**实验4模块与面向对象编程**

**姓名：宁浩 学号：2312190617 班级：软件2301 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，且在规定时间内完成本次实验，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。

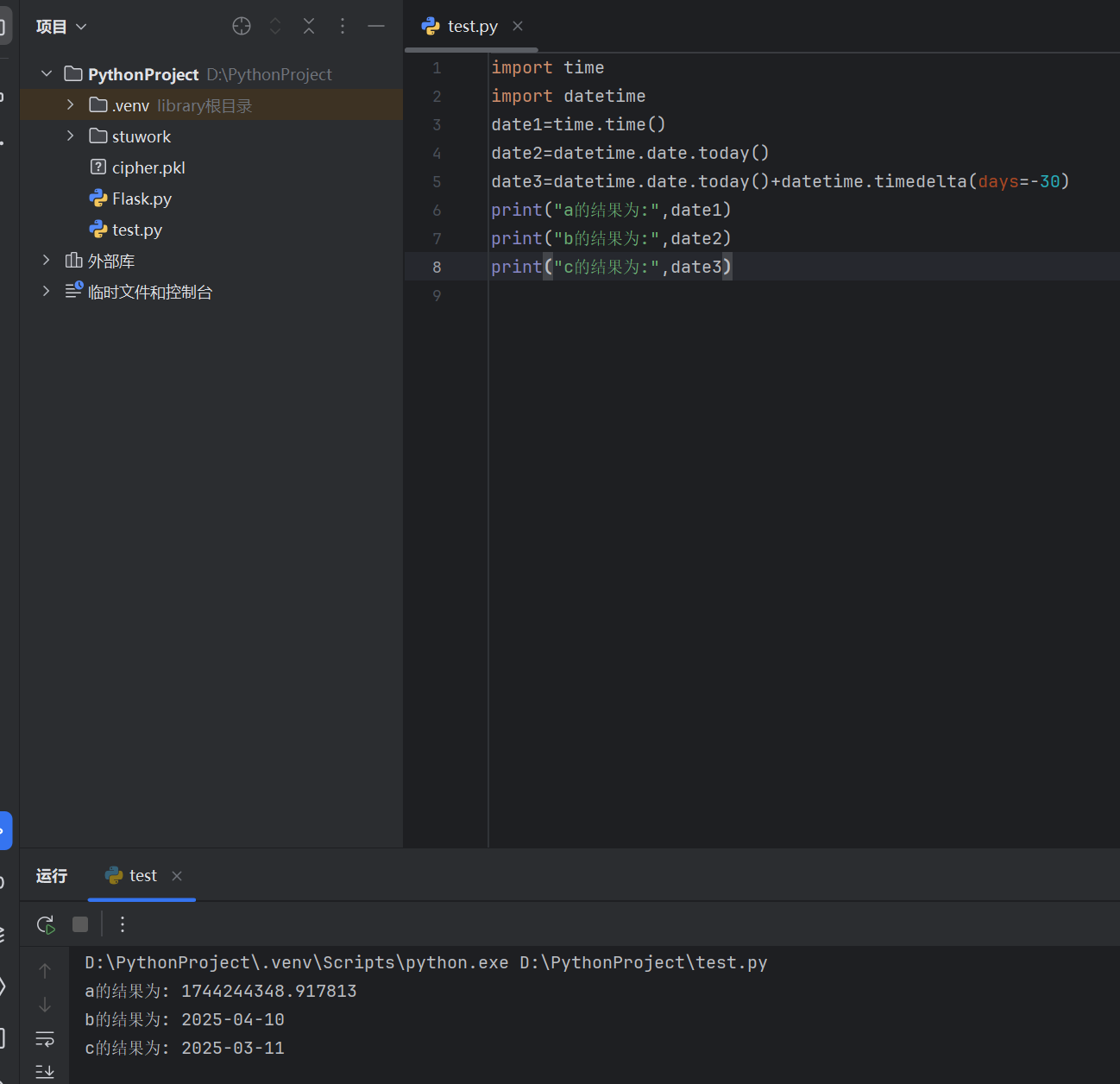
1. 了解时间模块time和datetime，在交互式环境中执行以下任务：

(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：



1. 设计一个学生成绩管理系统类 GradeManager，包含以下功能：

**数据属性：**

students：私有字典，存储学生信息（格式：{"学号": {"姓名": str, "成绩": float}}）。

**方法属性：**

add\_student(self, student\_id: str, name: str, grade: float)：添加学生信息。

remove\_student(self, student\_id: str)：删除指定学号的学生。

get\_grade(self, student\_id: str) -> float：获取学生成绩。

get\_average(self) -> float：计算全班平均成绩。

get\_top\_student(self) -> tuple：返回成绩最高的学生（学号、姓名、成绩）。

源代码：

class GradeManager:

student={}

def \_\_init\_\_(self):

self.student={}

def add\_student(self, student\_id: str, name: str, grade: float):

self.student[student\_id]={"姓名":name,"成绩":grade}

def remove\_student(self, student\_id: str):

self.student[student\_id]=None

def get\_grade(self, student\_id: str) -> float:

return self.student[student\_id]["成绩"]

def get\_average(self) -> float:

return sum(list(map(lambda x:x["成绩"],self.student.values())))/len(list(self.student.values()))

def get\_top\_students(self) -> tuple:

max=0.0

list=[]

for key in self.student.keys(): #找最大分

if self.student[key]["成绩"] > max:

max=self.student[key]["成绩"]

for key in self.student.keys(): #处理同分

if self.student[key]["成绩"] == max:

list.append((key,self.student[key]["姓名"],self.student[key]["成绩"]))

item=list[0]

if len(list) > 1:

for it in list:

item+=it

return item

G=GradeManager()

G.add\_student("1","一",100)

G.add\_student("2","二",90)

