桂林电子科技大学

实验 2 文件操作和异常处理 实验报告

实验名称	文件操作和异常处理						
院　　系	计算机与信息安全学院		专业	软件工程			
学　　号	2200350204		姓名	李禹佳			
实验日期	2023	年	12	月	8	日	

辅导员意见：

成绩　　辅导员签　名

一、实验目的

1. 掌握不同文件类型的读写和异常处理；
2. 掌握目录遍历的方法；
3. 掌握 json 解析器的设计与实现；
4. 掌握装饰器的使用；
5. 掌握类的设计和定制

二、实验内容

1. 读取 txt 文件内容；
2. 素数文件写入文件；
3. 遍历输出文件目录；
4. 读写 json 文件；
5. 工资结算计算；
6. Json 解析器；
7. 类的设计；
8. 类的定制。

三、实验环境

在 Educoder 平台进行实验

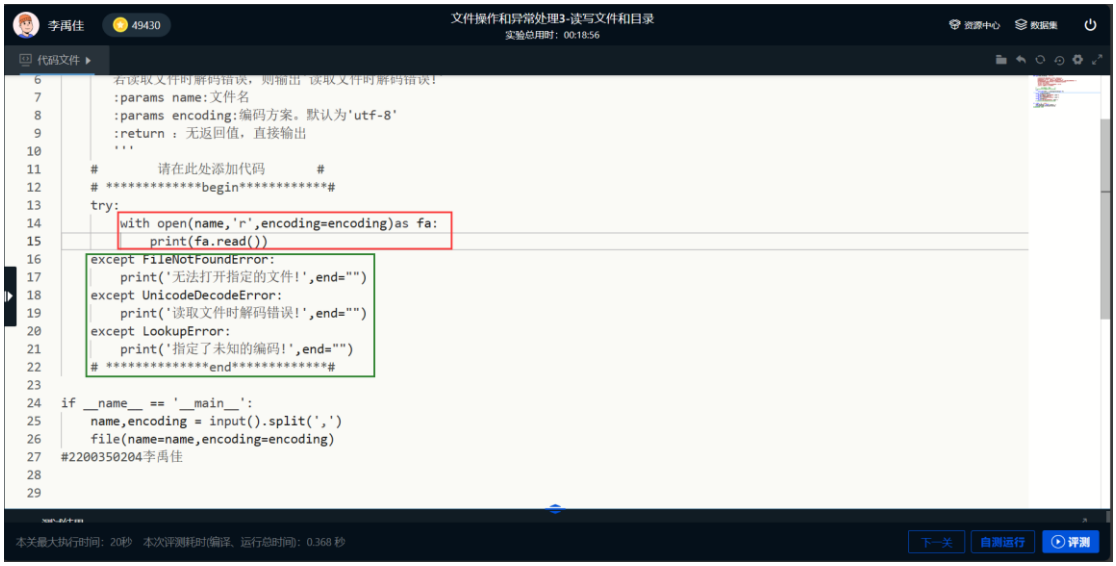
四、实验要求

根据每个实训的每个关卡要求完成代码提交和测评

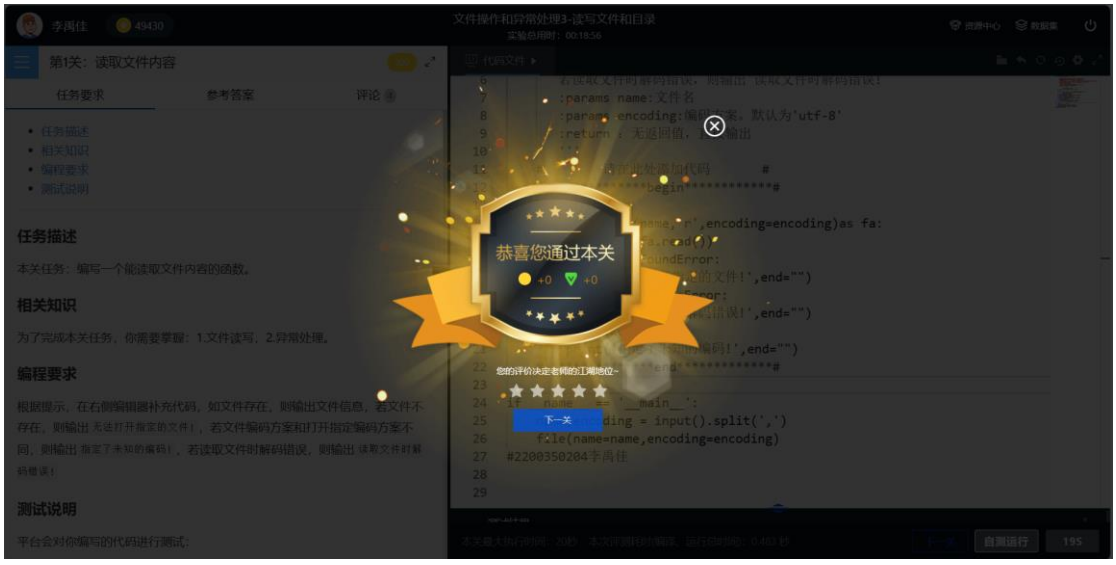
五、实验步骤

1. 读取 txt 文件内容
   代码截图：

在代码中我对异常情况进行了处理，情况分别为文件不存在，文件解码错误，文件编码错误。异常捕获为：FileNotFoundError ， UnicodeError ， LookupError 如果没有上述异常错误，则打印目标文件内容

