

Python 的 50+ 練習：資料科學學習手冊

使用類別組織函數與資料

數據交點 | 郭耀仁 <https://linktr.ee/yaojenkuo>

這個章節會登場的保留字、屬性與方法有：

- `class` 保留字。
- `pass` 保留字。
- `__doc__` 屬性。
- `__init__()` 方法。
- `__repr__()` 方法。

複習：Python 提供三種機制供使用者組織程式碼

視應用範疇由小到大依序為：

1. 函數（Function）。
2. 類別（**Class**）。
3. 模組（Module）。

複習： 如何理解程式碼組織機制的層次

- 數行程式碼可以組織為一個函數。
- 數個函數可以組織為一個類別。
- 數個函數或類別可以組織為一個模組。
- 數個模組可以組織為一個功能更多的模組。

與函數的概念相似，類別同樣有四個來源

1. 來自內建。
2. 來自標準模組。
3. 來自第三方模組。
4. 來自使用者的定義。

目前還沒有載入標準/第三方模組類別的需求，之後在「使用模組包裝函數與類別」章節會說明

關於類別

什麼是類別

自行設計資料或者資料結構的機制，能夠將多個函數與資料組織起來使用，定義「類別」也是入門物件導向程式設計的第一步。

兩種不同的程式設計

1. 程序型程式設計（Procedural programming）。
2. 物件導向程式設計（Object-oriented programming, OOP）。

程序型程式設計

以函數為主體的撰寫程式型態稱為「程序型程式設計（Procedural programming）」，把即將要執行的程式碼組織為函數，並依序使用這些函數來完成任務。

```
def function_one():  
    ...  
    return ...  
  
def function_two():  
    ...  
    return ...  
  
function_one()  
function_two()
```

物件導向程式設計

除了程序型程式設計，另外一種在軟體開發中被採用的撰寫程式型態稱為「物件導向程式設計（Object-oriented programming, OOP）」。

```
class class_one:
    def method_one(self):
        ...
        return ...

object_one = class_one()
object_one.method_one()
```

程序型程式設計 vs. 物件導向程式設計

- 線狀 vs. 放射狀。
- 點餐式 vs. 自助餐式。

類別、實例與物件

定義類別是一種讓使用者自行設計資料或資料結構的機制。

- `type()` 內建函數所顯示的 `class` 就是「類別」。
- 物件 (Object) 是類別 (Class) 的實例 (Instance)，因此建立物件的程式碼常被稱為實例化 (Instantiation)。

luke 物件是 str 類別的實例

In [1]:

```
luke = "Luke Skywalker" # instantiation  
type(luke)
```

Out[1]:

```
str
```

skywalkers 物件是 list 類別的實例

In [2]:

```
skywalkers = ["Luke Skywalker", "Anakin Skywalker", "Darth Vader"] # instantiation  
type(skywalkers)
```

Out[2]:

```
list
```

類別之於物件的關係

- 類別如同藍圖一般的存在。
- 物件如同依照藍圖所創造的產品。

為什麼需要定義類別

- 當內建類別或模組所提供的類別無法滿足需求時，我們定義類別。
- 定義「類別」是物件導向程式設計的第一步。

常用的內建類別

- 資料

- `int`
- `float`
- `str`
- `bool`
- `NoneType`

常用的內建類別（續）

- 資料結構

- `list`
- `tuple`
- `dict`
- `set`

資料科學模組主要提供的類別

- 資料結構：
 - `ndarray`
 - `Index`
 - `Series`
 - `DataFrame`

資料科學模組主要提供的類別（續）

- 視覺化類別：
 - `Figure`
 - `AxesSubplot`
- 機器學習估計器類別：
 - 轉換器
 - 預測器

設計類別時可以定義函數與資料

- 在類別程式區塊中定義的函數，實例化後稱為物件的方法（Methods）。
- 在類別程式區塊中定義的資料，實例化後稱為物件的屬性（Attributes）。

使用內建函數 `dir()` 檢視物件的方法與屬性

- 前後有兩個底線 `__` 命名的方法或屬性是所謂的特殊方法、特殊屬性。
- 特殊方法或屬性具有 Python 指定好的功能，例如接下來會用到的 `__doc__`、`__init__()` 方法與 `__repr__()` 方法。

