Python_10

類別與物件導向

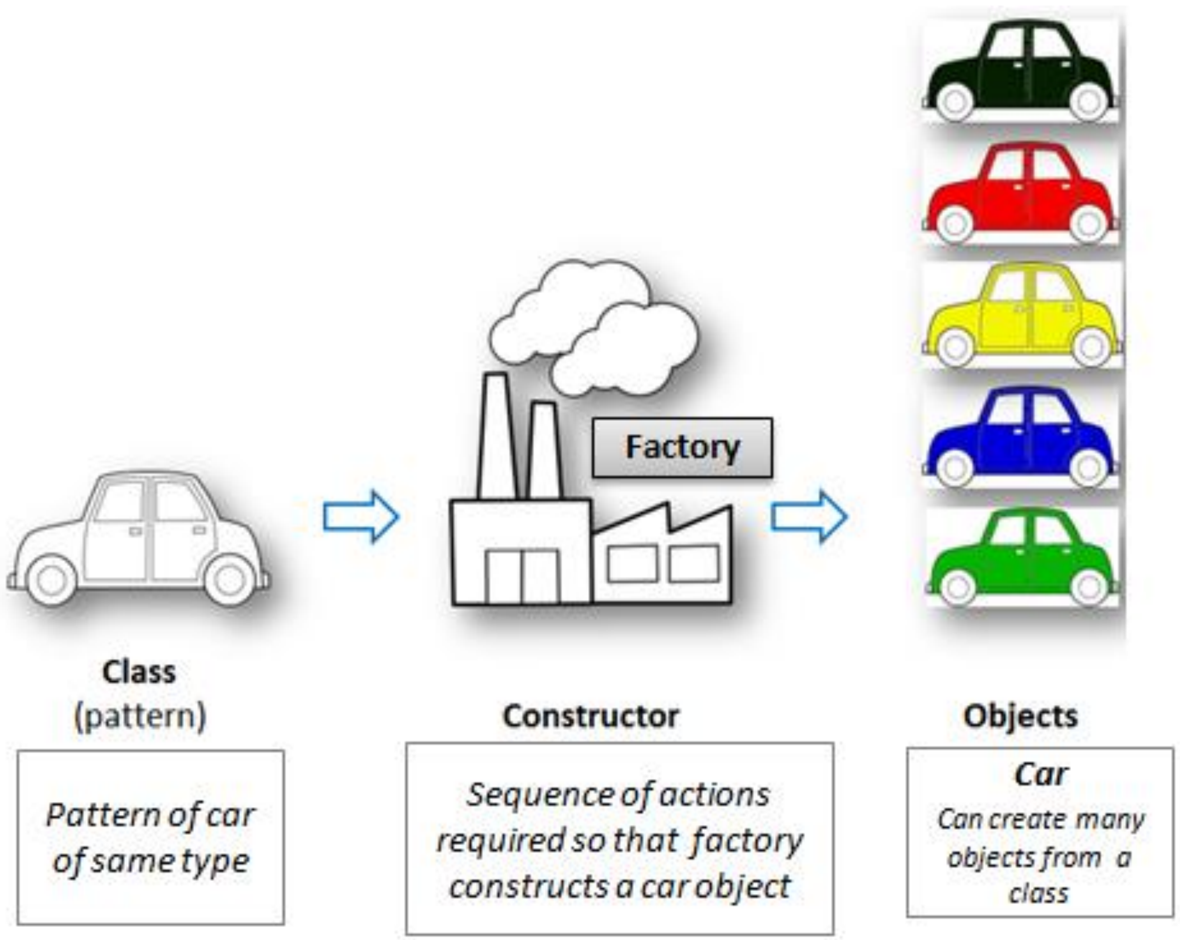
物件導向程式設計(OOP)