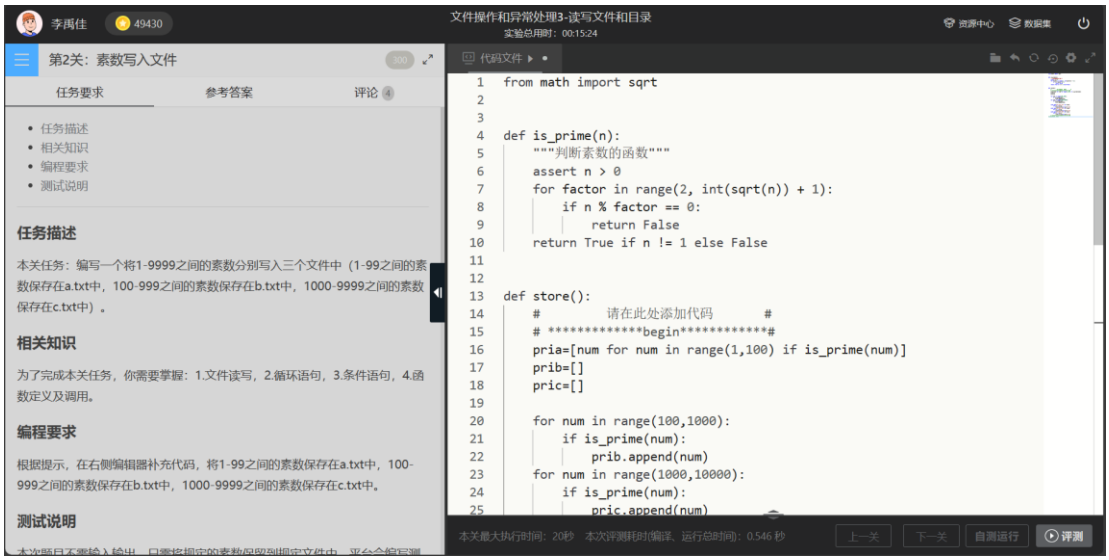


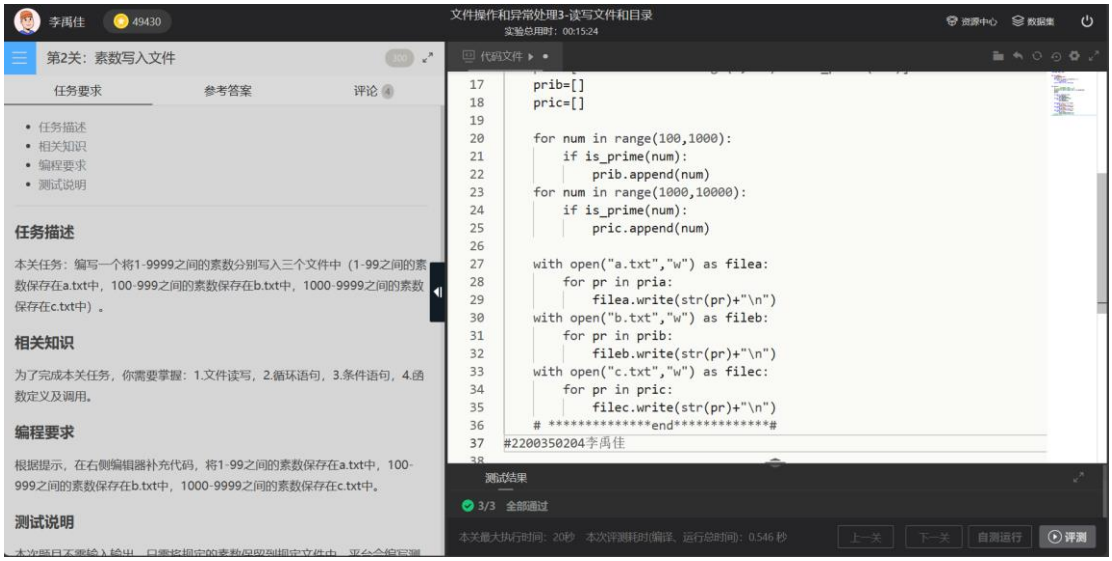
通关



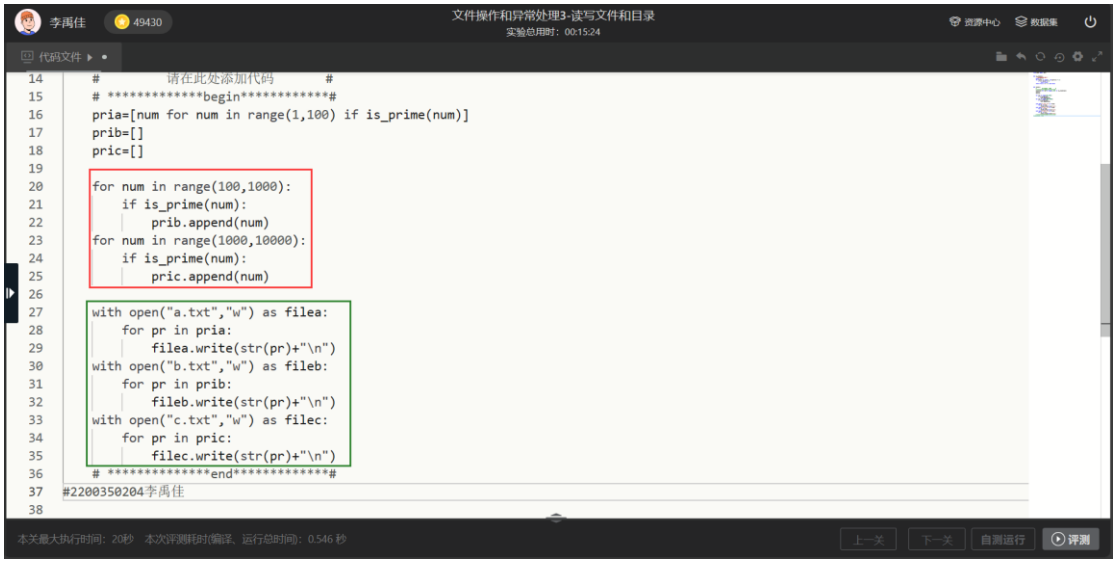
2. 素数文件写入文件

代码截图：

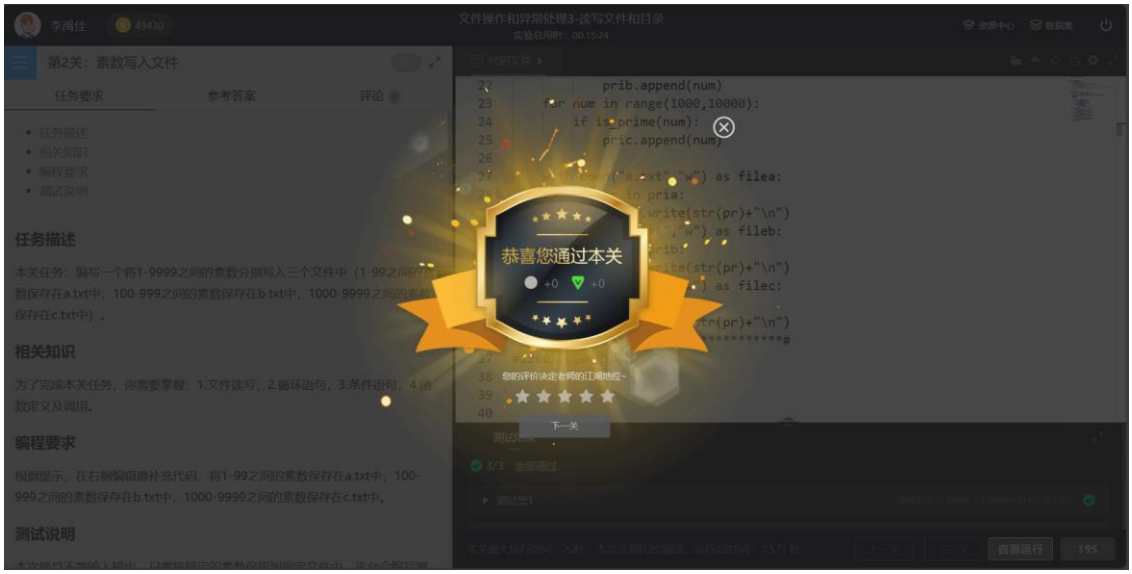




由题将不同范围的素数存入相对应文件，在代码中我先是用循环来获取不同区间的数再依次判断是否为素数，后尝试使用简写放在了第一个数组中。后使用 open 打开文件并使用 write 用循环将存入数组的数依次写入相应文件

