**实验4模块与面向对象编程**

**姓名： 徐超信 学号：1935010102 班级：计科1902 成绩：**

**1. 实验内容与目标**

本次实验旨在了解Python的模块、包、类等概念，熟悉常用模块的用法，学会自定义模块，理解Python的程序组织方式，具备面向对象的编程能力。需掌握的相关知识点主要包括：

**1) 模块及其使用：**为最高级别的程序组织单元，将程序代码和数据封装起来以便重用；具有一定功能的程序块，分别用不同的文件名(“\*.py”文件)存放，使代码更容易维护和管理；**导入模块**的方式：**import** module\_name 或 **import** module\_name **as** \* 或 **from** module\_name **import** \*…。

常见模块包括：random、time、os、sys、math、string、re等

另外，Python脚本和模块都是一个以.py扩展名文件，如何区分？很多脚本的最后都有一段类似下面的判断语句，限制只能以脚本方式运行，不作为模块使用：

if \_\_name\_\_ == '\_\_main\_\_':

执行语句…

**2) 包：**Python文件的一种组织方式，就是将几个功能相近的模块可组成一个Python包，存放到一个目录结构中，通过输入包的路径来调用包中模块的相应对象，如变量，函数与类等等。要创建一个包，就要建一个与包名同名的目录，接着在该目录下创建“\_\_init\_\_. py”文件，该文件是包的初始化文件，可以为空，也可定义相关代码；接着将相关的模块放入这个目录下，这样就创建一个包，这个目录下的模块就属于这个包。

常见的包(package)有：numpy、scipy、matplotlib、pandas、sklearn等

**3) 类与面向对象编程：**类是一种描述相同属性与方法的对象的集合。类和对象是面向对象编程的两个主要方面，类创建一个数据结构新类型，而对象是这个类的实例。一个简单例子如下：

|  |  |
| --- | --- |
| **Bird类的定义** | **类的实例化，即对象的创建** |
| class Bird():  def \_\_init\_\_(self):  self.hungry = True  def eat(self):  if self.hungry:  print('Aaaah...')  self.hungry = False  else:  print('No,thanks!') | >>> b1=Bird()  >>> b1.eat ()  Aaaah...  >>> b1.eat ()  No,thanks!  >>> b1.eat ()  No,thanks!  >>> b1.hungry  False |

请完成下列实验练习题，报告书写要求同第1次实验作业。注意，请在本报告中将自己的姓名、学号、班级书写正确，文件名按以下格式写出具体的班级、学号、姓名，样例如下：

“软件1902-1912190001-张三-实验4\_模块与面向对象编程.docx”

最后，请在规定时间内完成本次实验，并将报告的word文档(非pdf版本)以附件形式提交至学习通平台。

**2. 实验练习题**

**答题要求**：将正确运行的**源代码书写在题目下方**，并紧跟着贴上一份**源代码的截图**和运行结果的**截图**。答题格式同第1次实验作业。

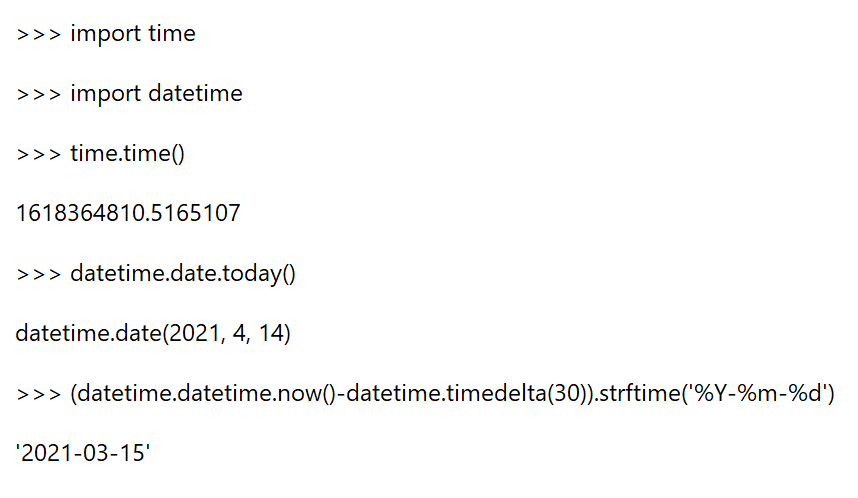
## (1)了解时间模块time和datetime，在交互式环境中执行以下任务：

(a) 利用time.time()获取当前的时间戳。

(b) 用datetime.date.today()获取当前的日期。

(c) 自行查询相关使用方法，利用datetime模块返回30天前的日期。

执行过程截图：



>>> import time

>>> import datetime

>>> time.time()

1618364810.5165107

>>> datetime.date.today()

datetime.date(2021, 4, 14)

>>> (datetime.datetime.now()-datetime.timedelta(30)).strftime('%Y-%m-%d')

'2021-03-15'

## (2) 数据随机存储，掌握os与random模块的用法以及文件读写方法。

定义一个函数fs(dirname,s)，其中参数dirname表示文件夹路径名，s为字符串，表示需要保存的数据。函数fs的功能是随机地将字符串s存储到dirname路径下某个文本文件(“\*.txt”文件)中。假如dirname路径下已有”a.txt”、 ”b.txt”、 ”c.txt”三个或更多文本文件，字符串s可能被保存到”a.txt”，或”b.txt”，或 ”c.txt”，或其他文件，机会是相同的，并且不能覆盖原有的文本文件。如果dirname路径下不存在任何文本文件，则新创建“new.txt”文件来保存字符串s。

