Chapter 12
類別的定義與使用

12-1：類別的定義與使用

- Python 是物件導向 (Object Orient Programing) 語言。
  - 所有資料類型都是物件
  - 允許使用者自創資料類型，叫做類別 (class)
- class 的語法如下：
  - class Classname():　#類別名稱第一個字母建議大寫

    statement1

    .....

    statement*n*

```
1   class Banks():
2       """定義銀行類別"""
3       bankName = "中國信託商業銀行"   #定義屬性
4       def motto(self):               #定義方法
5           return "We are Family"
```

- 類別內的變數稱為屬性
- 類別內的函數稱作方法

12-1：類別的定義與使用

- 操作類別的屬性與方法
  - object.類別的屬性
  - object.類別的方法( )

```
1  class Banks():
2      """定義銀行類別"""
3      bankName = "中國信託商業銀行"   #定義屬性
4      def motto(self):               #定義方法
5          return "We are Family"
6
7  userBank = Banks()       #宣告一個類別變數
8  print(f"銀行名稱為： {userBank.bankName}")
9  print(f"銀行座右銘為： {userBank.motto()}")
```

```
銀行名稱為： 中國信託商業銀行
銀行座右銘為： We are Family
```

12-1：類別的定義與使用

- 類別的建構方法
  - 透過初始化方法 (method) 初始化整個類別，稱作建構方法 (constructor)。
  - __init__(): 程式內宣告這個類別的物件時，將自動執行這個方法。

```
1   class Banks():
2       """定義銀行類別"""
3       bankName = "中國信託商業銀行"  #定義屬性
4       def __init__(self, userName, userMoney):
5           self.name = userName
6           self.money = userMoney
7       def getMoney(self):
8           return self.money
9
10  userBank = Banks("Wu", 1000)      #宣告一個類別變數
11  print(f"{userBank.name} 的存款餘額是 {userBank.getMoney()}")
```

```
Wu 的存款餘額是 1000
```

- constructor/method 都至少要有一個參數 (不見得要叫 self)
- self 表示呼叫這個 method 的那個變數

12-1：類別的定義與使用

```python
class Banks():
    """定義銀行類別"""
    bankName = "中國信託商業銀行"  #定義屬性
    def __init__(self, userName, userMoney):
        self.name = userName
        self.money = userMoney
    def saveMoney(self, money):
        self.money += money
        print(f"存進 {money} 元，完成。")

    def withdrawMoney(self, money):
        self.money -= money
        print(f"提 {money} 元，完成。")

    def showMoney(self):
        print(f"{self.name} 目前餘額 {self.money} 元")

userBank = Banks("Wu", 1000)      #宣告一個類別變數
userBank.showMoney()
userBank.saveMoney(300)
userBank.showMoney()
userBank.withdrawMoney(100)
userBank.showMoney()
```

```
Wu 目前餘額 1000 元
存進 300 元，完成。
Wu 目前餘額 1300 元
提 100 元，完成。
Wu 目前餘額 1200 元
```

12-1：類別的定義與使用

```python
class Banks():
    """定義銀行類別"""
    bankName = "中國信託商業銀行"  #定義屬性
    def __init__(self, userName, userMoney):
        self.name = userName
        self.money = userMoney
    def saveMoney(self, money):
        self.money += money
        print(f"存進 {money} 元，完成。")

    def withdrawMoney(self, money):
        self.money -= money
        print(f"提 {money} 元，完成。")

    def showMoney(self):
        print(f"{self.name} 目前餘額 {self.money} 元")

wuBank = Banks("Wu", 1000)      #宣告一個類別變數
linBank = Banks("Lin", 1500)    #宣告一個類別變數
wuBank.saveMoney(300)
linBank.withdrawMoney(100)
wuBank.showMoney()
linBank.showMoney()
```

```
存進 300 元，完成。
提 100 元，完成。
Wu 目前餘額 1300 元
Lin 目前餘額 1400 元
```

12-1：類別的定義與使用

- 屬性初始值的設定
  - 通常屬性初始值得設定會寫在 __init__() 內。

```python
class Banks():
    """定義銀行類別"""
    def __init__(self, userName):
        self.name = userName
        self.money = 0
        self.bankName = "中國信託商業銀行"
    def saveMoney(self, money):
        self.money += money
        print(f"存進 {money} 元，完成。")

    def withdrawMoney(self, money):
        self.money -= money
        print(f"提 {money} 元，完成。")

    def showMoney(self):
        print(f"{self.name} 目前餘額 {self.money} 元")

wuBank = Banks("Wu")       #宣告一個類別變數
print(f"{wuBank.name} 在 {wuBank.bankName} 開了個戶頭")
wuBank.showMoney()
wuBank.saveMoney(300)
wuBank.showMoney()
```

```
Wu 在 中國信託商業銀行 開了個戶頭
Wu 目前餘額 0 元
存進 300 元，完成。
Wu 目前餘額 300 元
```

12-2：類別的訪問權限 – 封裝(encapsulation)

