实验七 Python面向对象编程

班级： 21计科1

学号： B20210302131

姓名：李佳琪

Github地址：https://github.com/Seven116

CodeWars地址：https://www.codewars.com/users/Seven116

实验目的

1. 学习Python类和继承的基础知识
2. 学习namedtuple和DataClass的使用

实验环境

1. Git
2. Python 3.10
3. VSCode
4. 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!

代码提交地址：
https://www.codewars.com/kata/54fe05c4762e2e3047000add

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。

代码提交地址:
https://www.codewars.com/kata/55b75fcf67e558d3750000a3

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
```

代码提交地址：
https://www.codewars.com/kata/515bb423de843ea99400000a

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：

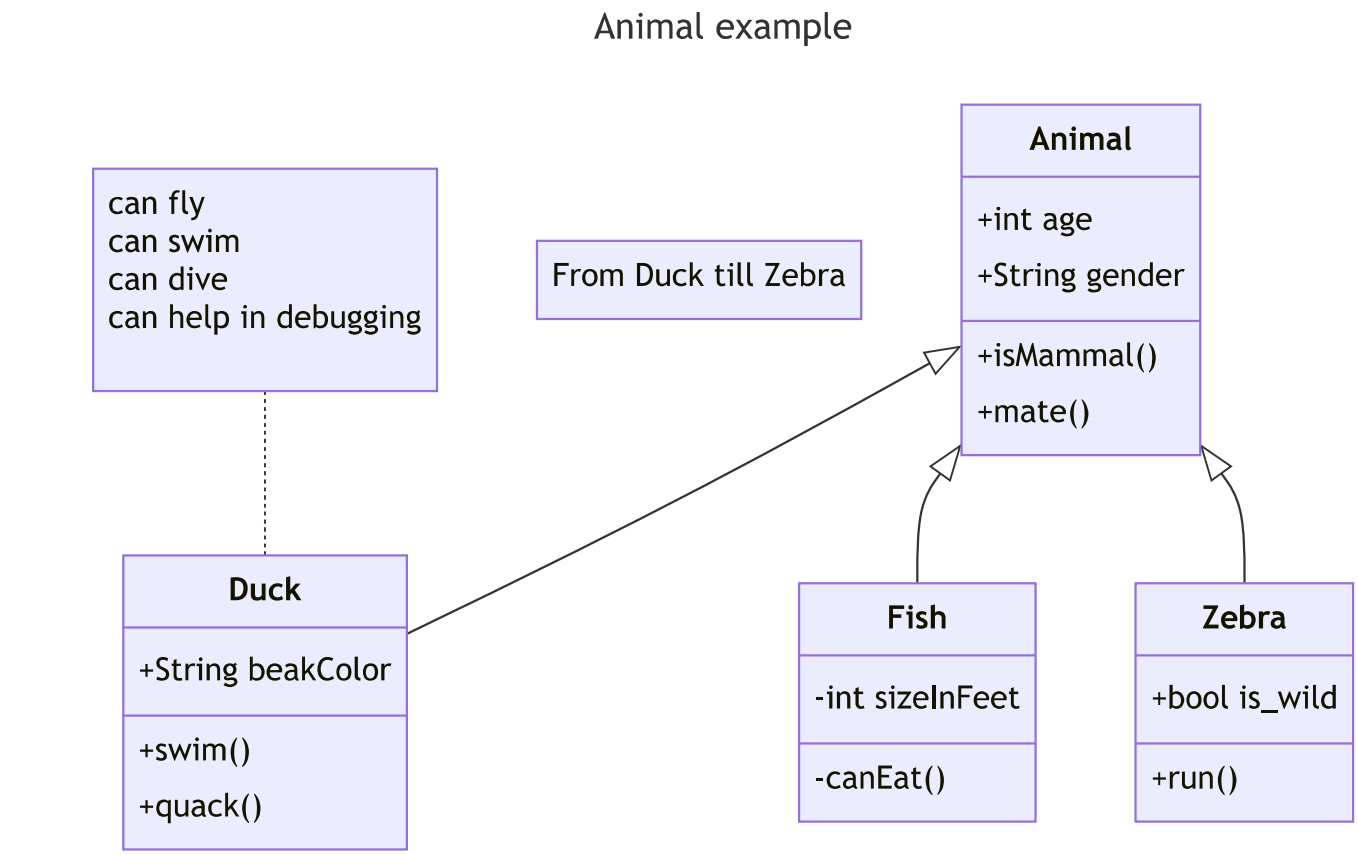
- Markdown Preview Mermaid Support

- 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：