源代码：

import os

import random

path\_string\_fix = "D:/OneDrive - pop.zjgsu.edu.cn/PythonPath/exp4/"

text\_string = "\ntest data to write"

def fs(dirname, s):

    """[to save the string s randomly to the txt file in the dirname path ]

    Args:

        dirname (str): file path

        s (str): data to save

    """

    list\_dir = os.listdir(path\_string\_fix)

    txt\_files\_list = list(filter(lambda file: file.endswith(".txt"), list\_dir))

    if len(txt\_files\_list)==0:

        with open(path\_string\_fix+"new.txt","w") as file\_output\_stream:

            file\_output\_stream.write(text\_string)

    else:

        file\_random\_index=random.randint(0, len(txt\_files\_list)-1)

        print(txt\_files\_list[file\_random\_index]+" will be written")

        # print(list(txt\_files\_list))

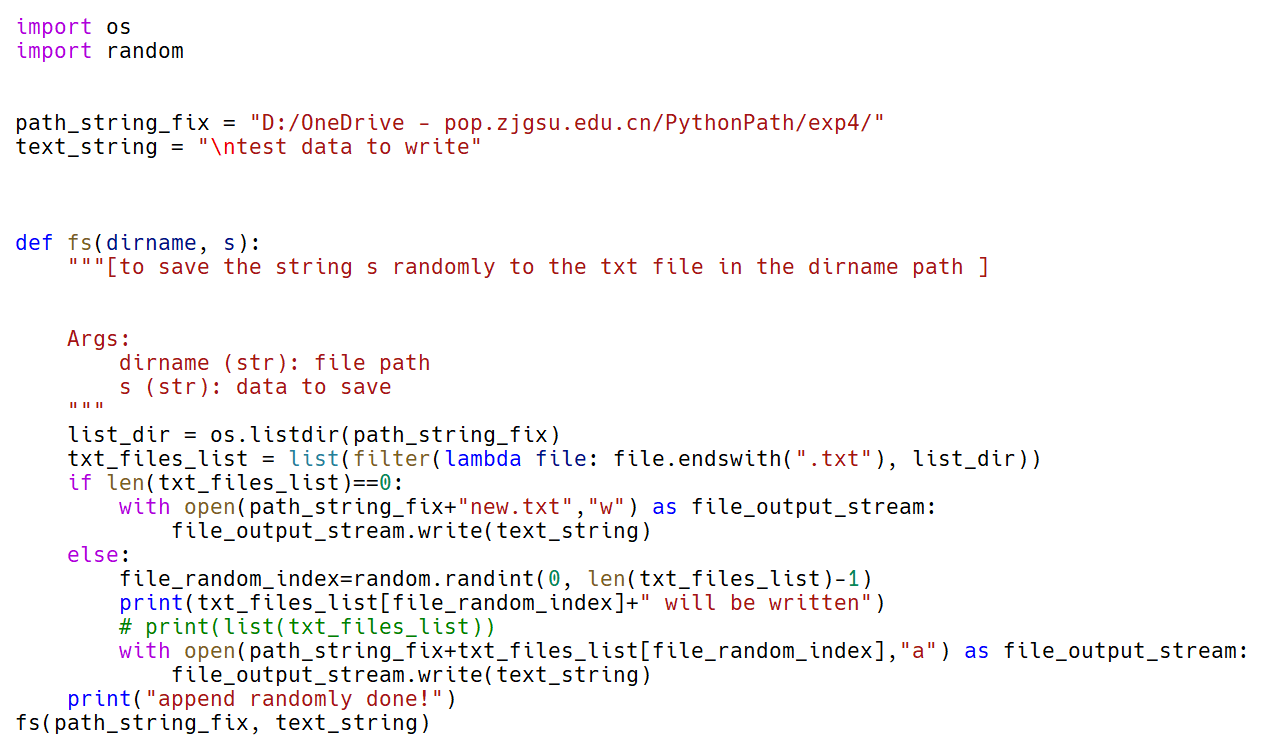
        with open(path\_string\_fix+txt\_files\_list[file\_random\_index],"a") as file\_output\_stream:

            file\_output\_stream.write(text\_string)

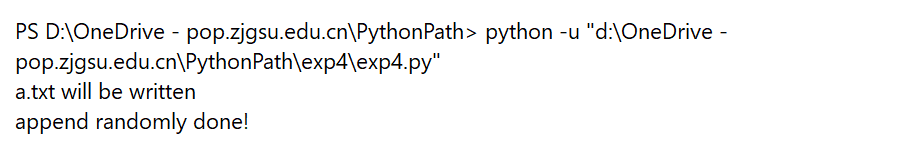
    print("append randomly done!")

