LV5

类与单元测试

BBCLOUD 林凡

https://github.com/lvancer/course_python

lin029011@163.com

LV5

大纲

面向对象

类

继承

多态

静态

单元测试

练习

lin029011@163.com

LV5

面向对象

面向对象编程是一种程序设计思想。

我们到目前为止所编写的程序，都是基于面向过程的。

面向过程就是我们在写程序过程中，

将需求拆成多个流程，每个封装一个方法，

逐个调用。

比如**买菜**、**洗菜**、**炒菜**、**装盘**这四个步骤，

我们对于的方法就是buy、wash、cook、dish。

如果需求有变化，如我还要买米、做饭，

用铁锅炒菜，用微波炉做菜等等。

这时候用面向过程就非常麻烦且会越来越乱。

如何给女朋友解释什么是面向对象编程?

lin029011@163.com

LV5

面向对象

面向对象则是从另一个角度来思考，它用对象来取代方法作为程序的基础。

对象是对事物的抽象，如人可以抽象成名字、年龄以及能做的事情，这就是类。

我们在程序设计时，就只要创建这个对象，让它来做它能做的事情。

还是**买菜**、**洗菜**、**炒菜**、**装盘**这个四个步骤，现在我们就可以有超市、厨师两个对象。

超市负责买菜，厨师负责洗菜、炒菜、装盘。

进一步厨师用什么洗菜，用什么做菜怎么做、怎么装盘又是一系列的对象来处理。

超市这个对象提供**买米**、**买菜**、**买肉**等等功能，必要时还可以继续添加。

除了基本的功能外，沃尔玛和永辉卖的东西就有差别，即使同样是卖米也会有差异。

这就是继承，沃尔玛和永辉都是超市的继承，但并不是相同的。

lin029011@163.com

LV5

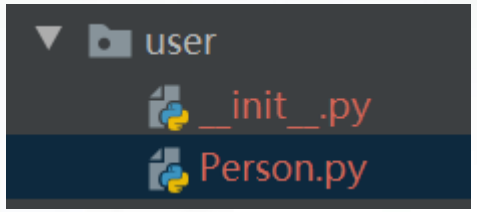
类

类【class】就是对对象的描述。

我们要把人的名字和年龄以及做的事情用python的类写出来。

创建一个类：class。

我们创建一个user包，在包里创建一个Person模块。在模块中编写Person类。




```
class Person:
    def __init__(self):
        pass
```

class后面就是类名Person，其下的代码块就是类的定义。类名使用驼峰写法。

一个文件一个类，大部分情况我们会这样做。Python中并不反对一个文件多个类。

第一个在类中要定义的方法就是构造方法，方法名为__init__。这是类默认存在的方法。

构造方法里的第一个参数是self，代表对象自己。默认类的方法第一个参数都是self。

lin029011@163.com

LV5

类

实例化：将类创建出来进行使用。

我们创建一个main.py来实例化这个类。


```
from user import Person
p1 = Person.Person()
```

```
from user.Person import Person
p1 = Person()
```

上面两种方式是在包下的类实例化方式，与方法的调用时一样的。

对于类，由于一文件一类的原则，我们更希望能简单一点调用到包里的类。如下：

```
from user import Person
p1 = Person()
p2 = Person()
```

创建了两个人

那么我们就要在包的__init__.py文件里做文章了，写入 `from .Person import Person`

然后我们就可以直接从包导入Person类了。

lin029011@163.com

LV5

类

构造方法：用于初始化，每个类被实例化时会默认调用的方法。

```
def __init__(self, name, age):
    self.name = name    # 名字
    self.age = age      # 年龄
```

我们加入Person类有的数据，名字和年龄。

self代表自己，它点出来的变量称为成员变量。

self.name和self.age分别被赋值给构造方法传入的参数name和age，用于初始化。

实例化并获取成员变量：

```
p1 = Person('Green', 12)
print(p1.name, p1.age)
```

使用构造方法进行创建，然后直接用创建好的变量，点出成员变量即可。

成员变量原则上只能在构造方法中声明。

lin029011@163.com

LV5

类

成员方法：构造方法其实是一个特殊的成员方法。

```
def sleep(self, t):
    print('{} sleeps for {} seconds'.format(self.name, t))
```

