实验七 Python面向对象编程 班级： 21计科1

学号： B20210302112

姓名： 张栋棋

Github地址：https://github.com/yourusername/pykc7

实验目的 学习Python类和继承的基础知识 学习namedtuple和DataClass的使用 实验环境 Git Python 3.10 VSCode VSCode插件 实验内容和步骤 第一部分 Python面向对象编程

完成教材《Python编程从入门到实践》下列章节的练习：

第9章 类 第二部分 在Codewars网站注册账号，完成下列Kata挑战：

第一题：面向对象的海盗 难度： 8kyu

啊哈，伙计!

你是一个小海盗团的首领。而且你有一个计划。在OOP的帮助下，你希望建立一个相当有效的系统来识别船上有大量战利品的船只。 对你来说，不幸的是，现在的人很重，那么你怎么知道一艘船上装的是黄金而不是人呢?

你首先要写一个通用的船舶类。

class Ship: def **init**(self, draft, crew): self.draft = draft self.crew = crew 每当你的间谍看到一艘新船进入码头，他们将根据观察结果创建一个新的船舶对象。

draft吃水 - 根据船在水中的高度来估计它的重量 crew船员 - 船上船员的数量 Titanic = Ship(15, 10)

任务

你可以访问船舶的 "draft(吃水) "和 "crew(船员)"。"draft(吃水) "是船的总重量，"船员 "是船上的人数。 每个船员都会给船的吃水增加1.5个单位。如果除去船员的重量后，吃水仍然超过20，那么这艘船就值得掠夺。任何有这么重的船一定有很多战利品! 添加方法 is_worth_it 来决定这艘船是否值得掠夺。

例如：

Titanic.is_worth_it() False 祝你好运，愿你能找到金子!

代码： class Ship:

```
def __init__(self, draft, crew):
    self.draft = draft
    self.crew = crew

def is_worth_it(self):
    return self.draft - self.crew * 1.5 > 20
```

Titanic = Ship(15, 10) print(Titanic.is_worth_it())

treasure_ship = Ship(35.1, 10) print(treasure_ship.is_worth_it())

第二题： 搭建积木 难度：7kyu

写一个创建Block的类（Duh.） 构造函数应该接受一个数组作为参数，这个数组将包含3个整数，其形式为[width, length, height]，Block应该由这些整数创建。

定义这些方法:

get_width() return the width of the Block get_length() return the length of the Block get_height() return the height of the Block get_volume() return the volume of the Block get_surface_area() return the surface area of the Block 例子：

b = Block([2,4,6]) # create a `Block` object with a width of `2` a length of `4` and a height of `6` b.get_width() # return 2
b.get_length() # return 4 b.get_height() # return 6 b.get_volume() # return 48 b.get_surface_area() # return 88
注意： 不需要检查错误的参数。

代码： class Block:

```
def __init__(self, args):
    self.width = args[0]
    self.length = args[1]
    self.height = args[2]

def get_width(self):
    return self.width

def get_length(self):
    return self.length

def get_height(self):
    return self.height

def get_volume(self):
    return self.width * self.length * self.height

def get_surface_area(self):
    return 2 * (self.width * self.length + self.width * self.height + self.length
* self.height)
```

第三题： 分页助手 难度：5kyu

在这个练习中，你将加强对分页的掌握。你将完成PaginationHelper类，这是一个实用类，有助于查询与数组有关的分页信息。 该类被设计成接收一个值的数组和一个整数，表示每页允许多少个项目。集合/数组中包含的值的类型并不相关。

下面是一些关于如何使用这个类的例子：

helper = PaginationHelper(['a','b','c','d','e','f'], 4) helper.page_count() # should == 2 helper.item_count() # should == 6 helper.page_item_count(0) # should == 4 helper.page_item_count(1) # last page - should == 2 helper.page_item_count(2) # should == -1 since the page is invalid

page_index takes an item index and returns the page that it belongs on

helper.page_index(5) # should == 1 (zero based index) helper.page_index(2) # should == 0 helper.page_index(20) # should == -1 helper.page_index(-10) # should == -1 because negative indexes are invalid 代码： import math