```
---
title: Animal example
---
classDiagram
    note "From Duck till Zebra"
    Animal <|-- Duck
    note for Duck "can fly\ncan swim\ncan dive\ncan help in debugging"
    Animal <|-- Fish
    Animal <|-- Zebra
    Animal : +int age
    Animal : +String gender
    Animal: +isMammal()
    Animal: +mate()
    class Duck{
        +String beakColor
        +swim()
        +quack()
    }
    class Fish{
        -int sizeInFeet
        -canEat()
    }
    class Zebra{
        +bool is_wild
        +run()
    }
```

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

9-1 餐馆：创建一个名为Restaurant的类，其方法__init__()设置两个属性：restaurant_name和cuisine_type。创建一个名为describe_restaurant()的方法和一个名为open_restaurant()的方法，其中前者打印前述两项信息，而后者打印一条消息，指出餐馆正在营业。

```
class Restaurant():

  def __init__(self,restaurant_name, cuisine_type):
      self.restaurant_name = restaurant_name
      self.cuisine_type = cuisine_type

  def describe_restaurant(self):
      print(f"restaurant name is {self.restaurant_name}")
      print(f"cuisine type is {self.cuisine_type}")

  def open_restaurant(self):
      print("Open")

```

```
restaurant = Restaurant("chuancai", "hot pot")

restaurant.describe_restaurant()
restaurant.open_restaurant()
```

9-3 用户：创建一个名为User的类，其中包含属性first_name和last_name，还有用户简介通常会存储的其他几个属性。在类User中定义一个名为describe_user()的方法，它打印用户信息摘要；再定义一个名为greet_user()的方法，它向用户发出个性化的问候。

创建多个表示不同用户的实例，并对每个实例都调用上述两个方法。

```
class User():
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def describe_user(self):
        print(f"username is: " + self.first_name + " "+ self.last_name)

    def greet_user(self):
        print("hello, {}! ".format(self.last_name))

user_1 = User("John","Smith")
user_1.describe_user()
user_1.greet_user()

user_2 =User("xiao", "ming")
user_2.describe_user()
user_2.greet_user()
```

9-4 就餐人数：在为完成练习9-1而编写的程序中，添加一个名为number_served的属性，并将其默认值设置为0。根据这个类创建一个名为restaurant的实例；打印有多少人在这家餐馆就餐过，然后修改这个值并再次打印它。

添加一个名为set_number_served()的方法，它让你能够设置就餐人数。调用这个方法并向它传递一个值，然后再次打印这个值。

添加一个名为increment_number_served()的方法，它让你能够将就餐人数递增。调用这个方法并向它传递一个这样的值：你认为这家餐馆每天可能接待的就餐人数。

```
class Restaurant():

    def __init__(self,restaurant_name, cuisine_type):
        self.restaurant_name = restaurant_name
        self.cuisine_type = cuisine_type
        self.number_served = 0

    def describe_restaurant(self):
```

```
        print(f"restaurant name is {self.restaurant_name}")
        print(f"cuisine type is {self.cuisine_type}")

    def open_restaurant(self):
        print("Open")

    def set_number_served(self, number):
        self.number_served = number
        print(f"{self.number_served} person has served!")

    def increment_number_served(self, numbers):
        self.number_served += numbers
        print(f"{self.number_served} person has served!")