- 公有與私有的屬性與方法
  - 前面介紹到的都是公有的屬性(public attribute)與方法(public method)。
    - 可以從外部直接修改到類別內部的屬性值。可以直接從外部呼叫方法。

```python
class Banks():
    """定義銀行類別"""
    def __init__(self, userName):
        self.name = userName
        self.money = 0
        self.bankName = "中國信託商業銀行"
    def showMoney(self):
        print(f"{self.name} 目前餘額 {self.money} 元")

wuBank = Banks("Wu")       #宣告一個類別變數
print(f"{wuBank.name} 在 {wuBank.bankName} 開了個戶頭")
wuBank.showMoney()
wuBank.money = 1000000
wuBank.showMoney()
```

```
Wu 在 中國信託商業銀行 開了個戶頭
Wu 目前餘額 0 元
Wu 目前餘額 1000000 元
```

    - 造成資料不安全

12-2：類別的訪問權限 – 封裝(encapsulation)

- 公有與私有的屬性與方法
  - 私有屬性 (private attribute)：
    - 確保類別內屬性的安全與不被外部修改。
    - 宣告屬性時，屬性名稱前面加上兩個底線 "__"。

```python
class Banks():
    """定義銀行類別"""
    def __init__(self, userName):
        self.__name = userName
        self.__money = 0
        self.__bankName = "中國信託商業銀行"
    def saveMoney(self, money):
        self.__money += money
        print(f"存進 {money} 元，完成。")

    def withdrawMoney(self, money):
        self.__money -= money
        print(f"提 {money} 元，完成。")

    def showMoney(self):
        print(f"{self.__name} 目前餘額 {self.__money} 元")

wuBank = Banks("Wu")      #宣告一個類別變數
print(f"{wuBank.__name} 在 {wuBank.__bankName} 開了個戶頭")
wuBank.showMoney()
wuBank.__money = 10000
wuBank.showMoney()
```

```
Traceback (most recent call last):
  File "e:\PythonTest\test.py", line 19, in <module>
    print(f"{wuBank.__name} 在 {wuBank.__bankName} 開了個戶頭")
AttributeError: 'Banks' object has no attribute '__name'
```

12-2：類別的訪問權限 – 封裝(encapsulation)

- 公有與私有的屬性與方法
  - 私有屬性 (private attribute)：

```python
class Banks():
    """定義銀行類別"""
    def __init__(self, userName):
        self.__name = userName
        self.__money = 0
        self.__bankName = "中國信託商業銀行"
    def saveMoney(self, money):
        self.__money += money
        print(f"存進 {money} 元，完成。")

    def withdrawMoney(self, money):
        self.__money -= money
        print(f"提 {money} 元，完成。")

    def showMoney(self):
        print(f"{self.__name} 目前餘額 {self.__money} 元")

wuBank = Banks("Wu")      #宣告一個類別變數
wuBank.showMoney()
wuBank.__money = 10000
wuBank.showMoney()
```

```
Wu 目前餘額 0 元
Wu 目前餘額 0 元
```

12-2：類別的訪問權限 – 封裝(encapsulation)

- 公有與私有的屬性與方法
  - 私有屬性 (private attribute)：

```python
class Banks():
    """定義銀行類別"""
    def __init__(self, userName):
        self.__name = userName
        self.__money = 0
        self.__bankName = "中國信託商業銀行"
    def saveMoney(self, money):
        self.__money += money
        print(f"存進 {money} 元，完成。")

    def withdrawMoney(self, money):
        self.__money -= money
        print(f"提 {money} 元，完成。")

    def showMoney(self):
        print(f"{self.__name} 目前餘額 {self.__money} 元")

wuBank = Banks("Wu")      #宣告一個類別變數
wuBank.showMoney()
wuBank.__money = 10000
wuBank.showMoney()
```

```
Wu 目前餘額 0 元
Wu 目前餘額 0 元
```

```python
wuBank = Banks("Wu")      #宣告一個類別變數
wuBank.showMoney()
wuBank._Banks__money = 10000
wuBank.showMoney()
```

```
Wu 目前餘額 0 元
Wu 目前餘額 10000 元
```

物件名稱._類別名稱私有屬性名稱

實質上，私有屬性還是可以被外界調用。

12-2：類別的訪問權限 – 封裝(encapsulation)