class PaginationHelper:

```
def __init__(self, collection, items_per_page):
    self.collection = collection
    self.items_per_page = items_per_page

def item_count(self):
    return len(self.collection)

# 总页数
def page_count(self):

    # 总条目数 / 每页条目数，然后向上取整
    return math.ceil(self.item_count() / self.items_per_page)

def page_item_count(self, page_index):

    # 页数为负数或者页数超过总页数
    if page_index < 0 or page_index >= self.page_count():
        return -1

    # 最后一页
    elif page_index == self.page_count() - 1:

        # 如果是6%4，那么最后一页就是2
        # 如果是8%4，那么最后一页就是0，说明最后一页是满的，应该返回4
        last_page = self.item_count() % self.items_per_page

        return self.items_per_page if last_page == 0 else last_page

    # 其他页
    else:
        return self.items_per_page

def page_index(self, item_index):
    # 非法的情况
    if item_index < 0 or item_index >= self.item_count():
```

```
            return -1
        else:
            return item_index // self.items_per_page
```

helper = PaginationHelper(['a','b','c','d'], 4) print(helper.page_count()) print(helper.page_item_count(0)) print(helper.page_index(4))

第四题： 向量（Vector）类 难度： 5kyu

创建一个支持加法、减法、点积和向量长度的向量（Vector）类。

举例来说：

a = Vector([1, 2, 3]) b = Vector([3, 4, 5]) c = Vector([5, 6, 7, 8])

a.add(b) # should return a new Vector([4, 6, 8]) a.subtract(b) # should return a new Vector([-2, -2, -2]) a.dot(b) # should return 1*3* + *2*4 + 3*5 = 26 a.norm() # should return sqrt(1^2 + 2^2 + 3^2) = sqrt(14) a.add(c) # raises an exception 如果你试图对两个不同长度的向量进行加减或点缀，你必须抛出一个错误。 向量类还应该提供：

一个 **str** 方法，这样 str(a) === '(1,2,3)' 一个equals方法，用来检查两个具有相同成分的向量是否相等。 注意：测试案例将利用用户提供的equals方法。

代码提交地址： https://www.codewars.com/kata/526dad7f8c0eb5c4640000a4

第五题： Codewars风格的等级系统 难度： 4kyu

编写一个名为User的类，用于计算用户在类似于Codewars使用的排名系统中的进步量。

业务规则：

一个用户从等级-8开始，可以一直进步到8。 没有0（零）等级。在-1之后的下一个等级是1。 用户将完成活动。这些活动也有等级。 每当用户完成一个有等级的活动，用户的等级进度就会根据活动的等级进行更新。 完成活动获得的进度是相对于用户当前的等级与活动的等级而言的。 用户的等级进度从零开始，每当进度达到100时，用户的等级就会升级到下一个等级。 在上一等级时获得的任何剩余进度都将被应用于下一等级的进度（我们不会丢弃任何进度）。例外的情况是，如果没有其他等级的进展（一旦你达到8级，就没有更多的进展了）。 一个用户不能超过8级。 唯一可接受的等级值范围是-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8。任何其他的值都应该引起错误。 逻辑案例：

如果一个排名为-8的用户完成了一个排名为-7的活动，他们将获得10的进度。 如果一个排名为-8的用户完成了排名为-6的活动，他们将获得40的进展。 如果一个排名为-8的用户完成了排名为-5的活动，他们将获得90的进展。 如果一个排名-8的用户完成了排名-4的活动，他们将获得160个进度，从而使该用户升级到排名-7，并获得60个进度以获得下一个排名。 如果一个等级为-1的用户完成了一个等级为1的活动，他们将获得10个进度（记住，零等级会被忽略）。 代码案例：

user = User() user.rank # => -8 user.progress # => 0 user.inc_progress(-7) user.progress # => 10 user.inc_progress(-5) # will add 90 progress user.progress # => 0 # progress is now zero user.rank # => -7 # rank was upgraded to -7 代码提交地址： https://www.codewars.com/kata/51fda2d95d6efda45e00004e

第三部分 使用Mermaid绘制程序的类图

安装VSCode插件：

Markdown Preview Mermaid Support Mermaid Markdown Syntax Highlighting 使用Markdown语法绘制你的程序绘制程序类图（至少一个），Markdown代码如下：

程序类图

显示效果如下：

查看Mermaid类图的语法-->点击这里

使用Markdown编辑器（例如VScode）编写本次实验的实验报告，包括实验过程与结果、实验考查和实验总结，并将其导出为 PDF格式 来提交。

实验过程与结果 请将实验过程与结果放在这里，包括：

第一部分 Python面向对象编程 第二部分 Codewars Kata挑战 第三部分 使用Mermaid绘制程序流程图 注意代码需要使用markdown的代码块格式化，例如Git命令行语句应该使用下面的格式：

