Python Object-Oriented Programming

馬誠佑

2025/04/11

Class & Object

- 類別（class）為一自訂的資料型態，裡面可包含資料（變數/屬性）與程式碼（函式/方法）。

- 物件（object）為一以自訂的類別（class）為型態的實體。

class

```
class ClassName:
    data = 0
    ....
    def __init__(self, para1, para2, ...):
        ....
        ....
    def classFunction1(self, para1, para2, ...):
        ....
        ....
```

- 繼承
- 多型

```python
class Cat:
    pass

a_cat = Cat()
another_cat = Cat()
print(a_cat)
a_cat.name = 'Mimi'
a_cat.age = 2
a_cat.nemesis = another_cat
print(a_cat.name)
print(a_cat.age)
print(a_cat.nemesis)
another_cat.name = 'Garfield'
print(a_cat.nemesis.name)
a_cat.nemesis.name = 'Felix'
print(another_cat.name)
```

✓ 0.0s

```
<__main__.Cat object at 0x111a153a0>
Mimi
2
<__main__.Cat object at 0x111a54ca0>
Garfield
Felix
```

初始化

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

a_cat = Cat('Mimi', 2)
another_cat = Cat('Garfield', 5)
print(a_cat.name)
print(a_cat.age)
print(another_cat.name)
print(another_cat.age)
```

✓ 0.0s

```
Mimi
2
Garfield
5
```

類別定義不一定要有__init__()方法

繼承

- 當你想要改寫某些類別或是擴充它的功能時，直接修改舊類別可能將它複雜化，或甚至破壞本來的正常功能。此時可以使用繼承。
- 繼承：用既有的類別建立一個新類別，加入一些新東西或修改它。
- 當新類別繼承一個類別時，新類別可以自動使用舊類別的所有程式碼。
- 被繼承的類別稱為父類別(parent)、超類別(superclass)或基礎類別(base class)
- 新類別稱為子類別(child、subclass)或衍生類別(derived class)

```python
class Car:
    def __init__(self, wheels_number=4, car_doors=4, passengers=4):
        self.wheels_number = wheels_number
        self.car_doors = car_doors
        self.passengers = passengers

    def drive(self):
        print('driving the car.')
class SUV(Car):
    def __init__(self, wheels_number, car_doors, passengers, brand_name='', air_bag=2, sunroof=False):
        super().__init__(wheels_number, car_doors, passengers)
        self.brand_name = brand_name
        self.air_bag = air_bag
        self.sunroof = sunroof

    def drive(self):
        print('driving the SUV.') #多型

    def getDetails(self):
        print("Brand:", self.brand_name)
        print("Wheels number:", self.wheels_number)  # 可直接呼叫父類別的變數(屬性)
        print("Doors number:", self.car_doors)  # 可直接呼叫父類別的變數(屬性)
        print("Air-bags number:", self.air_bag)
        print("Sunroof:", self.sunroof)

toyota_rav = SUV(4, 5, 5, "Toyota RAV", 4, True)
toyota_rav.getDetails()
bmw_x5 = SUV(4, 5, 5, "BMW X5", 6, True)
bmw_x5.getDetails()
```

✓ 0.0s

```
Brand: Toyota RAV
Wheels number: 4
Doors number: 5
Air-bags number: 4
Sunroof: True
Brand: BMW X5
Wheels number: 4
Doors number: 5
Air-bags number: 6
Sunroof: True
```

- 子類別可增加新的變數（屬性）
- 子類別可覆寫或添加方法（Function）

- issubclass()可查看一個類別是否衍生自另一個類別

```python
issubclass(SUV,Car)
```

✓ 0.0s

```
True
```

- 可使用super()來取得父類別的方法

多重繼承

Python 的繼承取決於方法解析順序（method resolution order）。每一個 Python 類別都有一個稱為 mro（）的特殊方法，這個方法會回傳一串類別，Python 會在那些類別中尋找該類別的物件的方法或屬性，此外還有一個類似的屬性，_mro_，它是這些類別的tuple。第一個被找到的勝出。

```
class Animal:
    def says (self):
        return 'I speak!'

class Horse(Animal):
    def says (self) :
        return 'Neigh!'

class Donkey(Animal):
    def says(self):
        return 'Hee-haw!'