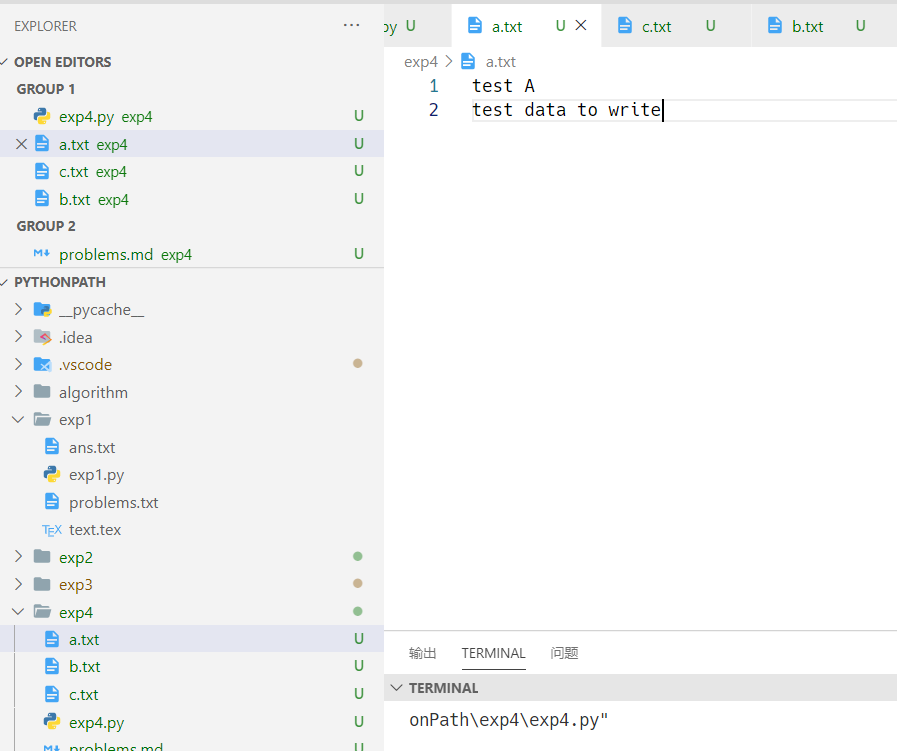
fs(path\_string\_fix, text\_string)

源代码截图：



运行结果截图：





## (3) 自定义一个模块，并调用模块中的函数。

首先, 定义一个函数fex (dirpwd)，用os模块的相关方法，返回一个列表，列表包括：dirpwd路径下所有文件不重复的扩展名，如果有2个".py"的扩展名，则返回一个".py"。提示：可用os.path.splitext()分离文件名和扩展名。然后，将定义该函数的代码保存在”exname.py”中，作为模块文件。

进一步，创建一个测试文件” mytest.py”，编写相关代码：调用exname模块中的fex方法，返回某个路径下的所有文件的扩展名（要求不重复）。

模块源代码：

import os

path\_string\_fix = "D:/OneDrive - pop.zjgsu.edu.cn/PythonPath/exp4/"

def fex(dirpwd):

    """return a set contains extension of files without repateing

    Args:

        dirpwd (string): path string

    Returns:

        set: file extension names

    """

    list\_dir=os.listdir(dirpwd)

    file\_post\_set=set()

    for file in list\_dir:

        file\_post=os.path.splitext(path\_string\_fix+file)[-1]

        if(file\_post):

            file\_post\_set.add(file\_post)

        # if file\_post in

    return file\_post\_set

主程序源代码：

import os

from exp4 import path\_string\_fix

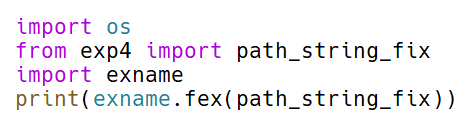
import exname

print(exname.fex(path\_string\_fix))

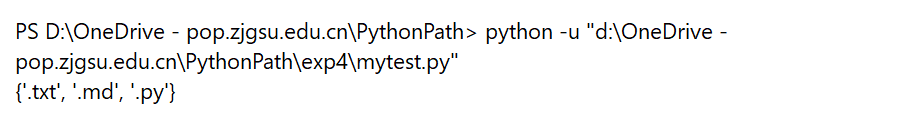
模块源代码截图：



主程序源代码截图：



运行结果截图：



## (4)类的定义以及实例化测试。

要求定义一个学生类Student，含有下面的**类属性：**

(a)姓名name

(b)年龄age

(c)成绩scores, 类型为列表或元组，形如（语文，数学，英语)

**类方法：**

(a)获取学生的姓名：get\_name()

(b)获取学生的年龄：get\_age()

(c)返回3门科目中最高的分数: get\_course()

写好类以后，给定1个同学进行**实例化测试**，例如:

zm = Student('zhangming', 20, [69,88,100])

**返回结果**：

zhangming

20

100

源代码：

class Student():

    name=""

    age=0

    scores=[]

    def \_\_init\_\_(self,name,age,scores):

        self.name=name

        self.age=age

        self.scores=scores

    def get\_name(self):

        return self.name

    def get\_age(self):

        return self.age

    def get\_course(self):

        return max(self.scores)

    def print\_all\_info(self):

        print(self.get\_name())