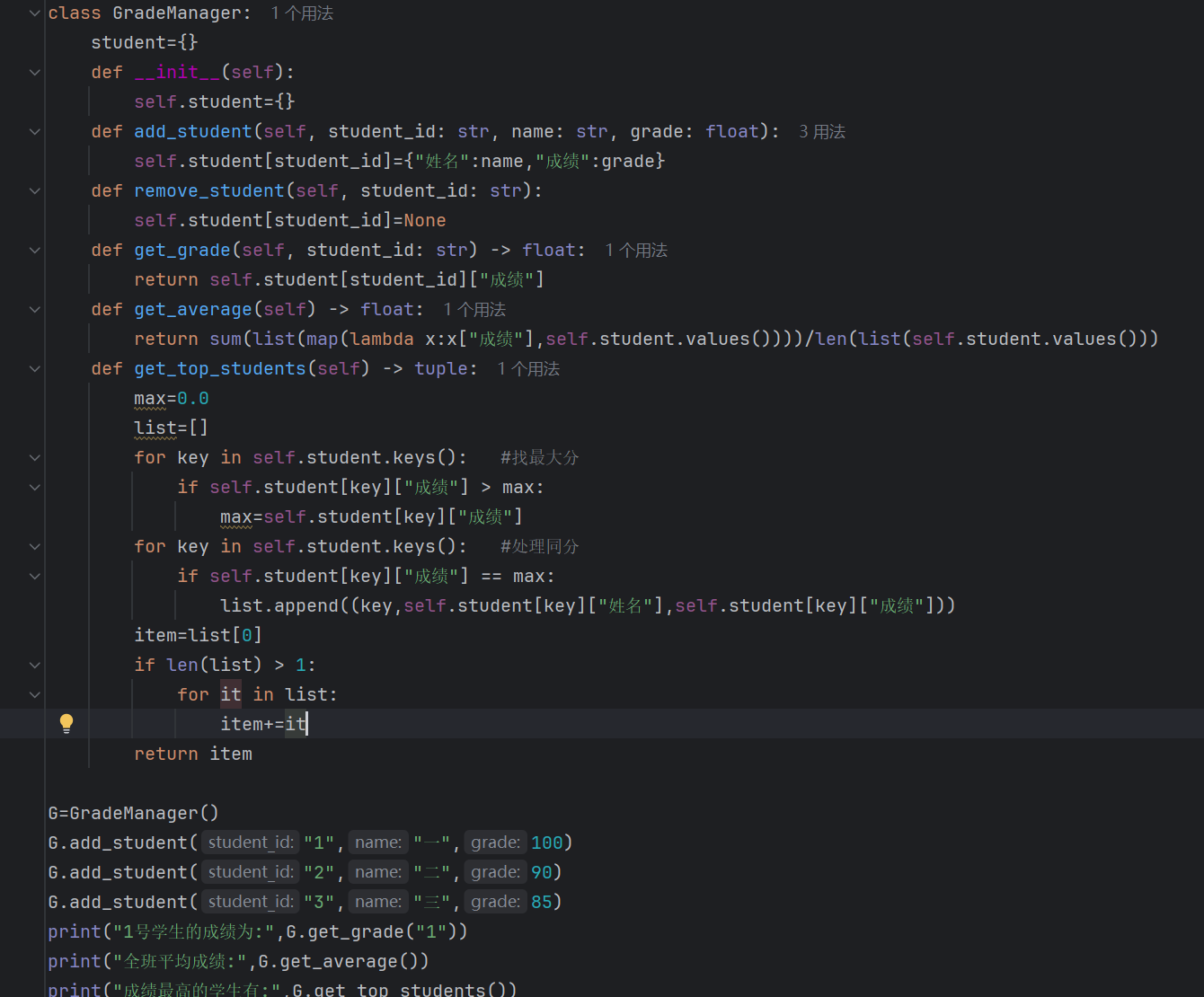
G.add\_student("3","三",85)

print("1号学生的成绩为:",G.get\_grade("1"))

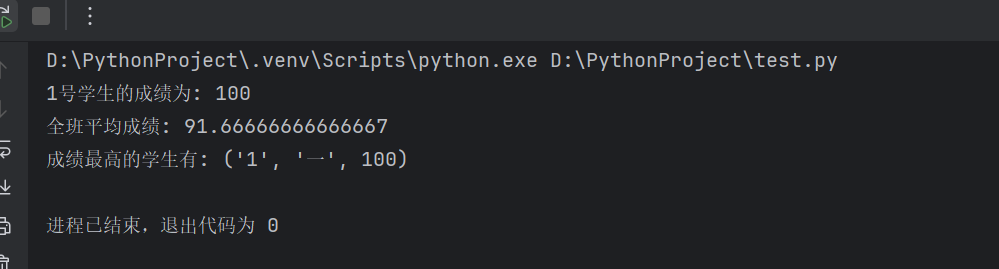
print("全班平均成绩:",G.get\_average())

print("成绩最高的学生有:",G.get\_top\_students())

源代码截图：



运行结果截图：



1. 设计一个简易计算器类Calculator

**方法属性：**

add(self, a: float, b: float) -> float：返回 a + b。

subtract(self, a: float, b: float) -> float：返回 a - b。

multiply(self, a: float, b: float) -> float：返回 a \* b。

divide(self, a: float, b: float) -> float：返回 a / b（需处理除零错误）。

power(self, a: float, b: float) -> float：返回 a \*\* b。

history(self) -> list：返回所有历史计算记录（格式：[("add", 2, 3, 5), ...])

源代码：

class Calculator:

list=[]

def \_\_init\_\_(self):

self.list=[]

def add(self,a:float,b:float) -> float:

self.list.append(("add",a,b,a+b))

return a+b

def subtract(self,a:float,b:float) -> float:

self.list.append(("subtract",a,b,a-b))

return a-b

def multiply(self,a:float,b:float) -> float:

self.list.append(("multiply",a,b,a\*b))

return a\*b

def divide(self,a:float,b:float) -> float:

try:

self.list.append(("divide",a,b,a/b))

return a/b

except ZeroDivisionError:

print("分母为0,除零错误")

def power(self,a:float,b:float) -> float:

self.list.append(("power",a,b,a\*\*b))

return a\*\*b

def history(self):

return self.list

C=Calculator()

a=4

b=2

print("a为:",a,",b为:",b)

print("a+b=",C.add(a,b))

print("a-b=",C.subtract(a,b))