定义了一个sleep方法，同样的第一个参数是self，代表该方法是成员方法。

在成员方法内部，我们就可以用self来获得成员变量。

调用：同样使用点的方式调用该方法。

```
p2 = Person('Lucy', 14)
p2.sleep(10)
```

这时如果我们需要在sleep前添加一步到床上的步骤，我们可以这样做，

添加一个go2bed成员方法，并在sleep方法中用self调用这个成员方法。

lin029011@163.com

LV5

类

目前完成的Person类。

```
class Person:

    def __init__(self, name, age):
        self.name = name    # 名字
        self.age = age      # 年龄

    def sleep(self, t):
        self.go2bed()
        print('{} sleeps for {} seconds.'.format(self.name, t))

    def go2bed(self):
        print(self.name + ' go to bed.')
```

lin029011@163.com

LV5

类

访问限制：对成员变量和成员方法进行限制，不让使用者可以直接使用。

成员变量作为一个类的数据，不应该可以直接被使用者修改，保证类被正确使用。

变量名用 两个下划线 【__ 】开头的就是被限制访问的私有变量，只可内部访问。

```
def __init__(self, name, age):
    self.__name = name      # 名字
    self.__age = age        # 年龄
```

```
p1 = Person('Green', 12)
print(p1.__name, p1.__age)
```

如果需要获取这两个变量的值，或进行修改，我们定义get，set方法进行。

```
def get_age(self):
    return self.__age

def set_age(self, age):
    self.__age = age
```

```
def get_name(self):
    return self.__name
```

我们希望age可以进行设置，而name没有set方法。

这样成员变量的任何修改都在我们的控制范围内。

lin029011@163.com

LV5

类

成员方法也使用两个下划线来做限制。

__go2bed就是只能内部访问的方法了。

主要目的是：

1、不希望使用者调用到。

2、对用户提供的方法也可以叫做接口。

接口就是使用者可以调用到的所有方法了

这样对于使用者来说也是最好的。

```
class Person:

    def __init__(self, name, age):
        self.__name = name     # 名字
        self.__age = age       # 年龄

    def sleep(self, t):
        self.__go2bed()
        print('{} sleeps for {} seconds.'
              .format(self.__name, t))

    def __go2bed(self):
        print(self.__name + ' go to bed.')

    def get_name(self):
        return self.__name

    def get_age(self):
        return self.__age
```

lin029011@163.com

LV5

继承

继承是在已有类【父类】的基础上，再编写一个子类，其拥有父类的所有内容。

```
from user import Person
class Student(Person):
    pass
```

在类名Student后面用括号抱起来的Person就是父类。

此时这个子类没有任何的代码，但是它已经继承了父类所有的内容，使用方法一样。

```
from user import Student
student1 = Student('Green', 12)
print(student1.get_name())
```

所有方法变量自动继承。

lin029011@163.com

LV5

继承

object类：默认的Python继承类。

没有做继承的类，其实都默认继承了python的object类【对象类】。

```
class Person(object):
```

object可以不写。

lin029011@163.com

LV5

继承

添加成员变量和方法：子类一般拥有比父类更强大的功能。

在Students类中，我们添加一个grade变量来标识年级，并添加一个新方法。

```
class Student(Person):

    def __init__(self, name, age, grade=1):
        Person.__init__(self, name, age)    # 父类构造方法
        self.__grade = grade                # 年级

  def get_grade(self):
        return self.__grade
```

代码中的Person.__init__表示了我们在新的构造方法中先调用了Person父类的构造方法。

这样我们就减少了重复的代码。

lin029011@163.com

LV5

继承

复写成员方法：修改父类的方法内容。

方法的复写必须保证方法名和参数与父类一致。

```
def sleep(self, t):
    print('student ' + self.get_name())
    Person.sleep(self, t)                # 父类方法
```

通过调用父类的方法来扩充该方法。或者直接不调用父类方法全部重写也是可以的。

```
def sleep(self, t):
    print('student {} sleeps for {} seconds.'
          .format(self.__name, t))
```

子类的方法覆盖了父类的方法。复写实现了面向对象中多态的概念。

此时我们发现代码报错了，原因在于__name变量是只能Person类内部访问的成员变量。

只能使用self.get_name()来获得名字，子类要如何调用父类的私有变量？

lin029011@163.com

LV5

继承

继承中的访问限制：

两个下划线使得该成员变量和方法只能在这个类中内部使用。

但在继承中，成员变量和方法除了不能被外部调用外，其子类应该是可以正常使用的。

使用一个下划线【_】开头定义的成员变量和方法就可以实现。

```
def __init__(self, name, age):
    self._name = name     # 名字
    self._age = age       # 年龄

def _go2bed(self):
    print(self._name + ' go to bed.')
```

这样刚才的复写方法中就可以使用了。

```
def sleep(self, t):
    print('student {} sleeps for {} seconds.'.format(self._name, t))
```

LV5

继承

访问限制分类：

public：公开权限，无下划线【self.name】，任何地方都可以访问修改。

protected：保护权限，单下划线【self._name】，只允许自己和其子类访问修改。

private：私有权限，双下划线【self.__name】，只允许自己访问修改。

根据我们的需要，选择对应的权限进行开发。

lin029011@163.com

LV5

多态

判断变量类型：type、isinstance。

```
p1 = Person('Green', 12)
s1 = Student('Green', 12)
print(type(s1))
print(type(p1))
```