In [3]:

```
# object luke is an instance of str class
luke = "Luke Skywalker"
print(dir(luke))
```

```
['__add__', '__class__', '__contains__', '__delattr__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__getitem__', '__getnewargs__', '__
__gt__', '__hash__', '__init__', '__init_subclass__', '__
__iter__', '__le__', '__len__', '__lt__', '__mod__', '__
__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex
__', '__repr__', '__rmod__', '__rmul__', '__setattr__',
 '__sizeof__', '__str__', '__subclasshook__', 'capitaliz
e', 'casefold', 'center', 'count', 'encode', 'endswit
h', 'expandtabs', 'find', 'format', 'format_map', 'inde
x', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdi
git', 'isidentifier', 'islower', 'isnumeric', 'isprinta
ble', 'isspace', 'istitle', 'isupper', 'join', 'ljust',
'lower', 'lstrip', 'maketrans', 'partition', 'replace',
'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rs
trip', 'split', 'splitlines', 'startswith', 'strip', 's
wapcase', 'title', 'translate', 'upper', 'zfill']
```

In [4]:

```
# object skywalkers is an instance of list class
skywalkers = ["Luke Skywalker", "Anakin Skywalker", "Darth Vader"]
print(dir(skywalkers))
```



```
['__add__', '__class__', '__contains__', '__delattr__',  
 '__delitem__', '__dir__', '__doc__', '__eq__', '__forma  
t__', '__ge__', '__getattribute__', '__getitem__', '__g  
t__', '__hash__', '__iadd__', '__imul__', '__init__',  
 '__init_subclass__', '__iter__', '__le__', '__len__',  
 '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',  
 '__reduce_ex__', '__repr__', '__reversed__', '__rmul_  
__', '__setattr__', '__setitem__', '__sizeof__', '__str_  
__', '__subclasshook__', 'append', 'clear', 'copy', 'cou  
nt', 'extend', 'index', 'insert', 'pop', 'remove', 'rev  
erse', 'sort']
```

函數與方法

使用 `class` 保留字定義類別

類別的命名習慣與物件、函數不同，非採用蛇形命名法，而是採用首字大寫的 *CapWords* 命名法。

來源：<https://www.python.org/dev/peps/pep-0008/#class-names>

pass 保留字的作用

類似在「使用流程控制管理程式區塊的執行」章節介紹過的 `continue`，完成程式區塊但沒有作用。

In [2]:

```
class SimpleCalculator:
    pass

simple_calculator = SimpleCalculator() # instantiation
print(type(simple_calculator))
print(dir(simple_calculator))
```

```
<class '__main__.SimpleCalculator'>
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']
```

關於 `<class`

`'__main__.SimpleCalculator'>` 中的

`__main__`

- Python 組織程式碼的機制有一個層次是「數個函數或類別可以組織為一個模組」。
- 如果沒有指定，自行定義的函數或類別會預設組織到 `__main__` 特殊模組。

In [8]:

```
def power(x, n):  
    """  
    Equivalent to x**n.  
    """  
    return x**n  
  
help(power)  
type(simple_calculator)
```

Help on function power in module __main__:

```
power(x, n)  
    Equivalent to x**n.
```

Out[8]:

```
__main__.SimpleCalculator
```

在類別程式區塊中使用 `def` 保留字定義函數

在類別程式區塊中定義的函數，實例化後稱為物件的方法。

```
class ClassName:
    def method_name(self):
        ...
        return ...

object_name = ClassName()
object_name.method_name()
```

如何理解 `self`

- 使用物件的方法必須要指定物件名稱：`object.method()`
- 可是設計類別的人不會知道物件名稱為何，因為實例化的物件名稱是由使用者所決定的。
- 若是要在定義類別的階段描述方法，就必須先給物件一個代名詞：`self` 作為第一個參數名稱。

In [6]:

```
class SimpleCalculator:
    def add(self, a, b):
        return a + b

simple_calculator = SimpleCalculator()
simple_calculator.add(5, 6)
```

Out[6]:

11

如何理解 `self` (續)

- 事實上，也不一定要用 `self` 作為第一個參數的名稱，可以用偏好的命名。
- 但是 Python 的使用者都習慣 `self`

來源: <https://www.python.org/dev/peps/pep-0008/#function-and-method-arguments>

In [7]:

```
class SimpleCalculator:
    def add(this, a, b): # use this instead of self
        return a + b

simple_calculator = SimpleCalculator()
simple_calculator.add(55, 66)
```

Out [7]:

121

資料與屬性

在類別程式區塊加入 `str` 字面值作為說明

- 我們可以加入 `str` 字面值描述自行定義的類別，與函數的說明（docstring）相似。
- 實例化後使用 `object.__doc__` 屬性閱讀說明。

In [8]:

```
class SimpleCalculator:
    """
    This class creates an instance with default attributes.
    """
    pass

simple_calculator = SimpleCalculator()
print(simple_calculator.__doc__)
```

This class creates an instance with default attributes.