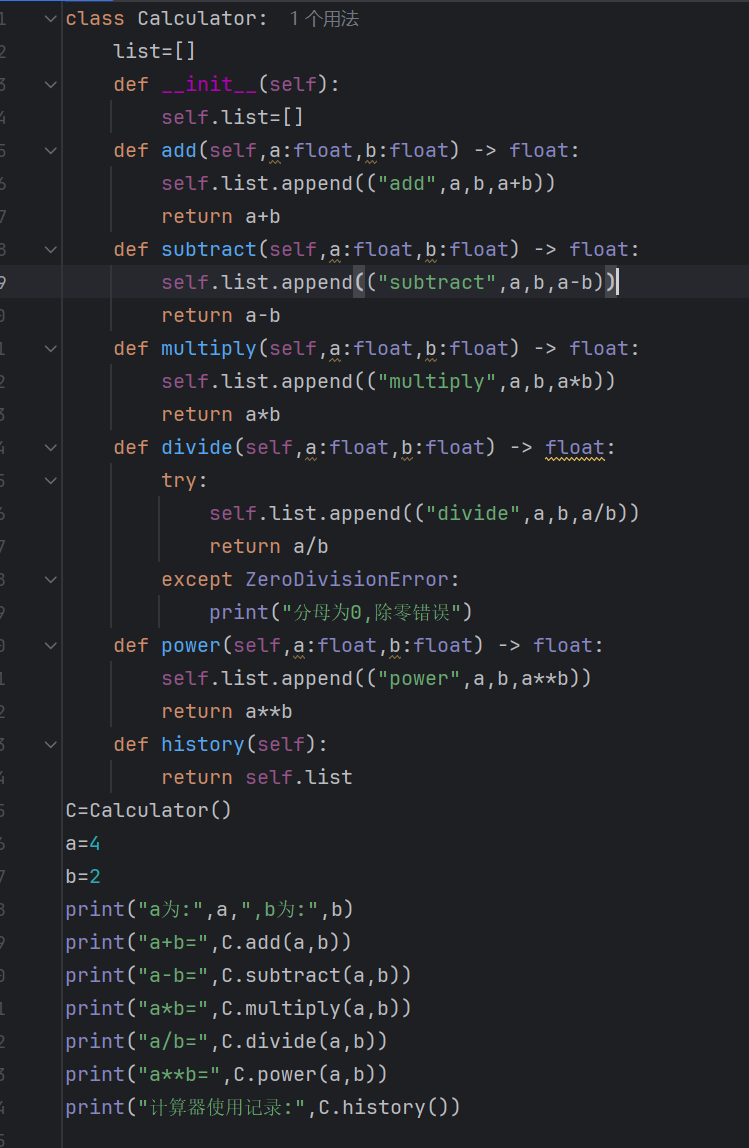
print("a\*b=",C.multiply(a,b))

print("a/b=",C.divide(a,b))

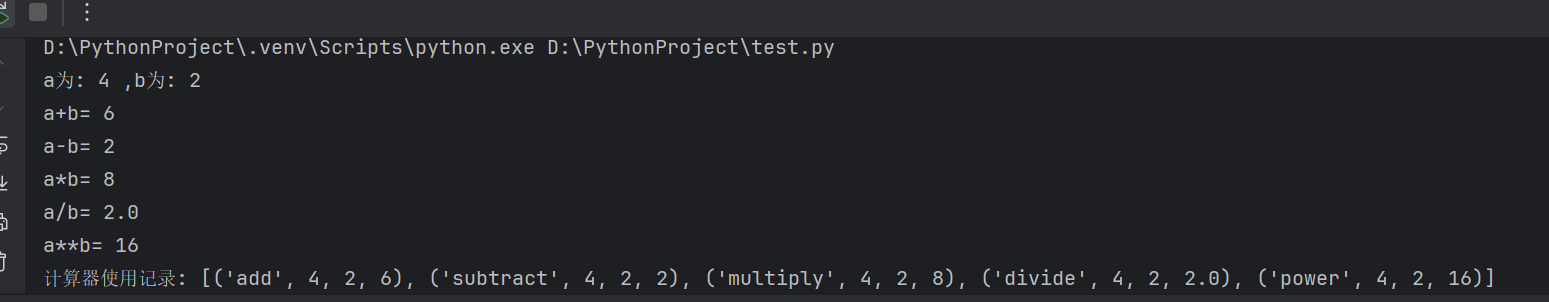
print("a\*\*b=",C.power(a,b))

print("计算器使用记录:",C.history())

源代码截图：



运行结果截图：



1. 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

def fs(dirname,s):

filename=list(os.listdir(dirname))

if len(filename)==0:

with open(os.path.join(dirname,r'new.txt'),"a",encoding='utf\_8') as f:

print("路径下不存在任何文本文件，新创建new.txt文件进行存储")

f.write(s)

else:

index=random.randint(0,len(filename)-1)

des=os.path.join(dirname,filename[index])

with open(des,"a",encoding='utf\_8') as f:

print("随机到的存储文件为:",filename[index])

f.write(s)

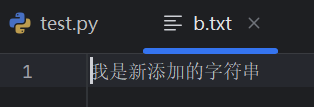
src=r'D:\PythonProject\filename'

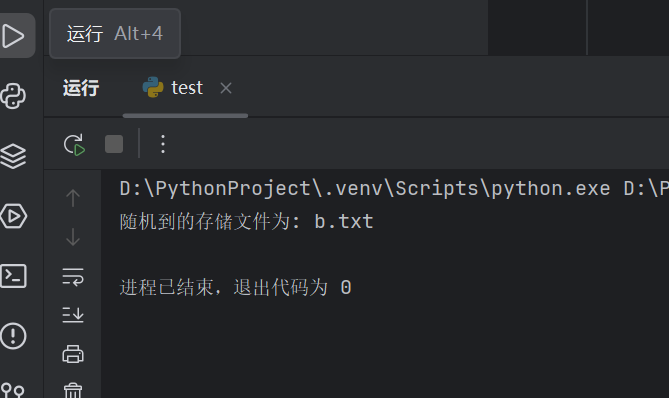
fs(src,"我是新添加的字符串")

源代码截图：



运行结果截图：





1. 类的定义以及实例化测试。

设计一个字典操作类 DictInfo，包含以下方法：

(a)添加键值对：add\_item(key, value)； [key: 键（字符串或数字类型）value: 值（任意类型）]

(b)获取键值：get\_value(key); [key: 键（字符串或数字类型）]

(c)合并字典：merge\_dict(dict2) [dict2: 另一个字典]

(d)删除最后一个键值对：del\_last\_item();[（格式：(key, value)）]

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：

dict\_info = DictInfo({'a': 1, 'b': 2, 3: 'c'})

源代码：

class DictInfo:

dic={}

def \_\_init\_\_(self,dict1):

self.dic=dict1

def add\_item(self,key,value):

self.dic[key]=value

def get\_value(self,key):

return self.dic[key]

def merge\_dict(self,dict2):

for key in dict2.keys():

self.dic[key]=dict2[key]

return self.dic

def del\_last\_item(self):

key=list(self.dic.keys())[-1]

self.dic.pop(key)

return self.dic

dict\_info = DictInfo({'a': 1, 'b': 2, 3: 'c'})

