实验4模块与面向对象编程

姓名：朱俊哲 学号：22121290220班级：计科2202 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. 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**)**:**

    txtt **=** [f **for** f **in** os**.**listdir(dirname) **if** f**.**endswith('.txt')]

**if** **not** txtt**:**

        new\_file\_path **=** os**.**path**.**join(dirname**,** "new.txt")

**with** open(new\_file\_path**,** 'w') **as** f**:**

            f**.**write(s)

            output **=** "s is written to new.txt"

**else:**

        filee **=** random**.**choice(txtt)

        filee\_path **=** os**.**path**.**join(dirname**,** filee)

**with** open(filee\_path**,** 'a') **as** f**:**

            f**.**write(s)

            output **=** f"s is written to {filee}"

**return** output

s **=** "I am the new one"

dirr **=** "D:/vscode/python/py\_lab/week4"

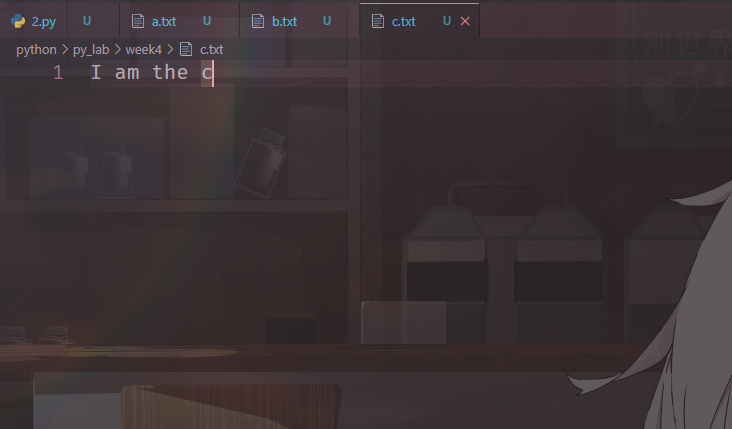
print(fs(dirr**,**s))