输出：

```
<class 'user.Student.Student'>
<class 'user.Person.Person'>
```

type可以获得该变量的类型。

```
print(isinstance(p1, Person))
print(isinstance(p1, Student))
```

输出：

```
True
False
```

isinstance可以直接判断变量是不是某个类型，对于父类变量结果与下面的代码相同：

```
print(type(p1) == Person)
print(type(p1) == Student)
```

但对于子类，情况并不相同。

lin029011@163.com

LV5

多态

子类的类型判断：

```
type(s1) == Person              # False
type(s1) == Student             # True
isinstance(s1, Person)          # True
isinstance(s1, Student)         # True
```

我们发现子类变量在type的相等判断中，并不等于父类。

但在isinstance判断中，子类被判断为与父类一样，这就是多态。

一个类可以是自己，也可以是其父类，逻辑上也是通的，学生也是一个人。

反之则不对，一个人未必是学生。

lin029011@163.com

LV5

多态

多态的用处：

```
def poly_test(person):
    if not isinstance(person, Person):
        raise Exception('数据类型传入错误')
    return person.sleep(10)
```

这个方法里，我们对传入的类型进行了限定。

这样我们就可以直接传入Student、Person或者其他子类都可以。

sleep也是父类就有方法，直接调用不会有问题。

当我们新增一个子类，如Programmer类，我们就不需要对这个方法进行修改，直接使用。

根据对象的真实类型来运行，就是多态的用处。

```
try:
    poly_test(p1)
    poly_test(s1)
except ValueError as e:
    print(e)
```

lin029011@163.com

LV5

静态

除了成员变量和成员方法，类中还可以定义静态属性：类变量和类方法。

静态的意思是不需要实例化就可以使用。

类变量的数据属于所有实例共享一个。

直接创建在类的定义中与方法同级。调用时使用类名.变量名。

这里做了一个统计，

统计同一姓名的人有几个。

每创建一个实例就统计一次。

```
class Programmer(Person):

    programmers = {}     # 静态变量

    def __init__(self, name, age):
        Person.__init__(self, name, age)
        if name not in Programmer.programmers:
            Programmer.programmers[name] = 0
        Programmer.programmers[name] += 1
```

```
from user import Programmer
p = Programmer('Green', 18)
print(Programmer.programmers)
```

LV5

静态

类方法是直接属于类的方法。

定义时，与成员方法不同的是第一个参数不是self而是cls【这个类】。

并需要在方法上添加一个装饰器。

```
@classmethod
def count_by_name(cls, name):
    return cls.programmers[name]
```

以@开头并写在def上面一行的就是装饰器，具体内容以后的高阶语法再介绍。

@classmethod就定义了下面的方法是一个类方法，且没有self参数，因为没有实例化。

第一参数是cls，代表了当前类，可以通过cls调用类变量，相当于Programmer.programmers。

使用时直接通过类名.方法调用，无需实例化。

```
count = Programmer.count_by_name('Green')
```

lin029011@163.com

LV5

静态

静态方法与类方法类似，但更为独立，仅仅表现为放在类里面托管的方法。

定义时，没有self参数，也没有cls参数。下面是上个方法的静态方法版本。

```
@staticmethod
def count_by_name(name):
    return Programmer.programmers[name]
```

@staticmethod就定义了下面的方法是一个静态方法。其他都与类方法一样。

使用上，静态方法一般与类没有什么关系，不会涉及类的操作，是一个比较独立的方法。

继承问题：类变量与子类共享，父类没有该类变量。

类方法同样可以继承和复写，有复写时要调用这个类的类方法。

静态方法也一样。

lin029011@163.com

LV5

单元测试

单元测试是一个测试框架。

我们直接来看一个简单的示例代码。

unittest是要引入的包。

创建一个类继承unittest.TestCase类。

然后其他方法就不需要了。

我们直接开始写测试用例。

以test开头的成员方法就是一个用例。

定义完类后，我们启动该测试unittest.main()。

前面的if语句判断了该文件是否启动文件，也是一种常用的方法。

启动后，单元测试会自动执行所有用例，并给出结果。

```
import unittest          # 单元测试框架
from user import Student     # 要测试的类


class TestStudentMethods(unittest.TestCase):

    def test_student(self):          # 测试用例
    s = Student('Green', 12)
        self.assertEqual(12, s.get_age())
        self.assertEqual('Green', s.get_name())


if __name__ == '__main__':  # 当作为启动文件时
  unittest.main()
```

lin029011@163.com

LV5

单元测试

```
def test_student(self):                # 测试用例
  s = Student('Green', 12)
    self.assertEqual(12, s.get_age())
    self.assertEqual('Green', s.get_name())
```