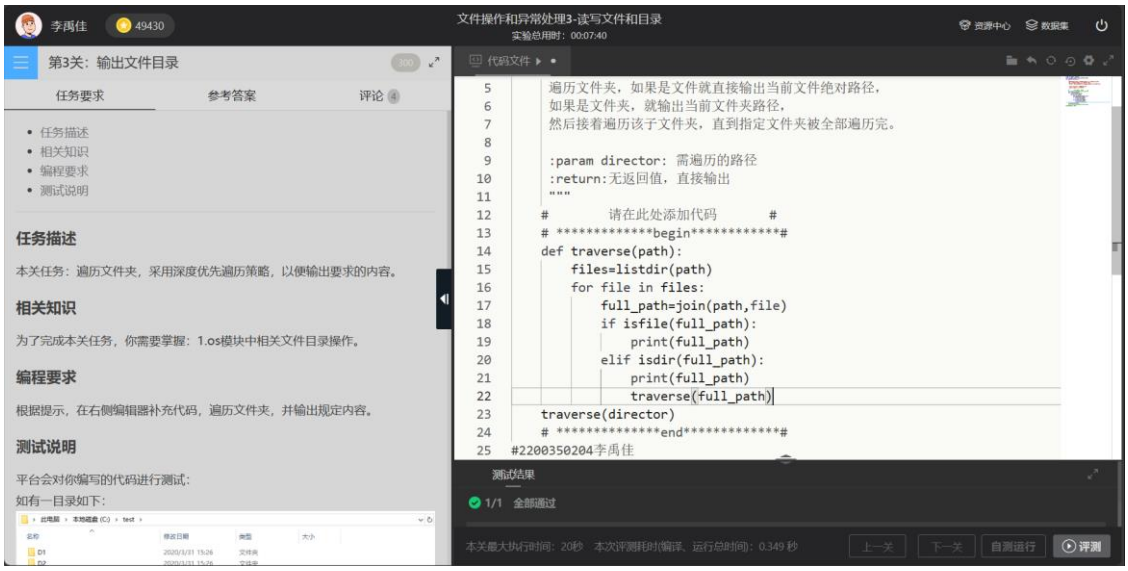


通关：



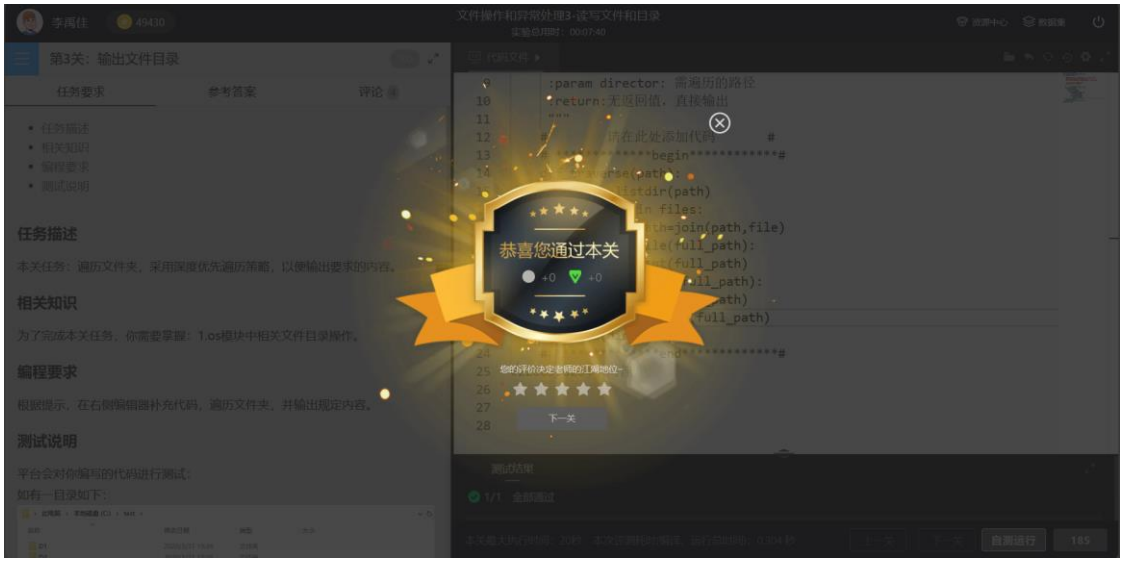
3. 遍历输出文件目录

代码截图：



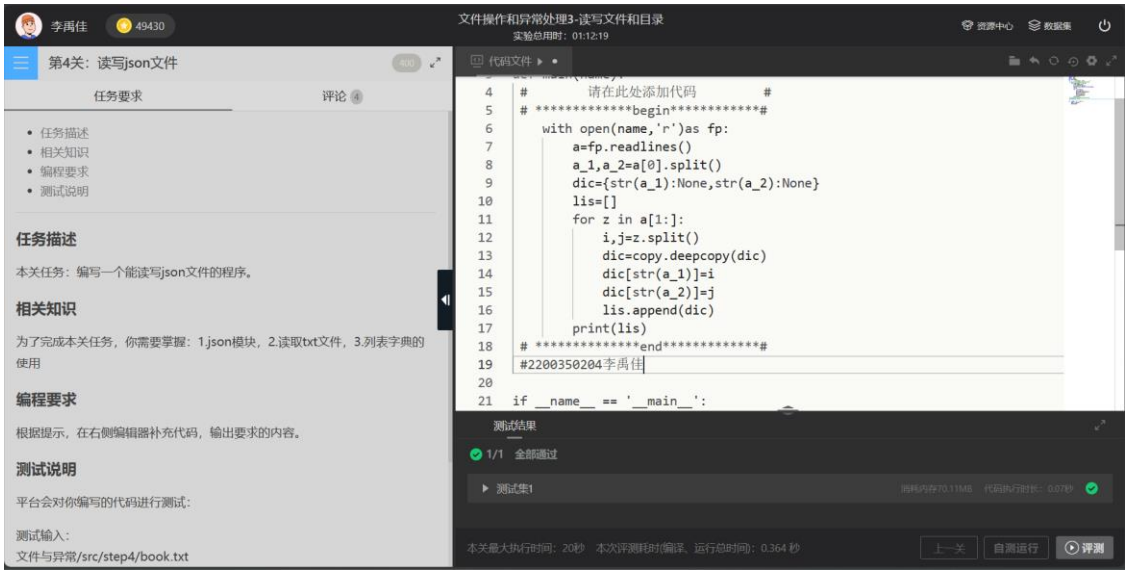
Traverse 方法首先通过 listdir(path) 获取指定路径下的所有文件和文件夹的列表，将它们存储在变量 files 中，然后循环遍历 files 列表。使用 join(path, file)获取当前文件或文件夹的完整路径，存储在变量 full_path 中。通过 isfile(full_path) 判断当前遍历的是否是一个文件，是文件则直接打印该文件的完整路径；是文件夹则打印该文件夹下的文件和文件夹。最后，代码调用 traverse(director) 函数，传入一个路径作为参数，从该路径开始递归遍历

通关：

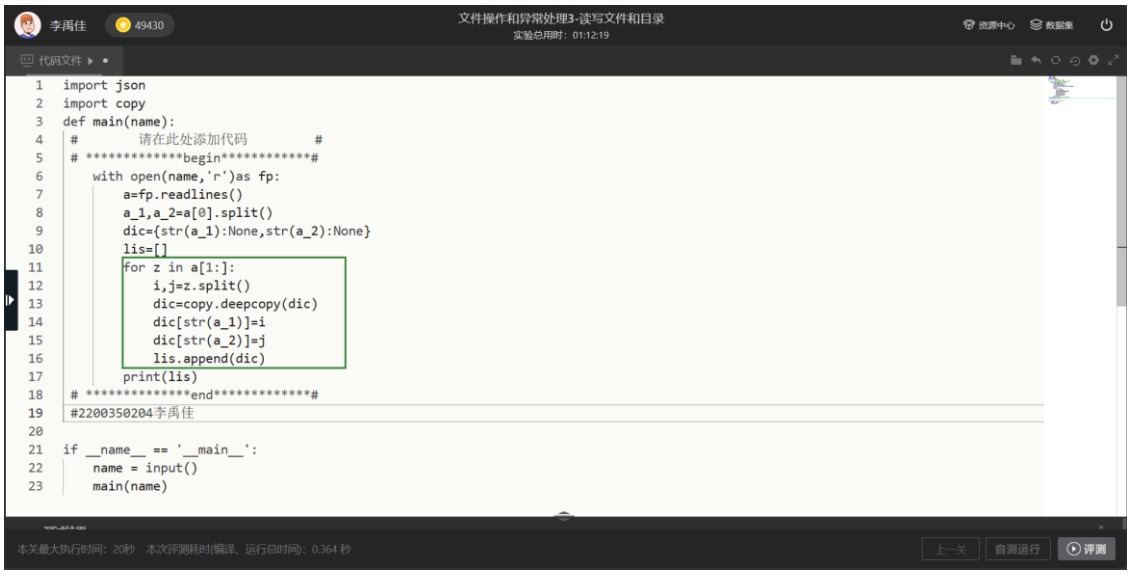


4. 读写 json 文件

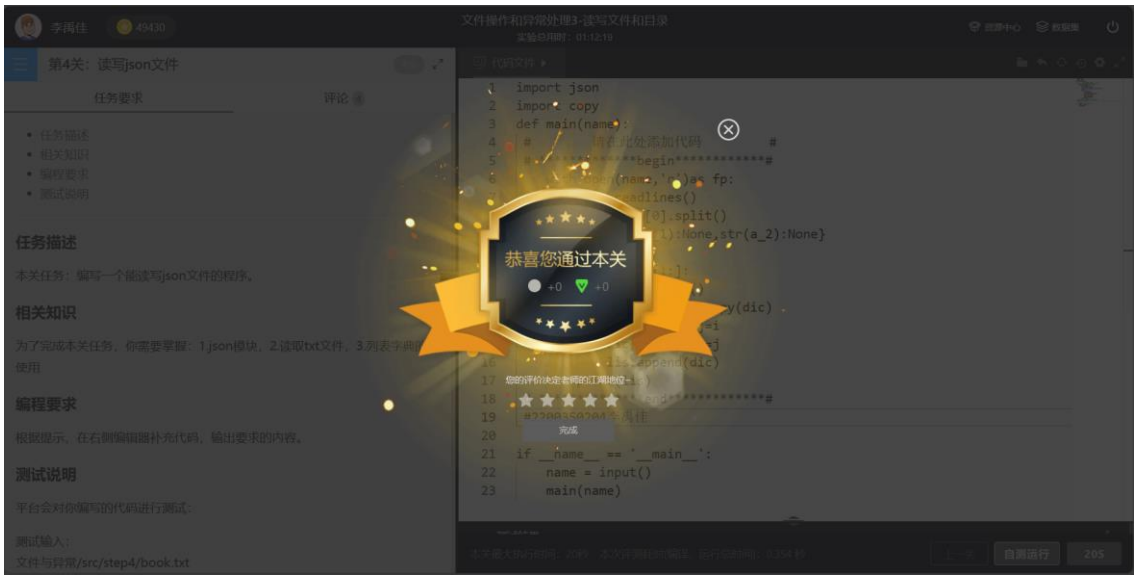
代码截图：



打开 name 值所在文件，逐行读取文件内容，并存入 a。通过 split() 方法按空格分割字符串，并将结果存储入 a_1 和 a_2。循环遍历从第二行开始的所有行，将每行的内容按空格分割为两个值，存储在 i 和 j 中。通过 copy.deepcopy(dic) 创建字典 dic 的深拷贝，将深拷贝结果存回。将 i 赋值给 dic[str(a_1)]，将 j 赋值给 dic[str(a_2)]。将 dic 添加到列表 lis 中并打印 lis