class Mule(Donkey,Horse):
    pass

class Hinny(Horse,Donkey):
    pass

print(Mule.mro())
print(Hinny.mro())
mule = Mule()
hinny = Hinny()
print(mule.says())
print(hinny.says())
```

✓ 0.0s

```
[<class '__main__.Mule'>, <class '__main__.Donkey'>, <class '__main__.Horse'>, <class '__main__.Animal'>, <class 'object'>]
[<class '__main__.Hinny'>, <class '__main__.Horse'>, <class '__main__.Donkey'>, <class '__main__.Animal'>, <class 'object'>]
Hee-haw!
Neigh!
```

當我們尋求 mule 的方法或屬性時，Python 會按照順序查看下列的東西：

1.物件本身（型態為 Mule）
2.物件的類別（Mule）
3.該類別的第一個父類別（Donkey）
4.該類別的第二個父類別（Horse）
5.祖父類別（Animal）

Mixin (輔助類別)

```
class PrettyMixin:
    def dump(self):
        import pprint
        pprint.pprint(vars(self))

class Dog(PrettyMixin):
    pass

puff = Dog()
puff.name = 'Puff'
puff.legs = 4
puff.color = 'yellow'
puff.age = 2

puff.dump()
```

✓ 0.0s

```
{'age': 2, 'color': 'yellow', 'legs': 4, 'name': 'Puff'}
```

- 輔助類別沒有任何方法與其他的父類別一樣，可避免方法解析時產生混亂
- 通常用於記錄等“邊緣”工作。

self

```
a_mule = Mule()
print(a_mule.says())

print(Mule.says(a_mule))
```

✓ 0.0s

```
Hee-haw!
Hee-haw!
```

- Python使用self引數來尋找正確的物件屬性與方法

- 呼叫class method時python做的事：
  1. 查看物件a_mule的類別（Mule）
  2. 用self參數將物件a_mule傳給Mule類別的says()方法。

屬性存取

```
class Duck:
    def __init__(self, input_name):
        self.name = input_name

fowl = Duck('Daffy')
print(fowl.name)
fowl.name = 'Ducky'
print(fowl.name)
```

✓ 0.0s

```
Daffy
Ducky
```

```
class Duck:
    def __init__(self, input_name):
        self.hidden_name = input_name

    def get_name(self):
        print('inside the getter')
        return self.hidden_name

    def set_name(self, input_name):
        print('inside the setter')
        self.hidden_name = input_name

don = Duck('Donald')
print(don.get_name())
don.set_name('Donna')
print(don.get_name())
```

✓ 0.0s

```
inside the getter
Donald
inside the setter
inside the getter
Donna
```

property

```
class Duck:
    def __init__(self, input_name):
        self.hidden_name = input_name

    def get_name(self):
        print('inside the getter')
        return self.hidden_name

    def set_name(self, input_name):
        print('inside the setter')
        self.hidden_name = input_name
    name = property(get_name, set_name)

don = Duck('Donald')
print(don.get_name())
don.set_name('Donna')
print(don.get_name())

print(don.name)
don.name = 'Duckie'
print(don.name)
```

✓ 0.0s

```
inside the getter
Donald
inside the setter
inside the getter
Donna
inside the getter
Donna
inside the setter
inside the getter
Duckie
```

```
class Duck:
    def __init__(self, input_name):
        self.hidden_name = input_name
    @property
    def name(self):
        print('inside the getter')
        return self.hidden_name
    @name.setter
    def name(self, input_name):
        print('inside the setter')
        self.hidden_name = input_name


fowl = Duck('Huff')
print(fowl.name)
fowl.name = 'Puff'
print(fowl.name)
```

✓ 0.0s

```
inside the getter
Huff
inside the setter
inside the getter
Puff
```

property

```python
class Circle:
    def __init__(self, radius):
        self.radius = radius

    @property
    def diameter(self):
        return self.radius * 2

c = Circle(4)
print(c.diameter)

c.radius = 16
print(c.diameter)

#如果你沒有為屬性指定setter property，你就不能直接從外面設定它
c.diameter = 20
```

⊗ 0.2s

```
8
32


---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[23], line 16
     13 print(c.diameter)
     15 #如果你沒有為屬性指定setter property，你就不能直接從外面設定它
---> 16 c.diameter = 20