restaurant = Restaurant("chuancai", "hot pot")
restaurant.set_number_served(10)
restaurant.increment_number_served(20)
restaurant.increment_number_served(30)
```

9-5 尝试登录次数：在为完成练习9-3而编写的User类中，添加一个名为login_attempts的属性。编写一个名为increment_login_attempts()的方法，它将属性login_attempts的值加1。再编写一个名为reset_login_attempts()的方法，它将属性login_attempts的值重置为0。

根据User类创建一个实例，再调用方法increment_login_attempts()多次。打印属性login_attempts的值，确认它被正确地递增；然后，调用方法reset_login_attempts()，并再次打印属性login_attempts的值，确认它被重置为0。

```
class User():
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name
        self.login_attempts = 0

    def increment_login_attempts(self):
        self.login_attempts += 1

    def reset_login_attempts(self):
        self.login_attempts = 0

user_1 = User("John","Smith")
user_1.increment_login_attempts()
user_1.increment_login_attempts()
user_1.increment_login_attempts()
print(user_1.login_attempts)

user_1.reset_login_attempts()
print(user_1.login_attempts)
```

9-6 冰淇淋小店：冰淇淋小店是一种特殊的餐馆。编写一个名为IceCreamStand的类，让它继承你为完成练习9-1或练习9-4而编写的Restaurant类。这两个版本的Restaurant类都可以，挑选你更喜欢的那个即可。添加一个名为flavors的属性，用于存储一个由各种口味的冰淇淋组成的列表。编写一个显示这些冰淇淋的方法。创建一个IceCreamStand实例，并调用这个方法。

```
class Restaurant():

    def __init__(self,restaurant_name, cuisine_type):
        self.restaurant_name = restaurant_name
        self.cuisine_type = cuisine_type

    def describe_restaurant(self):
        print(f"restaurant name is {self.restaurant_name}")
        print(f"cuisine type is {self.cuisine_type}")

    def open_restaurant(self):
        print("Open")

class IceCreamStand(Restaurant):
    def __init__(self, restaurant_name, cuisine_type):
        super().__init__(restaurant_name, cuisine_type)
        self.flavors = []

    def show_icecream(self):
        for flavor in self.flavors:
            print(flavor)

icecream = IceCreamStand("haagen-Dazs", "icecream")
icecream.flavors = ["a", "b", "c"]

icecream.show_icecream()
```

9-7 管理员：管理员是一种特殊的用户。编写一个名为Admin的类，让它继承你为完成练习9-3或练习9-5而编写的User 类。添加一个名为privileges的属性，用于存储一个由字符串（如"can add post"、"can delete post"、"can ban user"等）组成的列表。编写一个名为show_privileges()的方法，它显示管理员的权限。创建一个Admin实例，并调用这个方法。

```
class User():
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def describe_user(self):
        print(f"username is: " + self.first_name + " "+ self.last_name)

    def greet_user(self):
        print("hello, {}! ".format(self.last_name))

class Admin(User):
```

```
    def __init__(self, first_name, last_name):
        super().__init__(first_name, last_name)
        self.privileges =["can add post", "can delete post", "can ban user"]

    def show_privileges(self):
        for privilege in self.privileges:
            print(privilege)

admin = Admin("John", "Smith")

admin.show_privileges()
```

9-8 权限：编写一个名为Privileges的类，它只有一个属性——privileges，其中存储了练习9-7所说的字符串列表。将方法show_privileges()移到这个类中。在Admin类中，将一个Privileges实例用作其属性。创建一个Admin实例，并使用方法show_privileges()来显示其权限。