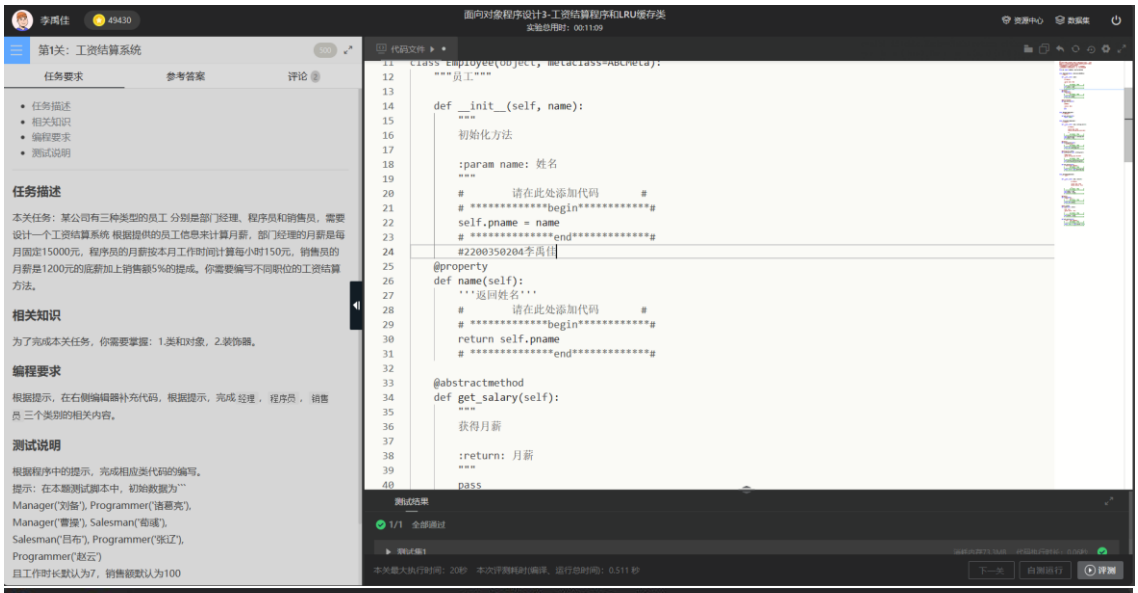


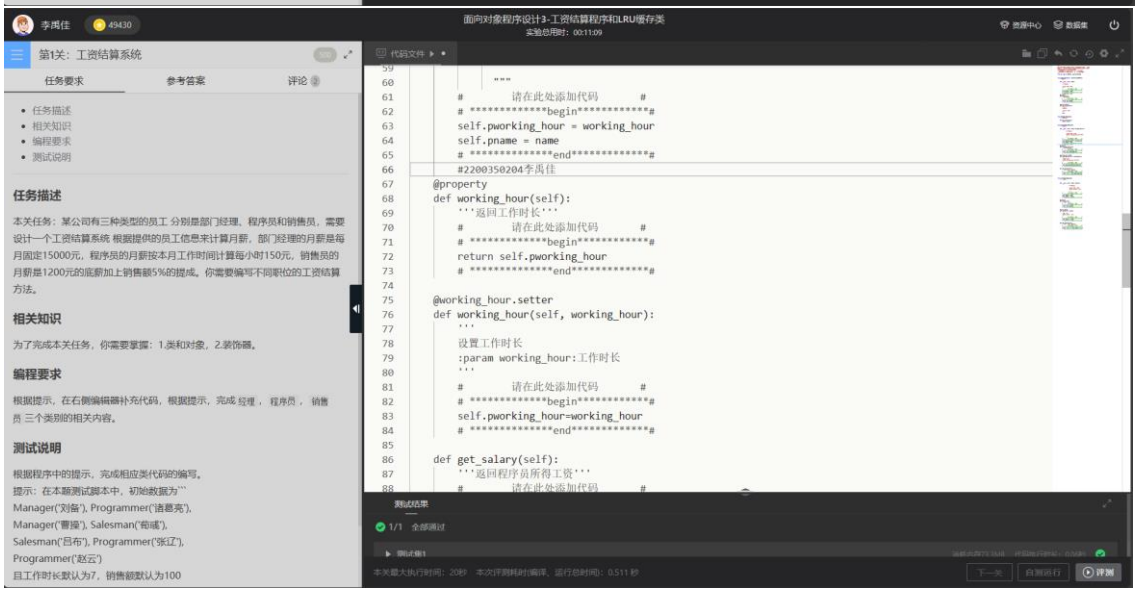
通关：

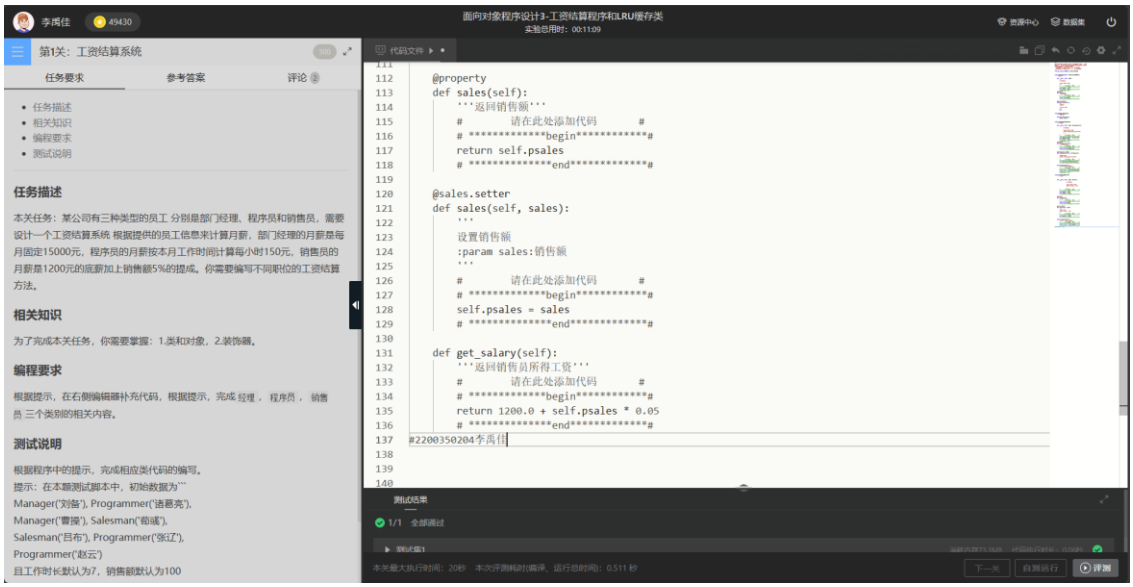


5. 工资结算计算

代码截图：





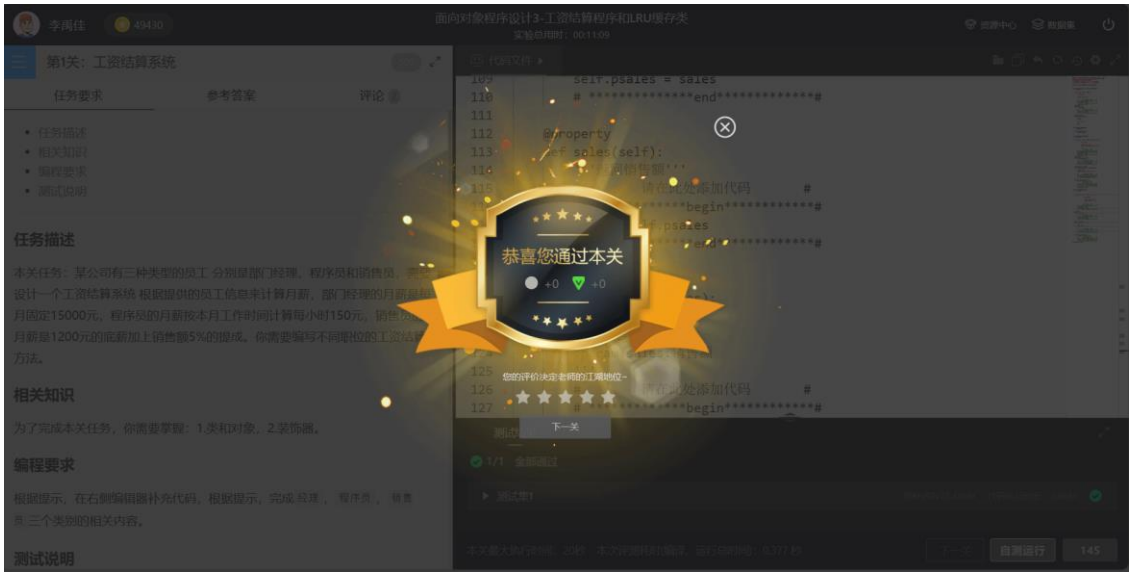


再__init__方法中，使用参数 name 初始化 pname,将参数值传入，在 name 方法中使用 getter 方法返回实例的姓名

在__init__方法中，初始化 pname 和 pworking_hour，在 working_hour 方法中使用 getter 方法中返回实例的工作时长，在 get_salary 中返回*150 后的工资数

