Python 基础

Day 06. Python 面向对象编程

1. 类和实例

OOP

- Object-oriented programming

类

> 抽象的模版

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

实例

> 具体的对象

```
class ClassName(object):
    pass
```

```
class Human(object):
    pass


human = Human()
print(type(human))
```

属性

> 自由绑定属性

```
human.name = 'Tom'

print(human.name)
```

> 强制定义属性

`__init__`

> All classes have a function called `__init__()`, which is always executed when the class is being initiated
>
> Use the `__init__()` function to assign values to object properties, or other operations that are necessary to do when the object is being created

`self`

> The `self` parameter is a reference to the class itself, and is used to access variables that belongs to the class
>
> It does not have to be named `self`, you can call it whatever you like, but it has to be the first parameter of any function in the class

```
class Human(object):

    def __init__(self, name, age):
```

```
        self.name = name
        self.age = age


tom = Human('Tom', 18)

print(tom)

print(tom.name)
print(tom.age)
```

方法

```
class Human(object):

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def get_name(self):
        return self.name


tom = Human('Tom', 18)

print(tom.get_name())
```

2. 封装

公有

私有

`__attribute_name` 私有属性

`_attritube_name` 视为私有属性

`__attribute_name__` 特殊属性 可以直接访问

```
class Human(object):

    def __init__(self, name, age):
        self.name = name
        self.__age = age


tom = Human('Tom', 18)

print(tom.name)
print(tom.__age)
```

- getter & setter

```
class Human(object):

    def __init__(self, name, age):
        self.name = name
        self.__age = age

    def get_age(self):
```

```
        return self.__age

    def set_age(self, age):
        if age > 0:
            self.__age = age
        else:
            raise ValueError('age can not be less than zero!')


tom = Human('Tom', 18)

print(tom.get_age())
tom.set_age(19)
print(tom.get_age())
# tom.set_age(-1)

tom.__age = 20  # this is another attribute of object tom
print(tom.get_age())
print(tom.__age)
```

3. 继承

继承的实现

```
class Human(object):
    pass


class Chinese(Human):
    pass
```

```
```

继承的含义

- 子类继承父类的属性和方法
- 子类可以有新的属性和方法

```
class Human(object):

    def __init__(self, name):
        self.name = name

    def study(self):
        print('human can studying...')


class Chinese(Human):
    pass


zhangsan = Chinese('Zhang San')

zhangsan.study()

print(zhangsan.name)
```

4. 多态

多态的实现

- 子类覆盖 `override` 或重写 `overwrite` 父类的方法

```
class Human(object):

    def study(self):
        print('human can studying...')


class Chinese(Human):

    def study(self):
        print('Chinese can studying...')  # override


zhangsan = Chinese()

zhangsan.study()
```

`isinstance`

```
print(isinstance(zhangsan, Chinese))
print(isinstance(zhangsan, Human))
```

```
def fn_study(human):
    human.study()


fn_study(zhangsan)
```

开闭原则

- Open–closed principle

鸭子类型

- Duck typing

```
class Duck(object):

    def study(self):
        print('Duck can study?')


yellow = Duck()

fn_study(yellow)
```

4. 获取对象信息

`type`

- `type` 判断对象类型

```
print(type(123))
print(type('123'))
print(type(abs))
```

```
print(type(zhangsan))
```

```
import types


def fn():
    pass


print(type(fn) == types.FunctionType)

print(type(abs) == types.BuiltinFunctionType)
```

isinstance

- `isinstance` 判断继承关系

```
print(isinstance(zhangsan, Human))
print(isinstance(zhangsan, Chinese))
print(isinstance(123, int))
print(isinstance(b'abc', bytes))
print(isinstance(123.456, (int, float)))
print(isinstance([], (tuple, list)))
```

dir

- `dir` 获取对象的属性和方法

```
print(dir(zhangsan))
print(zhangsan.__class__)

print(dir('abc'))

print('abc'.__len__())
```