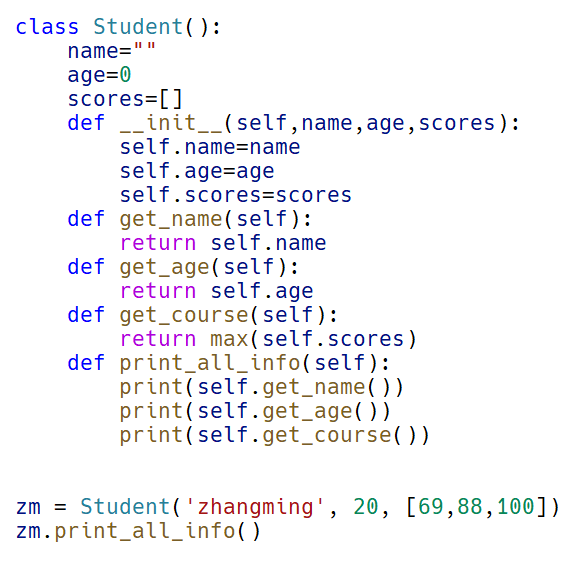
        print(self.get\_age())

        print(self.get\_course())

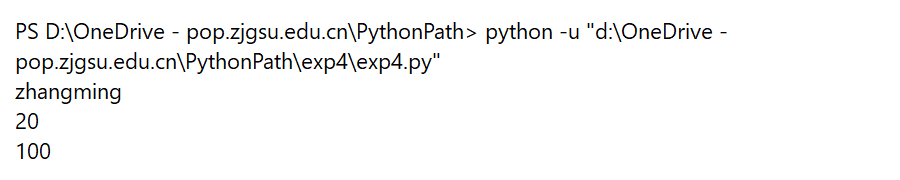
zm = Student('zhangming', 20, [69,88,100])

zm.print\_all\_info()

源代码截图：



运行结果截图：



## (5)类的定义以及实例化测试。

定义一个列表的操作类Listinfo，包括的方法:

(a)列表元素添加: add\_elem (elename) [elename: 字符串或者整数类型]

(b)列表元素取值：get\_ elem(num) [num: 整数类型]

(c)列表合并：merge\_list(ls) [ls: 列表类型]

(d)删除最后一个元素，并返回这个元素：del\_lastone()

写好这个类之后，进行实例化测试，某个实例对象创建可以如下：

list\_info = Listinfo([44,222,111,333,454,'sss','333'])

源代码：

class Listinfo():

    list=[]

    def \_\_init\_\_(self,list\_info):

        Listinfo.list=list\_info

        # self.list=list\_info

    def add\_elem(self,elename):

        Listinfo.list.append(elename)

        return Listinfo.list

    def get\_elem(self,num):

        return Listinfo.list[num]

    def merge\_list(self,ls):

        Listinfo.list=Listinfo.list+ls

        return Listinfo.list

    def del\_lastone(self):

        return Listinfo.list.pop()

    # def print\_test\_info():

    #

list\_info=Listinfo([44,222,111,333,454,'sss','333'])

# list\_info.add\_elem(5)

print("before operation:",list\_info.list)

print("list\_info.add\_elem(\"test add\") result:",list\_info.add\_elem("test add"))

print("list\_info.get\_elem(5):",list\_info.get\_elem(5))

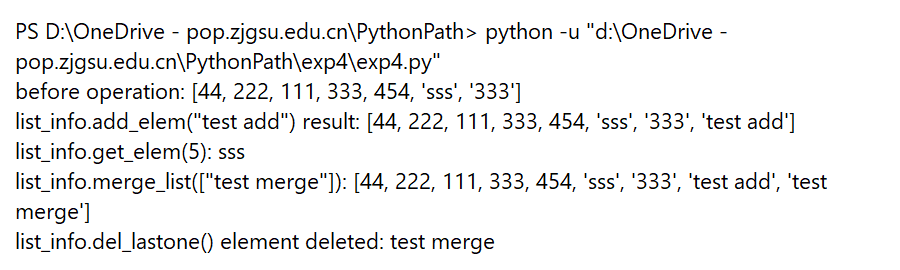
print("list\_info.merge\_list([\"test merge\"]):",list\_info.merge\_list(["test merge"]))

print("list\_info.del\_lastone() element deleted:",list\_info.del\_lastone())

源代码截图：



运行结果截图：



## (6)通过Python包与模块的创建，理解Python项目的组织结构。

创建一个**包（Package）**，并命名为“mypack”，注意包的目录下需要包含“\_\_init\_\_. py”文件。

在“mypack”目录下创建“aa.py”文件以及一个名为“subpack”的**子包**，在子包“subpack”下创建“bb.py”文件。最后创建一个和“mypack”同级的“test.py”文件。项目结构如下:

test.py

mypack

|-- \_\_init\_\_.py

|-- aa.py

|-- subpack

|-- \_\_init\_\_.py

|-- bb.py

其中， 需将各个“\*.py”源代码具体定义为：

(i)在“aa.py”中定义一个函数add(x, y)，该方法能够打印输出x、y的两数之和；(ii) 在“bb.py”中定义一个函数sub(x, y)，该方法能够打印输出x、y的两数之差；

