOGS", "GCATS", "CBIRDS"]
if "PDOGS" in animals:
    print("it's time to do homework!")

classes = ["PD", "DSAP"]
if "PBC" not in classes:
    print("You don't have PBC this semester!")
```

Variable Scope

Local variables vs. Global variables

- **區域變數 (local variables)**
 - 在 function 內宣告
 - **只存在** function 內，不能在該 function 外被使用
 - 不同 function 內變數名稱可以重複 (但不建議)
 - 當 function 執行結束，區域變數在記憶體的位置會釋放，該區域變數消失
 - 可以透過 “**global**” keyword 在 function 內取得全域變數
- **全域變數 (global variables)**
 - 在 function 外宣告
 - 可以在**任何** function 內取得該值
 - 但是如果 function 內有和全域變數相同名字的區域變數，全域變數會**被覆寫掉**

Local variables vs. Global variables

```
# 區域變數只存在於 function 內
def add_together(a):
    b = 1
    sum = a + b
    return sum
print(add_together(2)) # 3
print(b)
# NameError: name 'b' is not defined

# 全域變數會被區域變數暫時覆蓋
b = 2
def add_together(a):
    b = 1
    sum = a + b
    return sum
print(add_together(2)) # 3
```

```
# 全域變數可在 function 內使用 (不建議)
b = 1
def add_together(a):
    sum = a + b
    return sum
print(add_together(2)) # 3

# 用 “global” 取得在 function 外的值
b = 2
def add_together(a):
    global b
    b = 1
    sum = a + b
    return sum
print(add_together(2)) # 3
print(b) # 1
```

Immutable vs. Mutable

- **Immutable objects**
 - 不能被改變
 - `string, int, float, bool...`
 - 傳遞數值到 function 時，數值會被複製一份丟到 function (**call by value**)
 - 所以 function 內的區域變數**不會**影響全域變數
- **Mutable objects** (Mutable: 易變的)
 - 可以被改變
 - `dict, list...`
 - 傳遞數值到 function 時，傳進去的是儲存數值的位置 (**call by reference**)
 - 所以 function 內的區域變數**會**影響全域變數

Passing in immutable and mutable

```
# Passing in immutable
def change_int(a, b):
    save = b
    b = a
    a = save
    print(a, b)
a1, b1 = 3, 5
change_int(a1, b1) # 5, 3
print(a1, b1) # 3, 5
```

```
# Passing in mutable
def change_list(a_list):
    a_list[0] = 4
    print(a_list)
b = [1, 2, 3]
change_list(b) # [4, 2, 3]
print(b) # [4, 2, 3]
```

Return Values

Return Values

- `return` : 回傳程式執行的結果，可以一次傳多個
- 回傳的同時會離開 `function`，將程式的執行控制權交回給 `caller`
 - 可以利用這個特性省略掉很多 `else` 和 `break`

Return 兩個參數
故用兩個參數去接



```
def sumdiff(x, y):  
    sum = x + y  
    diff = x - y  
    return sum, diff  
s, d = sumdiff(7, 4)  
print(s, d) # 11 3
```

Return Values

- 若有多個 `return`，一跑到 `return` 程式即結束

```
def return_even(x):
    if x % 2 == 0:
        return x
    x += 1
    return x

print(return_even(6))
```

Return Values

- Python function 不論是否有 `return statement`，皆有回傳值
- 若沒有指定，則會傳回 `None`
- 若是有回傳值的 function，不可以忽略 `return statement`

```
def haha():
    b = 3
    return b
haha() # 
print(haha())
# 3
```

```
def haha_again():
    b = 3
    haha_again() # 
    print(haha_again())
# None
```

```
def haha_third():
    c = 3
    print(c)
haha_third() # 3
print(haha_third()) # 3
# None
```

Recursion

Recursion

- 遞迴 (Recursion) 是在函式中呼叫自身同名函式
 - 呼叫者本身會先被置入記憶體堆疊 (Stack) 中
 - 等到被呼叫者執行完畢之後，再從堆疊中取出之前被置入的函式繼續執行
- 將問題拆解為
 - base case (無法再分解出子問題)
 - sub-problem case (比原問題容易)

Factorial

- 階層 (factorial) 計算 (遞迴方式)
- $0! = 1$
- $n! = n \times (n - 1)!$

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

factorial	result
call	6
call	2
call	1
call	1

Coursera week3

該怎麼找錢（進階版）解答

題目敘述

如果你在一家零售店幫消費的客人結帳，你可能需要快速地挑出合適且數量正確的鈔票與零錢。假設客人的消費金 a 一定是 1 到 1000 之間的整數，而你有無限量的 500、100、50、10、5、1 這些面額的鈔票和零錢，我們希望你能依照下面的規則找錢：

你找的錢的總額要是 $1000 - a$ 。與其給客人五張 100 元，不如給他一張 500 元；與其給客人兩個 50 元，不如給他一張 100 元……依此類推。

以下是一些範例：

如果客人消費 200 元，你應該找給他 1 張 500 元和 3 張 100 元。

如果客人消費 286 元，你應該找給他 1 張 500 元、2 張 100 元、1 個 10 元和 4 個一元。

如果客人消費 925 元，你應該找給他 1 個 50 元、2 個 10 元和 1 個 5 元。

在本題中，你將會被給予上述的整數 a ，而你要找出符合上述規則的唯一找錢方式。

輸入輸出格式

在每筆測試資料中，會有一個整數 a 代表客人的消費金額， a 會介於 1 到 999 之間（包含 1 跟 999）。讀入 a 之後，你會依照題目指定的規則找出每種面額的鈔票或銅板應該要給幾張或幾個，然後由面額大至面額小依序輸出所需鈔票張數或銅板個數，但如果不需要找給客人某個面額的鈔票或銅板，就跳過該面額不要輸出。因為這樣一來可能只輸出少於 6 個數字，會不知道怎麼對應到面額，因此現在要把面額與所需張數（個數）成對地輸出，中間用一個逗點和一個空格隔開，而面額與面額之間用一個分號和一個空格隔開。

Sample Input

286

Sample Output

500, 1; 100, 2; 10, 1; 1, 4

解答

- Coursera week3 進度版 (無迴圈、無 list)