源代码截图：



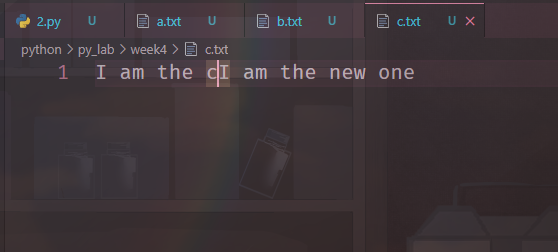
运行结果截图：

**存在txt**

执行前

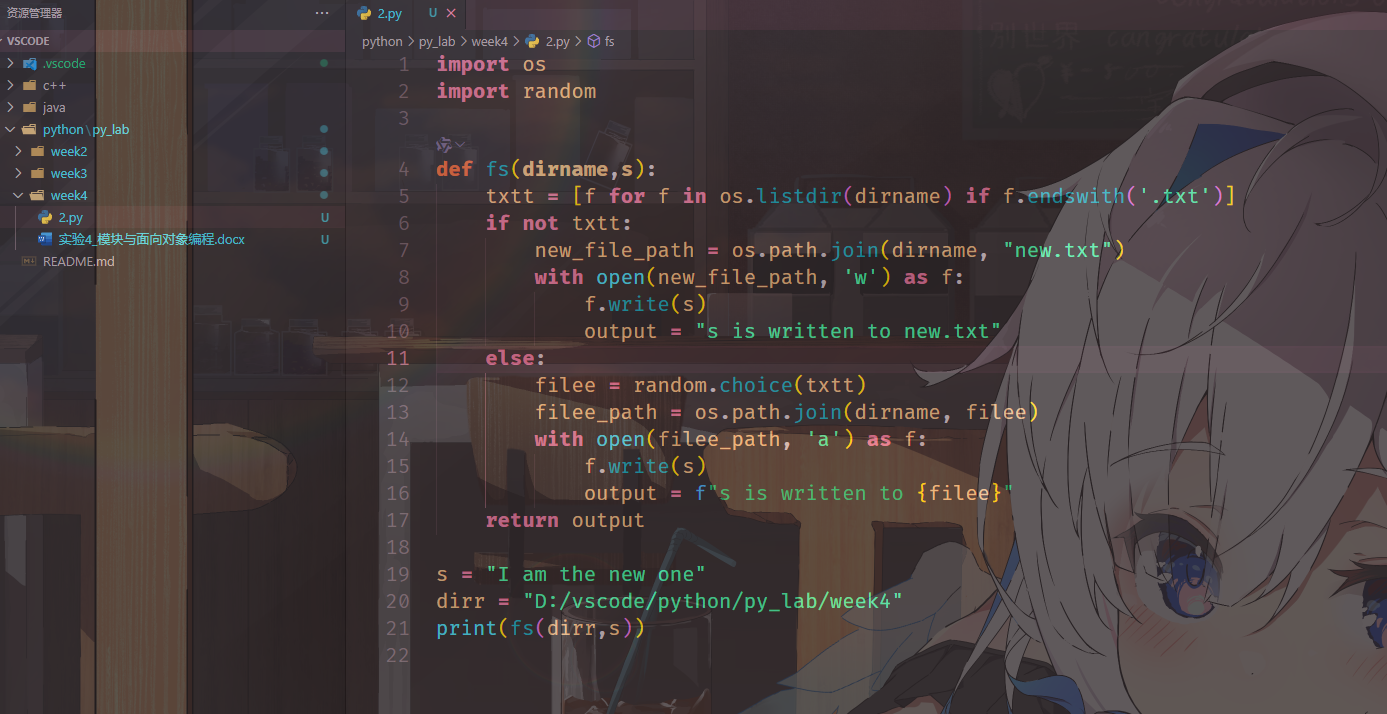
执行后



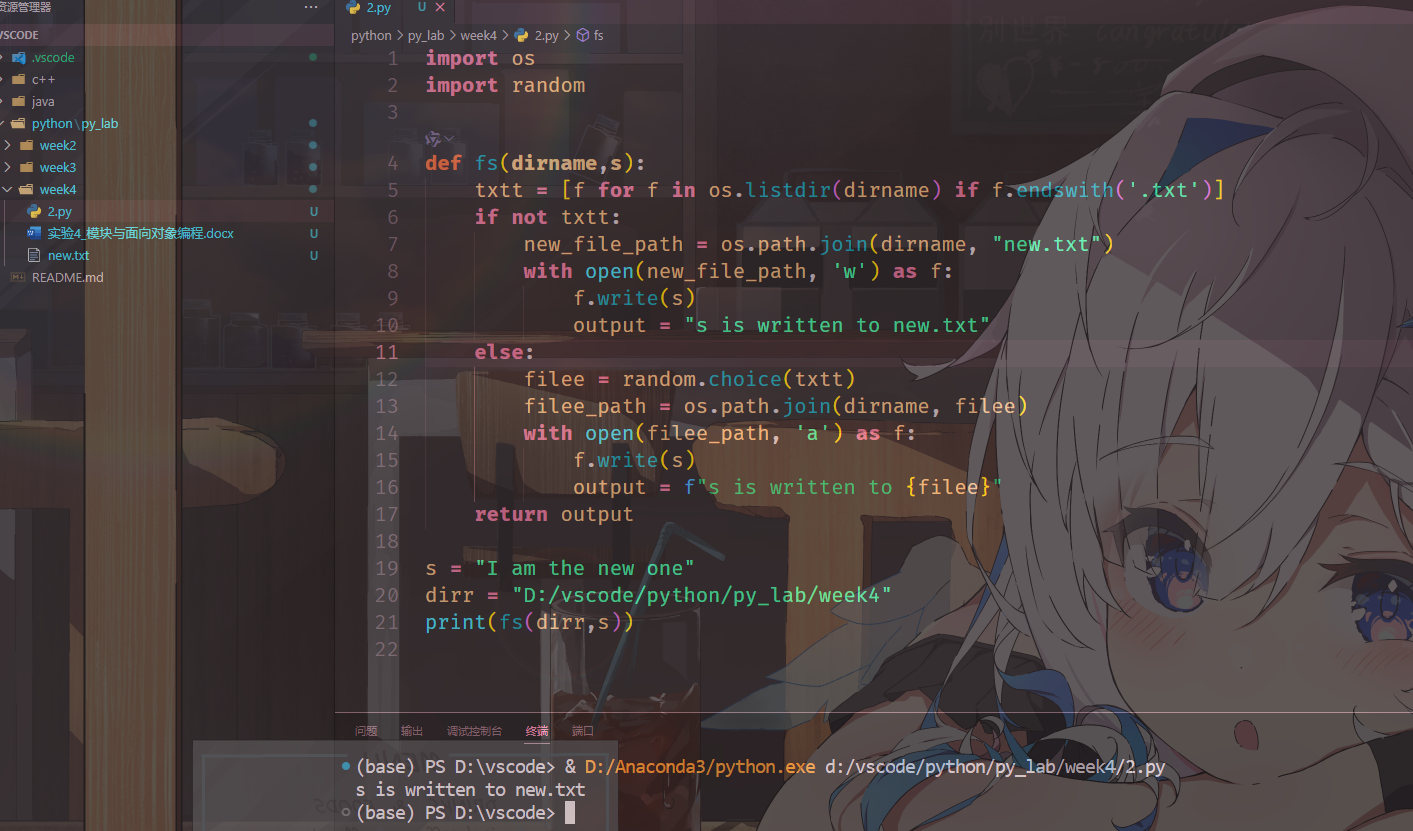


**不存在txt**

运行前



运行后



****

1. 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**:**

**def** \_\_init\_\_(*self***,** **init\_list**)**:**

*self***.**lst **=** init\_list

**def** add\_elem(*self***,** **elename**)**:**

**if** isinstance(elename**,** int) **or** isinstance(elename**,** str)**:**

*self***.**lst**.**append(elename)

**else:**

**raise** ValueError("Element must be an integer or a string")

**def** get\_elem(*self***,** **num**)**:**

**if** 0 **<=** num **<** len(*self***.**lst)**:**

**return** *self***.**lst[num]

**else:**

**raise** IndexError("List index out of range")

**def** merge\_list(*self***,** **ls**)**:**

**if** isinstance(ls**,** list)**:**

*self***.**lst**.**extend(ls)

**else:**

**raise** ValueError("Argument must be a list")

**def** del\_lastone(*self*)**:**

**if** *self***.**lst**:**

**return** *self***.**lst**.**pop()

**else:**

**raise** IndexError("Delete from empty list")

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

list\_info**.**add\_elem(777)

list\_info**.**add\_elem("new\_element")

*#list\_info.add\_elem(list\_info)*

print(list\_info**.**lst)

get\_element **=** list\_info**.**get\_elem(2)

print(get\_element)

merged\_list **=** [1**,** 2**,** 3]

list\_info**.**merge\_list(merged\_list)

*#list\_info.merge\_list("merged\_list")*

print(list\_info**.**lst)

deleted\_element **=** list\_info**.**del\_lastone()

print(list\_info**.**lst)

print(deleted\_element)

源代码截图：



运行结果截图：



1. 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**.**subpack **import** bb

**from** mypack **import** aa

**from** mypack **import** a**,** b

m **=** 4

n **=** 3

re1 **=** bb**.**sub(m**,** n)

re2 **=** aa**.**add(m**,** n)