- 公有與私有的屬性與方法
  - 私有方法 (private method)：
    - 確保類別內方法的安全與不被外部調用。
    - 宣告方法時， 方法名稱前面加上兩個底線 "__"。

```python
class Banks():
    """定義銀行類別"""
    def __init__(self, userName):
        self.__name = userName
        self.__money = 0
        self.__bankName = "中國信託商業銀行"
        self.__rate = 30.0
        self.__serviceFee = 0.01

    def USD2NTD(self, usd):
        self.result = self.__calRate(usd)
        return self.result

    def __calRate(self, usd):
        return (int)(usd * self.__rate * (1-self.__serviceFee))


wuBank = Banks("Wu")      #宣告一個類別變數
USD = 50
print(f"{USD} 美金可換得 {wuBank.USD2NTD(USD)} 新台幣")
print(f"{USD} 美金可換得 {wuBank._Banks__calRate(USD)} 新台幣")
print(f"{USD} 美金可換得 {wuBank.__calRate(USD)} 新台幣")
```

```
50 美金可換得 1485 新台幣
50 美金可換得 1485 新台幣
Traceback (most recent call last):
  File "e:\PythonTest\test.py", line 22, in <module>
    print(f"{USD} 美金可換得 {wuBank.__calRate(USD)} 新台幣")
AttributeError: 'Banks' object has no attribute '__calRate'
```

12-2：類別的訪問權限 – 封裝(encapsulation)

- 從存取屬性(attribute)值看Python風格property( )

```
1   class Score():
2       def __init__(self, score):
3           self.score = score
4
5   stu = Score(60)
6   print(stu.score)
7   stu.score = 100
8   print(stu.score)
```

```
60
100
```

資料保護

```
1   class Score():
2       def __init__(self, score):
3           self.__score = score
4
5       def getScore(self):
6           print("getScore method")
7           return self.__score
8
9       def setScore(self, score):
10          print("setScore method")
11          self.__score = score
12
13  stu = Score(0)
14  print(stu.getScore())
15  stu.setScore(90)
16  print(stu.getScore())
```

```
getScore method
0
setScore method
getScore method
90
```

12-2：類別的訪問權限 – 封裝(encapsulation)

- 從存取屬性(attribute)值看Python風格
  property( )
  - 使用 getter 跟 setter 存取私有屬性。
  - 新式屬性 = property(getter [, setter [, fdel [,doc]]])
    - getter 是獲取屬性值函數
    - setter 是設定屬性值函數
    - fdel 是刪除屬性值函數
    - doc 是屬性描述

```
getScore method
0
setScore method
getScore method
90
```

```python
class Score():
    def __init__(self, score):
        self.__score = score

    def getScore(self):
        print("getScore method")
        return self.__score

    def setScore(self, score):
        print("setScore method")
        self.__score = score

    sc = property(getScore, setScore)

stu = Score(0)
print(stu.sc)
stu.sc = 90
print(stu.sc)
```

12-2：類別的訪問權限 – 封裝(encapsulation)

- 從存取屬性(attribute)值看Python風格property( )
  - 也可以用 @property 與 @property名稱.setter

```
1   class Score():
2       def __init__(self, score):
3           self.__score = score
4
5       def getScore(self):
6           print("getScore method")
7           return self.__score
8
9       def setScore(self, score):
10          print("setScore method")
11          self.__score = score
12
13      sc = property(getScore, setScore)
14
15  stu = Score(0)
16  print(stu.sc)
17  stu.sc = 90
18  print(stu.sc)
```

```
getScore method
0
setScore method
getScore method
90
```

```
1   class Score():
2       def __init__(self, score):
3           self.__score = score
4       @property
5       def sc(self):
6           print("getScore method")
7           return self.__score
8
9       @sc.setter
10      def sc(self, score):
11          print("setScore method")
12          self.__score = score
13
14  stu = Score(0)
15  print(stu.sc)
16  stu.sc = 90
17  print(stu.sc)
```

12-2：類別的訪問權限 – 封裝(encapsulation)

- 方法與屬性的類型
  - 可將類別的方法/屬性區分為實例方法/屬性與類別方法/屬性。
  - 實例方法/屬性
    - self.屬性 / def 方法 (self)
    - 建立類別物件時，屬於物件的一部分。
    - 使用實需建立此類別物件，後由物件調用。前面介紹的全部都是屬於此類。
  - 類別方法/屬性
    - 方法前面加上 @classmethod ，第一個參數習慣使用 cls。
    - 不需要實體化即可由類別本身調用
    - 類別屬性會隨時被更新。

12-2：類別的訪問權限 – 封裝(encapsulation)

- 方法與屬性的類型
  - 類別方法/屬性

```python
class Counter():
    cnt = 0                    #類別屬性
    count = 0
    def __init__(self):
        Counter.cnt += 1
        self.count +=1

    def showCount(self):
        print(f"Counter = {self.cnt}/{self.count}")

    @classmethod
    def showCnt(cls):   #類別方法
        print("Class Method")
        print(f"Counter = {cls.cnt}")
        print(f"Counter = {Counter.cnt}")

one = Counter()
two = Counter()
three = Counter()
Counter.showCnt()
one.showCount()
two.showCount()
three.showCount()
```

```
Class Method
Counter = 3
Counter = 3
Counter = 3/1
Counter = 3/1
Counter = 3/1
```

12-2：類別的訪問權限 – 封裝(encapsulation)

