Python小练习：class使用中的一些技巧

作者：凯鲁嘎吉 - 博客园 http://www.cnblogs.com/kailugaji/

本文主要介绍Python代码中@property、pass、@abc.abstractmethod、raise NotImplementedError、super().__init__、*args与**kwargs等所起的作用。

1. python_class.py

```python
# -*- coding: utf-8 -*-
# Author：凯鲁嘎吉 Coral Gajic
# https://www.cnblogs.com/kailugaji/
# python class使用中的一些技巧
'''
部分代码参考于：
    https://blog.csdn.net/weixin_42681866/article/details/83376484
    https://blog.csdn.net/qq_40666620/article/details/105026716
'''
import abc # 抽象类
from typing import Dict, List, Any, Union


class Person(object):
    def __init__(self, name, age):
        self.name = name
        self.__age = 18 # 双下划线的age定义为私有变量
        # 这里的成员属性__age需要与成员方法age()区分开

    # python @property的使用
    # property装饰器的作用：将方法变成属性调用
    @property
    def age(self):
        return self.__age
    # 方法加入@property后，这个方法相当于一个属性，该属性可以让用户进行使用，而且用户有没办法随意修改

    @age.setter
    def age(self, age):
        if age < 18:
            print('年龄必须大于18岁')
            return
        self.__age = age
```

```
33          return self.__age
34
35      # pass的使用
36      # pass的作用：代码的占位符
37      def Test_pass(self):
38          pass
39      # 现在这个方法我还没想好里面要写啥东西，但又不能空着不写，所以用pass先占个位
40
41      # @abc.abstractmethod的使用
42      # @abc.abstractmethod的作用：实现抽象方法
43      # 基类不能实例化，子类实现了该抽象方法才能被实例化
44      @abc.abstractmethod
45      def Test_abstract(self, weight):
46          pass
47
48      # raise NotImplementedError的使用
49      # raise NotImplementedError的作用：如果子类不重写这个方法，就会抛出异常
50      def Test_Error(self):
51          raise NotImplementedError("没有重写Test_Error方法，因此抛出NotImplementedError异常！")
52
53  # 父类与子类的使用
54  # raise NotImplementedError的使用
55  # 当子类没有重写父类中的成员函数，然后子类对象调用该函数时，会提示这个错误！
56  # pass的使用
57  class PersonTwo(Person): # 继承于父类Person
58      # 对父类的Test_Error方法进行了重写
59      def Test_Error(self):
60          print('你好！\n'
61                '我重写了父类Person中的Test_Error方法！\n'
62                '因此没有抛出NotImplementedError异常！')
63
64      # 对父类的Test_pass方法进行了重写
65      def Test_pass(self):
66          print('我现在想好了，准备写这段话！')
67
68      # 对父类的Test_abstract方法进行了重写
69      def Test_abstract(self, weight):
70          print('体重：', weight)
71
72  # 没有重写Test_Error方法
73  class PersonThree(Person):
74      def Test_Error2(self):
75          print('Sorry！')
76
77  # super().__init__的使用
78  # super().__init__()作用：用来调用super class内的__init__构造方法
79  class PersonFour(Person):
```

```
80      def __init__(self, name, age, height):
81          super().__init__(name, age)
82          self.height = height
83
84  # *args与**kwargs的使用
85  # *args的作用：arguments，位置参数，在当传入的参数个数未知，且不需要知道参数名称时使用
86  # **kwargs的作用：keyword arguments，关键字参数，当传入的参数个数未知，但需要知道参数的名称时使用
87  def roster(para1, *args, **kwargs):
88      print('必选参数：', para1)
89      print('位置参数：', args)
90      print('位置参数求和结果：', sum(args))
91      print("关键字参数数据类型：", type(kwargs))
92      print('************花名册************')
93      count = 0
94      for key, value in kwargs.items():
95          count = count +1
96          print('第%d个学生的姓名：' % count, key)
97          print('第%d个学生的信息：' % count)
98          for i, j in value.items():
99              print(i, '：', j)
100
101 print('*args与**kwargs的用法：')
102 tuple = (1, 2, 3, 4)
103 dict1 = {"小明":{"年龄": 15, "身高": 165, "体重": 55},
104          "小红":{"年龄": 18, "身高": 170, "体重": 60},
105          "小王":{"年龄": 25, "身高": 180, "体重": 66},
106          "小张":{"年龄": 30, "身高": 175, "体重": 58}}
107 roster('花名册', *tuple, **dict1)
108
109 print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
110 print('@property的用法：')
111 # 创建实例
112 xm = Person('小明', 20)
113 xm.name='小红'
114 xm.__age=5
115 print('姓名：', xm.name) # 按最新的来
116 print('----------------------')
117 print('情况1：')
118 print('年龄：%d岁' % xm.age) # 默认18岁，xm.__age改了，不关xm.age的事
119 # 使用@property装饰器来创建只读属性
120 # @property装饰器会将方法转换为相同名称的只读属性
121 # 可以与所定义的属性配合使用，这样可以防止属性被修改
122 print('----------------------')
123 print('情况2：')
124 xm.age = 10
125 print('年龄：%d岁' % xm.age)
126 print('----------------------')
```

```python
print('情况3：')
xm.age=20
# 使用@property装饰器来修饰方法，使方法可以像属性一样访问
# 加了@property后，可以用调用属性的形式来调用方法,后面不需要加()
print('年龄：%d岁' % xm.age)

# pass的使用
print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
print('pass的用法：')
print('情况1：父类')
c = Person('小李', 28)
c.Test_pass()
print('----------------------')
print('情况2：PersonTwo子类')
d = PersonTwo('小李', 28)
d.Test_pass()
print('----------------------')
print('情况3：PersonThree子类')
e = PersonThree('小王', 35)
e.Test_pass()

# @abc.abstractmethod的使用
print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
print('@abc.abstractmethod的用法：')
g = PersonTwo('小李', 28)
g.Test_abstract(55) # 实例化，体重55KG

# super().__init__的使用
print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
print('super().__init__的用法：')
f = PersonFour('小张', 26, 170)
print('姓名：', f.name)
print('年龄：', f.age) # 不是26，还是默认的18
print('身高：', f.height)

# raise NotImplementedError的使用
print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
print('raise NotImplementedError的用法：')
print('情况1：')
a = PersonTwo('小李', 28)
a.Test_Error()
print('----------------------')
print('情况2：')
b = PersonThree('小王', 35)
b.Test_Error2()
b.Test_Error() # 删除这行不报错
```