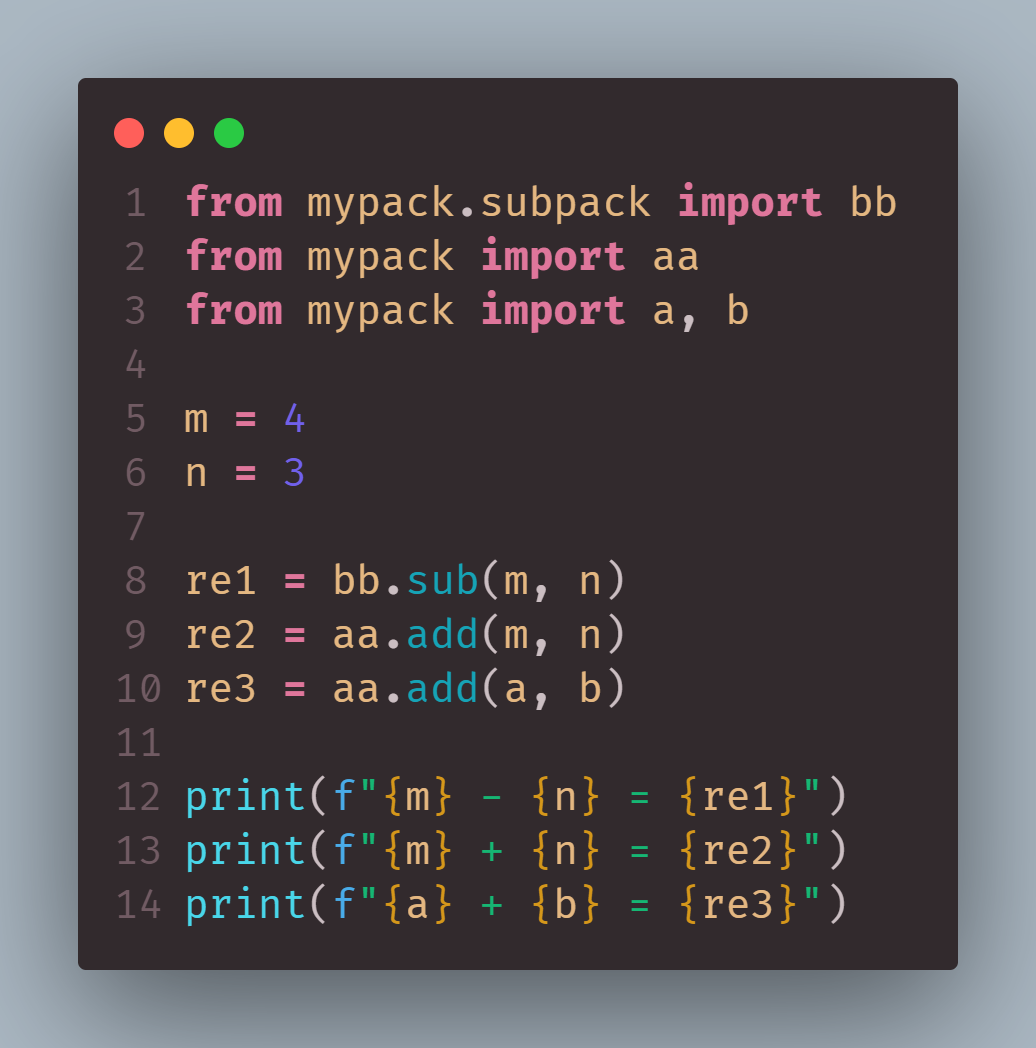
re3 **=** aa**.**add(a**,** b)

print(f"{m} - {n} = {re1}")

print(f"{m} + {n} = {re2}")

print(f"{a} + {b} = {re3}")

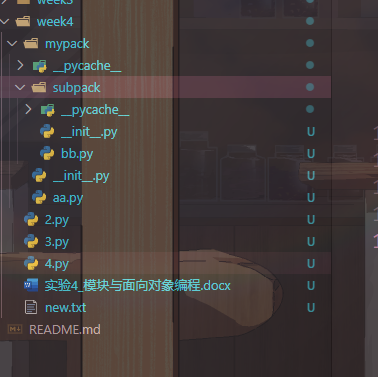
源代码(test.py)截图：



运行结果截图：



项目实际目录截图：



1. 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** do\_homework(*self*)**:**

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

class Student(*Human*)**:**

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

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

*self***.**homework **=** homework

**def** do\_homework(*self*)**:**

        super()**.**do\_homework()

        print("作业为: " **+** *self***.**homework)

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

print(f"stu's name is {stu**.**name}")

print(f"stu's age is {stu**.**age}")

print(f"stu's homework is {stu**.**homework}")

stu**.**do\_homework()

stu**.**get\_name()

源代码截图：



运行结果截图：



1. 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

class student(*person*)**:**

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

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

*self***.**\_\_grades **=** grades

**def** get\_avg\_grade(*self*)**:**

**return** sum(*self***.**\_\_grades) **/** len(*self***.**\_\_grades)

源代码(test.py)：

**from** mc **import** student

stu1 **=** student('Alice'**,** 22**,** 'Female'**,** [88**,** 92**,** 95])

stu2 **=** student('Bob'**,** 23**,** 'Male'**,** [75**,** 80])

