Class & OOP

2024 資訊之芽 py 班
黃千睿

Class

類別 (class)

如果要存入各種食物的各種資料，可以怎麼做?

```
# [name, price, weight]
food = [["bread", 40, 100],["croissant", 100, 150]]
```

```
# {name : [price, weight]}
food = {"bread": [40, 100],"croissant" : [100, 150]}
```

類別 (class)

- 我們可以用一個定義好的 `class` ，來產生具有相似構造的物件 `(object)`
- `class` 像是物件的說明書，告訴你產生的物件具有哪些構造
- 對 `class` 而言，每個用它產生的物件就稱作這個 `class` 的 `instance`

基本語法

- 利用自己定義的 class 建立一個物件
- class 的名稱大家會習慣大寫

```
class Food():           # 宣告 要加括號
    pass                # class 的內部細節

bread = Food()          #建立物件
```

Attribute

屬性 (attribute)

- 一個物件可以存放資料
- ex : 食物的價錢、重量、成分等等

可以在創造物件後加入屬性

```
class Food():              # 宣告 要加括號
    pass                   # class 的內部細節

bread = Food()             #建立物件
bread.price = 40           #新增屬性
```

屬性 (attribute)

如果這個屬性是這個 class 產生的所有物件都要有的呢？

也可以在創造物件時就加入屬性

```
class Food():
      def __init__(self):  # 要加一個位置參數(self)，代表新建立的物件
            self.price = 40  # 透過初始化函式(__init__)設定屬性

bread = Food()
print(bread.price) #取值方法
```

位置參數 (self) ？？ 初始化函式 (init)？？

init ?? self ??

- `init` 是創造物件時會自動被呼叫的函式，因為是特殊的函式，所以要寫成 `__init__` 的形式
- 會給你即將出爐的物件 當作函式的第一個參數 (`self`)
- 參數的名字不一定要叫 `self` ， 可以隨便你取

```
class Food():
      def __init__(s): # 來看看第一個傳入的東西到底是啥
            print(s)
            print(type(s))
            print(id(s))

bread = Food()
print(id(bread))
```

- 同前面，參數的名字不一定要叫 `self` ， 可以隨便你取
- `s` 是一個 `Food` 這個 `class` 的 `object`
- 透過 `id()` ，我們可以確認 `s` 和我們創造出來的物件 `bread` 是同一個東西

More on attribute

```
class Food():
      def __init__(self,price):
            self.price = price  # 透過初始化函式設定屬性

bread = Food(40)
print(bread.price) #取值方法
```

- 第二個位置的參數才會是 Food() 中的第一個參數
- 第三，四個以此類推

舉個例子

```
class Food():
      def __init__(self, price, size, rating):
            self.price = price
            self.size = size
            self.rating = rating

cupcake = Food(50,"M",100)
print(cupcake.size, cupcake.price)
```

改變屬性的值

```
class Food():
      def __init__(self, price, size, rating):
            self.price = price
            self.size = size
            self.rating = rating

cupcake = Food(50,"M",100)
print(cupcake.rating)

cupcake.rating = 50
print(cupcake.rating)
```

刪除

```
class Food():
      def __init__(self,price):     # 加入屬性的方法
            self.price = price

bread = Food(40)
cupcake = Food(50)
del cupcake              # 刪除物件

del bread.price
print(bread.price)     # 會壞掉
```

Class attribute

- 剛剛使用的 self.price, self.size, … 叫做 instance attribute ，是屬於物件本身的屬性
- class attribute 是屬於 class 的屬性

```
class Food():
      sold = 0            # class attribute
      def __init__(self, price):
            self.price = price
            Food.sold += 1

bread = Food(40)
print(Food.sold) # 1

cupcake = Food(50)
print(Food.sold) # 2
```

Class attribute (補充)

- 我們也可以透過物件存取 `class attribute`
- 但如果透過物件改變 `class attribute` 的值，他就會變成屬於該物件的獨立 `attribute`

```
print(bread.sold)          #2
bread.sold += 4

print(bread.sold)          #6
print(cupcake.sold)        #2
print(Food.sold)           #2

donut = Food(30)
print(bread.sold)          #6
print(cupcake.sold)        #3
print(Food.sold)           #3
```

練習時間

1. 利用 神奇寶貝圖鑑，紀錄 (三個以上) 神奇寶貝的編號、體重和屬性

2. 使用 `class attribute` 紀錄所有神奇寶貝的平均體重

* 完成之後先把程式留著，下一次練習繼續使用

Method

方法(method)

Q：物件裡面能放資料，那能不能放函式？

A：可以

```
class Food():
      def __init__(self, price):
            self.price = price

      def check_price(self):
            if self.price > 50:
                  print("expensive")

croissant = Food(100)
croissant.check_price()
```

Instance method

- instance method 是針對個別物件的 method
- python 會把呼叫 instance method 的物件傳入第一個參數 (self)

```
class Food():
	def __init__(self, price):
		self.price = price

	def modify_price(self, factor):
		self.price = factor * self.price

croissant = Food(100)
croissant.modify_price(4)
print(croissant.price)
```

