Python, Part. 2

2024 臺大資管營

Benson Chiu

2024/01/30

國立臺灣大學資訊管理學系

本節內容

1. 函數
2. 物件
3. 模組與套件
4. 雜談

函數

函數的定義與呼叫 (1) - 什麼是函數？

- 在 Python 中，函數 (function) 是一段**組織好的、可以重複使用**的程式碼，用於執行特定的任務。
- 可以把函數想像成一個**小型的機器**，當你向它提供一些資料 (稱為**參數** parameters) 時，它會執行一系列操作，然後可能會給你一些結果 (稱為**回傳值** return values)。

函數的定義與呼叫 (2) - 函數的定義方式

- 若要定義一個函數，你需要使用 **`def` 關鍵字**，然後是你的**函數名稱**和**一對括號**，最後以冒號作結。
- 在這對括號裡，你可以放入一些**參數**。
- 在接下來的各行的程式碼描述了函數**要執行的具體操作**。
- 最後，你可以使用 `return` 關鍵字來指定**回傳值**。

函數的定義與呼叫 (3) - 無回傳值的實例

```
def roast(stuff):
    print(" 你看起來就像", stuff, sep="")
    # 沒有回傳值

roast(" 茶碗蒸")
```

如果不需要任何回傳值，可以**不寫 `return`** 或者讓函數**回傳 `None`**。

函數的定義與呼叫 (4) - 無回傳值的實例

```python
def introduce(name, country, skill):
    print(" 我是", name, "，來自", country, "，擅長", skill, sep="")
    # 沒有回傳值

# 用順序傳入函數
introduce(" 椅子", " 香港", " 評論醋飯")

# 用關鍵字傳入函數
introduce(country=" 台灣", name=" 超哥", skill=" 超派鐵拳")
```

除了依照順序將參數對應的值傳入函數之外，在 Python 中，我們可以**用關鍵字傳參數**，在傳入的值前面「註明」其所對應的參數名稱。

函數的定義與呼叫 (5) - 有回傳值的實例

```
def f(x):
    return 2 * x + 5
a = 10
b = f(a)
```

此時，b 被賦予的值為何？

遞迴函數 (1)

遞迴函數 (recursive function) 是一種在其定義中**調用自身**的函數，可用來解決那些**可以分解為相同問題的較小版本**的問題。

範例：階乘

假設 n 是非負整數且 $f(n) = n!$ ，階乘的**遞迴關係式**可以表示為

$$\begin{cases} f(0) = 1 & \text{初始條件} \\ f(n) = n \times f(n-1) & \text{遞迴關係} \end{cases}$$

遞迴函數 (2)

```
def factorial(n):
    if n == 0: # 初始條件
        return 1
    else:
        return n * factorial(n - 1) # 遞迴關係

print(factorial(5))
print(factorial(10))
```

b513. 判斷質數

質數是一個大於 1 的自然數，且除了 1 和它本身以外，無法被其他自然數整除的數字。例如：2、3、5、7 等都是質數。

你的任務是完成一個函數 *is_prime(number)*，這個函數將接受一個整數作為輸入，並返回一個布林值（True 或 False），表明該數字是否為質數。

```
def is_prime(number):
  # 請在這邊填上你的程式碼，並在最後繳交「完整程式碼」

n = int(input())
for i in range(n):
  x = int(input())
  if is_prime(x):
    print('Y')
  else:
    print('N')
```

c002. f91

定義一個遞迴的函數叫做 f91 。它輸入一個正整數 N 並且依據以下的規則傳回一個正整數：

- 如果 $N \leq 100$, 那麼
  $f91(N) = f91(f91(N + 11))$
- 如果 $N \geq 101$, 那麼 $f91(N) = N - 10$

請你寫一個程式來計算 f91 的值。

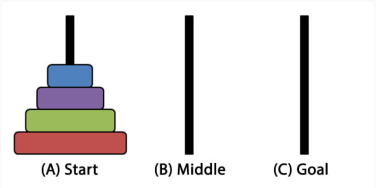
```python
def f91(n):
  # 請在這邊填上你的程式碼
  # 並在最後繳交「完整程式碼」

while True:
  try:
    n = int(input())
    if n == 0:
      break
    else:
      print(f"f91({n}) = {f91(n)}")
  except:
    break
```

a227. 河內塔 (1)

河內塔 (towers of Hanoi) 是一個著名的數學遊戲或謎題。這個遊戲包括三根柱子和一系列大小不同的盤子，這些盤子開始時按照大小順序疊放在一根柱子上，最大的盤子在最下面。遊戲目標是將所有盤子從起始柱子移動到終點柱子上，遵循以下規則：

- 每次只能移動一個盤子。
- 盤子只能從柱頂移動到另一柱的頂端。
- 任何時候，大盤子不能放在小盤子上面。

給定下圖 A 柱的起始盤數 N，輸出把 A 上 N 個環移動到 C 的方法。



a227. 河內塔 (2)

```
def hanoi(n, start, middle, goal):
  if n == 1:
    # 請在這邊填上你的程式碼，並在最後繳交「完整程式碼」
  else:
    # 請在這邊填上你的程式碼，並在最後繳交「完整程式碼」

while True:
  try:
    n = int(input())
    # 請在這邊填上你的程式碼，並在最後繳交「完整程式碼」
  except:
    break
```

