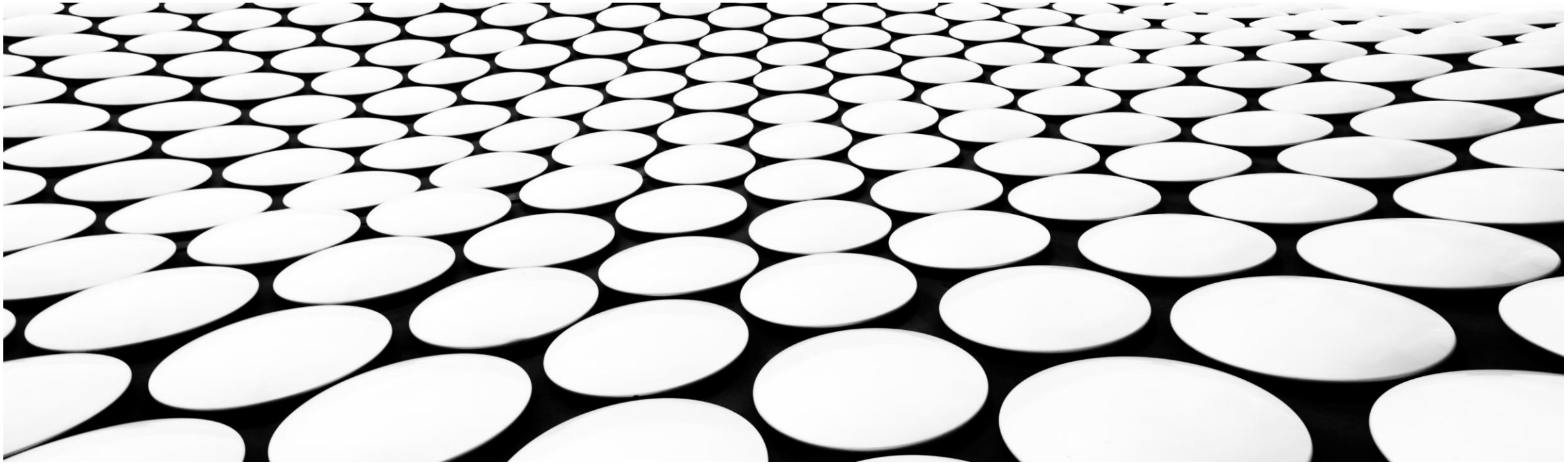
# Python 物件導向概念與實務

MARs教育團隊 – 孫汶琳 交大電機博士候選人



# 目錄

# 為什麼要物件導向程式設計?

- 接續動物園遊戲的例子，如果用傳統程式設計開發的話...

一隻獅子的資料以及會做的動作

```
lion_weight = 100
lion_height = 2.2
lion_food = [ 'chicken', 'pig', 'cow' ]
# ......
def lion1_run():
# ...
def lion1_eat():
```

另一隻獅子的資料以及會做的動作

```
another_lion_weight = 90
another_lion_height = 2.22
another_lion_food = [ 'chicken', 'cow' ]
# ......
def another_lion_run():
# ...
def another_lion_eat():
```

北極熊的資料以及會做的動作

```
polar_bear_weight = 150
polar_bear_height = 2.3
polar_bear_food = [ ‘seal', 'penguin' ]
# ......
def polar_bear_swim():
# ...
def polar_bear_eat():
```

一隻亞洲獅的資料以及會做的動作

```
asian_lion_weight = 85
asian_lion_height = 2.24
asian_lion_food = [ 'chicken', ‘monkey' ]
# ......
def asian_lion_run():
# ...
def asian_lion_eat():
```

# 為什麼要物件導向程式設計?

- 如果用物件導向程式設計來寫的話...

獅子的資料以及會做的動作

```
class Lion(Animal):
# 基本資訊在Animal類別定義了
# 僅需新增或修改和其他動物不同的行為
    def roar():
    # 不須加前綴字詞來分辨不同物種了
```

創建獅子個體

```
# 第一隻獅子
a_lion = Lion()
# 第二隻獅子
another_lion = Lion()
# 第三隻獅子
the_other_lion = Lion()
```

北極熊的資料以及會做的動作

```
class PolarBear(Animal):
# 基本資訊在Animal類別定義了
# 僅需新增或修改和其他動物不同的行為
    def swim():
    # 不須加前綴字詞來分辨不同物種了
```

亞洲獅的資料以及會做的動作

```
class AsianLion(Lion):
# 基本資訊在Animal和Lion類別定義了
# 僅需新增或修改和其他品種獅子不同的行為
    def climb_tree():
    # 不須加前綴字詞來分辨不同物種了
```

# 物件導向程式設計的四大特色

- 封裝性 (Encapsulation)
- 繼承性 (Inheritance)
- 多型性 (Polymorphism)
- 抽象性 (Abstraction)

# 類別 (Class)

- 對物件 (Object) 的一般性描述、抽象定義
- 因不具實體、僅作定義，故無法直接被使用
- 可以使用的定義包含兩種:
  - 資料成員 Data Member
    - 也就是類別 Class 中的變數 Variable
  - 函式成員 Function Member
    - 又稱 方法 Method
    - 也就是類別 Class 中的函式 Function

```
class Animal():
    species = 'animal' # 資料成員
    def intro(self): # 函式成員 (方法)
        print('I am an ', self.species)
```