(iii)在“mypack”的“\_\_init\_\_. py”文件中定义两个变量a和b，并为其赋值a=2, b=1；

(iv)在“test.py”文件中定义两个变量m和n，并为其赋值m=4, n=3。

且要求在“test.py”文件中，进行如下操作：

(a)使用from ... import ...的方式导入模块“bb.py”中的sub(x, y)方法，将m、n传入sub(x,y)方法中，得到输出结果；

(b)使用from ... import ...的方式导入模块“aa.py”中的add(x,y)方法，将m、n传入add(x,y)方法中，得到输出结果；

(c)使用from ... import ...的方式导入“mypack”里“\_\_init\_\_. py”文件的a、b，将a、b传入add(x,y)方法中，得到输出结果。

源代码(test.py)：

m=4

n=3

from mypack.subpack.bb import sub

sub(m,n)

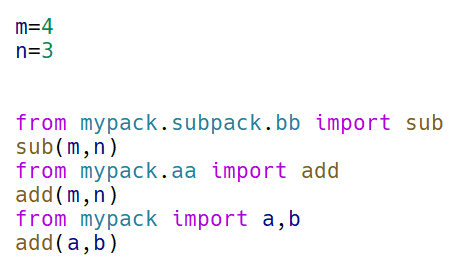
from mypack.aa import add

add(m,n)

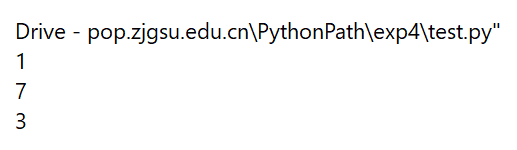
from mypack import a,b

add(a,b)

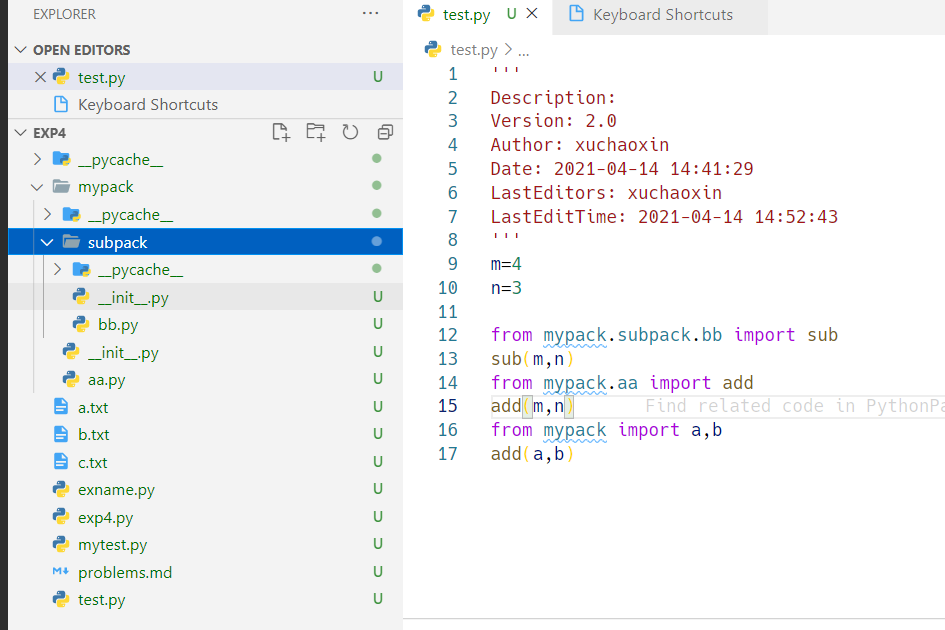
源代码(test.py)截图：



运行结果截图：



项目实际目录截图：



## (7)类的继承及其实例化测试。

定义一个类Human，包括的方法: (i) 构造函数\_\_init\_\_()：包含参数name、age; (ii) get\_name(): 打印输出name的内容; (iii) do\_homework()：打印输出语句‘There is no homework from the parent!’;

另外再定义一个类Student，让其**继承**Human类，包含的方法：(i)构造函数\_\_init\_\_()：包含参数name、age、homework; (ii) do\_homework()：打印输出属性homework内容的语句，如print(“作业为:”+ self.homework)，并利用super()方法继承Human类的do\_homework()方法。

接下来，进行实例化测试，如stu = Student(‘John’, 20, ‘Python实验’)，进一步引用并打印输出对象stu的三个属性name、age、homework的内容，然后调用方法do\_homework()和get\_name()，查看运行结果。

源代码：

class Human():

    def \_\_init\_\_(self, name, age):

        self.name = name

        self.age = age

    def get\_name(self):

        print(self.name)

    def get\_age(self):

        print(self.age)

    def do\_homework(self):

        print("there is no homework from the parent!")

    # def test(self):

    #     self.do\_homework()

class Student(Human):

    def \_\_init\_\_(self, name, age, homework):

        Human.\_\_init\_\_(self, name, age)

        self.homework = homework

    # def get\_homework(self):

    def do\_homework(self):

        print("作业为："+self.homework)

        super().do\_homework()

# stu=Student()

# human = Human("sdfj", 5)

# human.test()

stu = Student("‘John’", 20, "‘Python实验’")

print(stu.name)