- 方法與屬性的類型
  - 靜態方法
    - 靜態方法是由@staticmethod開頭，不需原先的self或cls參數

```python
class Counter():
    cnt = 0                     #類別屬性
    def __init__(self):
        Counter.cnt += 1

    @staticmethod
    def demo():   #靜態方法
        print("Counter class with counter.")
        print(f"Counter = {Counter.cnt}")

one = Counter()
two = Counter()
three = Counter()
Counter.demo()
```

```
Counter class with counter.
Counter = 3
```

12-3：類別的繼承

- 基本的類別設計好後，設計不同衍生類別包含不同特性與功能。
  - 例如：基本類別為車子，衍生類別賓士、BMW、保時捷...
  - 這種動作叫繼承
- 繼承類別：
  - 被繼承的類別叫父類別 (parent class)、基底類別 (base class) 或超類別 (super class)。
  - 繼承的類別叫子類別 (child class)或衍生類別 (derived class)。
- 類別繼承最大的優點是許多父類別的公有方法或屬性，在子類別中不用重新設計，可以直接引用。

12-3：類別的繼承

- 基底類別必須寫在衍生類別之前。

```
1  class BaseClassName():                    #先定義基底類別
2      Base Class 的內容。
3
4  class DerivedClassName(BaseClassName):    #再定義衍生類別
5      Derived Class 的內容。
```

12-3：類別的繼承

- 衍生類別繼承基底類別的實例應用

```python
class Father():
    def hometown(self):
        print("我住在台南")

class Son(Father):
    pass

hung = Father()
ivan = Son()
hung.hometown()
ivan.hometown()
```

```
我住在台南
我住在台南
```

```python
class Banks():
    """定義銀行類別"""
    def __init__(self, userName):
        self.name = userName
        self.money = 0
        self.bankName = "中國信託商業銀行"
    def saveMoney(self, money):...

    def withdrawMoney(self, money):...

    def showMoney(self):...

class BanksTainan(Banks):
    pass

wuBank = Banks("Wu")
linBank = BanksTainan("Lin")
print(f"{wuBank.name} 在 {wuBank.bankName} 開了個戶頭")
print(f"{linBank.name} 在 {linBank.bankName} 開了個戶頭")
```

```
Wu 在 中國信託商業銀行 開了個戶頭
Lin 在 中國信託商業銀行 開了個戶頭
```

12-3：類別的繼承

- 取得基底類別的私有屬性
  - 類別定義外無法直接取得類別內的私有屬性。
  - 只能透過 return 的方式，回傳私有屬性內容。

```python
class Father():
    def __init__(self):
        self.__address = "台南市大學路"
    def getAddress(self):
        return self.__address

class Son(Father):
    pass

hung = Father()
ivan = Son()
print(f"Base class: {hung.getAddress()}")
print(f"Derived class: {ivan.getAddress()}")
```

```
Base class: 台南市大學路
Derived class: 台南市大學路
```

```python
class Banks():
    """定義銀行類別"""
    def __init__(self, userName):
        self.__name = userName
        self.__money = 10000
        self.__bankName = "中國信託商業銀行"
    def saveMoney(self, money):...

    def withdrawMoney(self, money):...

    def showMoney(self):...

    def bankName(self):
        return self.__bankName

class BanksTainan(Banks):
    pass

wuBank = Banks("Wu")
linBank = BanksTainan("Lin")
print(f"Wu 在 {wuBank.bankName()} 開了個戶頭")
print(f"Lin 在 {linBank.bankName()} 開了個戶頭")
```

```
Wu 在 中國信託商業銀行 開了個戶頭
Lin 在 中國信託商業銀行 開了個戶頭
```

12-3：類別的繼承

- 衍生類別與基底類別有相同名稱的屬性
  - 衍生類別也可以有自己的初始化 __init__() 方法。
  - 衍生類別也可能有和基底類別重複名稱的屬性與方法。
    - 優先使用衍生類別的屬性/方法。

```python
class Person():
    def __init__(self, name):
        self.name = name

class PersonJob(Person):
    def __init__(self, name):
        self.name = name + " 教授"

curtis = Person("Curtis")
curtisJob = PersonJob("Curtis")
print(f"{curtis.name}")
print(f"{curtisJob.name}")
```

```
Curtis
Curtis 教授
```

```python
class Banks():
    """定義銀行類別"""
    def __init__(self, userName):
        self.name = userName
        self.money = 0
        self.bankName = "中國信託商業銀行"
    def saveMoney(self, money):...

    def withdrawMoney(self, money):...

    def showMoney(self):...

class BanksTainan(Banks):
    def __init__(self, userName):
        self.bankName = "中國信託商業銀行 台南分行"


wuBank = Banks("Wu")
linBank = BanksTainan("Lin")
print(f"Wu 在 {wuBank.bankName} 開了個戶頭")
print(f"Lin 在 {linBank.bankName} 開了個戶頭")
```

```
Wu 在 中國信託商業銀行 開了個戶頭
Lin 在 中國信託商業銀行 台南分行 開了個戶頭
```