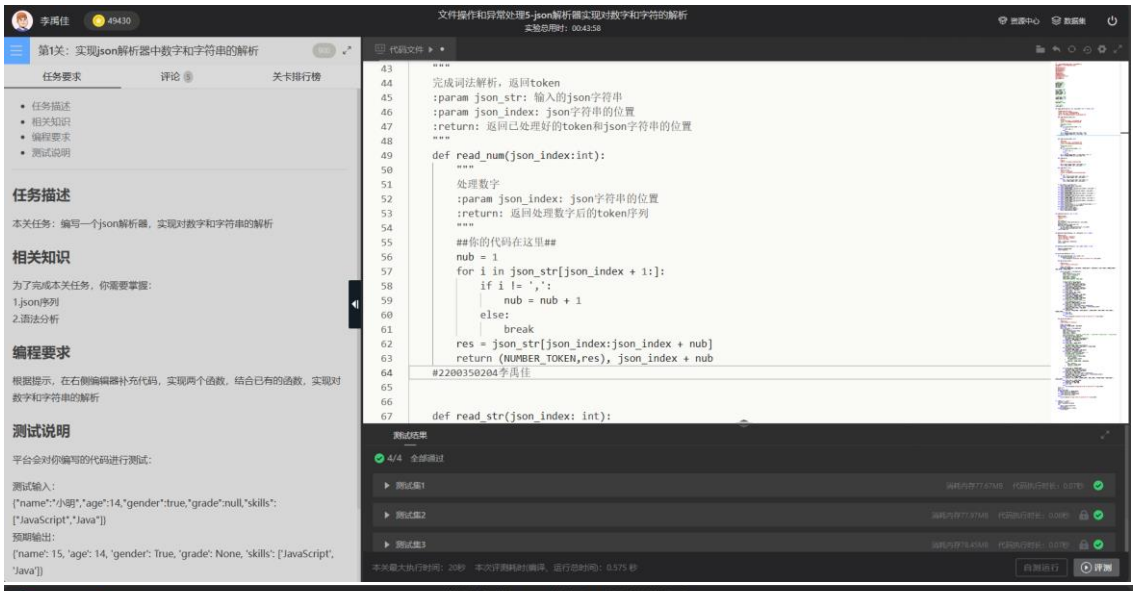
在__init__方法中初始化 pname 和 psales 并传入数据，在@property sales 方法中使用 getter 返回实例的销售额，在 @sales.setter sales 方法中使用 setter 将销售额属性改为可修改，在 get_salary 方法中*1200 返回销售员工资

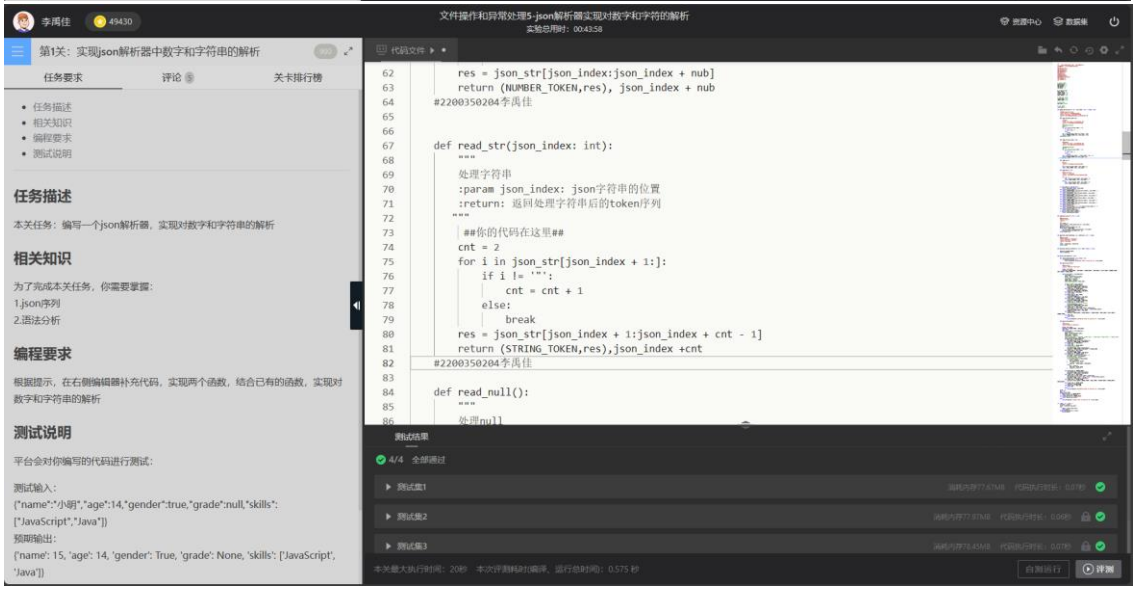
通关：



6. Json 解析器

代码截图：



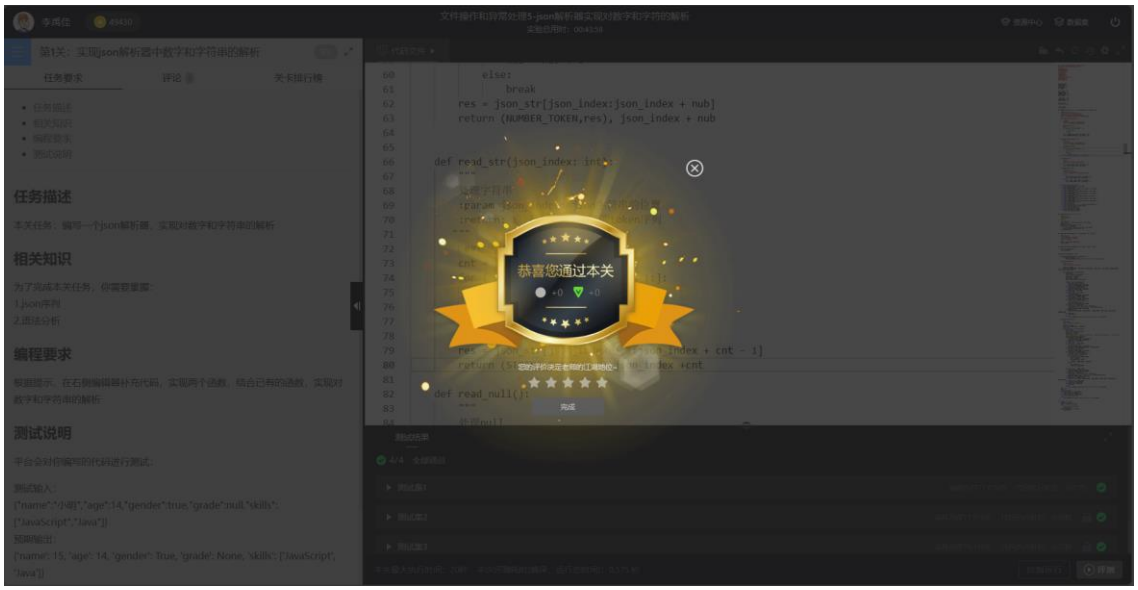


找到数字以,结束

找到以”开始再以”结尾的字符串

```
        else:
            break
    res = json_str[json_index:json_index + nub]
    return (NUMBER_TOKEN,res), json_index + nub
#2200350204李禹佳


def read_str(json_index: int):
    """
    处理字符串
    :param json_index: json字符串的位置
    :return: 返回处理字符串后的token序列
    """
    ##你的代码在这里##
    cnt = 2
    for i in json_str[json_index + 1:]:
        if i != '"':
            cnt = cnt + 1
        else:
            break
    res = json_str[json_index + 1:json_index + cnt - 1]
    return (STRING_TOKEN,res),json_index +cnt
#2200350204李禹佳

def read_null():
    """
    处理null
    :return: 返回处理null后的token序列
    """
    rem = json_str[json_index: json_index + 4]
    return (NULL_TOKEN, rem), json_index + 4

def read_bool(s: str):
```