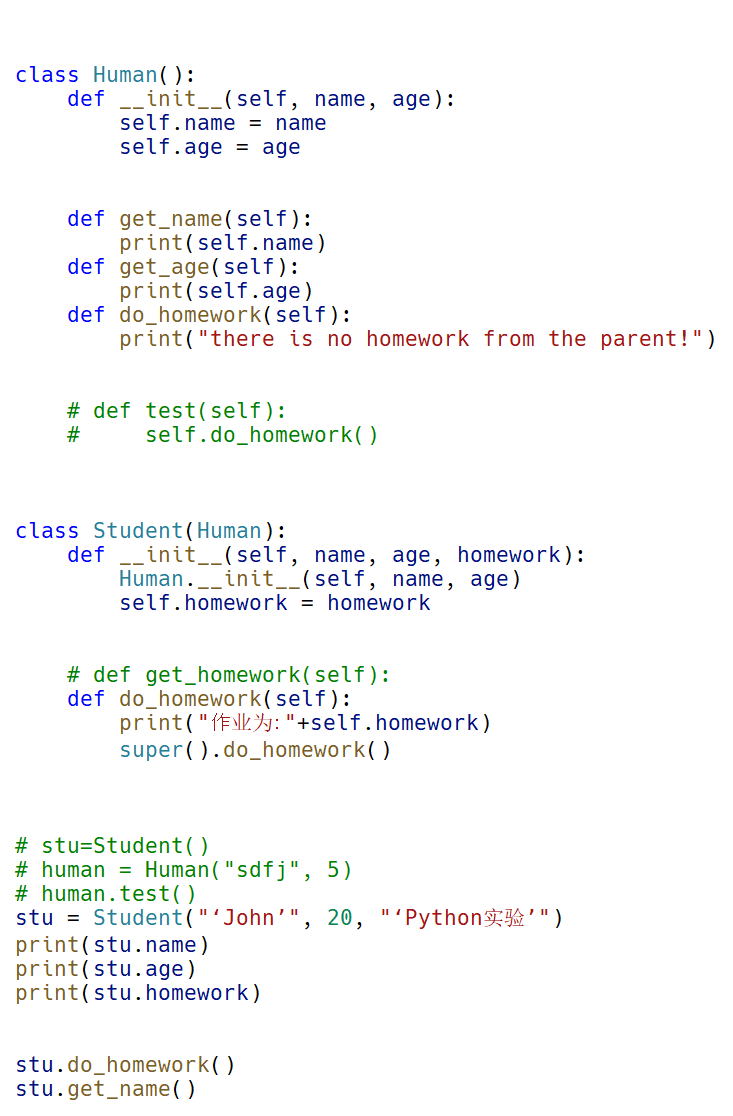
print(stu.age)

print(stu.homework)

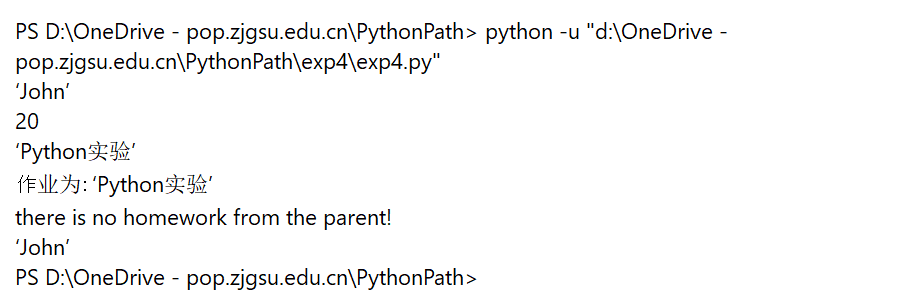
stu.do\_homework()

stu.get\_name()

源代码截图：



运行结果截图：



## (8)类成员的访问以及实例化测试。

定义一个计数器类Counter，包括的变量及方法: **(i)**首先定义两个**类变量**，分别为：**公开变量**publicCount，为其赋初始值0；**私有变量**secretCount（私有变量的命名方式为：\_\_variableName），为其赋初始值0；**(ii)**定义方法count()：能够使得私有变量secretCount和共有变量publicCount都自增1，并打印输出私有变量secretCount的值。

接下来，进行实例化测试，创建类的实例对象，形如counter = Counter()，并进行以下操作：

(a)使用类对象调用count()方法；

(b)尝试使用类对象直接调用私有变量secretCount。如果报错，思考如何使用类对象直接调用私有属性，并打印输出结果。

源代码：

class Counter():

    publicCount=0

    \_\_secretCount=0

    def count(self):

        self.publicCount+=1

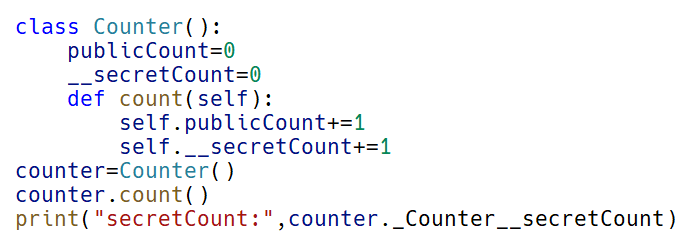
        self.\_\_secretCount+=1

counter=Counter()

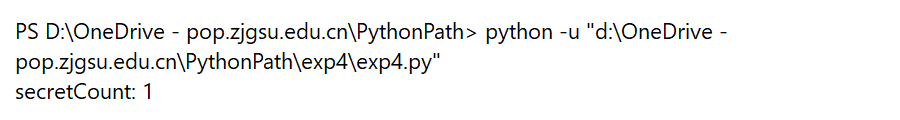
counter.count()

print("secretCount:",counter.\_Counter\_\_secretCount)