物件

物件的基本概念 (1)

- 物件 (object) 是 Python 非常基本且重要的一種資料結構，**幾乎任何事物都可以被視為一種物件**。
- 可以把物件想像成一個小盒子，裡面裝著一些數據（**屬性** attributes）和可以對這些數據進行操作的函數（**方法** methods）。
- **類別** (class) 定義物件的**藍圖**，規範了物件擁有的屬性和方法。
- 當類別被定義後，我們可以根據它來創建具體物件，被稱為**實例** (instance)。

物件的基本概念 (2)

類別

狗	
屬性	名字、品種、年齡、友善程度 .....
方法	吠叫、吃、撒嬌、睡覺、攻擊 .....

實例一

名字：多比　品種：薩摩耶

年齡：4　友善程度：90

實例二

名字：金寶　品種：黃金獵犬

年齡：2　友善程度：100

實例三

名字：可愛狗勾　品種：米克斯

年齡：未知　友善程度：0

內建物件 (1)

內建物件分成兩個主要類別：

- **可變物件** (Mutable Object) 的值「可以」在初始化後能被改變。
  例：list, set, dictionary
- **不可變物件** (Immutable Object) 的值「不可」在初始化後被改變。
  例：int, float, str, tuple

內建物件 (2)

可變物件

```
l = [1, 2, 3]
l[1] = 1
print(l) # [1, 1, 3]
```

不可變物件

```
s = "Hello"
s[1] = "a" # Error
print(s)
```

想想看

1. 假設變數 *a* 原本被賦予的值是 5，後來我們使用 *a = 10* 將變數 *a* 所代表的值改為 10。此時，*a* 的值看似被改變了，為什麼我們還是說 *int* 是一個不可變物件？
2. 假設我們以可變物件（例如：list）作為參數傳入到函數中，並且在函數中修改，請問這樣的行為會影響到函數外嗎？為什麼？

自訂物件 (1) - 類別的定義

在 Python 中，我們可以使用 *class* 關鍵字自定義物件。

```
class Dog:
    # 建構子：對物件的屬性進行初始化
    def __init__(self, name, age, breed):
        self.name = name
        self.age = age
        self.breed = breed
    # 方法
    def bark(self):
        print(" 我是", self.name, "，汪！")
```

自訂物件 (2) - 創建實例

```
# 建立 Dog 類別的實例
dog_1 = Dog(" 多比", 4, " 薩摩耶")
dog_2 = Dog(age=2, name=" 金寶", breed=" 黃金獵犬")
dog_3 = Dog(" 可愛狗勾", None, " 米克斯")

dog_1.bark() # 我是 多比 ，汪！
dog_2.bark() # 我是 金寶 ，汪！
dog_3.bark() # 我是 可愛狗勾 ，汪！
```

物件導向程式設計 (1)

- 英文全名為 Object-Oriented Programming，簡稱為 OOP。
- Python 的**主要設計原則**，旨在提高程式的**模組化**、**可維護性**和**可擴展性**。
- 物件導向的核心：**封裝** (encapsulation)、**繼承** (inheritance) 與**多型** (polymorphism)。

物件導向程式設計 (2) - 封裝

封裝 (encapsulation) 隱藏了具體的實作內容，使用者只需知道輸入格式和預期結果，不需要了解、修改內部的實作細節。

```
a = [1, 3, 2, 4, 5]
a.sort() # 使用者不需要知道具體的實作細節
print(a)
```

物件導向程式設計 (3) - 繼承

繼承 (inheritance) 描述了物件之間的父子關係。

```
class Dog():
    def __init__(self, name, age):
        self.name = name
        self.age = age

class GoldenRetriever(Dog): # 繼承自父類別 Dog
    def __init__(self, name, age):
        super().__init__(name, age)

class Labrador(Dog): # 繼承自父類別 Dog
    def __init__(self, name, age):
        super().__init__(name, age)

my_dog = GoldenRetriever('Willie', 6)
your_dog = Labrador('Lucy', 3)
```

物件導向程式設計 (4) - 多型

多型 (polymorphism) 指不同類別物件對相同的方法會有不同的實現。

```
# 創建不同種類的動物物件
dog = Dog()
cat = Cat()
cow = Cow()

# 調用相同的方法，但根據物件的實際類別，會呼叫不同的實現
print(dog.speak())  # 輸出：Woof!
print(cat.speak())  # 輸出：Meow!
print(cow.speak())  # 輸出：Moo!
```

模組與套件

什麼是模組和套件？

- **模組** (module) **是其他開發者編寫好的程式碼：**這些程式碼通常包含有用的類別和函數，並被儲存於以.py 為副檔名的檔案中。
- **套件** (package) **是更高層次的組織結構：**由多個在同一個資料夾中的相關模組構成。