```
pay = int(input())
change = 1000 - pay

# 將最終的答案儲存在一個字串裡面
answer = str()

if change//500 == 1:
    # 500元的鈔票只可能有一張
    answer += "500, 1"
    change -= 500
    if change != 0:
        answer += "; "
if change//100 >= 1:
    answer += ("100, " + str(change//100))
    change -= (change//100) * 100
    if change != 0:
        answer += "; "
```

```
if change//50 >= 1:
    answer += ("50, " + str(change//50))
    change -= (change//50) * 50
    if change != 0:
        answer += "; "
if change//10 >= 1:
    answer += ("10, " + str(change//10))
    change -= (change//10) * 10
    if change != 0:
        answer += "; "
if change//5 >= 1:
    answer += ("5, " + str(change//5))
    change -= (change//5) * 5
    if change != 0:
        answer += "; "
if change//1 >= 1:
    answer += ("1, " + str(change//1))

print(answer)
```

解答

- 目前進度版

```
pay = int(input())
change = 1000 - pay
money_list = [500, 100, 50, 10, 5, 1]

# 將最終的答案儲存在一個字串裡面
answer = str()

for m in money_list:
    if change // m >= 1:
        answer += (str(m) + ", " + str(change//m)))
        change -= (change//m) * m
    if change != 0:
        answer += ";""

print(answer)
```

重點整理

自訂 function

- 元素：`def`、`function_name`、`parameters`、`return values`
- 格式：加冒號、縮排
- `parameter` 形式參數 vs. `Argument` 實際參數
 - 實際參數跟形式參數名稱不用一樣
- 先定義完 `function` 才能呼叫

```
def multiplication(num1, num2=1):  
    result = num1 * num2  
    return result  
  
a = 2  
b = 3  
print(multiplication(a, b)) # 6  
print(multiplication(a)) # 2
```

Return values

- 回傳程式執行結果，回傳後會離開 function
- 若有多個 return，遇到第一個 return 程式即結束
- 可省略 else 和 break
- 若要回傳值，不可忽略 return

Recursion

- Recursive function
- 在一個函式中呼叫自己
- 將原問題拆成子問題，解出子問題後回推原問題

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

print(factorial(5)) # 120
```

```
n = 5  return 5 * factorial(4) = 5 * 24
n = 4  return 4 * factorial(3) = 4 * 6
n = 3  return 3 * factorial(2) = 3 * 2
n = 2  return 2 * factorial(1) = 2 * 1
n = 1  return 1 * factorial(0) = 1 * 1
n = 0  return 1
```

Practices

Problem 1

- 請設計一支簡易計算機程式，一開始可以選擇模式，接著輸入數字 A 及數字 B。若輸入 1 可以算出 $A + B$ ，若輸入 2 可以算出 $A - B$ ，若輸入 3 則可以算出 $A * B$ 。在本程式中，使用者必須可以輸入**計算模式 t** 以及想要計算的**數字 A** 與**數字 B**，輸出則為**計算結果**。
- 請顯示提示語，並用自訂 function 判斷計算模式，產生不同答案

Input	Sample 1 Input	Sample 2 Input	Sample 3 Input
Enter the type: t	1	2	3
Enter A: A	2	10	1.5
Enter B: B	3	3.8	3
Output	Sample 1 Output	Sample 2 Output	Sample 3 Output
計算結果	5.0	6.2	4.5

Problem 2

- 請設計一支簡易計算機程式，一開始可以選擇模式，若輸入 1 可以算出以 2 為底的對數值，若輸入 2 可以算出以 e 為底的對數值，若輸入 3 則可以算出平方值。在本程式中，使用者輸入計算模式 t 以及想要計算的數字 n，輸出則為計算結果。
- 請顯示提示語，並用自訂 function 判斷計算模式，產生不同答案
- Hint: 使用 library - math: math.log(x[, base])

Input	Sample 1 Input	Sample 2 Input	Sample 3 Input
Enter the type: t	1	2	3
Enter a number: n	8	100	4
Output	Sample 1 Output	Sample 2 Output	Sample 3 Output
計算結果	3.0	4.605170185988092	16.0

Problem 3

- 已知費氏數列為 0, 1, 1, 2, 3, 5, 8, 13,
- 其數學定義為 $F(0) = 0$, $F(1) = 1$, $F(n) = F(n-1) + F(n-2)$
- 輸入 n ，請求出 $F(n)$ 的值
- 請利用一個遞迴 function 計算。

Input	Sample 1 Input	Sample 2 Input
n	10	13
Output	Sample 1 Output	Sample 2 Output
$F(n)$	55	233

進階題

- 要將 p 顆蘋果及 q 顆橘子分給小朋友，每個小朋友拿到的每種水果個數都要一樣，水果不能被分割，請問最多可以分給幾個小朋友呢？
- 請顯示提示語，並利用一個遞迴 function 計算。
- Hint：輾轉相除法，兩個整數的最大公因數等於其中較小的數和兩數相除餘數的最大公因數。

Input	Sample 1 Input	Sample 2 Input
Apple: p	12	13
Orange: q	15	15
Output	Sample 1 Output	Sample 2 Output
小朋友數	3	1