- 物件導向程式設計(Object-Oriented Programming)
  - 以真實世界的物體運作狀態來設計程式
  - 以物件作為程式運作的基本單元

- 物件(Object)
  - 實際在記憶體中執行運作的實體(Instance)
  - 動態的，根據執行情況有狀態的變化
  - 有固定的行為與設計架構

- 類別(Class)
  - 類別是物件的定義，有如物件的規格。
  - 物件是根據類別的定義在記憶體建立實體

汽車的類別與物件



Pattern of car of same type

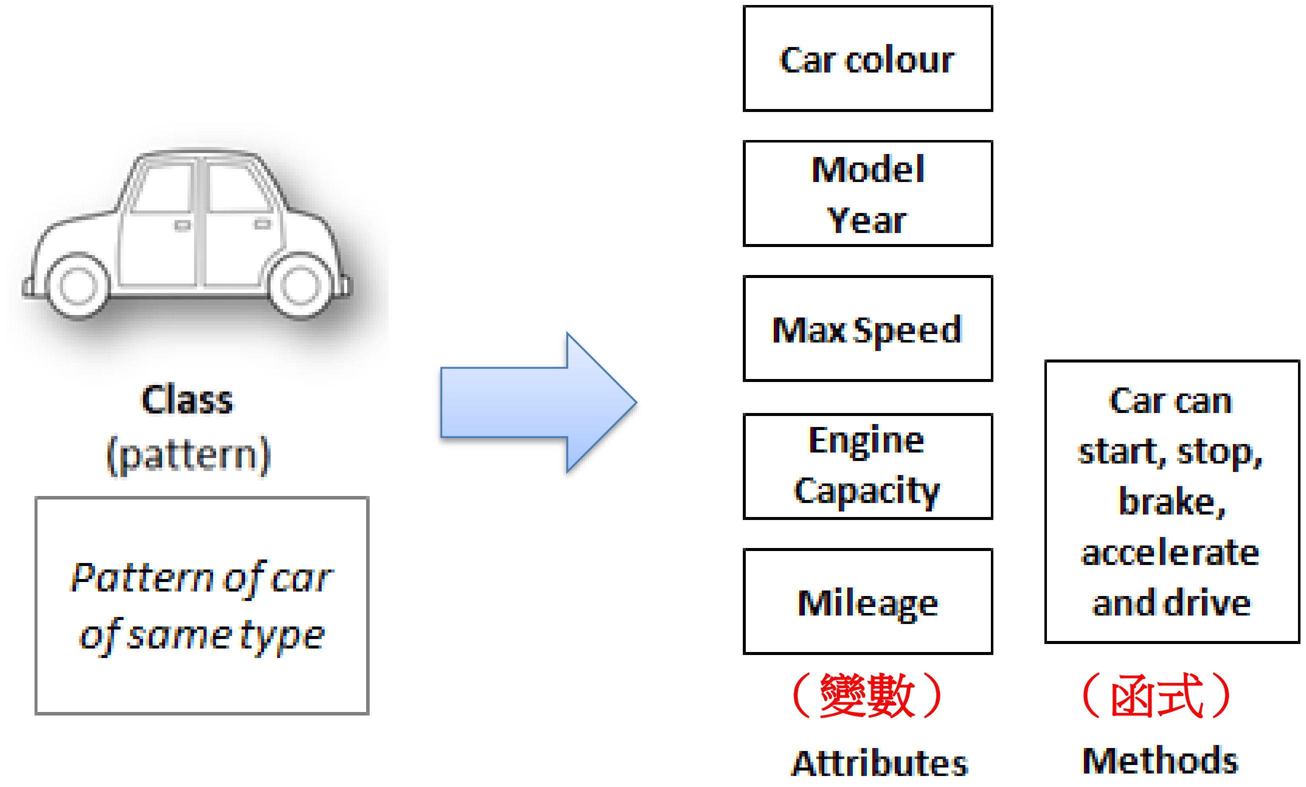
Sequence of actions required so that factory constructs a car object

Can create many objects from a class

類別(Class)

- 類別(Class)
  - 屬性 (attribute、Property)
  - 方法 (method 、Behavior)

類別成員



Python 類別定義

Car類別定義

```
class Car:
    # attributes
    year = 2019  # car model's year
    mpg = 30     # mileage
    speed = 0  # current speed

    # methods
    def accelerate(self):
        self.speed += 20

    def brake(self):
        self.speed -= 10

def main():
    car1 = Car()
    print('speed',car1.speed)
    car1.accelerate()
    car1.accelerate()
    print('speed', car1.speed)
    car1.brake()
    print('speed', car1.speed)

if __name__ == "__main__":
   main()
```

屬性

每個方法的第一個參數規定為self，代表實體自己

產生car1物件實體

car1物件的操作
car1.方法
car1.屬性

Python 類別定義

```
class Car:
    # attributes
    year = 2019   # car model's year
    mpg = 30      # mileage
    speed = 0   # current speed

    # methods
    def accelerate(self):
        self.speed += 20

    def brake(self):
        self.speed -= 10

def main():
    car1 = Car()
    car2 = Car()
    car1.accelerate()
    car1.accelerate()
    print(car1.speed)
    print(car2.speed)

if __name__ == "__main__":
   main()
```