- Python 的函式成員都預設有第一個參數，用來代表自己，方便使用同一個類別中的資料成員
  - 名字不限定為 self，只是位子一定是第一個，慣用上都命名為self

# 物件 (Object)

- 物件是類別的實體形式，一個類別可以有多個物件
- 透過建構子 Constructor 來將類別實體化為物件
- 舉例來說，假設有一個類別定義為:

```
class Animal():
    species = 'animal' # 資料成員
    def intro(self): # 函式成員 (方法)
        print('I am an ', self.species)
```

- 則可以透過建構子來創建物件 animal1，並透過此物件使用、修改所定義的成員

```
animal1 = Animal()
animal1.species = 'Elephant'
animal1.intro()  # I am an Elephant
```

# 建構子 (Constructor)

- 類別可以透過建構子 Constructor 實體化為物件，是在物件被創造時第一個執行的函式
- Python 中可以透過覆寫(Override) **__init__()** 來修改建構子
- 可以賦予建構子參數，限定所要創建的物件必須要有哪些資料成員
- 舉例來說，如果 Animal 類別要求需要填入品種(species)才能創建物件，則可以定義如下:

```
class Animal():
    def __init__(self, species): # 建構子
        self.species = species  # 資料成員可以在建構子中定義就好

    def intro(self): # 函式成員 (方法)
        print('I am an ', self.species)
```

- 對應的使用方式如下:

```
animal1 = Animal('Elephant')
animal1.intro()  # I am an Elephant
```

# 繼承 (Inheritence) I

- 繼承是物件導向程式設計很重要的一環!
- 缺少了繼承功能的物件導向，就單純只是把參數和函式打包而已
- 繼承允許我們藉由已經存在的類別去創建新的類別，並傳承他所有的成員定義
- 被繼承的類別我們稱作 父類別 (Parent Class)，繼承的類別我們稱作 子類別 (Child Class)
- 若我們要創建一個獅子 Lion 的類別，並繼承 Animal 類別的所有成員定義:

```
class Animal():
    def __init__(self, species): # 建構子
        self.species = species  # 資料成員可以在建構子中定義就好

    def intro(self): # 函式成員 (方法)
        print('I am an ', self.species)

class Lion(Animal): # 設定父類別為 Animal
    pass    # 代表沒有要做任何修改，直接繼承其父類別
```

# 繼承 (Inheritence) II

- 前面的範例中我們可以發現幾個問題:
  1. 都已經新增一個子類別 Lion 了，建立物件時還是要傳入種類 (species)，有點太冗贅了!
  2. 原本的函式成員 intro() 定義不符合需求，要把 'I am an' 換成 'I am a'
  3. 可以用的成員定義有點少，需要新增一些!

```
class Animal():
    def __init__(self, species): # 建構子
        self.species = species  # 資料成員可以在建構子中定義就好

    def intro(self): # 函式成員 (方法)
        print('I am an ', self.species)

class Lion(Animal): # 設定父類別為 Animal
    pass    # 代表沒有要做任何修改，直接繼承其父類別
```

```
lion = Lion('Lion')
lion.intro()  # I am an Lion
```

# 問題一: 修改 __init__()

- 針對問題一我們可以透過修改 __init__()來解決，這裡提供兩個解法:
  - 直接覆蓋 **__init__()**

```
class Lion(Animal): # 設定父類別為 Animal
    def __init__(self) : # 覆蓋父類別中的建構子
        self.species = 'Lion'
```

  - 使用 **super()** 保留父類別中的建構子 __init__() 定義

```
class Lion(Animal): # 設定父類別為 Animal
    def __init__(self) : # 覆蓋父類別中的建構子
        super().__init__('Lion') # 呼叫父類別所定義的建構子
```

```
class Animal():
    def __init__(self, species):
        self.species = species

    def intro(self):
        print('I am an ', self.species)
```

# 問題二: 修改所繼承的函式成員

- 和問題一類似，只不過欲更改的對象從建構子換成函式成員，同學可以參考問題一的解法自己練習看看!

```
class Animal():
    def __init__(self, species):
        self.species = species

    def intro(self):
        print('I am an ', self.species)

class Lion(Animal):
    def __init__(self) :
        self.species = 'Lion'
```

```
class Animal():
    def __init__(self, species):
        self.species = species

    def intro(self):
        print('I am an ', self.species)

class Lion(Animal):
    def __init__(self) :
        super().__init__('Lion')
```

# 問題三: 新增子類別的成員定義

- 既然會想要定義新的類別，必定是有與原本類別不一樣的定義想要新增
- 在處理完繼承自父類別的定義後，就可以直接定義新的成員囉!

```
class Animal():
    def __init__(self, species):
        self.species = species

    def intro(self):
        print('I am an ', self.species)
```

```
class Lion(Animal):
    def __init__(self, mane_color) :
        super().__init__('Lion')
        # 新增資料成員
        self.mane_color = mane_color

    def intro(self) :
        print('I am a ', self.species)

    def roar(self) :  # 新增函式成員 (方法)
        print('I am the king of the world!')
```

```
lion = Lion('brown')
lion.intro()  # I am a Lion
lion.roar()  # I am the king of the world!
```