Instance method

- `__init__` 就是一個把即將出爐的物件當作參數的 `instance method`
- 可以傳入其他參數或物件

```
class Food():
	def __init__(self, price, rating):
		self.price = price
		self.rating = rating

	def compare(self, other):
		return self.rating > other.rating

croissant = Food(100, 80)
bread = Food(40, 70)
print(croissant.compare(bread))
```

class method

- class method 是針對一個 class 的 method
- python 會把呼叫 class method 的 class 傳入第一個參數 (self)
- 可以影響 class 本人和用 class 生出來的 object

```
class Food():
      sold = 0
      def __init__(self, price):
            self.price = price
            Food.sold += 1
      @classmethod
      def foodSold(cls):
             print(f"Number of food sold : {cls.sold}")

Food.foodSold()     #0
croissant = Food(100)
Food.foodSold()     #1
```

class method

- @classmethod 叫做 decorator，簡單來說就是給一個 function 上 buff 的神奇語法

- 在這個例子裏面，decorator會自動幫我們把 foodSold(cls) 這個 function 所屬的 class 餵給 foodSold(cls) 當中的 cls 參數

```
class Food():
        sold = 0

        @classmethod
        def foodSold(cls):
                print(f"Number of food sold : {cls.sold}")
```

static method

- static method 不傳入物件或類別

```
class Food():
	sold = 0
	def __init__(self, price):
		self.price = price
		Food.sold += 1

	@staticmethod
	def promote():
		print("三年三班手工薯條超級好吃")

fries = Food(40)
fries.promote()     # 透過 object 呼叫
Food.promote()      # 透過 class 呼叫
```

Summary

	Instance method	Class method	Static method
對象	物件	類別	None
參數	`func(self)`	`func(cls)`	`func()`
呼叫	`obj.func()`	`cls.func()`	`cls.func()` `obj.func()`

練習時間

延續前一次的練習，請寫出下列的 `instance method`

- `isType(self, type: str) -> bool`，讓使用者可以檢查神奇寶貝是否為該屬性
- `modifyWeight(self, num: int) -> None`，讓使用者可以更改一個神奇寶貝的體重，同時更新平均體重

補充

- Magic method : https://www.tutorialsteacher.com/python/magic-methods-in-python

包在我身上

OOP

OOP

- Object Oriented Programming
- 把東西包成物件方便重複使用、管理
- 可以使用 python 中的 class / object
- 三大特色：

  - 封裝 Encapsulation
  - 繼承 Inheritance
  - 多型 Polymorphism

封装

封裝

- 將 attribute 分為 public 與 private
- private attribute 前面有雙底線，且只能透過 method 取值和更改

```
class Profile():
      def __init__(self, name, assets):
            self.name = name
            self.__assets = assets # 你的財產
      def getAssets(self):
            return self.__assets

member = Profile("Enzo", 10000)
print(member.getAssets())        # 10000
print(member.__assets)           # AttributeError
```

封裝

private attribute 前面有雙底線，且只能透過 method 取值和更改

```
class Profile():
      def __init__(self, name, assets):
            self.name = name
            self.__assets = assets # 你的財產
      def getAssets(self):
            return self.__assets
      def deposit(self, num):
            self.__assets += num

member = Profile("Enzo", 10000)
member.deposit(3000)
print(member.getAssets())        # 13000
member.__assets += 1000000       # AttributeError
```

練習

把 name 也改成 private attribute 吧 (需要存取用的 method)

繼承

繼承

我們有一個 class -> Person

```
class Person():
	def __init__(self, name, age):
		self.name = name
		self.age = age
	def printInfo(self):
		print(self.name, self.age)
```

我們還需要另一個 class -> Student，除了有 Person 原本的屬性和 method ，還需要再加上額外的屬性及 method

可以怎麼做呢？

- 利用繼承 (inheritance) 可以輕鬆的完成
- 在這個情況下，Person 為 parent ， Student 為 child
- 可以利用 super() 存取 parent 的 method

```
class Student(Person):
      def __init__(self, name, age, score):
            super().__init__(name, age)      # 存取 parent 的 __init__()
            self.score = score

      def isQualified(self):
            return self.score > 60

student1 = Student("Alice", 18, 70)
student1.printInfo()
print(student1.isQualified())
```

* Student 也可以被其他 class 繼承哦

多型

多型

- 當我們使用繼承時，child 可以自己重新定義 parent 提供的 method
- 一個 parent 可以被很多 child class 繼承
- 可以透過這個方式讓這些 class 的物件執行相同名稱，但不同效果的 method -> 多型 (polymorphism)

```
class Animal():
	def __init__(self):
		pass
	def sound(self):
		return "Ahhhh"

class Cow(Animal):
	def __init__(self):
		super().__init__()
	def sound(self):
		return "MooMoo"

class Bird(Animal):
	def __init__(self):
		super().__init__()
	def sound(self):
		return "ChuChu"

def make_sound(obj):
	print(obj.sound())
```

- 透過 Cow 和 Bird 裡的 sound method 輸出不同聲音

```
newCow = Cow()
newBird = Bird()

make_sound(newCow)
make_sound(newBird)
```

- 如果繼承的 class 沒定義 sound method 呢

```
class Cat(Animal):
        def __init__(self):
                super().__init__()
newCat = Cat()
make_sound(newCat)
```

會使用 parent 的 sound method ， 輸出 "Ahhhh"

Thank you