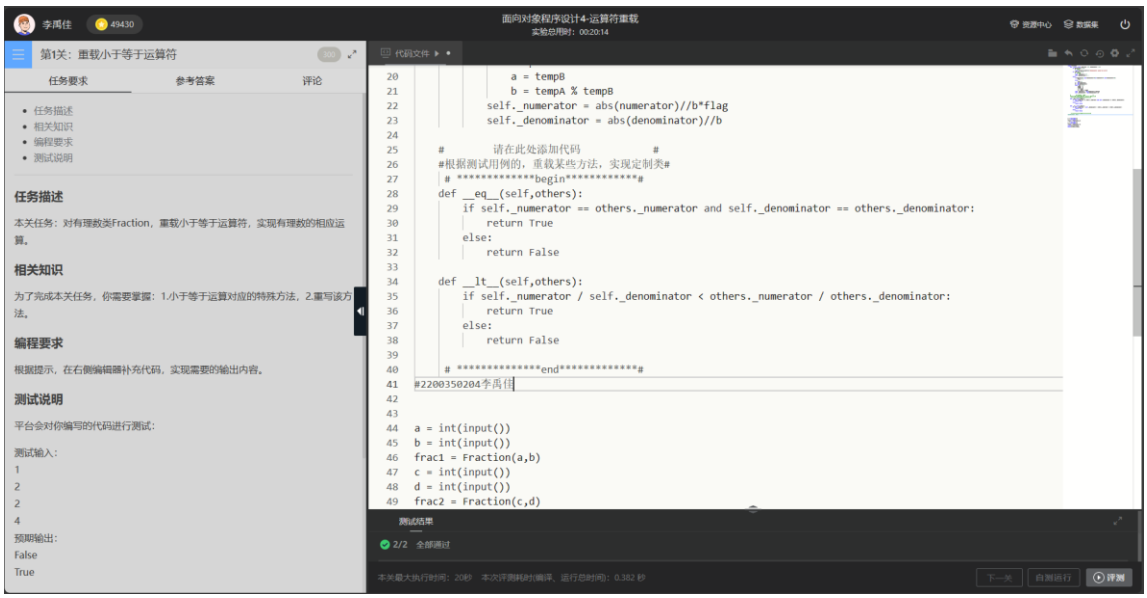
通关：



7. 类的设计

7-1 重载小于等于运算符

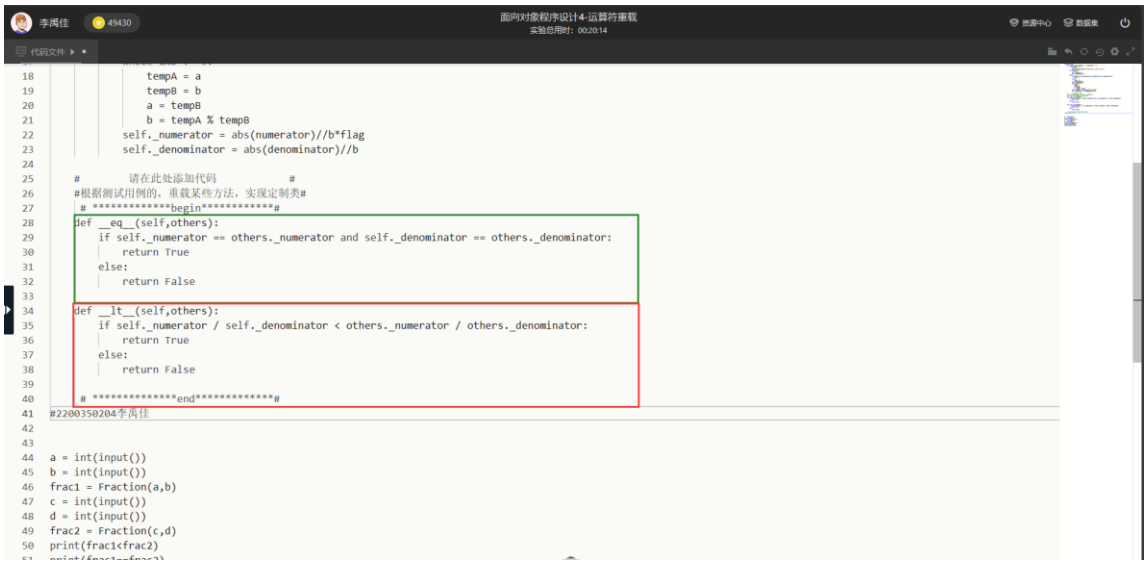
代码截图：



```
            a = tempB
            b = tempA % tempB
        self._numerator = abs(numerator)//b*flag
        self._denominator = abs(denominator)//b

    #         请在此处添加代码                #
    #根据测试用例的，重载某些方法，实现定制类#
    # ************begin***********#
    def __eq__(self,others):
        if self._numerator == others._numerator and self._denominator == others._denominator:
            return True
        else:
            return False

    def __lt__(self,others):
        if self._numerator / self._denominator < others._numerator / others._denominator:
            return True
        else:
            return False

    # ************end***********#
#2200350204李禹佳


a = int(input())
b = int(input())
frac1 = Fraction(a,b)
c = int(input())
d = int(input())
frac2 = Fraction(c,d)
```

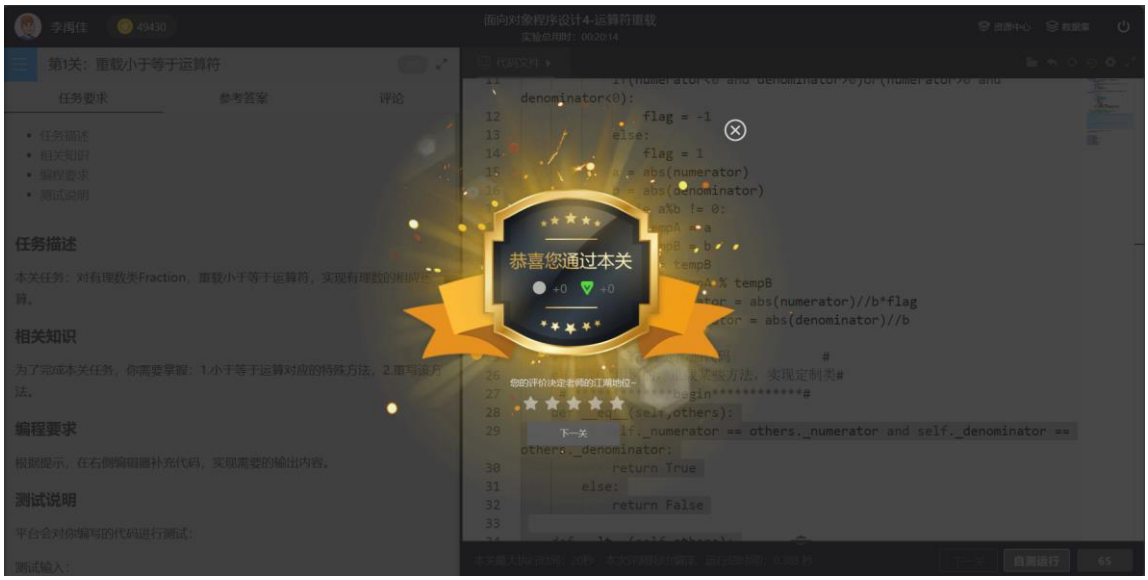
重载__eq__和__lt__方法，__eq__方法判断两个分数是否相等。如果两个分数的分子和分母都相等，则认为它们相等。方法中将 self._numerator 和 self._denominator 与另一个分数对象的对应属性进行比较。