每個物件實體有自己的屬性
每個物件的方法操作是獨立的

產生car1物件實體

產生car2物件實體

輸出

```
car1 spped 40
car2 speed 0
```

__init__( )

```
class Car:
    # attributes
    year = 2019   # car model's year
    mpg = 30      # mileage
    speed = 0   # current speed

    # methods
    def __init__(self):
        self.speed = 10

    def accelerate(self):
        self.speed += 20

    def brake(self):
        self.speed -= 10

def main():
    car1 = Car()
    print('car1 speed',car1.speed)

if __name__ == "__main__":
   main()
```

呼叫__init__()

- Python在建立物件時，會先執行__init__()
- __init__( ) 稱為建構子(constructor)，當類別定義了__init__( )，在建立物件實體時，會自動先呼叫類別定義的__init__( )，來進行物件的初始設定。

輸出

```
car1 speed 10
```

__init__( )

```
class Car:
    # attributes
    year = 2019  # car model's year
    mpg = 30     # mileage
    speed = 0  # current speed

    # methods
    def __init__(self, speed):
        self.speed = speed

    def accelerate(self):
        self.speed += 20

    def brake(self):
        self.speed -= 10

def main():
    car1 = Car(15)
    print('car1 speed',car1.speed)

if __name__ == "__main__":
   main()
```

__init__() 可以增加參數作為物件屬性的初始值

輸出

```
car1 speed 15
```

方法的分類

- 實體方法 (instance method)
  - 必須要帶預設參數 self，當作此實體
  - 需要產生實例來呼叫該方法
- 類別方法 (class method)
  - 必須要帶預設參數 cls，當作此類別
  - 不需要產生實例， 可使用類別呼叫該方法
- 靜態方法（ static method ）
  - 不用帶任何預設參數
  - 實例或類別都可呼叫該方法

方法的分類

```python
class A:
    # attributes
    count = 1
    # methods
    def __init__(self):
        self.count +=1

    @classmethod
    def get_cls_count(cls):
        print(cls.count)

    @staticmethod
    def usage():
        print('USAGE: This is class A')

    def get_self_count(self):
        print(self.count)
```

```python
def main():
    A.get_cls_count()

    a = A()
    a.get_self_count()

    A.get_cls_count()

    a.usage()
    A.usage()

if __name__ == "__main__":
   main()
```

屬性的其他設定方式

```
class Car:
    # attributes
    # methods
    def __init__(self, speed):
        self.speed = speed

    def accelerate(self):
        self.speed += 20

    def brake(self):
        self.speed -= 10

def main():
    Car.color = 'black'
    Car.brand = 'benz'
    car1 = Car(15)
    car1.type = 'SUV'

    print('car1 spped',car1.speed)
    print('car1 color', car1.color)
    print('car1 brand', car1.brand)
    print('car1 type', car1.type)
```

輸出

```
car1 spped 15
car1 color black
car1 brand benz
car1 type SUV
```

物件導向程式的三大特色

- 封裝（Encapsulation）
  - 對外部世界隱藏物件內部的工作細節
- 繼承（Inheritance）
  - 子類別可繼承父類別的屬性和方法
- 多型（Polymorphism）
  - 使用同樣的介面操作不同類的物件

物件導向與程序導向比較

物件導向程式設計	程序導向程式設計
以物件為單位	以函式為單位
模組由物件組成	模組由程序(函式)組成
物件透過封裝可決定開放程度，與那些成員可以公開。	無法決定開放程度，成員只有區域內部與全域。
物件可藉由繼承可擴充	函式宣告後無法擴充
透過多型可使用一個共同介面操作不同物件	無法用一個共同介面操作不同函式

封裝（Encapsulation）

- 封裝的對象為類別成員(屬性、方法)。
- 分為 public(公開)、protected(保護)、private(私有)
- Python類別裡的所有的類別成員(屬性、方法)，預設都是公開的(public)，Python一切都是物件，方法也方法物件(method object)，屬性是各種資料型態物件。
- 名稱加上單底線(_成員名稱)，成員為保護的(protected) 。
- 名稱加上雙底線(__成員名稱)，成員為私有的(Private) 。
- 名稱前後有雙底線(__成員名稱__)，是Python內建成員名稱，例如：__init()__ 。
- 基於封裝，外部無法存取私有成員。
- 基於封裝，匯入模組時,無法匯入該模組受到保護的成員。