在類別程式區塊中定義函數 `__init__()`

定義資料

- 在類別程式區塊中定義的資料，實例化後稱為物件的屬性。
- `__init__()` 函數顧名思義就是在實例化當下就會起作用的函數。

In [9]:

```
class SimpleCalculator:
    """
    This class creates an instance with 2 custom attributes.
    """
    def __init__(self, a, b):
        self.a = a
        self.b = b

simple_calculator = SimpleCalculator(55, 66)
print(simple_calculator.a)
print(simple_calculator.b)
```

55

66

如何理解 `self` (續)

- 在使用物件的屬性時必須要指定物件名稱: `object.attribute`
- 可是設計類別的人不會知道物件名稱為何，因為實例化的物件名稱是由使用者所決定的。
- 若是要在定義類別的時候描述屬性，就必須先給物件一個代名詞: `self` 作為第一個參數名稱。

以 `self.attribute` 在類別程式區塊中使用屬性

In [10]:

```
class SimpleCalculator:
    """
    This class creates an instance with 2 custom attributes and 1 custom method.
    """
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def add(self):
        return self.a + self.b # retrieve via self.a, self.b

simple_calculator = SimpleCalculator(55, 66) # self proxies object simple_calculator
simple_calculator.add()
```

Out[10]:

121

以 `self.method()` 在類別程式區塊中使用方法

In [11]:

```
class SimpleCalculator:
    """
    This class creates an instance with 2 custom attributes and 2 custom methods.
    """
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def add(self):
        return self.a + self.b # retrieve via self.a, self.b
    def add_then_square(self):
        return (self.add())**2 # use via self.add()

simple_calculator = SimpleCalculator(55, 66)
print(simple_calculator.add_then_square())
```

14641

自行定義的類別實例化後沒有顯示外觀

In [12]:

```
simple_calculator
```

Out[12]:

```
<__main__.SimpleCalculator at 0x7fa8d779c790>
```

在類別程式區塊中定義函數 `__repr__()` 描述顯示外觀

In [13]:

```
class SimpleCalculator:
    """
    This class creates an instance with 2 custom attributes and 2 custom methods.
    """
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def add(self):
        return self.a + self.b # retrieve via self.a, self.b
    def add_then_square(self):
        return (self.add())**2 # use via self.add()
    def __repr__(self):
        return f"a: {self.a}, b: {self.b}" # retrieve via self.a, self.b

simple_calculator = SimpleCalculator(55, 66)
print(simple_calculator)
```

a: 55, b: 66

物件導向程式設計的四個特性

1. 封裝。
2. 繼承。
3. 抽象。
4. 多型。

資料分析師「必須」瞭解的物件導向程式設計特性

- 封裝：將函數與資料整合為物件的方法與屬性。
- 類別程式區塊中定義的函數實例化後稱為物件的「方法」。
- 類別程式區塊中定義的資料實例化後稱為物件的「屬性」。

重點統整

- 類別是自行定義資料類別或者資料結構的機制，能夠將多個函數與資料組織起來使用，也是入門物件導向程式設計的第一步。
- 兩種不同的程式設計：
 - 程序型程式設計（Procedural programming）。
 - 物件導向程式設計（Object-oriented programming, OOP）。
- 類別是一種讓使用者自行設計資料類別或資料結構的機制。

重點統整（續）

- 物件（Object）是類別（Class）的實例（Instance），因此建立物件的程式碼常被稱為實例化（Instantiation）。
- 物件導向程式設計特性：封裝，指的是將函數與資料整合為物件的方法與屬性。
- 類別程式區塊中定義的函數實例化後稱為物件的「方法」。
- 類別程式區塊中定義的資料實例化後稱為物件的「屬性」。