AttributeError: can't set attribute
```

修飾名稱來保護隱私

```
class Duck:
    def __init__(self, input_name):
        self.__name = input_name

    @property
    def name(self):
        print('inside the getter')
        return self.__name
    @name.setter
    def name(self, input_name):
        print('inside the setter')
        self.__name = input_name

fowl = Duck('Huff')
print(fowl.name)
fowl.name = 'Puff'
print(fowl.name)
print(fowl.__name)#無法直接存取
```

⊗ 0.0s

```
inside the getter
Huff
inside the setter
inside the getter
Puff


---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[25], line 19
     16 fowl.name = 'Puff'
     17 print(fowl.name)
---> 19 print(fowl.__name)

AttributeError: 'Duck' object has no attribute '__name'
```

- Python會偷偷把以__（雙底線）開頭的變數名稱修飾成別的名稱。

```
print(fowl._Duck__name)
```

✓ 0.0s

```
Puff
```

類別與物件屬性

- 你可以將變數（屬性）指派給類別，他們會被該類別的物件繼承，但如果你改變物件的屬性的值，它不會影響類別屬性。

```
class Fruit:
    color = 'red'

grapes = Fruit()

print(Fruit.color)
print(grapes.color)
```

✓ 0.0s

```
red
red
```

```
grapes.color = 'purple'
print(Fruit.color)
print(grapes.color)
```

✓ 0.0s

```
red
purple
```

```
new_fruit = Fruit()
print(new_fruit.color)
```

✓ 0.0s

```
green
```

```
Fruit.color = 'green'
print(Fruit.color)
print(grapes.color)
```

✓ 0.0s

```
green
purple
```

方法（函數）的型態

- 實例方法：如果它的前面沒有decorator，它是實例方法，它的第一個引數應該是self，用來引用個別物件本身。

- 如果它的前面有@classmethod裝飾器，它是類別方法，它的第一個引數應該是cls(或任何名稱，只要不是保留字class即可)，引用類別本身

- 如果它的前面有@staticmethod裝飾器，他是靜態方法，它的第一個引數不是物件或類別。

實例方法

- 當類別定義裡面的方法第一個引數是self時，它就是實例方法。（前面講的都是這種方法）

- 當你呼叫方法時python會將該物件傳給這個方法

類別方法

- 類別方法會影響整個類別。你對類別做的任何改變都會影響它的所有物件。

- 在類別定義裡，@classmethod裝飾器代表他後面的函式是個類別方法。

- 類別方法的第一個參數是類別本身（cls）

類別方法

```
class Car:
    count = 0
    def __init__(self, wheels_number=4, car_doors=4, passengers=4):
        self.wheels_number = wheels_number
        self.car_doors = car_doors
        self.passengers = passengers
        Car.count += 1

    def exclaim(self):
        print('I am a car.')

    @classmethod
    def kids(cls):
        print('Car class has',cls.count,'objects.') #cls.count也可使用Car.count

honda = Car()
toyota = Car()
bmw = Car()
Car.kids()
```

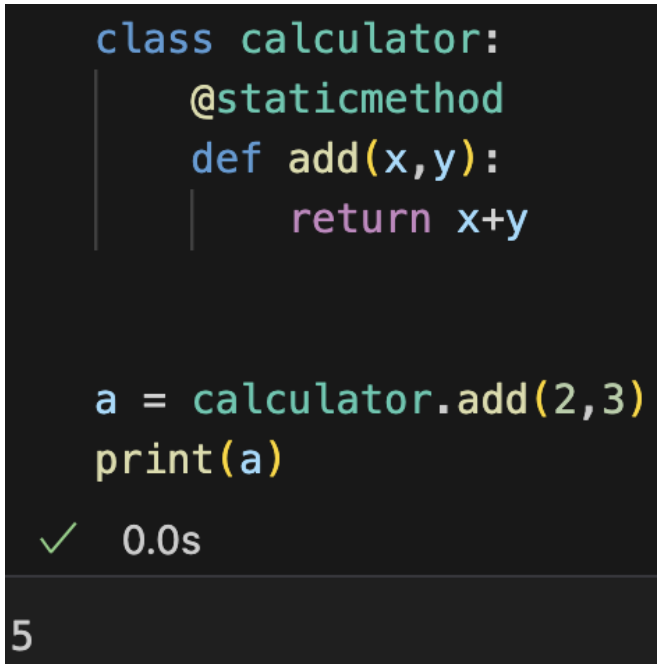
✓ 0.0s