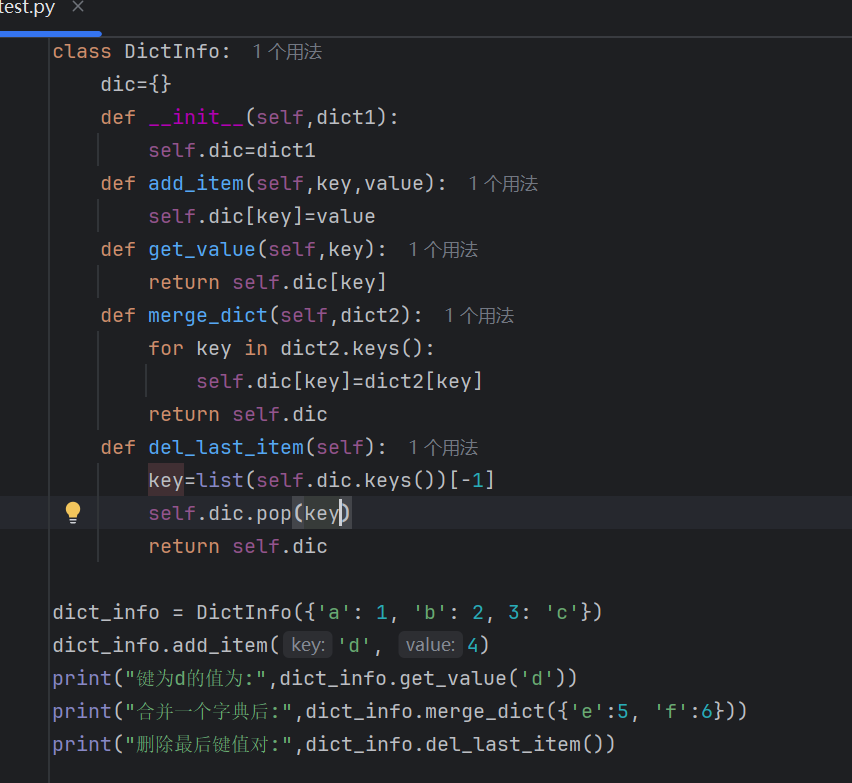
dict\_info.add\_item('d', 4)

print("键为d的值为:",dict\_info.get\_value('d'))

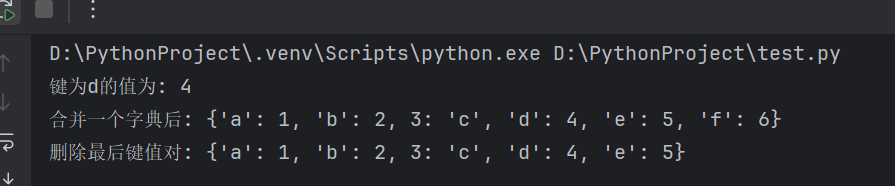
print("合并一个字典后:",dict\_info.merge\_dict({'e':5, 'f':6}))

print("删除最后键值对:",dict\_info.del\_last\_item())

源代码截图：



运行结果截图：



(6) 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:

name = ""

age = 0

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

self.name = name

self.age = age

def get\_name(self):

print(self.name)

def do\_homework(self):

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

class Student(Human):

name = ""

age = 0

homework=''

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

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

self.homework= homework

def do\_homework(self):

print('作业为:'+self.homework)

super().do\_homework()

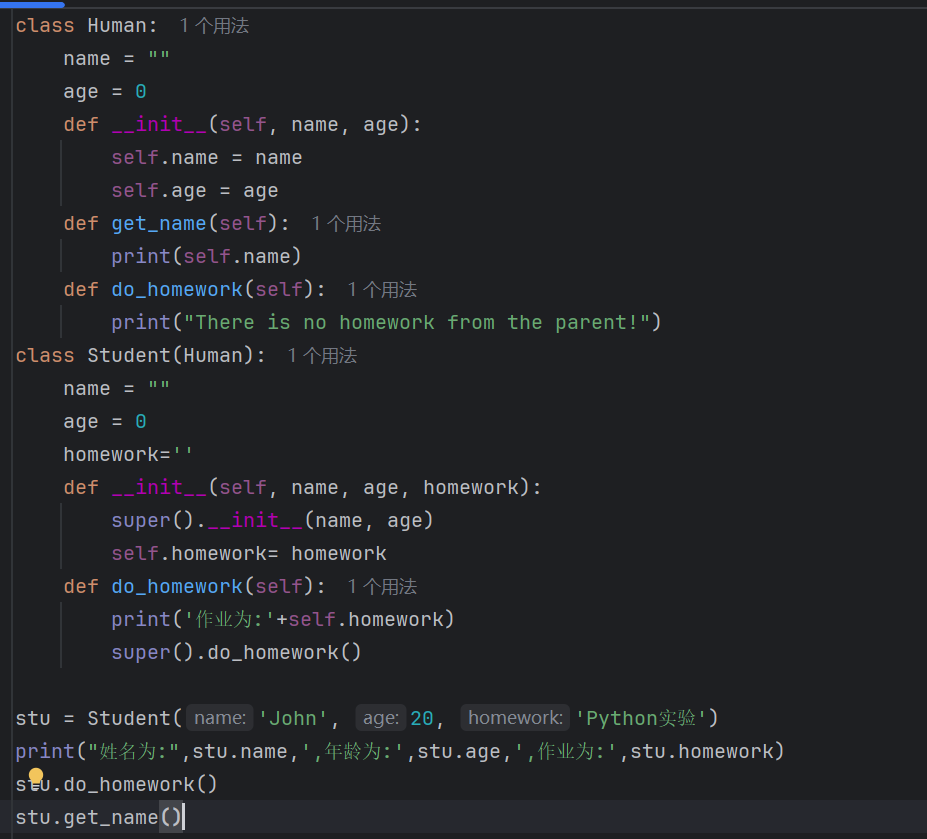
stu = Student('John', 20, 'Python实验')

print("姓名为:",stu.name,',年龄为:',stu.age,',作业为:',stu.homework)

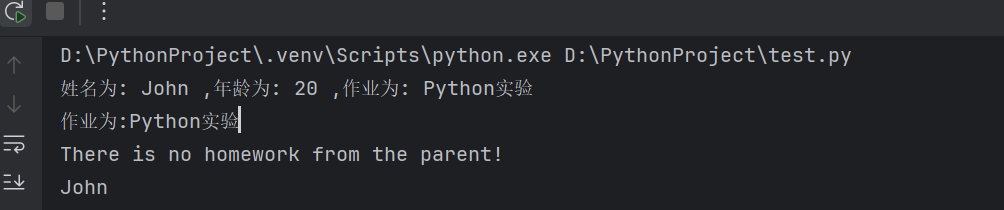
stu.do\_homework()

stu.get\_name()