2. 结果

```
D:\ProgramData\Anaconda3\python.exe "D:/Python code/exercise/python_class.py"
*args与**kwargs的用法：
必选参数： 花名册
位置参数： (1, 2, 3, 4)
位置参数求和结果： 10
关键字参数数据类型： <class 'dict'>
**********花名册**********
第1个学生的姓名： 小明
第1个学生的信息：
年龄 ： 15
身高 ： 165
体重 ： 55
第2个学生的姓名： 小红
第2个学生的信息：
年龄 ： 18
身高 ： 170
体重 ： 60
第3个学生的姓名： 小王
第3个学生的信息：
年龄 ： 25
身高 ： 180
体重 ： 66
第4个学生的姓名： 小张
第4个学生的信息：
年龄 ： 30
身高 ： 175
体重 ： 58
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

@property的用法：
姓名： 小红
----------------------
情况1：
年龄：18岁
----------------------
情况2：
年龄必须大于18岁
年龄：18岁
----------------------
情况3：
年龄：20岁
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

pass的用法：
情况1：父类
----------------------
```

```
情况2：PersonTwo子类
我现在想好了，准备写这段话！
--------------------
情况3：PersonThree子类
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

@abc.abstractmethod的用法：
体重：  55
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

super().__init__的用法：
姓名：  小张
年龄：  18
身高：  170
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

raise NotImplementedError的用法：
情况1：
你好！
我重写了父类Person中的Test_Error方法！
因此没有抛出NotImplementedError异常！
----------------------
情况2：
Sorry!

Traceback (most recent call last):
  File "D:/Python code/exercise/python_class.py", line 172, in <module>
    b.Test_Error() # 删除这行不报错
  File "D:/Python code/exercise/python_class.py", line 51, in Test_Error
    raise NotImplementedError("没有重写Test_Error方法，因此抛出NotImplementedError异常！")
NotImplementedError: 没有重写Test_Error方法，因此抛出NotImplementedError异常！

Process finished with exit code 1
```

注意：代码中最后一行是为了说明raise NotImplementedError的作用，把最后一行 b.Test_Error() 删掉后，程序就不会报错了。

补充：下面这个程序用于判断输入的类名称是否是子类，是，则返回该子类。

```python
# -*- coding: utf-8 -*-
# Author：凯鲁嘎吉 Coral Gajic
# https://www.cnblogs.com/kailugaji/
# 判断输入的类名称是否是子类，是，则返回该子类
class BaseClass:
    def __init__(self):
        self.value = 0

    def set_value(self, value):
        self.value = value

```

```
    def get_value(self):
        return self.value

class SubClass(BaseClass):
    def __init__(self):
        super().__init__()
        self.value = 1

    def set_value(self, value):
        super().set_value(value)
        self.value += 1

    def get_value(self):
        return self.value

# 这段代码定义了一个名为get_subclass的函数，它接受两个参数：base_class表示基类，class_name表示要查找的子类名称。
# 该函数使用递归的方式来查找基类的所有子类，并返回其中名称与子类名称相同的子类。如果找不到名称相同的子类，则返回None。
# 在函数体内，首先使用__subclasses__方法获取基类的所有子类，然后使用get_subclass函数递归地查找每个子类。
# 如果找到了名称与子类名称相同的子类，则返回该子类。如果没有找到，则返回None。
# 请注意，这段代码假设基类中的所有子类都是可枚举的，即它们都有一个唯一的名称。
# 如果基类中的子类是可变的，则需要使用其他方法来查找子类。
def get_subclass(base_class, class_name):
    for c in base_class.__subclasses__():
        if c.__name__ == class_name:
            return c
    for c in base_class.__subclasses__():
        temp_c = get_subclass(c, class_name)
        if temp_c is not None:
            return temp_c
    return None

subclass = SubClass()
result = get_subclass(BaseClass, "SubClass")
if result is not None:
    print(f"SubClass {subclass.value} is a subclass of BaseClass")
else:
    print(f"SubClass {subclass.value} is not a subclass of BaseClass")
```

结果：

```
SubClass 1 is a subclass of BaseClass
```

可以看到，SubClass属于基类BaseClass中的子类，因此调用子类，返回值1。