__lt__ 方法用于实现小于号 < 的比较操作。比较两个分数的大小实际上就是比较它们的大小关系，即比较两个分数的小数值的大小。实现中首先将两个分数转换为小数值进行比较，如果左边的分数小于右边的分数，则返回 True，否则返回 False。



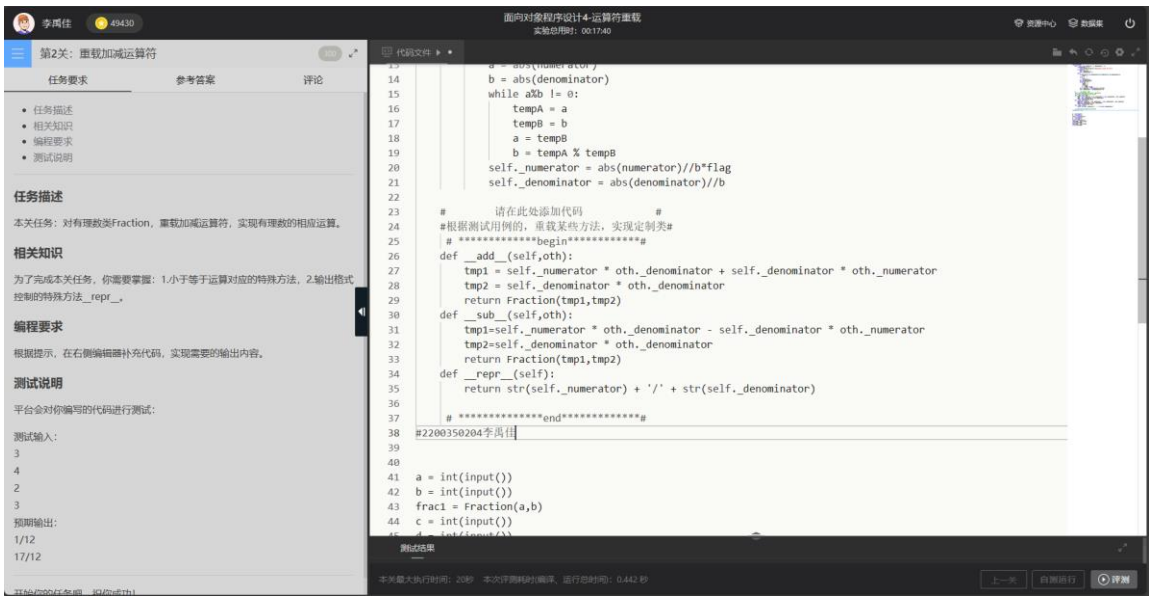
```
            tempA = a
            tempB = b
            a = tempB
            b = tempA % tempB
        self._numerator = abs(numerator)//b*flag
        self._denominator = abs(denominator)//b

    #        请在此处添加代码            #
    #根据测试用例的，重载某些方法，实现定制类#
    # **********begin**********#
    def __eq__(self,others):
        if self._numerator == others._numerator and self._denominator == others._denominator:
            return True
        else:
            return False

    def __lt__(self,others):
        if self._numerator / self._denominator < others._numerator / others._denominator:
            return True
        else:
            return False

    # **********end**********#
#2200350204李禹佳


a = int(input())
b = int(input())
frac1 = Fraction(a,b)
c = int(input())
d = int(input())
frac2 = Fraction(c,d)
print(frac1<frac2)
```

通关：



7-2 重载加减运算符

代码截图：

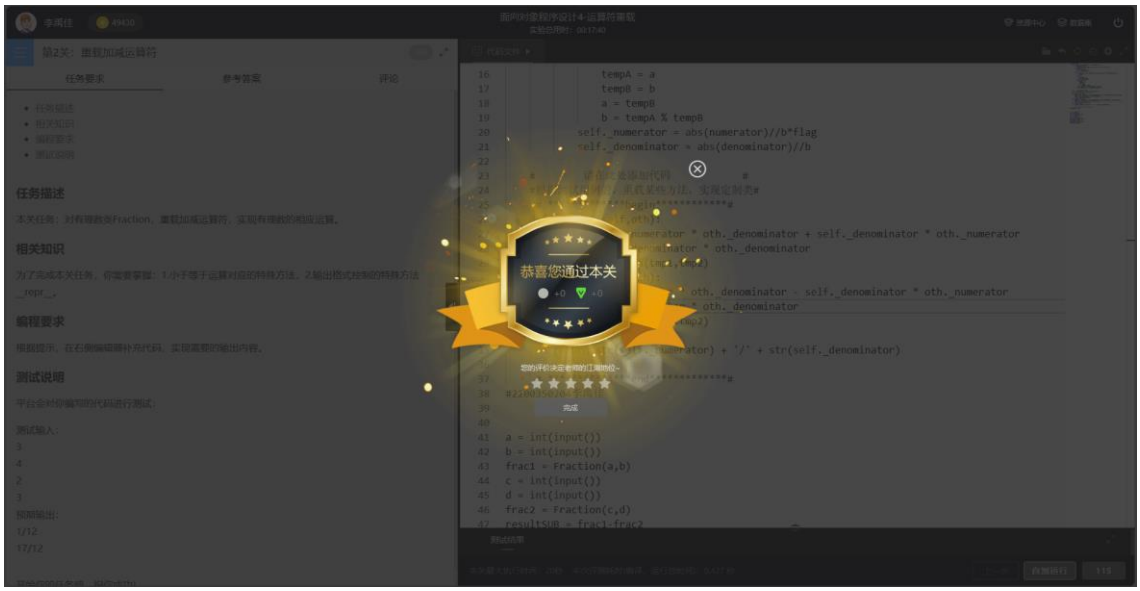



```
            b = abs(denominator)
            while a%b != 0:
                tempA = a
                tempB = b
                a = tempB
                b = tempA % tempB
            self._numerator = abs(numerator)//b*flag
            self._denominator = abs(denominator)//b

    #        请在此处添加代码            #
    #根据测试用例的，重载某些方法，实现定制类#
    # **********begin**********#
    def __add__(self,oth):
        tmp1 = self._numerator * oth._denominator + self._denominator * oth._numerator
        tmp2 = self._denominator * oth._denominator
        return Fraction(tmp1,tmp2)
    def __sub__(self,oth):
        tmp1=self._numerator * oth._denominator - self._denominator * oth._numerator
        tmp2=self._denominator * oth._denominator
        return Fraction(tmp1,tmp2)
    def __repr__(self):
        return str(self._numerator) + '/' + str(self._denominator)

    # **********end**********#
#2200350204李禹佳

a = int(input())
b = int(input())
frac1 = Fraction(a,b)
c = int(input())
```

重载__add__、__sub__ 和 __repr__方法；__add__ 方法将两个分数的分子分别乘以对方的分母再相加得到分子。分母则是两个分数的分母相乘。最后，将得到的分子和分母作为参数创建一个新的分数对象，并返回；__sub__ 方法和加法类似，将两个分数的分子分别乘以对方的分母再相减得到分子。分母则是两个分数的分母相乘。最后，将得到的分子和分母作为参数创建一个新的分数对象，并返回；__repr__ 方法用于定义分数类的字符串表示形式。将分数的分子和分母以 / 分隔并以字符串的形式返回。

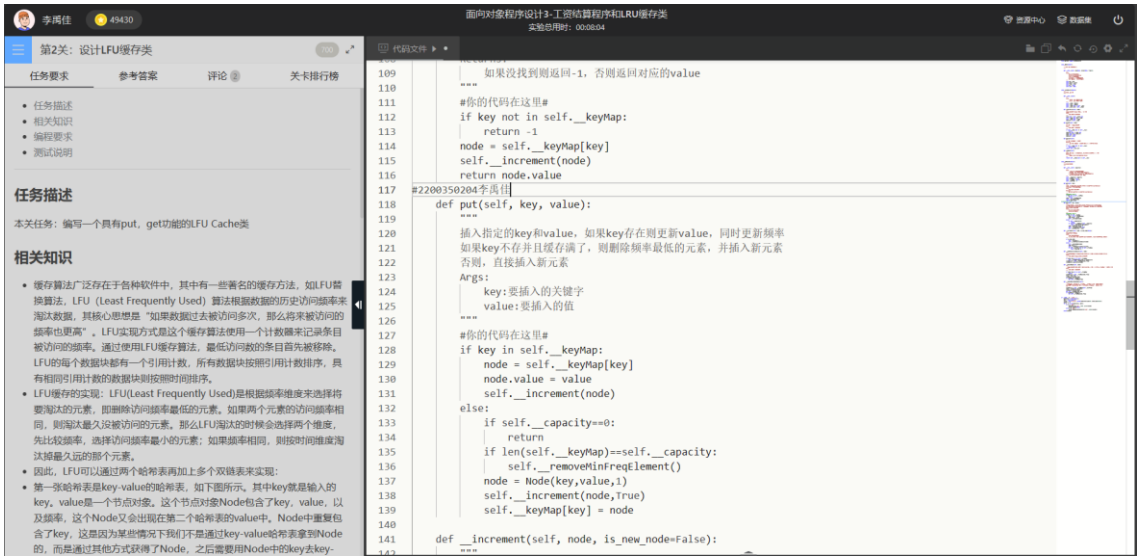
```
            tempA = a
            tempB = b
            a = tempB
            b = tempA % tempB
        self._numerator = abs(numerator)//b*flag
        self._denominator = abs(denominator)//b

    #        请在此处添加代码          #
    #根据测试用例的，重载某些方法，实现定制类#
    # ************begin************#
    def __add__(self,oth):
        tmp1 = self._numerator * oth._denominator + self._denominator * oth._numerator
        tmp2 = self._denominator * oth._denominator
        return Fraction(tmp1,tmp2)
    def __sub__(self,oth):
        tmp1=self._numerator * oth._denominator - self._denominator * oth._numerator
        tmp2=self._denominator * oth._denominator
        return Fraction(tmp1,tmp2)
    def __repr__(self):
        return str(self._numerator) + '/' + str(self._denominator)

    # ************end************#
#2200350204李禹佳


a = int(input())
b = int(input())
frac1 = Fraction(a,b)
c = int(input())
d = int(input())
frac2 = Fraction(c,d)
resultSUB = frac1-frac2
```

