第0章

# 啟程之前，請不要錯過我

## 0.1　Python精簡入門

設計模式與程式設計語言沒有關係，它是對物件導向思想的靈活應用和高度概括，你可以用任何一種語言來實現它，但還是需要用一種語言來舉例。除特別說明外，本書的所有示例源碼，均採用Python實現。如果你初次接觸Python，請務必先閱讀本章的內容；如果你已經很熟悉Python，可直接跳過本章的內容。

### 0.1.1　Python的特點

Python崇尚優美、清晰、簡單，是一種優秀並被廣泛使用的語言。

與Java和C++這些語言相比，Python最大的幾個特點是：

（1）語句結束不用分號“;”。

（2）代碼塊用縮進來控制，而不用大括弧“{}”。

（3）變數使用前不用事先聲明。

從其他語言剛轉到Python的時候可能會有點不適應，用一段時間就好了！

個人覺得，在所有的高級電腦語言中，Python是最接近人類的自然語言的。Python的語法、風格都與英文的書寫習慣非常接近，Python的這種風格被稱為Pythonic。如條件運算式，在Java和C++語言中是這樣的：

int min = x < y ? x : y

而在Python語言中是這樣的：

min = x if x < y else y

有沒有覺得第二種方式更接近人類的自然思維？

### 0.1.2　基本語法

1. 資料類型

Python是一種動態語言，定義變數時不需要在前面加類型說明， 而且不同類型之間可以方便地相互轉換。Python有五個標準的資料類型：

（1）Numbers（數字）

（2）String（字串）

（3）List（列表）

（4）Tuple（元組）

（5）Dictionary（字典）

其中List、Tuple、Dictionary為容器，將在下一部分介紹。Python支援四種不同的數位類型：int（有符號整型）、float（浮點型）、complex（複數）（說明：Python 3中已去除long類型，與int類型合併）。

每個變數在使用前都必須賦值，變數賦值以後才會被創建。

源碼示例0-1

age = 18 # int

weight = 62.51 # float

name = "Tony" # string

print("age:", age)

print("weight:", weight)

print("name:", name)

# 變數的類型可以直接改變

age = name

print("age:", age)

a = b = c = 5

# a、b、c三個變數指向相同的記憶體空間，具有相同的值

print("a:", a, "b:", b, "c:", c)

print("id(a):", id(a), "id(b):", id(b), "id(c):", id(c))

輸出結果：

age: 18

weight: 62.51

name: Tony

age: Tony

a: 5 b: 5 c: 5

id(a): 1457772400 id(b): 1457772400 id(c): 1457772400

2. 常用容器

##### 1）List

List（列表）是Python中使用最頻繁的資料類型，用“[ ]”標識。清單可以完成大多數集合類的資料結構實現，類似於Java中的ArrayList和C++中的Vector。此外，一個List中還可以同時包含不同類型的資料，支援字元、數位、字串，甚至可以包含清單（即嵌套）。

（1）列表中值的切割也可以用到變數[頭下標:尾下標]，這樣就可以截取相應的列表，從左到右索引預設從0開始，從右到左索引預設從-1開始，下標可以為空（表示取到頭或尾）。

（2）加號（+）是列表連接運算子，星號（\*）是重複操作。

源碼示例0-2

list = ['Thomson', 78, 12.58, 'Sunny', 180.2]

tinylist = [123, 'Tony']

print("list:", list) # 輸出完整列表

print("list[0]:", list[0]) # 輸出清單的第一個元素

print("list[1:3]:", list[1:3]) # 輸出第二個至第三個元素

print("list[2:]:", list[2:]) # 輸出從第三個開始至清單末尾的所有元素

print("tinylist \* 2 :", tinylist \* 2) # 輸出列表兩次

print("list + tinylist :", list + tinylist) # 列印組合的清單

list[1] = 100

print("設置list[1]:", list) # 輸出完整列表

list.append("added data")

print("list添加元素:", list) # 輸出增加後的列表

輸出結果：

list: ['Thomson', 78, 12.58, 'Sunny', 180.2]

list[0]: Thomson

list[1:3]: [78, 12.58]

list[2:]: [12.58, 'Sunny', 180.2]

tinylist \* 2 : [123, 'Tony', 123, 'Tony']

list + tinylist : ['Thomson', 78, 12.58, 'Sunny', 180.2, 123, 'Tony']