```
Car class has 3 objects.
```

靜態方法

- 在類別定義中前面有@staticmethod裝飾器的為靜態方法

- 它既不影響類別，也不影響物件。 它只是為了被歸類，不四處漂流而放在類別裡。



多型

```
class Quote:
    def __init__(self, person, words):
        self.person = person
        self.words = words
    def who(self):
        return self.person
    def says(self):
        return self.words + '.'

class QuestionQuote(Quote):
    def says(self):
        return self.words + '???'

class ExclamationQuote(Quote):
    def says(self):
        return self.words + '!!!!'

hunter = Quote('Elmer Fudd', "I'm hunting wabbits")
print(hunter.who(), 'says', hunter.says())

hunted1 = QuestionQuote('Bugs Bunny', "What's up, doc?")
print(hunted1.who(), 'says', hunted1.says())

hunted2 = ExclamationQuote('Daffy Duck', "It's rabbit season")
print(hunted2.who(), 'says', hunted2.says())
```

✓ 0.0s

```
Elmer Fudd says I'm hunting wabbits.
Bugs Bunny says What's up, doc????
Daffy Duck says It's rabbit season!!!!
```

```
class BabblingRabbit:
    def who(self):
        return 'rabbit'
    def says(self):
        return 'Babble'

rabbit = BabblingRabbit()

def who_says(obj):
    print(obj.who(), 'says', obj.says())

who_says(hunter)
who_says(hunted1)
who_says(hunted2)
who_says(rabbit)
```

✓ 0.0s

```
Elmer Fudd says I'm hunting wabbits.
Bugs Bunny says What's up, doc????
Daffy Duck says It's rabbit season!!!!
rabbit says Babble
```

魔術方法（特殊方法）

- 6+2 >>> 8, ‘who let’ + ‘the dogs out!’ >>>‘who let the dogs out!’
  （python怎麼知道數字相加是把值用加法而字串相加是接起來？）
- 開頭與結尾都是____（雙底線）

```
class Word:
    def __init__(self, text):
        self.text = text

    def equals(self, word2):
        return self.text.lower() == word2.text.lower()

first = Word('ha')
second = Word('Ha')
third = Word('eh')
print(first.equals(second))
print(first.equals(third))
```

✓ 0.0s

```
True
False
```

```
class Word:
    def __init__(self, text):
        self.text = text

    def __eq__(self, word2):
        return self.text.lower() == word2.text.lower()

first = Word('ha')
second = Word('Ha')
third = Word('eh')
print(first == second)
print(first == third)
```

✓ 0.0s

```
True
False
```

类别	方法名
字符串/字节序列表示形式	__repr__, __str__, __format__, __bytes__
数值转换	__abs__, __bool__, __complex__, __int__, __float__, __hash__, __index__
集合模拟	__len__, __getitem__, __setitem__, __delitem__, __contains__
迭代枚举	__iter__, __reversed__, __next__
可调用模拟	__call__
上下文管理	__enter__, __exit__
实例创建和销毁	__new__, __init__, __del__
属性管理	__getattr__, __setattr__, getattribute__, __setattribute__, __delattr__, __dir__
属性描述符	__get__, __set__, __delete__
跟类相关的服务	__prepare__, __instancecheck__, __subclasscheck__

类别	方法名和对应的运算符
一元运算符	__neg__ -, __pos__ +, __abs__ abs()
众多比较运算符	__lt__ <, __le__ <=, __eq__ =, __ne__ !=, __gt__ >, __ge__ >=
算术运算符	__add__ +, __sub__ -, __mul__ *, __truediv__ /, __floordiv //, __mod__ %, __divmod__ divmod(), __pow__ **或pow(), __round__ round()
反向算术运算符	__radd__, __rsub__, __rmul__, __rtruediv__, __rfloordiv__, __rmod__, __rdivmod__, __rpow__
增量赋值算术运算符	__iadd__, __isub__, __imul__, __itruediv__, __ifloordiv__, __imod__, __ipow__
位运算符	__invert__ ~, __lshift__ <<, __rshift__ >>, __and__ &, __or__ |, __xor__ ^
反向位运算符	__rlshift__, __rrshift__, __rand__, __rxor__, __ror__
增量赋值位运算符	__ilshift__, __irshift__, __iand__, __ixor__, __ior__