- `hasattr` `getattr` `setattr` 操作对象的状态

```
import math


class Circle(object):
    def __init__(self, x, y, r):
        self.x = x
        self.y = y
        self.r = r

    def area(self):
        return self.r * self.r * math.pi


c = Circle(1, 2, 3)

print(hasattr(c, 'x'))
print(hasattr(c, 'y'))

print(getattr(c, 'x'))
print(getattr(c, 'y'))

print(getattr(c, 'z', 'not found'))
```

```
setattr(c, 'z', 4)

print(getattr(c, 'z'))

print(hasattr(c, 'area'))

print(getattr(c, 'area'))

fn = getattr(c, 'area')

fn()

print(fn())
```

5. 实例属性和类属性

实例属性

- 实例属性属于各个实例所有

类属性

- 类属性属于类所有，所有实例共享一个属性
- **不要** 对实例属性和类属性使用相同的名字

```
class Circle(object):

    pi = 3.1415926
```

```
c = Circle()

print(c.pi)

c.pi = 4

print(c.pi)

print(Circle.pi)

del c.pi

print(c.pi)
```

6. 多重继承

一个类可以同时继承多个父类

- `multiple inheritance`

```
class SubClassName(BaseClassName1, BaseClassName2, ...):
    pass
```

```
class Clock(object):

    def __init__(self, hour, minute, second):
        self.hour = hour
        self.minute = minute
```

```
        self.second = second

    # getters & setters


class Calendar(object):

    def __init__(self, year, month, day):
        self.year = year
        self.month = month
        self.day = day

    # getters & setters


class CalendarClock(Calendar, Clock):

    def __init__(self, year, month, day, hour, minute, second):
        Calendar.__init__(self, year, month, day)
        Clock.__init__(self, hour, minute, second)

    def display(self):
        print('%d-%d-%d %d:%d:%d' % (self.year, self.month, self.day, self.hour, self.minute, self.second))


calendarClock = CalendarClock(2018, 11, 26, 12, 34, 56)
calendarClock.display()
```

7. 定制类

特殊方法 / 魔术方法

`__method_name__` 特殊方法 `special method` / 魔术方法 `magic method`

> 魔术方法有的不需要自定义，有的则通过一些简单的定义实现神奇的功能

Special method names

`__slots__`

- 为实例绑定任意的属性与方法

```
from types import MethodType

class Human(object):
    pass


tom = Human()
tom.name = 'Tom'
print(tom.name)


def set_name(self, name):
    self.name = name


tom.set_name = MethodType(set_name, tom)
tom.set_name('Thomas')

print(tom.name)
```

```
jerry = Human()
jerry.set_name('what?')  # AttributeError
```

- 使用 `__slots__` 限制实例的属性
- 对子类不起作用

```
class Human(object):

    __slots__ = 'age'


tom = Human()

tom.age = 18
print(tom.age)

tom.name = 'Tom'  # AttributeError
print(tom.name)
```

`__str__`

```


class Student(object):

    def __init__(self, name):
        self.name = name

    # def __str__(self):
    #     return self.name
```

```
tom = Student('Tom')
print(tom)
```

~~__iter__~~

~~__getitem__~~

~~__getattr__~~

~~__call__~~

8. 枚举类

9. 元类

10. 作业

1. 定义类，实现 power(x, n) 功能

```
print(Solution().power(2, -3));
print(Solution().power(3, 5));
print(Solution().power(100, 0));
```

2. 定义类，实现字符串逆序

```
print(Solution().reverse_words('hello.py'))
```

3. 定义三角形类，实现求三角形周长和面积的方法，属性为三个边长

```
rectangle = Rectangle(1, 2, 3)
print(rectangle.perimeter())
print(rectangle.area())
```

4. 定义立方体类，属性为长度、宽度、高度，通过方法来计算它的体积
5. 定义一个人类，包含姓名、性别、年龄等信息
    - 所有的变量必须私有。其他类只能通过该类的方法获取和修改
    - 实例化一个人类，试着通过该类的方法修改实例化的人的信息
6. 定义一个学生类，包含三个属性（学号，姓名，成绩）均为私有的
    - 分别给这三个属性定义两个方法，一个设置它的值，另一个获得它的值
    - 测试这些方法
7. 继承人类编写一个学生类，为学生类添加新的属性和方法，并进行测试