12-3：類別的繼承

- 衍生類別與基底類別有相同名稱的方法
  - 物件導向中的多型 (polymorphism)

```python
class Person():
    def __init__(self, name):
        self.name = name
    def job(self):
        return "沒工作"

class PersonJob(Person):
    def __init__(self, name):
        self.name = name + " 教授"
    def job(self):
        return "是教授"

curtis = Person("Curtis")
curtisJob = PersonJob("Curtis")
print(f"{curtis.name} {curtis.job()}")
print(f"{curtisJob.name} {curtisJob.job()}")
```

```
Curtis 沒工作
Curtis 教授 是教授
```

```python
class Banks():
    """定義銀行類別"""
    def __init__(self, userName):
        self.name = userName
        self.money = 0
        self.bankName = "中國信託商業銀行"
    def saveMoney(self, money):…

    def withdrawMoney(self, money):…

    def showMoney(self):…

    def bankTitle(self):
        return self.bankName

class BanksTainan(Banks):
    def __init__(self, userName):
        self.bankNameDerived = "中國信託商業銀行 台南分行"
    def bankTitle(self):
        return self.bankNameDerived

wuBank = Banks("Wu")
linBank = BanksTainan("Lin")
print(f"Wu 在 {wuBank.bankTitle()} 開了個戶頭")
print(f"Lin 在 {linBank.bankTitle()} 開了個戶頭")
```

```
Wu 在 中國信託商業銀行 開了個戶頭
Lin 在 中國信託商業銀行 台南分行 開了個戶頭
```

12-3：類別的繼承

- 衍生類別引用基底類別的方法
  - 使用 super()

```python
class Animal():
    def __init__(self, animalName, animalAge):
        self.name = animalName
        self.age = animalAge

    def run(self):
        print(f"{self.name.title()} is running.")

class Dog(Animal):
    def __init__(self, dogName, dogAge):
        super().__init__("My dog " + dogName, dogAge)

myCat = Animal("Lucy", 6)
print(f"{myCat.name.title()} is {myCat.age} years old.")
myCat.run()

myDog = Dog("Lily", 5)
print(f"{myDog.name.title()} is {myDog.age} years old.")
myDog.run()
```

```
Lucy is 6 years old.
Lucy is running.
My Dog Lily is 5 years old.
My Dog Lily is running.
```

12-3：類別的繼承

- 衍生類別有自己的方法

```python
class Animal():
    def __init__(self, animalName, animalAge):
        self.name = animalName
        self.age = animalAge

    def run(self):
        print(f"{self.name.title()} is running.")

class Dog(Animal):
    def __init__(self, dogName, dogAge):
        super().__init__("My dog " + dogName, dogAge)

    def sleeping(self):
        print(f"{self.name} is sleeping")

myCat = Animal("Lucy", 6)
print(f"{myCat.name.title()} is {myCat.age} years old.")
myCat.run()

myDog = Dog("Lily", 5)
print(f"{myDog.name.title()} is {myDog.age} years old.")
myDog.run()
myDog.sleeping()
```

```
Lucy is 6 years old.
Lucy is running.
My Dog Lily is 5 years old.
My Dog Lily is running.
My dog Lily is sleeping
```

12-3：類別的繼承

- 三代同堂的類別與取得基底類別的屬性super( )
  - 透過不斷呼叫 super( ).__init__( )　　# 將父類別的屬性複製

```python
class Grandfather():
    def __init__(self):
        self.grandfatherMoney = 10000
    def getInfo1(self):
        print("Grandfather's Info.")

class Father(Grandfather):
    def __init__(self):
        super().__init__()
        self.fatherMoney = 8000
    def getInfo2(self):
        print("Father's Info.")
```

```python
class Son(Father):
    def __init__(self):
        super().__init__()
        self.sonMoney = 2000
    def getInfo3(self):
        print("Son's Info.")
    def showMoney(self):
        print(f"\nSon 資產: {self.sonMoney}")
        print(f"Father 資產: {self.fatherMoney}")
        print(f"Grandfather 資產: {self.grandfatherMoney}")

ivan = Son()
ivan.getInfo3()
ivan.getInfo2()
ivan.getInfo1()
ivan.showMoney()
```

```
Son's Info.
Father's Info.
Grandfather's Info.

Son 資產: 2000
Father 資產: 8000
Grandfather 資產: 10000
```

12-3：類別的繼承

- 兄弟類別屬性的取得

```python
class Father():
    def __init__(self):
        super().__init__()
        self.fatherMoney = 8000
    def getInfo1(self):
        print("Father's Info.")

class Son1(Father):
    def __init__(self):
        super().__init__()
        self.sonMoney = 6000
    def getInfo2_1(self):
        print("Son1's Info.")

class Son2(Father):
    def __init__(self):
        super().__init__()
        self.sonMoney = 2000
    def getInfo2_2(self):
        print("Son2's Info.")
    def showMoney(self):
        print(f"\nSon2 資產: {self.sonMoney}")
        print(f"Son1 資產: {Son1().sonMoney}")
        print(f"Father 資產: {self.fatherMoney}")

ivan = Son2()
ivan.showMoney()
```

```
Son2 資產: 2000
Son1 資產: 6000
Father 資產: 8000
```