Git命令

显示效果如下：

git init git add . git status git commit -m "first commit" 如果是Python代码，应该使用下面代码块格式，例如：

Python代码

显示效果如下：

def add_binary(a,b): return bin(a+b)[2:] 代码运行结果的文本可以直接粘贴在这里。

注意：不要使用截图，Markdown文档转换为Pdf格式后，截图可能会无法显示。

实验考查 请使用自己的语言并使用尽量简短代码示例回答下面的问题，这些问题将在实验检查时用于提问和答辩以及实际的操作。

Python的类中__init__方法起什么作用？ 当创建一个类的实例时，Python会自动调用该类的__init__()方法，并将该对象作为第一个参数传递给它。在__init__()方法中，你可以设定对象的属性和执行一些必要的操作以使对象正确地初始化。通常情况下，在类中定义成员变量时，都会在__init__()方法中初始化这些属性值。 **init__方法在对象创建时自动调用，无需手动显式调用。在Python中还有另一种特殊方法__new**()，它负责创建对象并返回一个对象的引用。new()方法通常不需要手动实现，因为Python会自动调用它来创建对象。init()方法则负责初始化对象的状态。而new()方法通常用于实现自定义的对象创建逻辑，并且返回一个新的实例对象。

Python语言中如何继承父类和改写（override）父类的方法。

创建一个父类

class Base_father: def **init**(self,name,age): self.name = name self.age = age print('调用了父类的name')

创建子类

class Base_son(Base_father):

```
def getname(self):
    print(f'姓名：{self.name}')
    print(f'年龄：{self.age}')
    return '运行完毕！！'
```

num1 = Base_son('suliang',21) print(num1.getname())

创建一个父类

class Base_father: def **init**(self,name,age): self.name = name self.age = age print('调用了父类的name')

创建子类

class Base_son(Base_father): def **init**(self,name,age): #利用super调用父类的构造函数 super(Base_son, self).**init**(name ,age) print('-'*50) self.name = name self.age = age print('调用了我自己定义的方法！！') def getname(self): print(f'姓名：{self.name}') print(f'年龄：{self.age}') return '运行完毕！！' num1 = Base_son('suliang',21) print(num1.getname())

Python类有那些特殊的方法？它们的作用是什么？请举三个例子并编写简单的代码说明。

coding:utf-8

class A(object): date = '20201215' # 类变量

```
def __init__(self):
    self.name = 'Stephen'       # 实例变量

def __str__(self):
    return self.date + ' ' + self.name
```

new = A() print(new) print(new.date) print(new.name)

class Date: day = 10 # 类变量

```
def __init__(self, year, month, day):        # 构造函数
    self.year = year
    self.month = month
    self.day = day

def __str__(self):
    return ("{year}-{month}-{day}").format(year=self.year, month=self.month,
```

```
    day=self.day)

    def yesterday(Date):
        Date.day -= 1

    def print_year(self, number):    # 实例方法
        print(self.year + number)

    @property
    def weather(self):        # 属性方法
        return 'rainny'

    @staticmethod     # 静态方法：校验输入值类型和大小是否符合咱们的类型。
    def var_str(date_str):
        year, month, day = tuple(date_str.split("-"))
        if 0 < int(year) and 0 < int(month) <= 12 and 0 < int(day) < 32:
            return True
        else:
            return False

    @classmethod      # 类方法
    def class_method(cls, date_str):

        if cls.var_str(date_str):
            year, month, day = tuple(date_str.split("-"))
            return cls(int(year), int(month), int(day))
```

new_day = Date.class_method("2018-10-10") print(new_day)

打印结果

2018 - 10 - 10

class Person(): #定义类 def info(self): #实例方法 self.name='xiaoming' print("shi li fang fa") def **del**(self): #析构函数，类销毁自动执行 print("类销毁的时候就执行")

zwj=Person() zwj.info() del zwj #销毁类 print("========")

shi li fang fa 类销毁的时候就执行

实验总结 在这次实验中我学到了Python中一些继承和改写父类的方法，还有_init_方法的用法，收获很多，相信通过自己的努力学习能够学到更多的有关python的知识。