Python 之面向对象

笔记本：	Python		
创建时间：	2019/6/10 13:45	**更新时间：**	2019/6/10 13:49
作者：	kylerlimy@gmail.com		

面向对象与面向过程

面向过程

以函数（能够处理数据的代码块）设计来组织我们的程序。这被称作面向过程（Procedure-oriented）的编程方式。在面向过程编程中，首先要分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候一个一个依次调用。

代表性语言：C 语言

面向对象

在面向对象编程中，它将数据与功能进行组合，并将其包装在被称作“对象”的东西中。也即是把问题进行分解，将一个问题分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描叙某个事物在整个解决问题的步骤中的行为。

代表性语言：c++，c#，java

类与对象

类与对象是面向对象编程的两个主要方面。一个类（Class）能够创建一种新的类型（Type），其中对象（Object）就是类的实例（Instance）。

对象可以使用属于它的普通变量来存储数据。这种从属于对象或类的变量叫作字段（Field）。对象还可以使用属于类的函数来实现某些功能，这种函数叫作类的方法（Method）。这两个术语很重要，它有助于我们区分函数与变量，哪些是独立的，哪些又是属于类或对象的。总之，字段与方法通称类的属性（Attribute）。

实例变量与类变量

实例变量（Instance Variables）：属于某一类的各个实例或对象。
类变量（Class Variables）：从属于某一类本身。

类变量（Class Variable）是共享的（Shared）——它们可以被属于该类的所有实例访问。该类变量只拥有一个副本，当任何一个对象对类变量作出改变时，发生的变动将在其它所有实例中都会得到体现。

对象变量（Object variable）由类的每一个独立的对象或实例所拥有。每个对象都拥有属于它自己的字段的副本，也就是说，它们不会被共享，也不会以任何方式与其它不同实例中的相同名称的字段产生关联。

示例：

```
class Robot:
    """表示有一个带有名字的机器人。"""

    # 一个类变量，用来计数机器人的数量
    population = 0
    def __init__(self, name):
        """初始化数据"""
        self.name = name
        print("(Initializing {})".format(self.name))

        # 当有人被创建时，机器人
        # 将会增加人口数量
        Robot.population += 1

    def die(self):
        """我挂了。"""
        print("{} is being destroyed!".format(self.name))

        Robot.population -= 1

        if Robot.population == 0:
            print("{} was the last one.".format(self.name))
        else:
            print("There are still {:d} robots
working.".format(Robot.population))

    def say_hi(self):
        """来自机器人的诚挚问候

        没问题，你做得到。"""
        print("Greetings, my masters call me {}.".format(self.name))
```

```
    @classmethod
    def how_many(cls):
        """打印出当前的人口数量"""
        print("We have {:d} robots.".format(cls.population))

droid1 = Robot("R2-D2")
droid1.say_hi()
Robot.how_many()

droid2 = Robot("C-3PO")
droid2.say_hi()
Robot.how_many()

print("\nRobots can do some work here.\n")

print("Robots have finished their work. So let's destroy them.")
droid1.die()
droid2.die()

Robot.how_many()
```

输出：

(Initializing R2-D2)
Greetings, my masters call me R2-D2.
We have 1 robots.
(Initializing C-3PO)
Greetings, my masters call me C-3PO.
We have 2 robots.

Robots can do some work here.

Robots have finished their work. So let's destroy them.
R2-D2 is being destroyed!
There are still 1 robots working.
C-3PO is being destroyed!
C-3PO was the last one.
We have 0 robots.

- 通过 Robot.population 而非 self.population 引用 population 类变量。对于 name 对象变量采用 self.name 标记法加以引用，这是这个对象中所具有的方法。
- 同时还要注意当一个对象变量与一个类变量名称相同时，类变量将会被隐藏。
- how_many 实际上是一个属于类而非属于对象的方法。这就意味着我们可以将它定义为一个 classmethod（类方法） 或是一 static method（静态方法）.
- self.name 的值是指定给每个对象的，这体现了对象变量的本质。

- 只能使用 self 来引用同一对象的变量与方法。这被称作属性引用（Attribute Reference）
- 所有的类成员都是公开的。但有一个例外：如果你使用数据成员并在其名字中使用双下划线作为前缀，形成诸如 __privatevar 这样的形式，Python 会使用名称调整（Namemangling）来使其有效地成为一个私有变量.
- 需要遵循这样的约定：任何在类或对象之中使用的变量其命名应以下划线开头，其它所有非此格式的名称都将是公开的，并可以为其它任何类或对象所使用。

self

类方法与普通函数

区别:类方法必须多加一个参数在参数列表开头，这个名字必须添加到参数列表的开头，但是你不用在你调用这个功能时为这个参数赋值，Python 会为它提供。这种特定的变量引用的是对象本身，称为 self。