12-4：多型(polymorphism)

```python
class Animals():
    def __init__(self, animalName):
        self.name = animalName
    def which(self):
        return f"My pet {self.name.title()}"
    def action(self):
        return "sleeping"

class Dogs(Animals):
    def __init__(self, dogName):
        super().__init__(dogName.title())
    def action(self):
        return "running in the street"

class Monkeys():
    def __init__(self, monkeyName):
        self.name =  f"My monkey {monkeyName.title()}"
    def which(self):
        return self.name
    def action(self):
        return "running in the forest"
```

```python
def doing(obj):
    print(f"{obj.which()} is {obj.action()}")

myCat = Animals("kitty")
doing(myCat)

myDog = Dogs("Lily")
doing(myDog)

myMonkey = Monkeys("lucy")
doing(myMonkey)
```

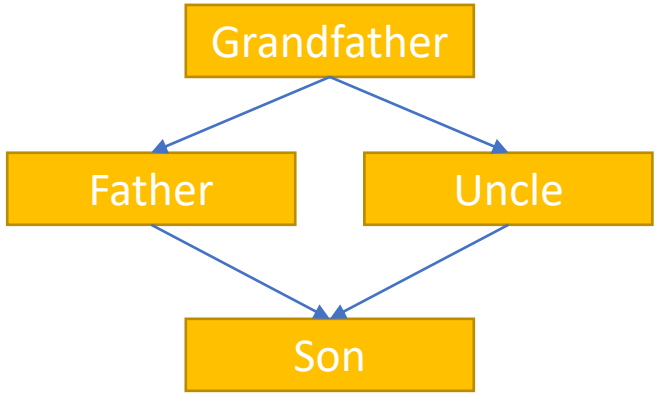
```
My pet Kitty is sleeping
My pet Lily is running in the street
My monkey Lucy is running in the forest
```

12-5：多重繼承

- 一個類別繼承多個類別的情形。
  - 同時繼承多個類別的方法。
- 語法為：

  class 類別名稱(父類別1, 父類別2, … , 父類別n):

  類別內容




```python
class Grandfather():
    def action1(self):
        print("Grandfather")

class Father(Grandfather):
    def action2(self):
        print("Father")

class Uncle(Grandfather):
    def action2(self):
        print("Uncle")

class Son(Father, Uncle):
    def action3(self):
        print("Son")

son = Son()
son.action3()   #Son
son.action2()   #Son -> Father
son.action1()   #Son -> Father -> Uncle -> Grandfather
```

```
Son
Father
Grandfather
```

12-5：多重繼承

```python
class Grandfather():
    def action1(self):
        print("Grandfather")

class Father(Grandfather):
    def action3(self):
        print("Father")

class Uncle(Grandfather):
    def action2(self):
        print("Uncle")

class Son(Father, Uncle):
    def action4(self):
        print("Son")

son = Son()
son.action4()   #Son
son.action3()   #Son -> Father
son.action2()   #Son -> Father -> Uncle
son.action1()   #Son -> Father -> Uncle -> Grandfather
```

```
Son
Father
Uncle
Grandfather
```

12-5：多重繼承

- super( )應用在多重繼承的問題

```
1   class A():
2       def __init__(self):
3           print("class A")
4
5   class B():
6       def __init__(self):
7           print("class B")
8
9   class AB(A, B):
10      def __init__(self):
11          super().__init__()
12          print("class AB")
13
14  x = AB()
```

```
class A
class AB
```

```
1   class A():
2       def __init__(self):
3           super().__init__()
4           print("class A")
5
6   class B():
7       def __init__(self):
8           super().__init__()
9           print("class B")
10
11  class AB(A, B):
12      def __init__(self):
13          super().__init__()
14          print("class AB")
15
16  x = AB()
```

拿掉其中一個會發生什麼事？

```
class B
class A
class AB
```

12-6：type與instance

- 大型程式中，往往會由多人合作設計，有時候想了解某個物件變數的資料型態或是所屬的類別關係，可以使用type與instance。

```python
class Grandfather():
    pass

class Father(Grandfather):
    pass

class Son(Father):
    def fn(self):
        pass

grandfather = Grandfather()
father = Father()
son = Son()

print(f"{type(grandfather) = }")
print(f"{type(father) = }")
print(f"{type(son) = }")
print(f"{type(son.fn) = }")
```

```
type(grandfather) = <class '__main__.Grandfather'>
type(father) = <class '__main__.Father'>
type(son) = <class '__main__.Son'>
type(son.fn) = <class 'method'>
```