测试用例：

先进行一些操作代码，就是我们的用例过程，这里创建了一个Student对象。

然后就要使用单元测试类给我们提供的assert系列方法来判断结果是否正确。

assertEqual是最常用的方法之一了，参数1写入期望值，参数2写入要验证的数据。

在运行过程中就会进行对比，如果两者相等则正确，不等则错误。

常用的assert方法还有：

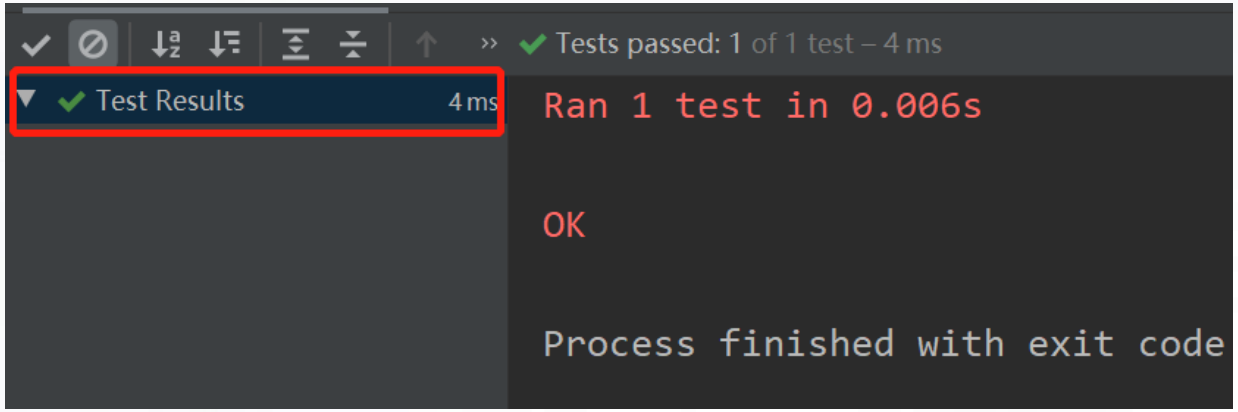
assertNotEqual、assertTrue 、assertFalse、assertIsNone、assertIsNotNone、assertIn等。

一个单元测试可以包含很多测试用例，一次进行测试。

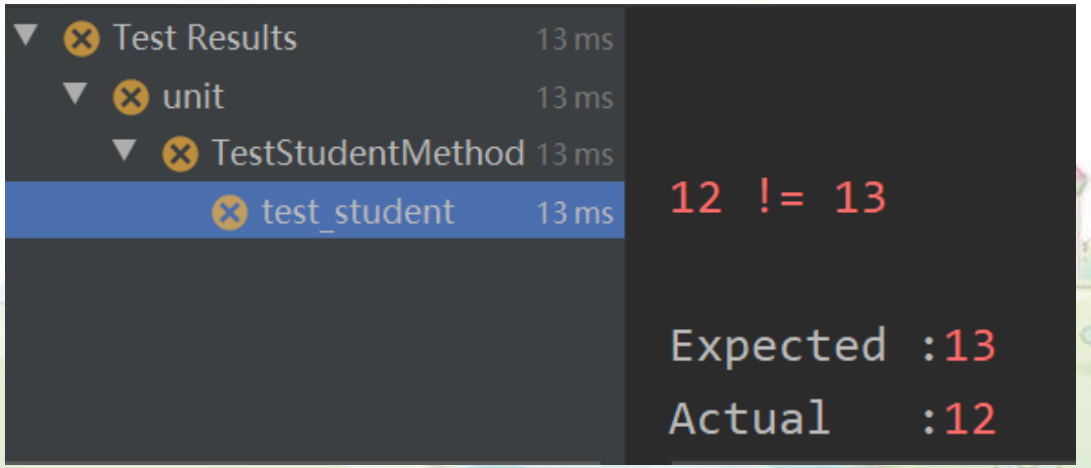
lin029011@163.com

单元测试

测试结果：所有assert都正确的情况下，即测试通过。



有没有通过的用例时，会告诉你哪个用例的哪个assert有问题，与期望值不符。



lin029011@163.com

LV5

练习

1、模拟一个小游戏，编写一个Sprite类【拥有血量和攻击力两个属性，并进行攻击】。

然后继承出Monster类和Hero类，让两者每回合互相攻击一次，直到有一方死亡。

每次攻击扣除对方的血量是攻击力加减N的随机值。

2、编写一个进行接口请求的单元测试。

lin029011@163.com