类

示例：

```
class Person:
    pass # 一个空的代码块
p = Person()
print(p)
```

输出：
<__main__.Person instance at 0x10171f518>

打印的结果告诉我们，我们在 Person 类的 __main__ 模块中拥有了一个实例。

方法

前面讨论过函数与类的方法（Method），唯一的不同在于类方法还拥有一个额外的 self 变量。

示例：

```
class Person:
    def say_hi(self):
        print('Hello, how are you?')

p = Person()
p.say_hi()
```

```
# 前面两行同样可以写作
# Person().say_hi()
```

输出结果：`Hello, how are you?`

注意 say_hi 方法不需要参数，但是依旧在函数定义中拥有 self 变量。

__init__ 方法

__init__ 方法会在类的对象被实例化（Instantiated）时立即运行。该方法可以对任何你想进行操作的目标对象进行初始化（Initialization）操作。

示例：

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

    def say_hi(self):
        print('Hello, my name is', self.name)

p = Person('kyle')
p.say_hi()
# 前面两行同时也能写作
# Person('kyle').say_hi()
```

输出：`Hello, my name is kyle`

在 Person 类下创建新的实例 p 时，采用的方法是先写下类的名称，后跟括在括号中的参数，形如：`p = Person('kyle')` 。

不能显式地调用 __init__ 方法。 这正是这个方法的特殊之处所在。

继承

面向对象编程的一大优点是对代码的重用（Reuse），重用的一种实现方法就是通过继承（Inheritance）机制。继承最好是想象成在类之间实现类型与子类型（Type and Subtype）关系的工具。

示例：

```
class SchoolMember:
    '''代表任何学校里的成员。'''
    def __init__(self, name, age):
```

```
        self.name = name
        self.age = age
        print('(Initialized SchoolMember: {})'.format(self.name))

    def tell(self):
        '''告诉我有关我的细节。'''
        print('Name:"{}" Age:"{}"'.format(self.name, self.age), end=" ")

class Teacher(SchoolMember):
    '''代表一位老师。'''
    def __init__(self, name, age, salary):
        SchoolMember.__init__(self, name, age)
        self.salary = salary
        print('(Initialized Teacher: {})'.format(self.name))

    def tell(self):
        SchoolMember.tell(self)
        print('Salary: "{:d}"'.format(self.salary))

class Student(SchoolMember):
    '''代表一位学生。'''
    def __init__(self, name, age, marks):
        SchoolMember.__init__(self, name, age)
        self.marks = marks
        print('(Initialized Student: {})'.format(self.name))

    def tell(self):
        SchoolMember.tell(self)
        print('Marks: "{:d}"'.format(self.marks))

t = Teacher('Mrs. Shrividya', 40, 30000)
s = Student('Swaroop', 25, 75)

# 打印一行空白行
print()

members = [t, s]
for member in members:
    # 对全体师生工作
    member.tell()
```

要想使用继承，在定义类时需要在类后面跟一个包含基类名称的元组。然后，会注意到基类的 __init__ 方法是通过 self 变量被显式调用的，因此这种方式可以初始化对象的基类部分。

初始化操作

在 Teacher 和 Student 子类中定义了__init__ 方法，Python 不会自动调用基类 (SchoolMember) 的构造函数，你必须自己显式地调用它。

相反，如果没有在一个子类中定义一个 __init__ 方法，Python 将会自动调用基类的构造函数.

当使用 SchoolMember 类的 tell 方法时，可以将 Teacher 或Student 的实例看作 SchoolMember 的实例。

继承中的方法调用

Python 总会从当前的实际类型中开始寻找方法。如果它找不到对应的方法，它就会在该类所属的基本类中依顺序逐个寻找属于基本类的方法，这个基本类是在定义子类时后跟的元组指定的。

多继承

如果继承元组（Inheritance Tuple）中有超过一个类，这种情况就会被称作多重继承（Multiple Inheritance）。

继承的特点
增加或修改了父类 的任何功能，它将自动反映在子类型中。对某一子类型作出的改动并不会影响到其它子类型。

多态性

可以将子类的对象看作是对父类对象的引用。在任何情况下，如果父类型希望，子类型都可以被替换，也就是说，该对象可以被看作父类的实例。