源代码截图：



运行结果截图：



(7) 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)：

from mypack.aa import add

from mypack.subpack.bb import sub

from mypack import a,b

m=4

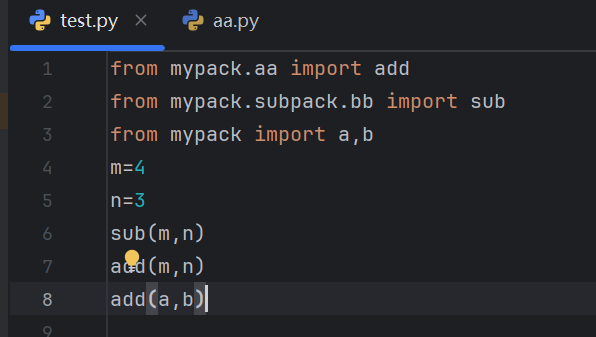
n=3

sub(m,n)

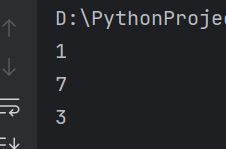
add(m,n)

add(a,b)

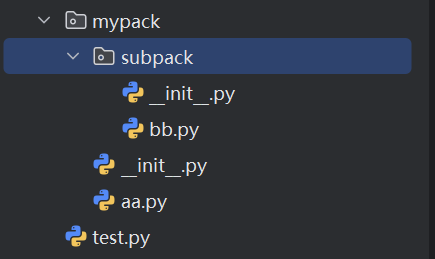
源代码(test.py)截图：



运行结果截图：



项目实际目录截图：



(8) 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:

name=''

age=0

gender=''

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

self.\_\_name=name

self.\_\_age=age

self.\_\_gender=sex

def getName(self):

return self.\_\_name

def getAge(self):

return self.\_\_age

def getGender(self):

return self.\_\_gender

def setName(self,name):

self.\_\_name=name

def setAge(self,age):

self.\_\_age=age

def setGender(self,gender):

self.\_\_gender=gender

class student(person):

grade={}

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

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

def add\_grade(self, subject, grade):

self.grade[subject]=grade

def get\_average(self):

return sum(self.grade.values())/len(self.grade)

源代码(test.py)：

from mc import student

stu1= student("一一",18,"男")

stu1.add\_grade("语文",95)

print("同学"+stu1.getName()+"的平均分为:",stu1.get\_average())

stu2= student("二二",18,"女")

stu2.add\_grade("语文",95)

stu2.add\_grade("数学",99)

print("同学"+stu2.getName()+"的平均分为:",stu2.get\_average())

stu3= student("三三",18,"男")

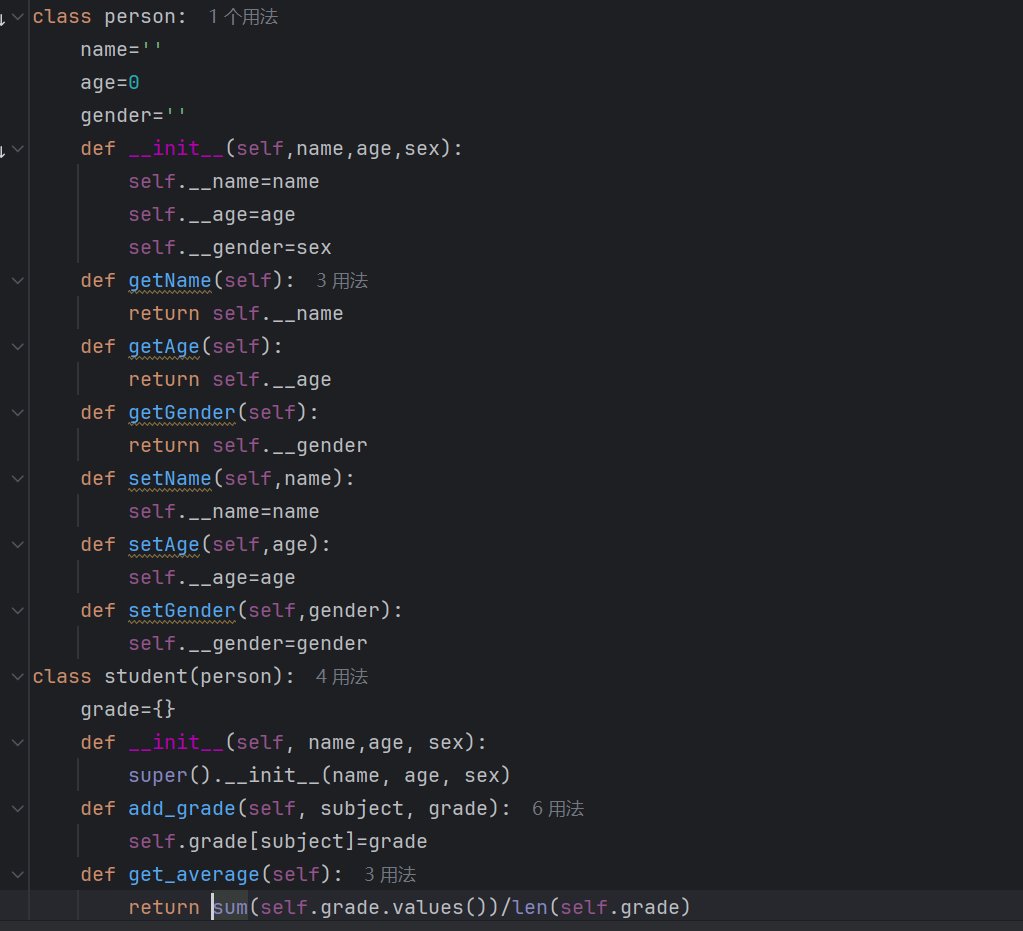
stu3.add\_grade("语文",95)

stu3.add\_grade("数学",96)

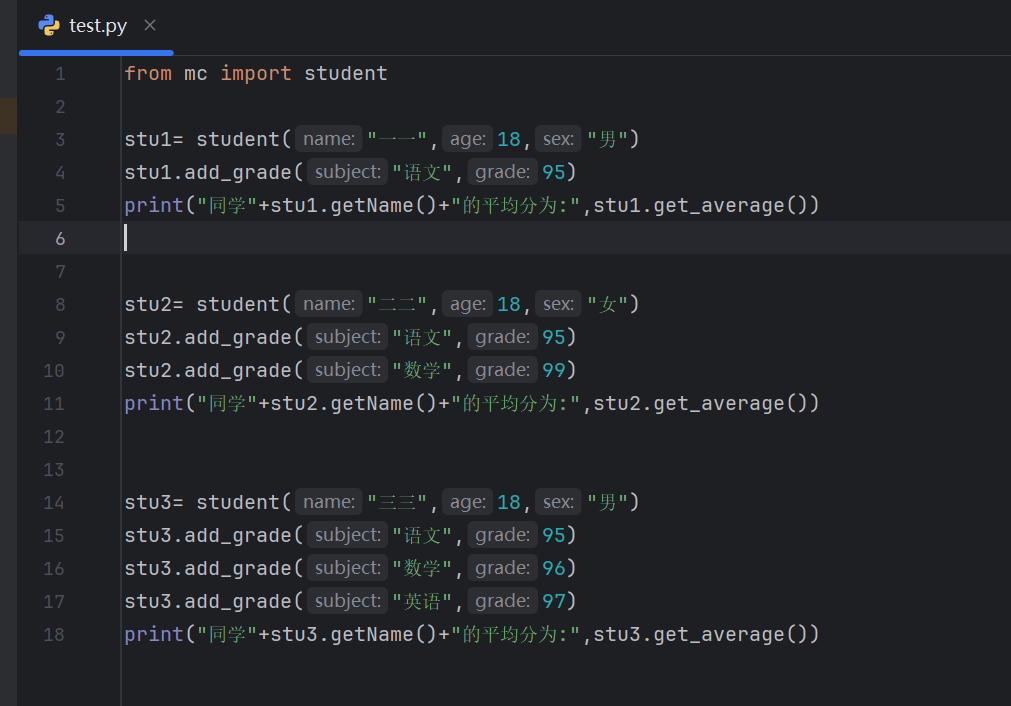
stu3.add\_grade("英语",97)