```
class Word:
    def __init__(self, text):
        self.text = text

first = Word('ha')
print(first)
first
```

✓ 0.0s

```
<__main__.Word object at 0x11400bf70>

<__main__.Word at 0x11400bf70>
```

```
class Word:
    def __init__(self, text):
        self.text = text

    def __str__(self):
        return self.text

    def __repr__(self):
        return 'Word("' + self.text + '")'

first = Word('ha')
print(first)
first
```

✓ 0.0s

```
ha

Word("ha")
```

使用dictionary 來給參數

```
class Duck:
    def __init__(self, name ,bill, tail):
        self.name = name
        self.bill = bill
        self.tail = tail

d = {'name':'Donald', 'bill':'large', 'tail':'short'}

donald = Duck(**d)
print(donald.name)
print(donald.bill)
print(donald.tail)
```

✓ 0.0s

```
Donald
large
short
```

HW6

1. 定義三個類別：Bear、Rabbit、Octothorpe。在每個類別中定義一個方法：eats()。 讓它回傳 ‘berries’ (Bear)、 ‘clover’ (Rabbit)與’campers’（Octothorpe）。用各個類別建立一個物件並印出它吃什麼東西。

2. 定義這些類別：Laser、Claw 與 SmartPhone，讓它們只有一個 does()方法，讓這個方法回傳 “disintegrate”（Laser）、”Crush”（Chaw）與”ring”（Smant Phone）。再定義 Robot 類別，讓它擁有上述類別的一個實例（物件）。為 Robot 定義 does()方法來印出它的元件做什麼事情。

3.

a. 製作一個稱為 Element 的類別，加入實例屬性 name、symbol 與 number。用值'Hydrogen'、'H'與1建立一個這種類別的物件。

b. 用這些鍵與值製作一個字典：‘name’：‘Hydrogen’、‘symbol’：‘H’、‘number’: 1 。再用 Element 類別與這個字典建立一個名為hydrogen 的物件。

c. 為 Element 類別定義一個名為 dump（）的方法，讓它印出物件屬性的值（name、symbol、number）。用這個新定義建立 hydrogen 物件，並使用 dump（）來印出它的屬性。

d. 呼叫 print（hydrogen）。在 Element 的定義中，將方法 dump的名稱改為 __str__，建立一個新的hydrogen 物件，並再次呼叫 print（hydrogen）。

e.修改 Element，讓 name、symbol 與 number 變成私用的。為每一個屬性定義 getter property，並回傳它的值。

4.（代數：一元二次方程式）請為一元二次方程式 $ax^2 + bx + c = 0$設計一個名為 QuadraticEquation 的類別。此類別包括：

- 代表三個係數的私有資料項目a、b、c。
- 為引數a、b、c所提供建構方法。
- 為a、b、c提供三個 get方法。
- 一個名為 getDiscriminant（），會回傳判別式（也就是$b^2 - 4ac$）的方法。
- 兩個分別名為 getRoot1（）與 getRoot2（），使用下列公式回傳方程式兩個根

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{與} r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

- 這些方法只有在判別式非負數時才有用。如果判別式為負數，我們便讓這些方法回傳0。

請撰寫一測試程式，提示使用者輸入a、b、c等值，接著根據判別式顯示結果。如果判別式為正數，便顯示兩個根。如果判別式為0，則顯示一個根。否則，則顯示“The equation has no roots”。

- 5.（馬表）請設計一個名為 Stop Watch 的類別，此類別包括：
  - 私有資料項目 startTime 與 endTime，以及其get方法。
  - 以目前時間初始化 startTime的建構方法。
  - 名為 start()的方法，用來將 startTime 重新設定為目前時間。
  - 名為 stop() 的方法，用來將 endTime 設定為目前時間。
  - 名為 getElapsedTime()的方法，回傳碼表所經過以毫秒為單位的時間。
  - 進行類別實作。請撰寫一測試程式，計算從1加到1,000,000 的執行時間。