源代码截图：



运行结果截图：



## (9) 模块文件中定义类及其实例化。

首先，创建一个名为“mc.py”的模块文件。在该模块文件中，编写一个通用的人员类（person），该类具有姓名（Name）、年龄（Age）、性别（Sex）等私有属性，如Name的私有属性可以写为“self.\_\_name”。为保护个人隐私，person类的所有数据属性均需定义为私有属性。然后，对person 类进行继承得到一个学生类（student），该类能够存放学生任意多门课的成绩（这里门数不定），并能求出平均成绩。

最后，另外创建一个主文件“test.py”，以from…import…方式导入student类，并给出三位学生的student实例，要求他们的课程门数不同，在实例化测试中对student类的功能进行验证。

源代码(mc.py)：

class Person():

    def \_\_init\_\_(self, name, age,sex):

        self.\_\_name=name

        self.\_\_age=age

        self.\_\_sex=sex

    def get\_all\_info(self):

        return self.\_\_name,self.\_\_age,self.\_\_sex

    def get\_string(self):

        tuple=self.get\_all\_info()

        string=""

        for item in tuple:

            string+=str(item)+" "

        return string

class Student(Person):

    def \_\_init\_\_(self,name,age,sex,courses\_dict):

        # Person.\_\_init\_\_(self,name, age,sex)

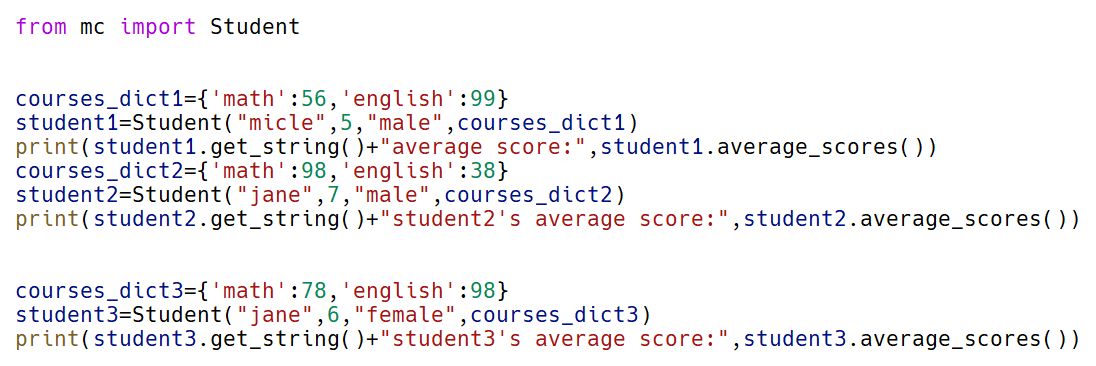
        super().\_\_init\_\_(name,age,sex)

        self.courses\_dict=courses\_dict

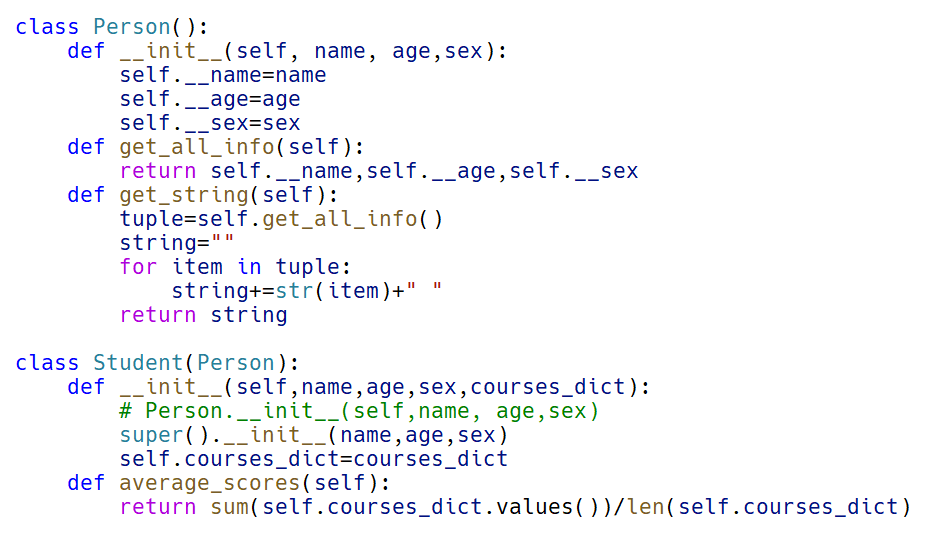
    def average\_scores(self):

        return sum(self.courses\_dict.values())/len(self.courses\_dict)

