模組與類別

朱克剛

Module

- 將以下程式碼存成 modop.py

```
def plus(x, y):
    return x + y

def minus(x, y):
    return x - y
```

- 在別的檔案中要引用 plus() 或 minus()
  - *import modop*
    - 使用方式為 modop.plus(5, 3)
  - *from modop import plus, minus*
  - *from modop import **
    - 使用時直接呼叫 plus() 或 minus()

是自己還是被當成Module？

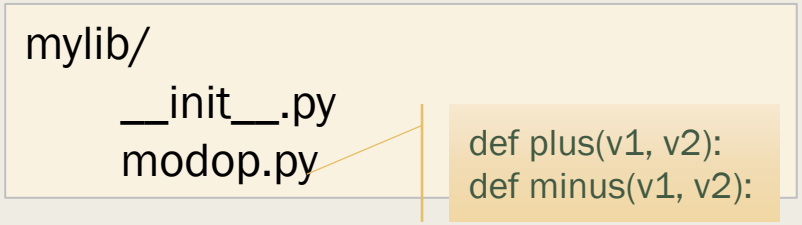
- 被當成 module 的 python檔案也可以自己執行，要如何區分此時是自己執行還是被 import 到別的 python 中當成 module 執行？

```
def echo(string):
    return string

if __name__ == "__main__":
    # 自己執行時會呼叫
    print(echo("test"))
```

Package

- Modules的集合
- 每個目錄中要有__init__.py檔，初始化程序可放此
  - *只要 mylib 第一次被匯入，__init__.py 中的程式碼就會被執行*




```
import mylib.modop
print (mylib.modop.plus(5, 3))

# OR
import mylib.modop as op
print (op.plus(5, 3))

# OR
from mylib.modop import *
print (minus(10, 7))
```

物件導向

Class

- 所有宣告的函數第一個參數為self，如果還有其它參數則依序放在self後面
- 當物件產生時，__init__(self) 會被呼叫，用來初始化變數

```
class Clock:
    def __init__(self, color=None):
        self.color = color

c1 = Clock('white')
c2 = Clock('black')
print('c1 is ' + c1.color)
print('c2 is ' + c2.color)
```

Instance vs Class Variable - I

- Instance（實體等級變數）：實體個別擁有
- Class（類別等級變數）：實體共同擁有

```
class Clock:
    def __init__(self, color=None):
        self.color = color
        Clock.price = 100

c1 = Clock('white')
c2 = Clock('black')
Clock.price = 200
print('c1 price {0}'.format(c1.price))
print('c2 price {0}'.format(c2.price))
```

Instance vs Class Variable - I

- 看看這樣會如何？

```
class Clock:
    def __init__(self, color=None):
        self.color = color
        Clock.price = 100

c1 = Clock('white')
c2 = Clock('black')
Clock.price = 200
c1.price = 50
print('c1 price {0}'.format(c1.price))
print('c2 price {0}'.format(c2.price))
```

加上這一行

存取等級

- 公有（public）
  - *一般變數（方法）名稱*
  - *例如：color*
- 私有（private）
  - *兩個底線開頭的變數（方法）名稱*
  - *例如：__color*
- 繼承（protected）
  - *一個底線開頭的變數（方法）名稱*
  - *例如：_color*

利用私有等級限制屬性值

- 讓Clock的顏色只能設定成白色或黑色

```
class Clock:
    def __init__(self, color=None):
        self.__color = color if color is None else self.setColor(color)

    def setColor(self, color):
        if color != 'white' and color != 'black':
            raise ValueError('wrong color')
        self.__color = color
        return color

    def getColor(self):
        return self.__color

c1 = Clock('black')
c1.__color = 'red'
```

試試這一行

繼承等級

```
class Clock(object):
    def __init__(self, color=None):
        self._color = color if color is None else self.setColor(color)

    def setColor(self, color):
        if color != 'white' and color != 'black':
            raise ValueError('wrong color')
        self._color = color
        return color

    def getColor(self):
        return self._color
```

覆寫

```
class SuperClock(Clock):
    def setColor(self, color):
        if color != 'white' and color != 'black' and color != 'red':
            raise ValueError('wrong color')
        self._color = color
        return color

try:
    c1 = SuperClock()
    c1.setColor('red')
    print(c1.getColor())
except Exception as e:
    print(e)
```