設置list[1]: ['Thomson', 100, 12.58, 'Sunny', 180.2]

list添加元素: ['Thomson', 100, 12.58, 'Sunny', 180.2, 'added data']

##### 2）Tuple

Tuple（元組）是另一種資料類型，用“()”標識，內部元素用逗號隔開。元組不能二次賦值，相當於唯讀列表，用法與List類似。Tuple相當於Java中的final陣列和C++中的const陣列。

源碼示例0-3

tuple = ('Thomson', 78, 12.58, 'Sunny', 180.2)

tinytuple = (123, 'Tony')

print("tuple:", tuple) # 輸出完整元組

print("tinytuple:", tinytuple) # 輸出完整元組

print("tuple[0]:", tuple[0]) # 輸出元組的第一個元素

print("tuple[1:3]:", tuple[1:3]) # 輸出第二個至第三個元素

print("tuple[2:]:", tuple[2:]) # 輸出從第三個開始至清單末尾的所有元素

print("tinytuple \* 2:", tinytuple \* 2) # 輸出元組兩次

print("tuple + tinytuple:", tuple + tinytuple) # 列印組合的元組

# tuple[1] = 100 # 不能修改元組內的元素

輸出結果：

tuple: ('Thomson', 78, 12.58, 'Sunny', 180.2)

tinytuple: (123, 'Tony')

tuple[0]: Thomson

tuple[1:3]: (78, 12.58)

tuple[2:]: (12.58, 'Sunny', 180.2)

tinytuple \* 2: (123, 'Tony', 123, 'Tony')

tuple + tinytuple: ('Thomson', 78, 12.58, 'Sunny', 180.2, 123, 'Tony')

##### 3）Dictionary

Dictionary（字典）是Python中除列表以外最靈活的內置資料結構類型。字典用“{ }”標識，由索引（key）和它對應的值value組成。相當於Java和C++中的Map。

清單是有序的物件集合，字典是無序的物件集合。兩者之間的區別在於：字典中的元素通過鍵存取，而不通過偏移存取。

源碼示例0-4

dict = {}

dict['one'] = "This is one"

dict[2] = "This is two"

tinydict = {'name': 'Tony', 'age': 24, 'height': 177}

print("tinydict:", tinydict) # 輸出完整的字典

print("tinydict.keys():", tinydict.keys()) # 輸出所有鍵

print("tinydict.values():", tinydict.values()) # 輸出所有值

print("dict['one']:", dict['one']) # 輸出鍵為'one' 的值

print("dict[2]:", dict[2]) # 輸出鍵為 2 的值

結果：

tinydict: {'name': 'Tony', 'age': 24, 'height': 177}

tinydict.keys(): dict\_keys(['name', 'age', 'height'])

tinydict.values(): dict\_values(['Tony', 24, 177])

dict['one']: This is one

dict[2]: This is two

3. 類的定義

使用class語句來創建一個新類，class之後為類的名稱並以冒號結尾，實例如下：

class ClassName:

'類的説明資訊' #類文檔字串

class\_suite #類體

類的説明資訊可以通過ClassName.\_doc\_查看，類體（class\_suite）由類成員、方法、資料屬性組成，舉例如下。

源碼示例0-5

class Test:

"這是一個測試類"

def \_\_init\_\_(self):

self.\_\_ivalue = 5

def getvalue(self):

return self.\_\_ivalue

其中\_init\_為初始化函數，相當於構造函數。

##### 1）存取權限

* \_foo\_：定義的是特殊方法，一般是系統定義名字，類似\_init\_()。
* \_foo：以單下畫線開頭時表示的是 protected 類型的變數，即保護類型只允許其本身與子類進行訪問，不能用於 from module import \*。
* \_\_foo：以雙下畫線開頭時，表示的是私有類型（private）的變數，即只允許這個類本身進行訪問。