print("同学"+stu3.getName()+"的平均分为:",stu3.get\_average())

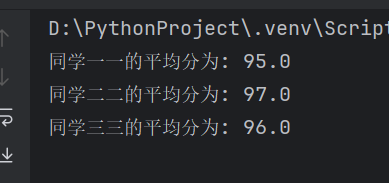
源代码截图(mc.py)：



源代码截图(test.py)：



运行结果截图：



1. 设计一个图书类Book，其中的数据成员有：price代表图书价格（私有属性）；title代表图书书名（私有属性）

请完成要求：

1)创建多个 Book 类实例（至少3本以上），初始化书名和价格。

2)对每本书的价格打8折（促销活动）。

3)对打折后的价格按从低到高排序。

4)输出排序后的书名和价格。

源代码：

class Book:

title = ""

price = 0

def \_\_init\_\_(self, title, price):

self.\_\_title = title

self.\_\_price = price

def get\_title(self):

return self.\_\_title

def get\_price(self):

return self.\_\_price

def set\_title(self, title):

self.\_\_title = title

def set\_price(self, price):

self.\_\_price = price

book1=Book("书1",100)

book2=Book("书2",110)

book3=Book("书3",120)

book4=Book("书4",130)

book=[book1,book2,book3,book4]

book1.set\_price(book1.get\_price()\*0.8)

book2.set\_price(book2.get\_price()\*0.8)

book3.set\_price(book3.get\_price()\*0.8)

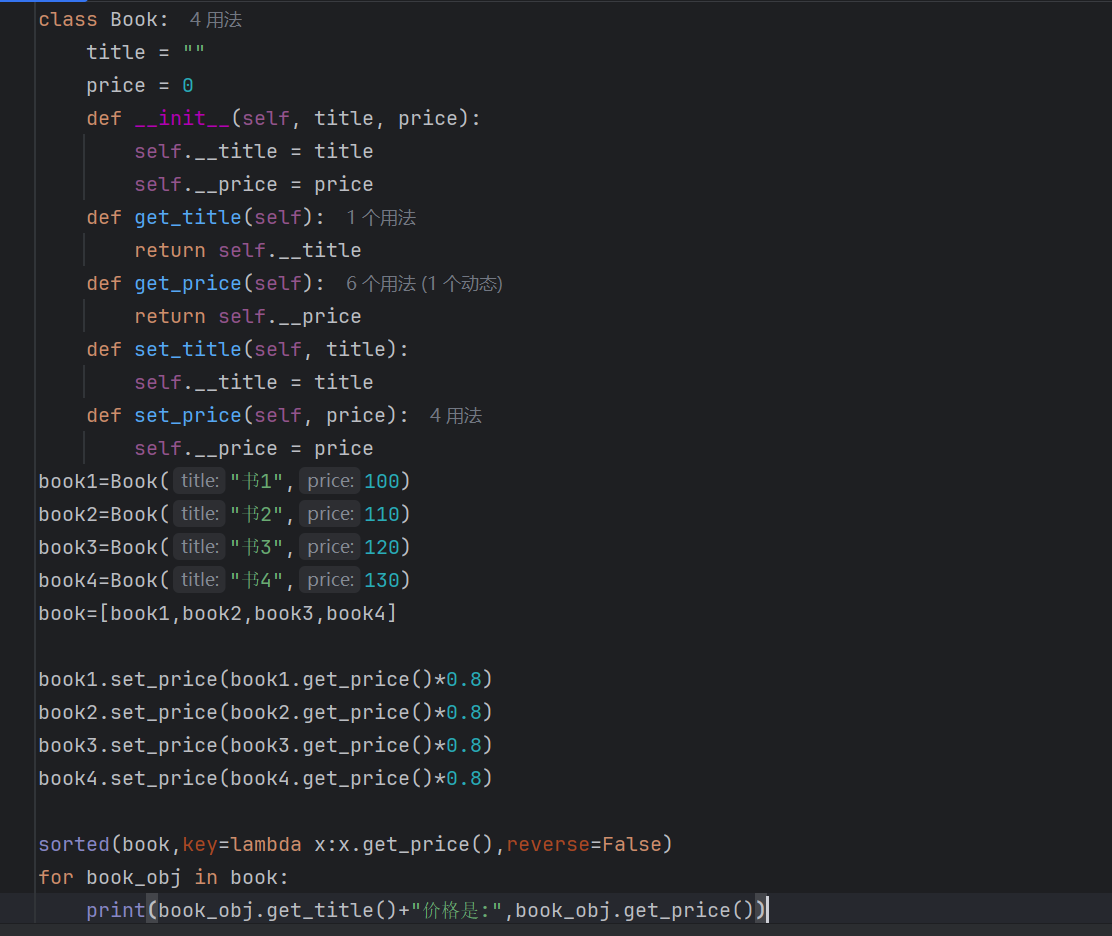
book4.set\_price(book4.get\_price()\*0.8)

sorted(book,key=lambda x:x.get\_price(),reverse=False)

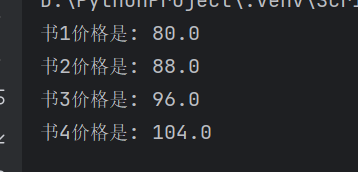
for book\_obj in book:

print(book\_obj.get\_title()+"价格是:",book\_obj.get\_price())

源代码截图：



运行结果截图：



1. 创建一个名为BankAccount的类，该类具有以下属性和方法:

1)属性：账户名（account\_name）、账户余额（balance）

2)方法：存款（deposit）、取款（withdraw）、打印账户信息（print\_info）

通过创建一个BankAccount类的实例，你可以输入一个账户名和初始余额，并使用deposit方法存款，使用withdraw方法取款，使用print\_info方法打印账户信息。

源代码：

class BankAccount:

account\_name=''

balance=0

def \_\_init\_\_(self,account\_name,balance):

self.\_\_account\_name=account\_name

self.\_\_balance=balance

def get\_account\_name(self):

return self.\_\_account\_name

def get\_balance(self):

return self.\_\_balance

def set\_balance(self,balance):

self.\_\_balance=balance

def set\_account\_name(self,account\_name):

self.\_\_account\_name=account\_name

def deposit(self,cash):

self.\_\_balance=self.\_\_balance+cash

def withdraw(self,cash):

if self.\_\_balance>cash:

self.\_\_balance=self.\_\_balance-cash

else:

print('你的账户余额不足')

def print\_info(self):

print("您好！您的账户名为："+self.\_\_account\_name+",账户余额为:",self.\_\_balance)

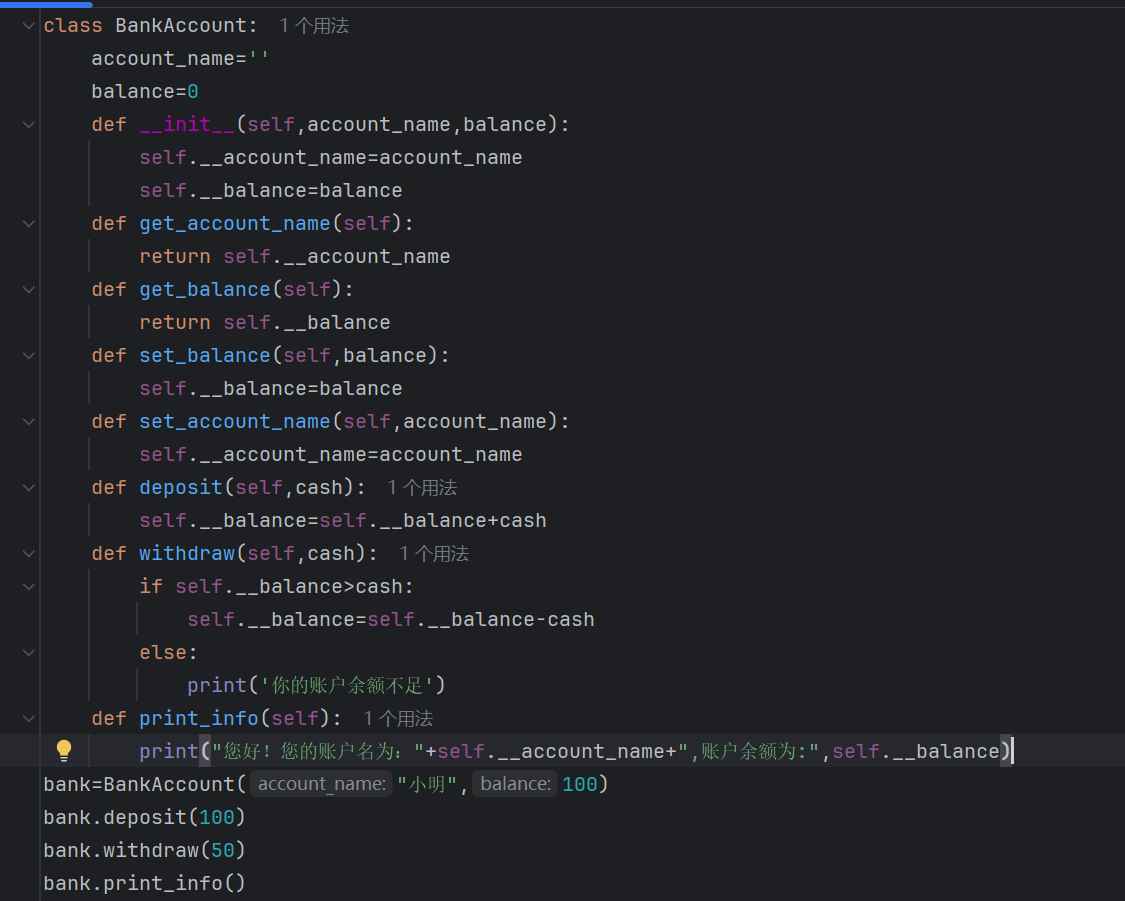
bank=BankAccount("小明",100)

bank.deposit(100)

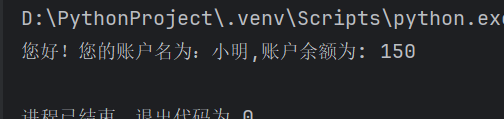
bank.withdraw(50)

bank.print\_info()

源代码截图：



运行结果截图：



1. 设计一个名为 MyRectangle 的矩形类来表示矩形。这个类包含：

1) 宽度和高度：width、height。

2) 构造方法：传入width，height。如果 width和 height 不传，则默认是 100。

3) 定义一个 getArea() 计算面积的方法。

4) 定义一个 getPerimeter()，计算周长的方法。

源代码：

class MyRectangle:

width = 0

height = 0

def \_\_init\_\_(self,width=100,height=100):

self.width = width

self.height = height

def get\_Area(self):

return self.width \* self.height

def get\_Perimeter(self):

return (self.width + self.height) \*2

r1 = MyRectangle(120, 130)

r2 = MyRectangle()

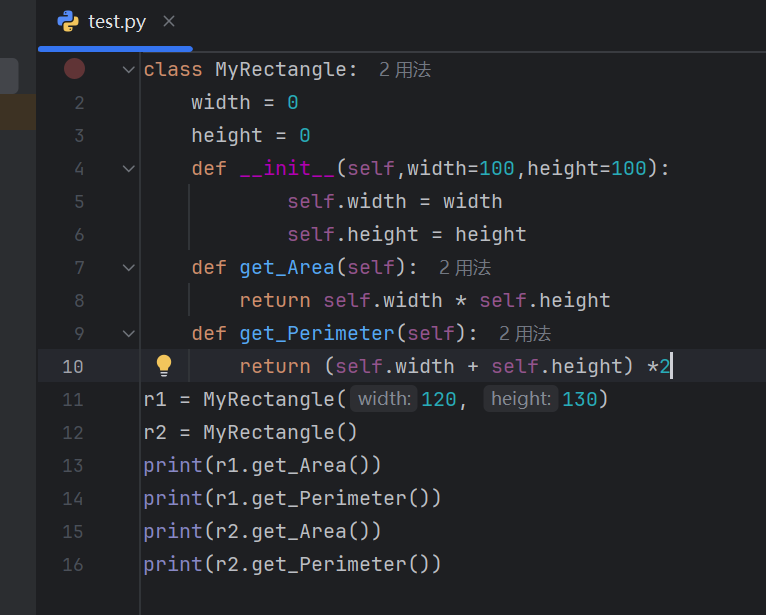
print(r1.get\_Area())

print(r1.get\_Perimeter())