封裝 (Encapsulation)

```python
class Book():
    def __init__(self, name ='', author='', price=0):
        self.name = name
        self.author = author
        self.__price = price   #私有

    @property
    def price(self):
        return self.__price

    @price.setter
    def price(self,value):
        self.__price = value


def main():
    book1 = Book('哈利波特神秘的魔法石','JK羅琳',350)
    print('書名：',book1.name)
    print('作者：',book1.author)
    book1.price = 300
    print('售價：', book1.price, '$NTD')

if __name__ == "__main__":
   main()
```

輸出

```
書名： 哈利波特神秘的魔法石
作者： JK羅琳
售價： 300 $NTD
```

Python類別內建方法

- __call__
  - 變成callable的實例，此方法內定義當實例被呼叫時要做的動作
- __del__
  - 當時實例被刪除的時候會被呼叫的方法
- __str__
  - 轉型為str型態時，若有實作此方法，會將其回傳值當作轉型 後的結果

Python 類別內建方法

```python
class Book():
    def __init__(self, name ='',author='',price=0):
        self.name = name
        self.author = author
        self.__price = price

    def __str__(self):
        return 'name,'+self.name+\
               ',author,'+self.author+',' \
                'price,'+str(self.__price)

    def __del__(self):
        print(self.name+'已移除')

    def __call__(self, name='' , author='', price=0):
        if len(name.strip())>0:
            self.name = name
        if len(author.strip()) > 0:
            self.author = author
        if price>=0:
            self.__price = price

def main():
    book1 = Book('哈利波特神秘的魔法石','JK羅琳',350)
    book1('哈利波特–消失的密室','',320)
    print(book1)

if __name__ == "__main__":
   main()
```

輸出

```
name,哈利波特—消失的
密室,author,JK羅
琳,price,320
哈利波特—消失的密室已
移除
```

繼承(Inheritance)

- 定義一個類別的時候，類別成員可以從某個現有的 類別繼承，新的class稱為子類（Subclass），而被繼承的class稱為父類別 (Superclass)

- 繼承可以重複利用父類別的所有功能。

- 子類別只需要新增自己特有的方法， 也可以把父類不適合的方法覆寫(override)

- Python中所有類別都自動繼承object類別

父類別 Car()

```
class Car:
    # attributes
    year = 1990 ;  mpg = 30000  ; speed = 0
    # methods
    def __init__(self, year,mpg, speed):
        self.year =year
        self.mpg = mpg
        self.speed =speed

    def accelerate(self):
        self.speed += 20
        print('speed up:',self.speed)

    def brake(self):
        self.speed -= 10
        print('speed down:', self.speed)
```

子類別 RaceCar()

```
class RaceCar(Car):
    def __init__(self,color='',make='',engine='',year ='',mpg=0, speed=0):
        self.color = color
        self.make = make
        self.engine = engine
        super(RaceCar,self).__init__(year,mpg,speed)

    def turbo_start (self):
        self.speed += 100
        print('Turbo Mode Start Speed:',self.speed )

    def brake(self):
        print('ABS Mode Start')
        super().brake()
```

overide　覆寫父類別的方法

子類別 RaceCar()

```python
def main():
    my_car = RaceCar('White','NISSAN GTR','V6','2007',3000,0)
    print(my_car.color,my_car.make)
    my_car.accelerate()
    my_car.turbo_start()
    my_car.accelerate()
    my_car.brake()

if __name__ == "__main__":
   main()
```

輸出

```
White NISSAN GTR
speed up: 20
Turbo Mode Start Speed: 120
speed up: 140
ABS Mode Start
speed down: 130
```

isinstance()與issubclass()

- isinstance() 可以判斷某一個物件 是否為某一個類別所建構的實體 (instance) ，若真則 回傳 True ，否則回傳 False。
- issubclass() 則可以判斷某一個類別是 否為另一個類別的子類別，同樣的，若真則回 傳 True ，否則回傳 False 。

```
print('my_car is instance of RaceCar:',isinstance(my_car,RaceCar))
print('my_car is instance of Car:',isinstance(my_car, Car))
print('RaceCar is sub class of Car',issubclass(RaceCar,Car))
```

輸出

```
my_car is instance of RaceCar: True
my_car is instance of Car: True
RaceCar is sub class of Car True
```