##### 2）類的繼承

類的繼承語法結構如下：

class 派生類名（基類名）：類體

Python中繼承中的一些特點：

（1）在繼承中基類的初始化方法\_init\_()不會被自動調用，它需要在其派生類的構造中親自專門調用。

（2）在調用基類的方法時，需要使用super()首碼。

（3）Python總是首先查找對應類型的方法，不能在派生類中找到對應的方法時，它才開始到基類中逐個查找（先在本類中查找調用的方法，找不到才去基類中找）。

如果在繼承元組中列了一個以上的類，那麼它就被稱作“多重繼承”。

##### 3）基礎重載方法

Python的類中有很多內置的基礎重載方法，我們可以通過重寫這些方法來實現一些特殊的功能。這些方法如表0-1所示。

表0-1　基礎重載方法

|  |  |  |  |
| --- | --- | --- | --- |
| 序　　號 | 方　　法 | 描　　述 | 簡單的調用 |
| 1 | \_*init\_* ( self [,args...] ) | 構造函數 | obj = className(args) |
| 2 | \_*del\_*( self ) | 析構方法，刪除一個物件 | del obj |
| 3 | \_*repr\_*( self ) | 轉化為供解譯器讀取的形式 | repr(obj) |
| 4 | \_*str\_*( self ) | 用於將值轉化為適於人閱讀的形式 | str(obj) |
| 5 | \_*cmp\_* ( self, x ) | 物件比較 | cmp(obj, x) |

### 0.1.3　一個例子讓你頓悟

我們將一段Java代碼對應到Python中來實現，進行對比閱讀，相信你很快就能明白其中的用法。

源碼示例0-6　Java代碼

class Person {

public static int visited;

Person(String name, int age, float height) {

this.name = name;

this.age = age;

this.height = height;

}

public String getName() {

return name;

}

public int getAge() {

return age;

}

public void showInfo() {

System.out.println("name:" + name);

System.out.println("age:" + age);

System.out.println("height:" + height);

System.out.println("visited:" + visited);

Person.visited ++;

}

private String name;

protected int age;

public float height;

}

class Teacher extends Person {

Teacher(String name, int age, float height) {

super(name, age, height);

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public void showInfo() {

System.out.println("title:" + title);

super.showInfo();

}

private String title;

}

public class Test {

public static void main(String args[]) {

Person tony = new Person("Tony", 25, 1.77f);

tony.showInfo();

System.out.println();

Teacher jenny = new Teacher("Jenny", 34, 1.68f);

jenny.setTitle("教授");

jenny.showInfo();

}

}

源碼示例0-7　對應的Python代碼

class Person:

"人"

visited = 0

def \_\_init\_\_(self, name, age, height):

self.\_\_name = name # 私有成員，存取權限為private

self.\_age = age # 保護成員，存取權限為protected

self.height = height # 公有成員，存取權限為public

def getName(self):

return self.\_\_name

def getAge(self):

return self.\_age

def showInfo(self):

print("name:", self.\_\_name)

print("age:", self.\_age)

print("height:", self.height)

print("visited:", self.visited)

Person.visited = Person.visited +1

class Teacher(Person):

"老師"

def \_\_init\_\_(self, name, age, height):

super().\_\_init\_\_(name, age, height)

self.\_\_title = None

def getTitle(self):

return self.\_\_title

def setTitle(self, title):

self.\_\_title = title

def showInfo(self):

print("title:", self.\_\_title)

super().showInfo()

def testPerson():

"測試方法"

tony = Person("Tony", 25, 1.77)

tony.showInfo()

print();

jenny = Teacher("Jenny", 34, 1.68);

jenny.setTitle("教授");

jenny.showInfo();

testPerson()

源碼示例0-6和源碼示例0-7的結果是一樣的，如下：

name: Tony

age: 25

height: 1.77

visited: 0

title: 教授

name: Jenny

age: 34

height: 1.68

visited: 1

### 0.1.4　重要說明

（1）為了降低程式複雜度，本書用到的所有示例代碼均不考慮多執行緒安全，望借鑒源碼的讀者注意。

（2）本書所有源碼均在Python 3.6.3下編寫，Python 3.0以上都可以正常運行。

（3）源碼地址：https://github.com/luoweifu/PyDesignPattern，本書所有原始程式碼可在此免費閱讀和下載。

## 0.2　UML精簡概述

### 0.2.1　UML的定義

UML是英文 Unified Modeling Language 的縮寫，簡稱**UML**（**統一模組化語言**），它是一種由一整套圖組成的標準化建模語言，用於説明系統開發人員闡明、設計和構建軟體系統。

UML的這一整套圖被分為兩組，一組叫結構性圖，包含類圖、組件圖、部署圖、物件圖、包圖、組合結構圖、輪廓圖；一組叫行為性圖，包含用例圖、活動圖（也叫流程圖）、狀態機圖、序列圖、通信圖、交互圖、時序圖。其中類圖是應用最廣泛的一種圖，經常被用於軟體架構設計中。