```
class User():
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def describe_user(self):
        print(f"username is: " + self.first_name + " "+ self.last_name)

    def greet_user(self):
        print("hello, {}! ".format(self.last_name))


class Privileges():
    def __init__(self):
        self.privileges =["can add post", "can delete post", "can ban user"]

    def show_privileges(self):
        for privilege in self.privileges:
            print(privilege)

class Admin(User):
    def __init__(self, first_name, last_name):
        super().__init__(first_name, last_name)
        self.privilege = Privileges()

    def show_privileges(self):
        self.privilege.show_privileges()

admin = Admin("John", "Smith")

admin.show_privileges()
```

9-9 电瓶升级：在本节最后一个electric_car.py版本中，给Battery类添加一个名为upgrade_battery()的方法。这个方法检查电瓶容量，如果它不是85，就将它设置为85。创建一辆电瓶容量为默认值的电动汽车，调用方法get_range()，然后对电瓶进行升级，并再次调用get_range()。你会看到这辆汽车的续航里程增加了。

```
class Car():
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        long_name = str(self.year) + ' ' + self.make + ' ' + self.model
        return long_name.title()

class Battery():
    """一次模拟电动汽车电瓶的简单尝试"""

    def __init__(self, battery_size=60):
        """初始化电瓶的属性"""
        self.battery_size = battery_size

    def describe_battery(self):
        """打印一条描述电瓶容量的消息"""
        print("This car has a " + str(self.battery_size) + "-kWh battery.")

    def get_range(self):
        """Print a statement about the range this battery provides."""
        if self.battery_size == 60:
            range1 = 140
        elif self.battery_size == 85:
            range1 = 185

        message = "This car can go approximately " + str(range1)
        message += " miles on a full charge."
        print(message)

    def upgrade_battery(self):
        self.battery_size = 85

class ElectricCar(Car):

    def __init__(self, manufacturer, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(manufacturer, model, year)
        self.battery = Battery()

```

```
47
48 my_tesla = ElectricCar('tesla', 'model s', 2016)
49 my_tesla.battery.get_range()
50 my_tesla.battery.upgrade_battery()
51 my_tesla.battery.get_range()
```

9-13 使用OrderedDict：在练习6-4中，你使用了一个标准字典来表示词汇表。请使用OrderedDict类来重写这个程序，并确认输出的顺序与你在字典中添加键—值对的顺序一致。

```
1  from collections import OrderedDict
2
3  dicts = {'list': '列表', 'str': '字符串', 'tuple': '元组', 'dict': '字典', 'int': '整型'
   }
4
5  dicts['split'] = '切片'
6  dicts['if'] =  '条件'
7  dicts['class'] = '类'
8  dicts['object'] = '对象'
9  dicts['boolean'] = '布尔'
10
11 order_dicts = OrderedDict()
12 for key, value in dicts.items():
13     order_dicts[key] = value
14
15 for key, value in order_dicts.items():
16     print(f"{key} : {value}")
```

9-14 骰子：模块random包含以各种方式生成随机数的函数，其中的randint()返回一个位于指定范围内的整数，例如，下面的代码返回一个1~6内的整数：

```
1 from random import randint
2 x = randint(1, 6)
```

请创建一个Die类，它包含一个名为sides的属性，该属性的默认值为6。编写一个名为roll_die()的方法，它打印位于1和骰子面数之间的随机数。创建一个6面的骰子，再掷10次。 创建一个10面的骰子和一个20面的骰子，并将它们都掷10次。

```
1  from random import randint
2
3  class Die():
4      def __init__(self, sides=6):
5          self.sides = sides
6
7      def roll_die(self):
8          x = randint(1, self.sides)
9          print(x, end=" ")
10
```

```
print("\n-----------6------------")
dice_6 = Die()

i = 0
while i < 10:
    dice_6.roll_die()
    i = i + 1

print("\n-----------10------------")
dice_10 = Die(10)

i = 0
while i < 10:
    dice_6.roll_die()
    i = i + 1

    dice_10 = Die(10)

print("\n-----------20------------")
dice_6 = Die(20)
i = 0
while i < 10:
    dice_6.roll_die()
    i = i + 1
```

- 第二部分 Codewars Kata挑战

第一题：面向对象的海盗