12-6：type與instance

- isinstance 函數可以傳回物件是否屬於某一類別。
- 語法為： isinstance(物件, 類別)　# 可傳回True或False

```python
class Grandfather():
    pass

class Father(Grandfather):
    pass

class Son(Father):
    def fn(self):
        pass

grandfather = Grandfather()
father = Father()
son = Son()
```

```python
print(f"son 屬於 Son 類別: {isinstance(son, Son)}")
print(f"son 屬於 Father 類別: {isinstance(son, Father)}")
print(f"son 屬於 Grandfather 類別: {isinstance(son, Grandfather)}")

print(f"father 屬於 Son 類別: {isinstance(father, Son)}")
print(f"father 屬於 Father 類別: {isinstance(father, Father)}")
print(f"father 屬於 Grandfather 類別: {isinstance(father, Grandfather)}")

print(f"grandfather 屬於 Son 類別: {isinstance(grandfather, Son)}")
print(f"grandfather 屬於 Father 類別: {isinstance(grandfather, Father)}")
print(f"grandfather 屬於 Grandfather 類別: {isinstance(grandfather, Grandfather)}")
```

```
son 屬於 Son 類別: True
son 屬於 Father 類別: True
son 屬於 Grandfather 類別: True
father 屬於 Son 類別: False
father 屬於 Father 類別: True
father 屬於 Grandfather 類別: True
grandfather 屬於 Son 類別: False
grandfather 屬於 Father 類別: False
grandfather 屬於 Grandfather 類別: True
```

12-7：特殊屬性

- __xx__ 的字串往往是系統保留的變數或屬性參數。
- 使用 dir() 列出 python 目前環境的變數、屬性、方法。

```
1   print(dir())
```

```
['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__p
ackage__', '__spec__']
```

12-7：特殊屬性

- 文件字串 __doc__：
  - Python 鼓勵程式設計師在設計函數或類別時，儘量為函數或類別增加文件的註解，之後可用 __doc__ 此特殊屬性列出此文件註解。

```
1  def getMax(x, y):
2      """文件字串實例
3      建議 x, y 為整數
4      回傳兩數中較大的數值"""
5      return x if(int(x) > int(y)) else y
6
7  print(getMax(4, 2))
8  print(getMax.__doc__)
```

```
4
文件字串實例
    建議 x, y 為整數
    回傳兩數中較大的數值
```

```
1  class MaxClass():
2      """文件字串實例
3  MaxClass 類別的應用"""
4      def getMax(self, x, y):
5          """文件字串實例
6          建議 x, y 為整數
7          回傳兩數中較大的數值"""
8          return x if(int(x) > int(y)) else y
9  max = MaxClass()
10
11
12 print(max.getMax(4, 2))
13 print(max.__doc__)
14 print(max.getMax.__doc__)
```

```
4
文件字串實例
MaxClass 類別的應用
文件字串實例
        建議 x, y 為整數
        回傳兩數中較大的數值
```

12-7：特殊屬性

- __name__屬性：
  - 在網路上看別人寫的程式，一定會經常在程式末端看到下列敘述：

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

  - 如果上述程式是自己執行，那麼__name__就一定是__main__。

```
1  print(f"module name: {__name__}")
```

```
module name: __main__
```

  - 若是程式被 import 到另外一個程式去時，則 __name__ 是本身的檔案名稱。之後會再詳細介紹。

12-8：類別的特殊方法

- __str__( )方法：
  - 類別的特殊方法，可以回傳容易讀取資訊的字串。

```
1   class Name():
2       def __init__(self, name):
3           self.name = name
4
5   me = Name("Curtis")
6   print(me)
```

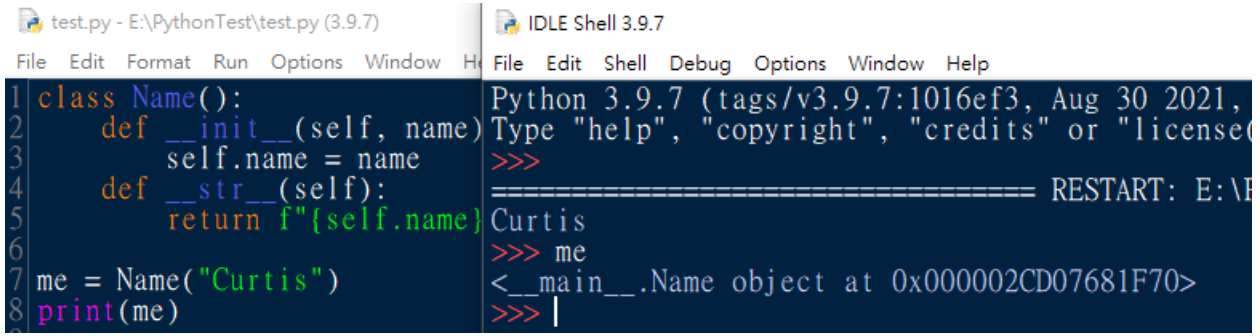
```
<__main__.Name object at 0x000002537F4813D0>
```

```
1   class Name():
2       def __init__(self, name):
3           self.name = name
4       def __str__(self):
5           return f"{self.name}"
6
7   me = Name("Curtis")
8   print(me)
```