通关：



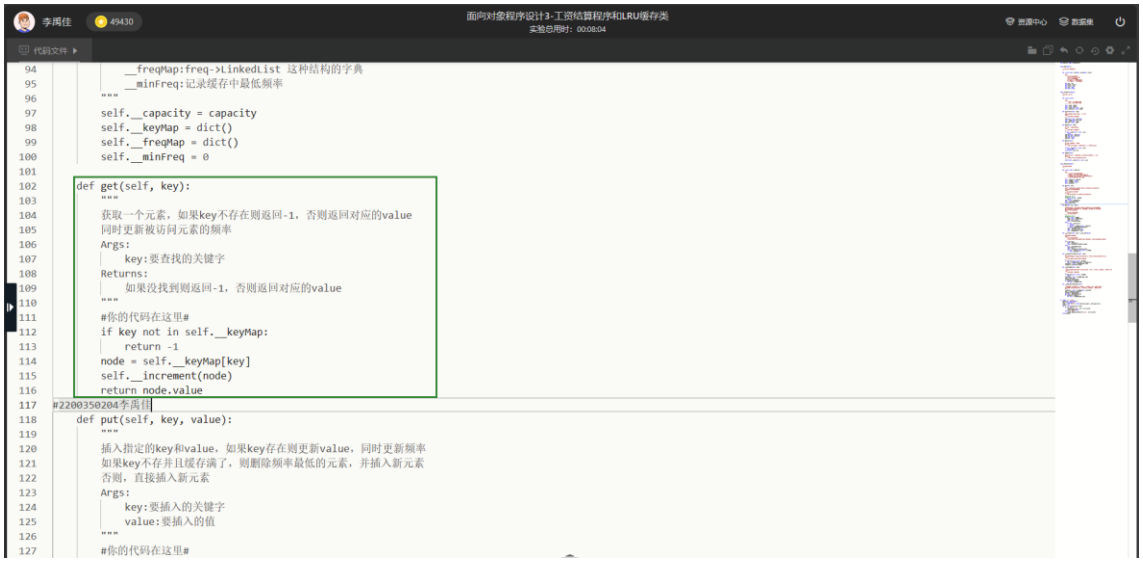
8. 类的定制

代码截图：



```
        如果没找到则返回-1，否则返回对应的value
        """
        #你的代码在这里#
        if key not in self.__keyMap:
            return -1
        node = self.__keyMap[key]
        self.__increment(node)
        return node.value
#2200350204李禹佳
    def put(self, key, value):
        """
        插入指定的key和value，如果key存在则更新value，同时更新频率
        如果key不存并且缓存满了，则删除频率最低的元素，并插入新元素
        否则，直接插入新元素
        Args:
            key:要插入的关键字
            value:要插入的值
        """
        #你的代码在这里#
        if key in self.__keyMap:
            node = self.__keyMap[key]
            node.value = value
            self.__increment(node)
        else:
            if self.__capacity==0:
                return
            if len(self.__keyMap)==self.__capacity:
                self.__removeMinFreqElement()
            node = Node(key,value,1)
            self.__increment(node,True)
            self.__keyMap[key] = node

    def __increment(self, node, is_new_node=False):
        """
```

Get 中获取到了 key 对应的元素值，如果 key 不存在返回-1，如果存在则返回对应的 value 值

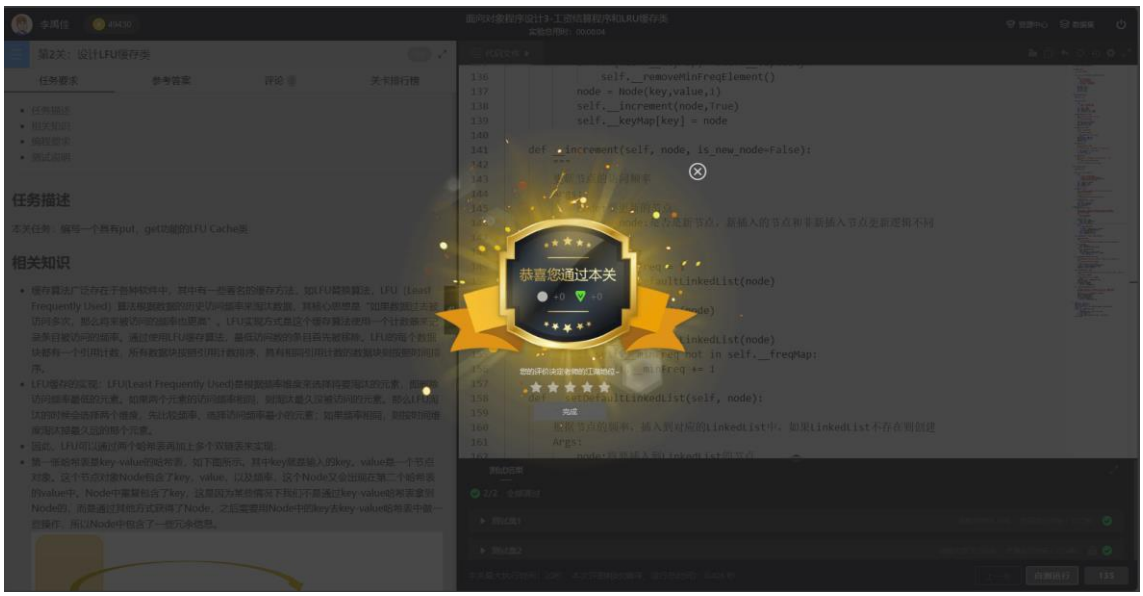


```
        __freqMap:freq->LinkedList 这种结构的字典
        __minFreq:记录缓存中最低频率
    """
    self.__capacity = capacity
    self.__keyMap = dict()
    self.__freqMap = dict()
    self.__minFreq = 0

def get(self, key):
    """
    获取一个元素，如果key不存在则返回-1，否则返回对应的value
    同时更新被访问元素的频率
    Args:
        key:要查找的关键字
    Returns:
        如果没找到则返回-1，否则返回对应的value
    """
    #你的代码在这里#
    if key not in self.__keyMap:
        return -1
    node = self.__keyMap[key]
    self.__increment(node)
    return node.value
#2200350204李禹佳
def put(self, key, value):
    """
    插入指定的key和value，如果key存在则更新value，同时更新频率
    如果key不存并且缓存满了，则删除频率最低的元素，并插入新元素
    否则，直接插入新元素
    Args:
        key:要插入的关键字
        value:要插入的值
    """
    #你的代码在这里#
```

Put 中插入指定的 key 和 value,如果 key 已经存在，则更新 value,如果 key 不存在，则删除频率最低的元并插入新元素

```
            self.__removeMinFreqElement()
        node = Node(key,value,1)
        self.__increment(node,True)
        self.__keyMap[key] = node

def __increment(self, node, is_new_node=False):
    """
    更新节点的访问频率
    Args:
        node:要更新的节点
        is_new_node:是否是新节点，新插入的节点和非新插入节点更新逻辑不同
    """
    if is_new_node:
        self.__minFreq = 1
        self.__setDefaultLinkedList(node)
    else:
        self.__deleteNode(node)
        node.freq += 1
        self.__setDefaultLinkedList(node)
        if self.__minFreq not in self.__freqMap:
            self.__minFreq += 1

def __setDefaultLinkedList(self, node):
    """
    根据节点的频率，插入到对应的LinkedList中，如果LinkedList不存在则创建
    Args:
        node:将要插入到LinkedList的节点
    """
    if node.freq not in self.__freqMap:
        self.__freqMap[node.freq] = LinkedList()
    linkedList = self.__freqMap[node.freq]
    linkedList.insertFirst(node)

def __deleteNode(self, node):
```

通关：



六、问题记录和实验总结（必写）

1. 对实现的方法，函数还不是很熟悉，需查找资料
2. 对代码的理解能力欠缺，长代码需耗时读完

ReadMe:
本实验报告中课内实验 2-1 的 1，2，3，4 对应实验内容的前四个，课内实验 2-3 进阶部分 1 中两关对应着实验内容的 7：类的设计(将两关分为 7-1 和 7-2)，课内实验 2-3 进阶部分 2 中两关对应着实验内容的 6：Json 解析器和 8：类的定制