```
  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

```
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

```
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())
```

第四题： 向量（Vector）类

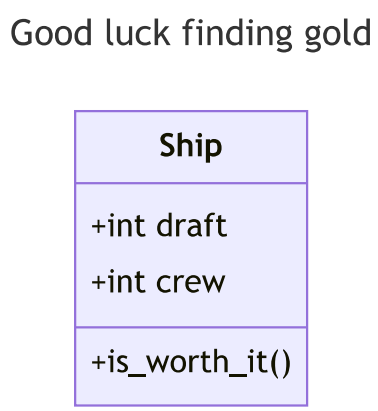
```
from math import sqrt

class Vector:

    def __init__(self, iterable):
        self._v = tuple(x for x in iterable)

    # 把打印元组时的空格去掉
    def __str__(self):
        return str(self._v).replace(' ', '')

    # 检查两个向量是否长度相等
    def check(self, other):
        if not len(self._v) == len(other._v):
            raise ValueError('Vectors of different length')

    def add(self, other):
        self.check(other)
        return Vector(s + o for s, o in zip(self._v, other._v))

    def subtract(self, other):
        self.check(other)
        return Vector(s - o for s, o in zip(self._v, other._v))

    def dot(self, other):
        self.check(other)
        return sum(s * o for s, o in zip(self._v, other._v))

    def norm(self):
        return sqrt(sum(x**2 for x in self._v))

    def equals(self, other):
        return self._v == other._v
```

第五题： Codewars风格的等级系统

```python
class User ():
    def __init__ (self):
        self.RANKS = [-8, -7, -6, -5, -4, -3, -2, -1, 1, 2, 3, 4, 5, 6, 7, 8]
        self.rank = -8
        self.rank_index = 0
        self.progress = 0

    def inc_progress (self, rank):
        if not rank in self.RANKS:
            raise ValueError
        rank_index = self.RANKS.index(rank)
        if rank_index == self.rank_index:
            self.progress += 3

        elif rank_index == self.rank_index - 1:
            self.progress += 1

        elif rank_index > self.rank_index:
            difference = rank_index - self.rank_index
            self.progress += 10 * difference * difference

        while self.progress >= 100 and self.rank < 8:
            self.rank_index += 1
            self.rank = self.RANKS[self.rank_index]
            self.progress -= 100

            if self.rank == 8:
                self.progress = 0
                return
```

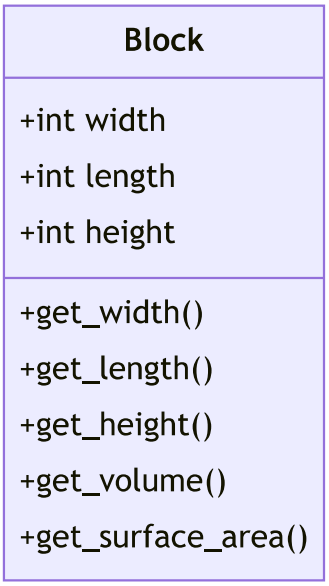
- 第三部分 使用Mermaid绘制程序的类图

第一题：面向对象的海盗

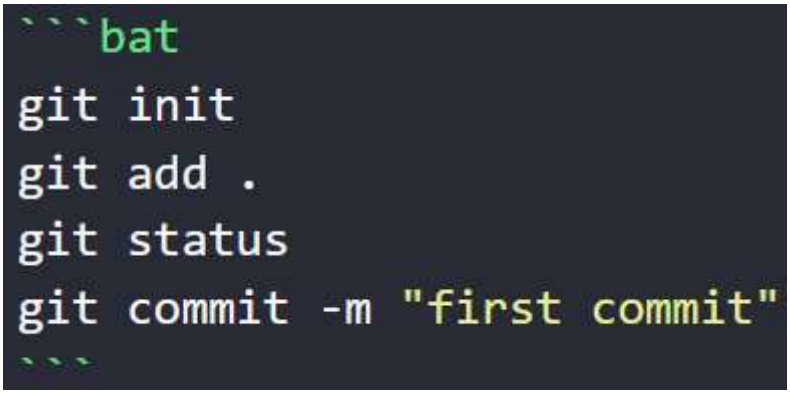


第二题： 搭建积木

Build the building blocks



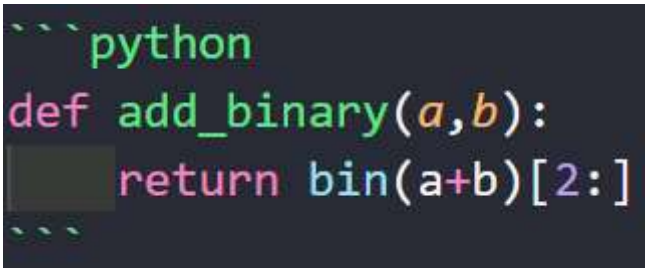
注意代码需要使用markdown的代码块格式化，例如Git命令行语句应该使用下面的格式：



显示效果如下：