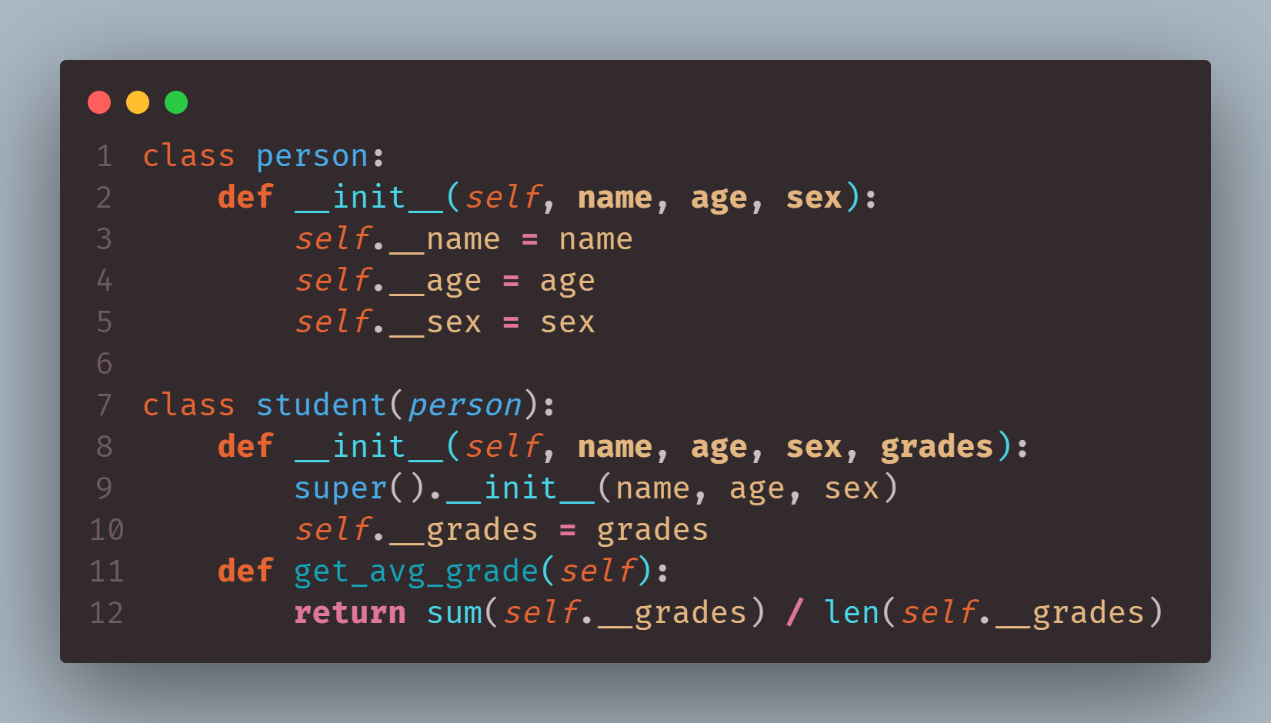
stu3 **=** student('Charlie'**,** 20**,** 'Male'**,** [70**,** 85**,** 90**,** 95])