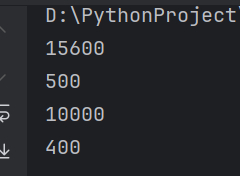
print(r2.get\_Area())

print(r2.get\_Perimeter())

源代码截图：



运行结果截图：

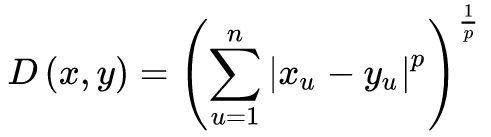


1. 定义一个高维空间样本点集类HDPoints，须包含以下数据属性与方法属性：

(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 random

class HDPoints:

points = []

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

self.points.append(points)

def centerpoint(self):

a=[0]\*1000

max\_len=0

list1=[]

for p in self.points:

if len(p)>max\_len:

max\_len = len(p)

for i in range(len(p)):

a[i]+=p[i]

for i in range(max\_len):

a[i]/=len(self.points)

list1.append(a[i])

return list1

def minkowski(self,x,y,p:int):

len\_max=len(x) if len(x)>len(y) else len(y)

list1=[]

sum=0

for i in range(len\_max):

list1.append(abs(x[i]-y[i]))

if p==0:

return max(list1)

else:

for l in list1:

sum+=l\*\*p

return sum\*\*(1/p)

def farthestpoint(self,p):

max=0

t=0

center=self.centerpoint()

for i in range(len(self.points)):

po=self.points[i]

if self.minkowski(po,center,p)>max:

max=self.minkowski(po,center,p)

t=i

return t,max

def farthest2point(self,p):

len\_p=len(self.points)

max=0

t1=t2=0

for i in range(len\_p):

for j in range(i+1,len\_p):

if self.minkowski(self.points[i],self.points[j],p)>max:

max=self.minkowski(self.points[i],self.points[j],p)

t1, t2 = i, j

return t1,t2,max

points = [[random.random() for i in range(6)] for \_ in range(50)] #维度6，50组

for p in points:

points\_handle=HDPoints(p)

p = random.randint(0, 5)

print("中心点:",points\_handle.centerpoint())

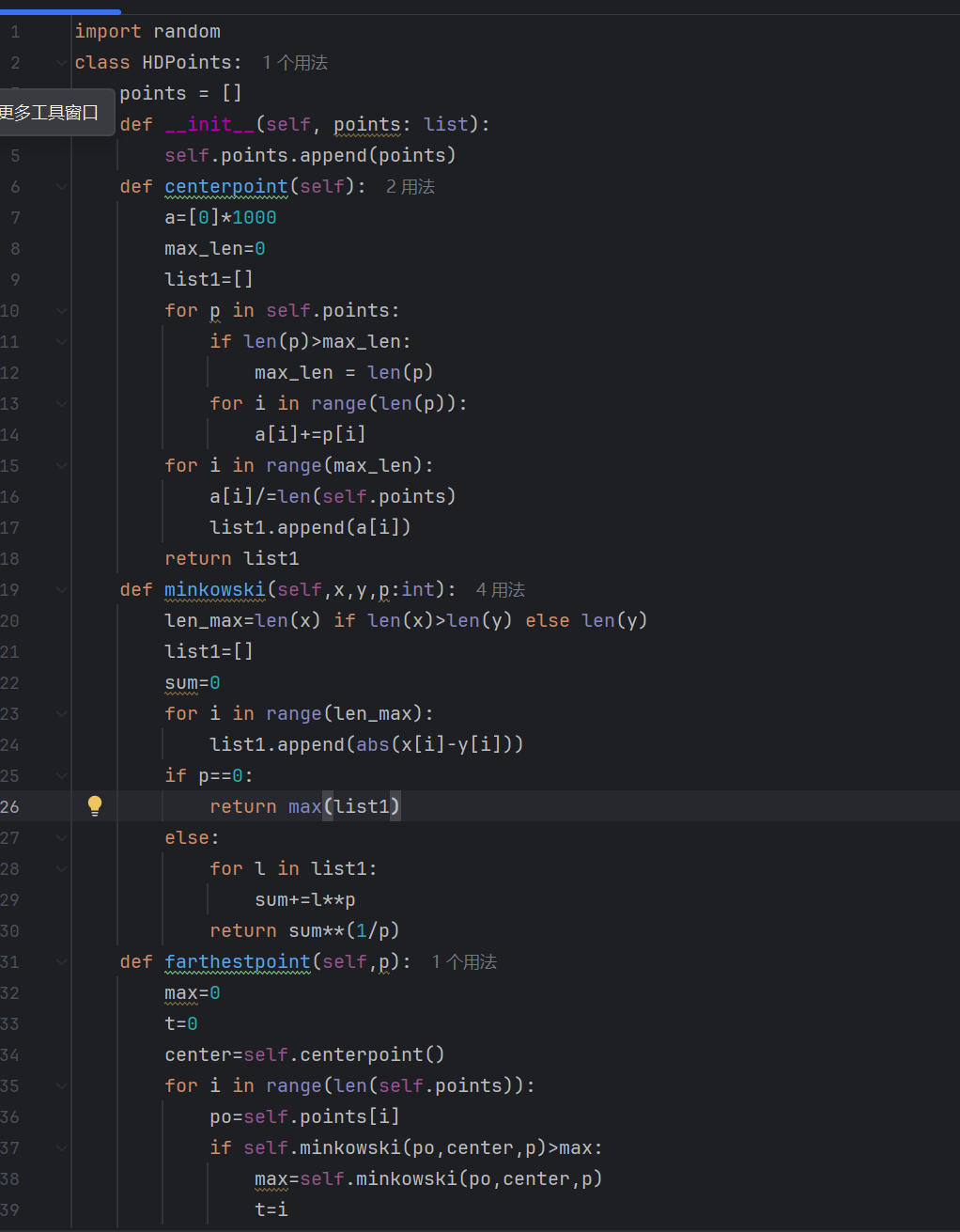
idx, dist = points\_handle.farthestpoint(p)

print(f"离中心最远的点索引: {idx}, 距离: {dist:.4f}")

i, j, d = points\_handle.farthest2point(p)

print(f"最远两点: ({i}, {j}), 距离: {d:.4f}")

源代码截图：





运行结果截图：