多重繼承

- 子類別 (subclass) 能夠繼承 (inherit) 多個父類別，使子類別可以有多種特性。
- 當子類別繼承超過一個來源的時候，會以寫在最左邊的父類別優先繼承。
- 當多個父類別如果有相同名稱的屬性 與方法 ，就會以最左 邊的父類別優先。例如 __init__() 、__str__() 等，就會以最左邊的父類別優先。

多重繼承

```
class Base(object):
    def __init__(self):
        print('I am Base')

class A(Base):
    def __init__(self):
        print('I am A')
        super().__init__()

class B(Base):
    def __init__(self):
        print('I am B')
        super().__init__()

class C(A, B):
    def __init__(self):
        super().__init__()

def main():
    c = C()

if __name__ == "__main__":
   main()
```

輸出

```
I am A
I am B
I am Base
```

多型(Polymorphism)

- 簡單來說，一個方法由不同的類別各自實現，可透過一個函式去呼叫不同物件的同名方法，但是不同物件表現出多種型式，稱為多型。

```
class Juicer:
    def open(self):
        print('開啟果汁機電源')

    def run(self):
        print('開始打果汁')


class Washer:
    def open(self):
        print('開啟洗衣機電源')

    def run(self):
        print('開始洗衣服')
```

```
def auto_run(thing):
    thing.open()
    thing.run()

def main():
    machine = Juicer()
    auto_run(machine)

    machine = Washer()
    auto_run(machine)

if __name__ == "__main__":
   main()
```

運算子多載

```python
class Vector3D:
    def __init__(self,x,y,z):
        self.x = x
        self.y = y
        self.z = z

    def __add__(self, other):
        v = Vector3D(self.x + other.x,self.y + other.y,self.z + other.z)
        return v

    def __sub__(self, other):
        v = Vector3D(self.x - other.x, self.y - other.y, self.z - other.z)
        return v

    def __str__(self):
        s = '('+str(self.x)+','+str(self.y)+','+str(self.z)+')'
        return s

def main():

    v1 = Vector3D(1,2,3)
    v2 = Vector3D(1,1,0)

    print(v1-v2)
    print(v1+v2)

if __name__ == '__main__':
    main()
```

內建的一些魔術方法

Operator Magic Methods	Description
__add__(self, other)	To get called on add operation using + operator
__sub__(self, other)	To get called on subtraction operation using - operator.
__mul__(self, other)	To get called on multiplication operation using * operator.
__floordiv__(self, other)	To get called on floor division operation using // operator.
__div__(self, other)	To get called on division operation using / operator.
__mod__(self, other)	To get called on modulo operation using % operator.
__pow__(self, other[, modulo])	To get called on calculating the power using ** operator.
__lt__(self, other)	To get called on comparison using < operator.
__le__(self, other)	To get called on comparison using <= operator.
__eq__(self, other)	To get called on comparison using == operator.
__ne__(self, other)	To get called on comparison using != operator.
__ge__(self, other)	To get called on comparison using >= operator.

內建的一些魔術方法

Initialization and Construction	Description
__new__(cls, other)	To get called in an object's instantiation.
__init__(self, other)	To get called by the __new__ method.
__del__(self)	Destructor method.

Attribute Magic Methods	Description
__getattr__(self, name)	Is called when the accessing attribute of a class that does not exist.
__setattr__(self, name, value)	Is called when assigning a value to the attribute of a class.
__delattr__(self, name)	Is called when deleting an attribute of a class.

內建的一些魔術方法

String Magic Methods	Description
__str__(self)	To get called by built-int str() method to return a string representation of a type.
__repr__(self)	To get called by built-int repr() method to return a machine readable representation of a type.
__unicode__(self)	To get called by built-int unicode() method to return an unicode string of a type.
__format__(self, formatstr)	To get called by built-int string.format() method to return a new style of string.
__hash__(self)	To get called by built-int hash() method to return an integer.
__nonzero__(self)	To get called by built-int bool() method to return True or False.
__dir__(self)	To get called by built-int dir() method to return a list of attributes of a class.
__sizeof__(self)	To get called by built-int sys.getsizeof() method to return the size of an object.

單元作業

- 請實作一個類別名稱Box，包含三個屬性：長(L)、寬(W)、高(H)，兩個方法：info()負責回傳箱子的長、寬、高資訊、volume()回傳箱子的體積。