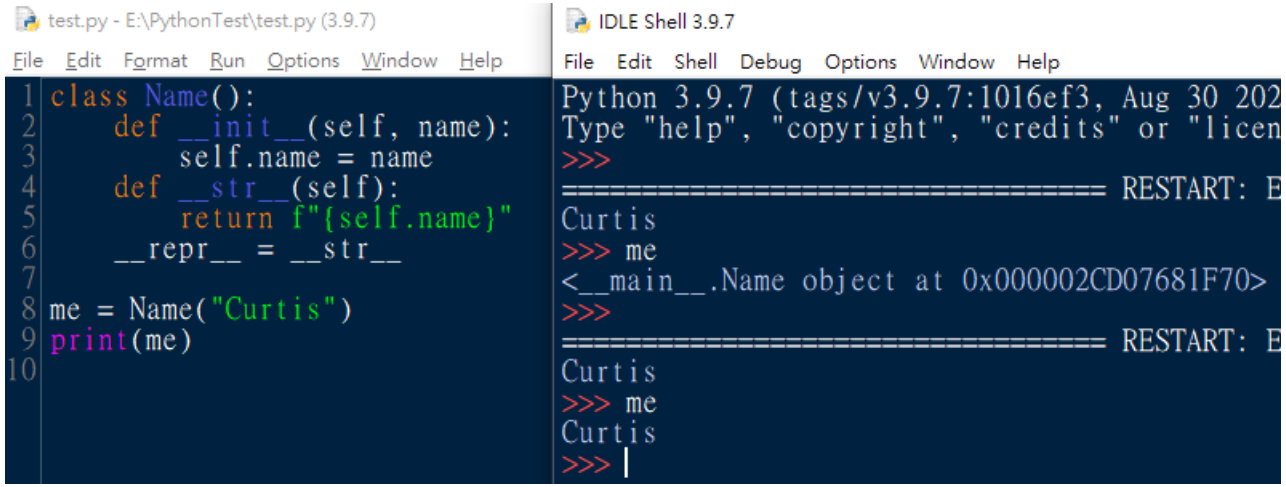
```
Curtis
```

12-8：類別的特殊方法

- __repr__( )方法



  - 若是在 python shell 內讀取類別變數 me，系統呼叫的是 __repr__方法。



12-8：類別的特殊方法

- __iter__( )方法：
  - 將類別設計成可迭代物件類別，類似 list, tuple, …供 for 迴圈使用。
  - 須搭配設計 __next__() 方法，取得下一個值，直到達到結束條件。
  - 使用 raise StopIteration 終止迴圈。

```
1   class Fib():
2       def __init__(self, max):
3           self.max = max
4
5       def __iter__(self):
6           self.n_1 = 0
7           self.n = 1
8           return self
9
10      def __next__(self):
11          fib = self.n_1
12          if fib > self.max:
13              raise StopIteration
14          self.n_1, self.n = self.n, self.n_1 + self.n
15          return fib
16
17  for i in Fib(100):
18      print(i)
```

```
0
1
1
2
3
5
8
13
21
34
55
89
```

12-8：類別的特殊方法

- __eq__( )方法：

```python
class City():
    def __init__(self, name):
        self.name = name
    #def equals(self, city2):
    def __eq__(self, city2):
        return self.name.lower() == city2.name.lower()

one = City("Taipei")
two = City("taipei")
three = City("Tainan")

#print(one.equals(two))
#print(one.equals(three))

print(one == two)
print(one == three)
```

```
True
False
```

12-8：類別的特殊方法

邏輯方法	說明
__eq__(self, other)	self == other # 等於
__ne__(self, other)	self != other # 不等於
__lt__(self, other)	self < other # 小於
__gt__(self, other)	self > other # 大於
__le__(self, other)	self <= other # 小於或等於
__ge__(self, other)	self >= other # 大於或等於

12-8：類別的特殊方法

數學方法	說明
__add__(self, other)	self + other # 加法
__sub__(self, other)	self – other # 減法
__mul__(self, other)	self * other # 乘法
__floordiv__(self, other)	self / /other # 整數除法
__truediv__(self, other)	self / other # 除法
__mod__(self, other)	self % other # 餘數
__pow__(self, other)	self ** other # 次方

12-9：動手練習

- 設計一個基底類別 Geometric。
  - 包含顏色與形狀名稱。
  - 提供設定與圖取顏色的 method。
- 設計衍生類別 Triangle, Rectangle 繼承 Geometric。
  - 利用 __init__ 將形狀名稱傳進基底類別並紀錄起來。
  - Triangle有底跟高的參數。Rectangle 有長跟寬的參數。
  - 提供計算 Triangle 跟 Rectangle 面積的 method。
- 設計一類別，內含一串列，其中元素全部都是數字。
  - 利用__xx__ 的特殊方法，設計串列元素的加減乘除。
  - listClass1 + listClass2 = 串列中元素相加