```bat
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：



显示效果如下：

```python
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。

1. Python的类中__init__方法起什么作用?

在Python中，init 是一个特殊的方法，通常被称为类的构造函数或初始化方法。当创建类的新实例时，init 方法会自动被调用，以初始化新创建的对象。

init 方法的主要作用是：

- 初始化属性：你可以在 init 方法中为新创建的对象初始化属性。
- 为对象设置初始状态：除了属性之外，init 还可以用于为新对象设置其他初始状态或执行其他初始化操作。
- 确保对象的完整性：通过在 init 中进行必要的检查或设置，可以确保对象在被使用之前已经正确初始化。

2. Python语言中如何继承父类和改写（override）父类的方法。

在Python中，可以使用class关键字来定义一个类，并通过在类定义时指定父类来实现继承。要改写（override）父类的方法，只需在子类中重新定义该方法即可。

3. 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。

Python类中有一些特殊的方法，它们以双下划线开头和结尾，例如__init__ 和 __str__以及__eq__等。这些方法在Python中被称为"魔法方法"或"双下方法"。它们的作用是允许我们自定义类的行为，覆盖Python默认的行为。例如：

init：这是一个构造方法，当一个对象被创建后会自动调用。它通常用于初始化对象的属性。

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p = Person("Alice", 25)
print(p.name)  # 输出: Alice
print(p.age)   # 输出: 25
```

str：这个方法返回一个对象的字符串表示。当我们尝试打印一个对象时，或者在将对象转换为字符串时，这个方法会被调用。

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"Person(name={self.name}, age={self.age})"

p = Person("Alice", 25)
print(p)  # 输出: Person(name=Alice, age=25)
```

eq：这个方法用于比较两个对象是否相等。当我们使用 == 运算符比较两个对象时，这个方法会被调用。

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __eq__(self, other):
        if isinstance(other, Person):
            return self.name == other.name and self.age == other.age
        return False

p1 = Person("Alice", 25)
p2 = Person("Bob", 25)
p3 = Person("Alice", 25)

print(p1 == p2)  # 输出: False
print(p1 == p3)  # 输出: True
```

实验总结

总结一下这次实验你学习和使用到的知识，例如：编程工具的使用、数据结构、程序语言的语法、算法、编程技巧、编程思想。

在本次实验中，我们进行了Python面向对象编程的实践。通过定义类和创建对象，学习了如何使用封装、继承和多态等核心概念进行编程。深入理解了面向对象编程的核心概念，掌握了类的定义。通过继承和多态的应用，感受到了代码的重用性和灵活性，为后续的编程工作奠定了基础。在实验过程中，可能存在对某些概念理解不够深入的问题。需要加强理论学习，以便更好地应用在实际开发中。后续可以尝试更多的实验案例，以加深对面向对象编程的理解和应用能力。可以通过实现更复杂的类和方法来锻炼自己的编程能力。