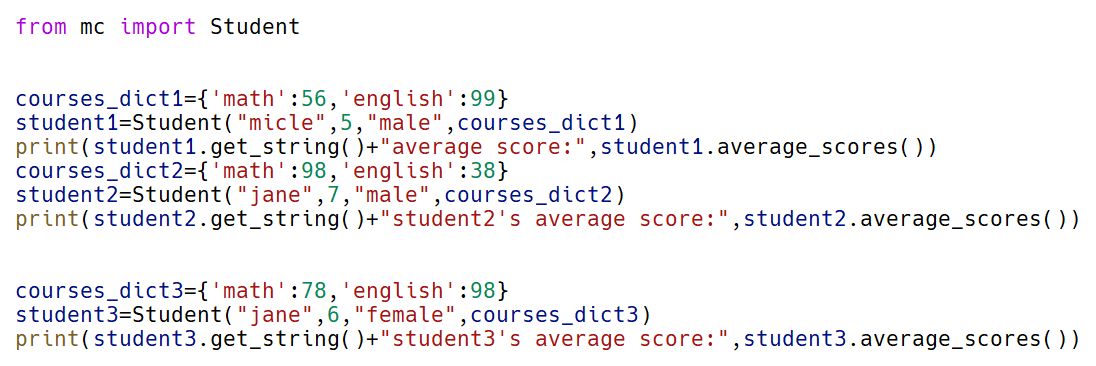
源代码(test.py)：



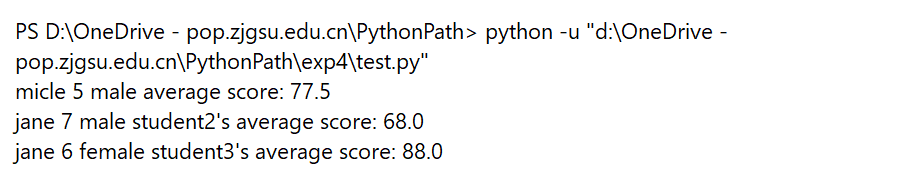
源代码截图(mc.py)：



源代码截图(test.py)：



运行结果截图：

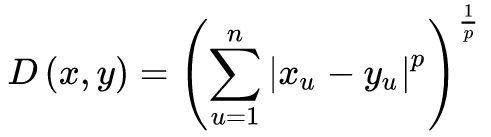


## (10) 定义一个高维空间样本点集类HDPoints，须包含以下数据属性与方法属性：

(a)数据属性self.points：类型为列表，由多个子列表构成，每个子列表表示高维空间中的一个数据点，且数据维度可以任意，并通过初始化构造函数获得。

(b)方法属性centerpoint(self)：计算点集的中心点。

(c)方法属性minkowski (self, x, y, p)：计算两点x和y之间的闵可夫斯基距离，p为非负整数，用p=0情形表示切比雪夫距离。由此定义的距离称为p-闵氏距离，其数学定义如下：



(d)方法属性farthestpoint(self, p)：找出离中心点p-闵氏距离最远的点，返回在self.points中的下标以及最大距离。

(e)方法属性farthest2points(self, p)：找出点集self.points中p-闵氏距离最远的两点，返回两点在self.points中的下标及其最大距离。

接一下来，实例化类HDPoints，利用random模块，随机产生至少50个高维空间数据点，样本点的维度至少在5以上，且每个分量取值服从区间[0,1]上的均匀分布。

同时，随机产生一个0~5之间的一个非负整数，赋值传递给p-闵氏距离函数中的参数p，对HDPoints实例对象的全部自定义方法属性（即centerpoint()、minkowski()、farthestpoint()和farthest2points()）进行功能测试。

源代码：

import numpy as np

from numpy import random

import itertools

class HDPoints():

    def \_\_init\_\_(self,HDPoints\_list):

        self.points =HDPoints\_list

    def centerpoint(self):

        ndarray=np.array(self.points)

        return sum(ndarray)/len(ndarray)

    def minkowski(self,x,y,p):

        abs\_list= [abs(x-y)\*\*p for x,y in zip(x,y)]

        return sum(abs\_list)\*\*(1/p)

    def farthestpoint(self,p):

        centerpoint=self.centerpoint()

        distances\_list=[self.minkowski(centerpoint,point,p) for point in self.points ]

        max\_distance= max(distances\_list)

        return  distances\_list.index(max\_distance),max\_distance

    def farthest2points(self,p):

        points\_index\_tuple\_list=[(point,i)for i,point in enumerate(self.points)]

        point\_pairs\_tuples=(itertools.combinations(points\_index\_tuple\_list,r=2))

            #element shape:(([point\_list1],index2),([point\_list2],index2))

        distances\_list=[(self.minkowski(tuple[0][0],tuple[1][0],p),(tuple[0][1],tuple[1][1])) for tuple in point\_pairs\_tuples ]

            #element shape:(minkowski\_distance,(index1,index2))

        max\_distance\_point=max(distances\_list,key=lambda tuple:tuple[0])

        return max\_distance\_point[1],max\_distance\_point[0]

# a=random.rand(1,0.5,2,3,6,3)

a=random.uniform(0,1,5).tolist()

""" get points list: """

points=[random.uniform(0,1,5).tolist() for i in range(50)]

hd\_Points=HDPoints(points)

p=random.randint(1,6)

print("the centerpoint is:",hd\_Points.centerpoint())

""" the minkowski method will be test contained in the farthestpoint() method! """

print("the farthest point:",hd\_Points.farthestpoint(p))

print(f"the farthest2point: index of the 2 pointes: {hd\_Points.farthest2points(p)[0]},the max minkowski distance is {hd\_Points.farthest2points(p)[1]}")

源代码截图：



运行结果截图：