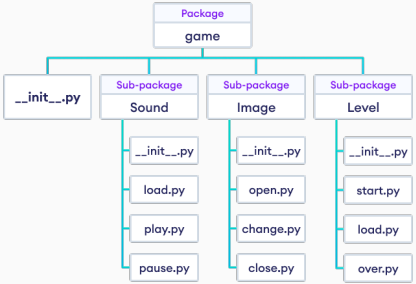


Figure 1: Source: Programiz

引入模組和套件的方法 (1)

```
import 模組或套件名稱
import 模組或套件名稱 as 別名
from 模組或套件名稱 import 子模組, 套件或函數名稱
```

引入模組和套件的方法 (2) - 範例

math 內建模組

```
import math

angle = math.radians(45)
sin_value = math.sin(angle)
cos_value = math.cos(angle)
tan_value = math.tan(angle)

print("sin(45°):", sin_value)
print("cos(45°):", cos_value)
print("tan(45°):", tan_value)
```

引入模組和套件的方法 (3) - 範例

小試身手[1]

1. 請用 *math* 內建模組計算 20 的階乘並印出。
2. 請用 *math* 內建模組計算 C_3^{20} 的值並印出。
3. 請用 *math* 內建模組計算 -0.00001 的絕對值後印出。
4. 請用 *math* 內建模組計算 8765 和 43210 的最大公因數後印出。

[1]請透過 *https://docs.python.org/zh-tw/3/library/math.html* 查詢你需要使用的函數

引入模組和套件的方法 (4) - 範例

datetime 內建模組

```
from datetime import datetime, date, time
# 獲取當前日期和時間
current_datetime = datetime.now()
print(" 當前日期和時間:", current_datetime)
# 創建一個特定的日期
specific_date = date(2023, 8, 31)
print(" 特定日期:", specific_date)

# 創建一個特定的時間
specific_time = time(14, 30, 0)
print(" 特定時間:", specific_time)

# 創建一個特定的日期和時間
specific_datetime = datetime(2023, 8, 31, 14, 30, 0)
print(" 特定日期和時間:", specific_datetime)
```

引入模組和套件的方法 (5) - 範例

小試身手[2]

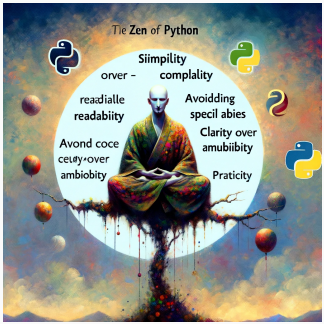
1. 請用 *datetime* 內建模組印出現在的日期和時間。
2. 請用 *datetime* 內建模組將字串 “24/01/30 14:00:00” 轉換成 *datetime* 物件。(提示：可以使用 *datetime.strptime( )*)
3. 請用 *datetime* 內建模組將 *datetime* 物件轉成格式為 “yyyy-mm-dd hh:mm:ss” 的字串。(提示：可以使用 *datetime_object.strftime( )*)

[2]請透過 *https://docs.python.org/zh-tw/3/library/datetime.html* 查詢你需要使用的函數

引入模組和套件的方法 (6) - 彩蛋

Python 之禪

```
import this
```



Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one– and preferably only one –obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than right now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea – let's do more of those!

常見的內建模組[3]

Python 的**標準函式庫**提供許多內建模組，一些常見的包括：

1. os：管理文件和目錄。
2. sys：存取與 Python 直譯器相關的資訊。
3. math：基本的數學運算。
4. datetime：處理日期和時間。
5. json：讀寫 JSON 格式的數據。
6. random：生成隨機數。

[3]關於內建模組，詳細的介紹請參閱 *https://docs.python.org/zh-tw/3/tutorial/stdlib.html*

第三方函式庫

除了本身提供的內建模組之外，Python 社群和開發者提供了大量的**第三方函式庫**[4]，它們豐富了 Python 的生態系統，使其成為功能強大且多用途的程式語言。

[4]關於第三方函式庫，詳細的索引請參閱 *https://pypi.org*

雑談

串列生成式 (List Comprehension)：沒教到，但很好用的東西

使用方式：

```
result = [expression for item in iterable]
```

範例：

```
# Before List Comprehension
squares = []
for x in range(10):
    squares.append(x**2)
print(squares) # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# After List Comprehension
squares = [x**2 for x in range(10)]
print(squares) # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

總結

課程目標：對 Python 語法有**最粗淺的認識**。

- **數位決策**課程會以 Python 作為工具完成資料分析的任務。
- 除此之外，Python 還有很多**廣泛的應用面向**等著大家去探索。
- 順便業配，歡迎大家參加**資訊之芽**。

一些建議：

- 高中生寫程式，不是只侷限於競賽和 APCS。
- 找到喜歡的學習模式。
- 培養自學、獨立解決問題與提問的能力。
- 不要比較，不要計較，不要自我侷限。

「勇氣」就是克服困難的活力。

缺乏勇氣的人，一旦遇到困難，就會墜入人生的黑暗深淵。—— Alfred Adler.

Feel free to ask us anything:)