**for** stu **in** (stu1**,** stu2**,** stu3)**:**

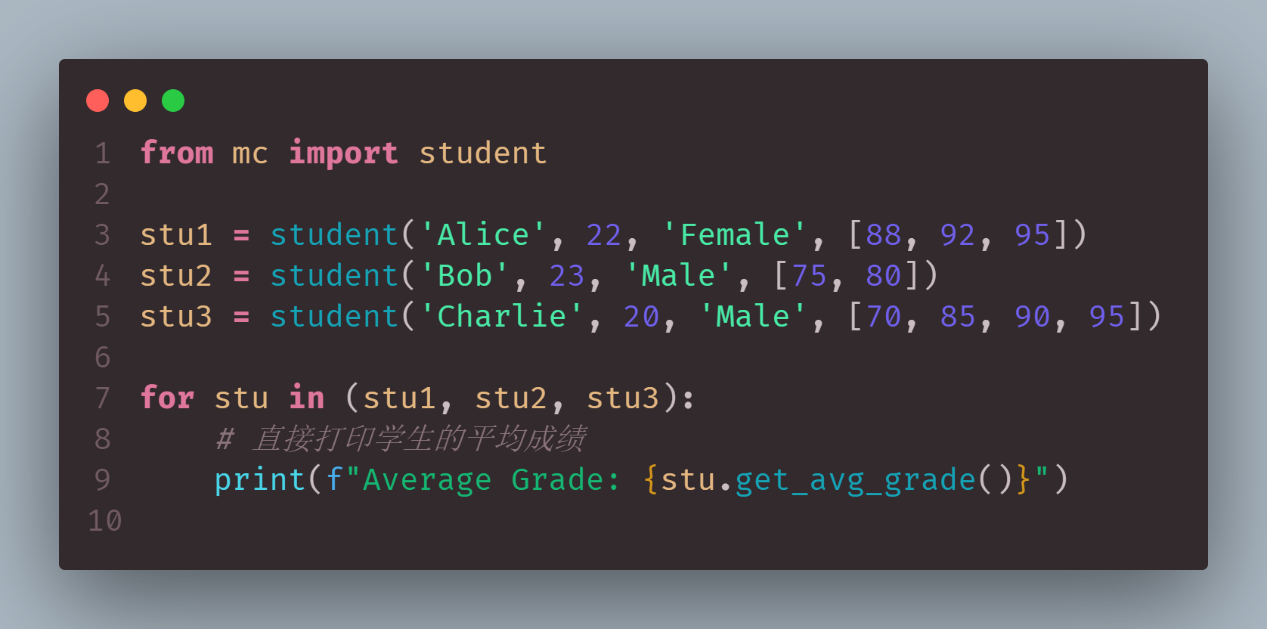
*# 直接打印学生的平均成绩*

    print(f"Average Grade: {stu**.**get\_avg\_grade()}")

源代码截图(mc.py)：



源代码截图(test.py)：



运行结果截图：

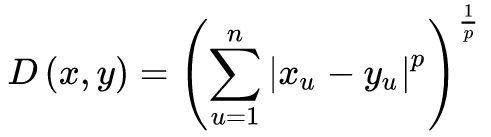


1. 定义一个高维空间样本点集类HDPoints，须包含以下数据属性与方法属性：

(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

**import** numpy **as** np

class HDPoints**:**

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

*self***.**points **=** points

**def** centerpoint(*self*)**:**

*# 计算每个维度的均值*

**return** [np**.**mean([point[dim] **for** point **in** *self***.**points]) **for** dim **in** range(len(*self***.**points[0]))]

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

**if** p **==** 0**:**  *# 切比雪夫距离*

**return** max(abs(a **-** b) **for** a**,** b **in** zip(x**,** y))

**else:**  *# p-闵氏距离*

**return** sum(abs(a **-** b) **\*\*** p **for** a**,** b **in** zip(x**,** y)) **\*\*** (1**/**p)

**def** farthestpoint(*self***,** **p**)**:**

        center **=** *self***.**centerpoint()

        max\_distance **=** 0

        farthest\_point\_index **=** 0

**for** i**,** point **in** enumerate(*self***.**points)**:**

            distance **=** *self***.**minkowski(center**,** point**,** p)

**if** distance **>** max\_distance**:**

                max\_distance **=** distance

                farthest\_point\_index **=** i

**return** farthest\_point\_index**,** max\_distance

**def** farthest2points(*self***,** **p**)**:**

        max\_distance **=** 0

        point1\_index**,** point2\_index **=** 0**,** 0

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

**for** j **in** range(i**+**1**,** len(*self***.**points))**:**

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

**if** distance **>** max\_distance**:**

                    max\_distance **=** distance

                    point1\_index**,** point2\_index **=** i**,** j

**return** (point1\_index**,** point2\_index**,** max\_distance)

*# 生成随机点*

num\_points **=** 50

dim **=** random**.**randint(5**,** 10)  *# 随机维度在5到10之间*

points **=** [[random**.**uniform(0**,** 1) **for** \_ **in** range(dim)] **for** \_ **in** range(num\_points)]

*# 实例化HDPoints类*

hd\_points **=** HDPoints(points)

*# 随机生成p*

p **=** random**.**randint(0**,** 5)

*# 测试方法*

center **=** hd\_points**.**centerpoint()

farthest\_point\_index**,** max\_distance **=** hd\_points**.**farthestpoint(p)