### 0.2.2　常見的關係

**類圖**用於表示不同的實體（人、事物和資料），以及它們彼此之間的關係。該圖描述了系統中物件的類型以及它們之間存在的各種靜態關係，是一切物件導向方法的核心建模工具。

UML類圖中最常見的幾種關係有：泛化（Generalization）、實現（Realization）、組合（Composition）、聚合（Aggregation）、關聯（Association）和依賴（Dependency）。這些關係的強弱順序為：泛化 = 實現 > 組合 > 聚合 > 關聯 > 依賴。

1. 泛化

**泛化（Generalization）**是一種繼承關係，表示一般與特殊的關係，它指定了子類如何特化父類的所有特徵和行為。

如：哺乳動物具有恒溫、胎生、哺乳等生理特徵，貓和牛都是哺乳動物，也都具有這些特徵，但除此之外，貓會捉老鼠，牛會耕地，如圖0-1所示。

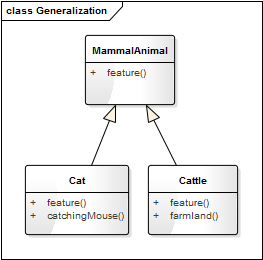


圖0-1　泛化

2. 實現

**實現（Realization）**是一種類與介面的關係，表示類是介面所有特徵和行為的實現。

如：蝙蝠也是哺乳動物，它除具有哺乳動物的一般特徵之外，還會飛，我們可以定義一個IFlyable的介面，表示飛行的動作，而蝙蝠需要實現這個介面，如圖0-2所示。

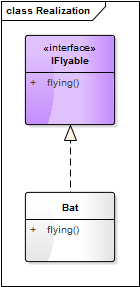


圖0-2　實現

3. 組合

**組合（Composition）**也表示整體與部分的關係，但部分離開整體後無法單獨存在。因此，組合與聚合相比是一種更強的關係。

如：我們的電腦由CPU、主機板、硬碟、記憶體組成，電腦與CPU、主機板、硬碟、記憶體是整體與部分的關係，但如果讓CPU、主機板等元件單獨存在，就無法工作，因此沒有意義，如圖0-3所示。

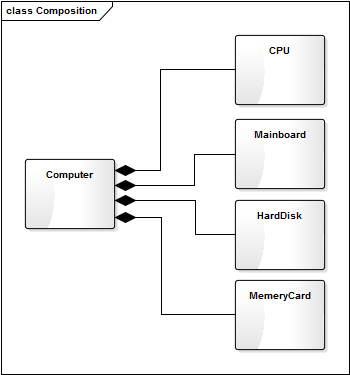


圖0-3　組合

4. 聚合

**聚合（Aggregation）**是整體與部分的關係，部分可以離開整體而單獨存在。

如：一個公司會有多個員工，但員工可以離開公司單獨存在，離職了依然可以好好地活著，如圖0-4所示。

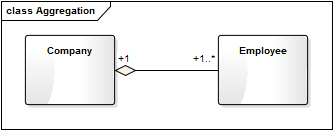


圖0-4　聚合

5. 關聯

**關聯（Association）**是一種擁有關係，它使一個類知道另一個類的屬性和方法。關聯可以是雙向的，也可以是單向的。

如：一本書會有多個讀者，一個讀者也可能會有多本書，書和讀者是一種雙向的關係（也就是多對多的關係）；但一本書通常只會有一個作者，是一種單向的關係（就是一對一的關係，也可能是一對多的關係，因為一個作者可能會寫多本書），如圖0-5所示。

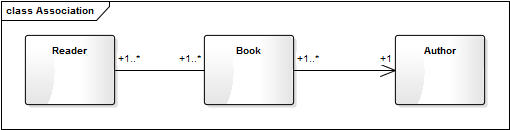


圖0-5　關聯

6. 依賴

**依賴（Dependency）**是一種使用的關係，即一個類的實現需要另一個類的協助，所以儘量不要使用雙向的互相依賴。

如：所有的動物都要吃東西才能活著，動物與食物就是一種依賴關係，動物依賴食物而生存，如圖0-6所示。

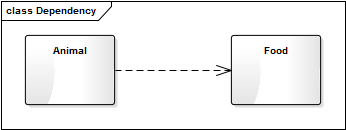


圖0-6　依賴