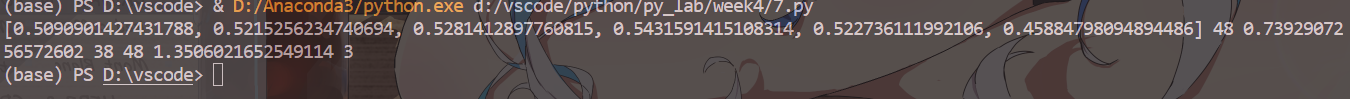
point1\_index**,** point2\_index**,** max\_2points\_distance **=** hd\_points**.**farthest2points(p)

print(center**,** farthest\_point\_index**,** max\_distance**,** point1\_index**,** point2\_index**,** max\_2points\_distance**,** p)

源代码截图：



运行结果截图：



1. 设计一个员工类(staff)，其中的数据成员有：salary代表员工工资，name代表员工姓名，为保护个人隐私，staff类的所有数据属性均需定义为私有属性。完成下面的功能：

1) 创建多个staff类实例（至少3个以上），初始化员工工资与姓名。

2) 给每个人涨100元工资。

3) 对涨后的工资进行排序。

4) 输出排序后的工资，以及姓名。

源代码：

class Staff**:**

**def** \_\_init\_\_(*self***,** **name,** **salary**)**:**

*self***.**\_\_name **=** name

*self***.**\_\_salary **=** salary

**def** get\_name(*self*)**:**

**return** *self***.**\_\_name

**def** get\_salary(*self*)**:**

**return** *self***.**\_\_salary

**def** increase\_salary(*self***,** **amount**)**:**

*self***.**\_\_salary **+=** amount

*# 创建Staff实例*

staff1 **=** Staff("Alice"**,** 3000)

staff2 **=** Staff("Bob"**,** 3500)

staff3 **=** Staff("Charlie"**,** 3200)

*# 每个员工工资增加100*

staff1**.**increase\_salary(100)

staff2**.**increase\_salary(100)

staff3**.**increase\_salary(100)

*# 对涨后的工资进行排序（降序）*

staff\_list **=** [staff1**,** staff2**,** staff3]

sorted\_staff\_list **=** sorted(staff\_list**,** **key=**lambda **x:** x**.**get\_salary()**,** **reverse=**True)

*# 输出排序后的工资及姓名*

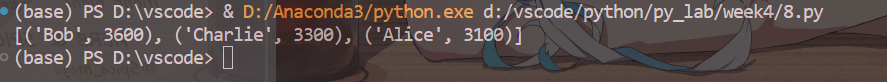
sorted\_salaries\_names **=** [(staff**.**get\_name()**,** staff**.**get\_salary()) **for** staff **in** sorted\_staff\_list]

print(sorted\_salaries\_names)

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**:**

**def** \_\_init\_\_(*self***,** **account\_name,** **balance=**0)**:**

*self***.**account\_name **=** account\_name

*self***.**balance **=** balance

**def** deposit(*self***,** **amount**)**:**

**if** amount **>** 0**:**

*self***.**balance **+=** amount

**else:**

            print("Please enter a positive amount to deposit.")

**def** withdraw(*self***,** **amount**)**:**

**if** 0 **<** amount **<=** *self***.**balance**:**

*self***.**balance **-=** amount

**elif** amount **<=** 0**:**

            print("Please enter a positive amount to withdraw.")

**else:**

            print("Insufficient balance.")

**def** print\_info(*self*)**:**

        print(f"Account Name: {*self***.**account\_name}")

        print(f"Balance: {*self***.**balance}")

*# 创建BankAccount实例*

account **=** BankAccount("John Doe"**,** 1000)

account**.**deposit(500)  *# 存款500*

account**.**withdraw(200)  *# 取款200*

account**.**print\_info()  *# 打印账户信息*

*# 返回最终的账户信息以确认*

print(account**.**account\_name**,** account**.**balance)

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**:**

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

*self***.**width **=** width

*self***.**height **=** height

**def** getArea(*self*)**:**

**return** *self***.**width **\*** *self***.**height

**def** getPerimeter(*self*)**:**

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

*# 创建几个MyRectangle实例来验证功能*

rect1 **=** MyRectangle()

rect2 **=** MyRectangle(50**,** 40)

*# 测试这些实例*

test\_results **=** {

    "rect1\_area"**:** rect1**.**getArea()**,**

    "rect1\_perimeter"**:** rect1**.**getPerimeter()**,**

    "rect2\_area"**:** rect2**.**getArea()**,**

    "rect2\_perimeter"**:** rect2**.**getPerimeter()**,**

}

print(test\_results)

源代码截图：



运行结果截图：

