报告人：严以鑫 学号：2023217533

实验一：Python 基础——程序控制结构与函数

实验详细要求

本课程主要学习 Python 的程序控制结构与函数设计，具体内容为分支结构

--条件表达式、循环结构--For 循环与 While 循环的基本语法、函数设计与使用。

(一) 实验目的

1. 掌握判断结构的语法

2. 掌握循环结构的语法

3. 掌握函数编程

(二)实验环境

Python 3.6 及以上 or PyCharm

(三)实验内容

1. 分支结构

编写代码：利用 if 判断来制作一个猜数字的小游戏

问题描述：程序运行时，系统在指定范围内生成一个随机数字，然后用户进行猜测，并根据用户输入进行必要的提示（right, too large, too small），如果猜对则提前结束程序，如果未有猜对，提示游戏结束并给出正确答案。

代码：

import random

num=random**.**randint(12, 20)

guessnum = int(input("12-20:"))

if guessnum>num:

    print("too large")

elif guessnum<num:

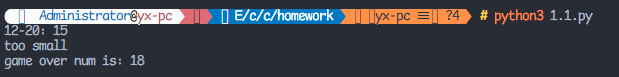
    print("too small")

else:

    print("right")

print("game over num is:",num)

截图:



2、循环结构

编写代码：利用 while 循环判断来制作一个猜数字的小游戏

问题描述：程序运行时，系统在指定范围内生成一个随机数字，然后用户进行猜测，并根据用户输入进行必要的提示（right, too large, too small），如果猜

对则提前结束程序，如果次数用完仍没有猜对，提示游戏结束并给出正确答案。

代码：

import random

num=random**.**randint(12, 20)

guessnum = int(input("12-20:"))

time =3

while guessnum!=num and time>0:

    if guessnum>num:

        print("too large")

    else:

        print("too small")

    guessnum = int(input())

    time=time-1

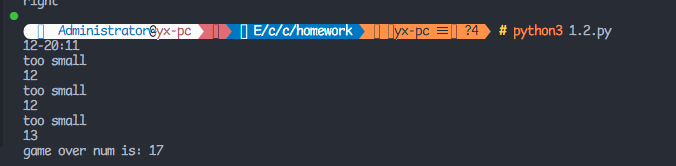
if(guessnum==num):

    print("right")

else:

    print("game over num is:",num)

截图:



3、函数结构

（1）设计函数用来计算斐波那契数列中小于参数 n 的所有值

def man1(num)->list:

    curr =1

    last =0

    list = []

    while curr<num:

        list**.**append(curr)

        tmp = curr

        curr = curr+last

        last = curr

return list

print(man1(10))

截图：



（2）利用列表实现筛选法求素数

问题描述：编写程序，输入一个大于 2 的自然数，然后输出小于该数字的所有素数组成的列表。

代码：

#*利用列表实现筛选法求素数*

def man2(num):

    m = math**.**ceil(math**.**sqrt(num)+1)

    for i in range(2, m):

        if num%i == 0 and i<num:

            return False

            break

        else:

            return True

def manman2():

    n = input("请输入一个大于2的自然数:")

    n = int(n)

    list2 =[]

    list2=list(filter(man2, [i for i in range(2, n+1)]))

print(list2)

manman2()

截图：



(3)编写函数：判断回文，也就是正读反读都一样的字符串

例如："abcba”

代码：

#*：判断回文*

def man3(str2):

    n = len(str2)

    right = n-1

    left = 0

    while left<n/2:

        if(str2[left]!=str2[right]):

            return False

        left+=1

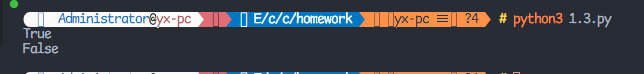
        right-=1

return True

print(man3('aba'))

print(man3('abaa'))

截图：



(4)编写函数：随机产生包含 n 个整数的列表，返回一个元组，其中第一个元素为所有参数的平均值，其他元素为所有参数中大于平均值的整数。例如： 随机产生长度为 3 的列表[2, 1, 3]，输出为（2.0, 3）

代码：

def man4(num):

    numbers = [random**.**randint(1, 100) for \_ in range(num)]

    print(numbers)

    avg = sum(numbers) / num

    greater\_than\_avg = [num1 for num1 in numbers if num1 > avg]

return (avg, \*greater\_than\_avg)

print(man4(3))

截图：



(5)编写函数： 一年 365 天， 每周工作 5 天，休息 2 天，休息日水平下降0.01，工作日要努力到什么程度一年后的水平才与每天努力 1%所取得的效果（即37.78 倍）一样呢？

代码：

def man5():

    log\_x = (math**.**log(37.78) - 105  \* math**.**log(0.99 )) / 260

    x = math**.**exp(log\_x)

return x

print(man5())

截图：



4、lambda 表达式

（1） 查找两个字符串首尾交叉的最大子串长度，连接两个字符串，首尾交叉部分只保留一份。例如，1234 和 2347 连接为 12347

要求：程序中使用 lambda 表达式以及函数

代码：

max\_overlap = lambda s1, s2: max([i for i in range(min(len(s1), len(s2)) + 1) if s1[-i:] == s2[:i]])

merge\_strings = lambda s1, s2: s1 + s2[max\_overlap(s1, s2):]

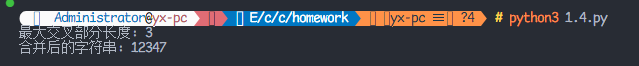
s1 = "1234"

s2 = "2347"

print(f"最大交叉部分长度: {max\_overlap(s1, s2)}")

print(f"合并后的字符串: {merge\_strings(s1, s2)}")

截图：



实验二：Python 基础二——序列数据类型

及应用

Python 序列类似于 C 中的一维、多维数组等，但功能要强大很多，使用也更加灵

活、方便。Python 中常用的序列结构有列表、元组、字典、字符串、集合、字典等。

（一）实验目的

1、掌握列表与列表推导式

2、掌握元组与生成器推导式

3、掌握切片的使用

4、掌握字典的用法

5、掌握集合的用法

6、掌握字符串及其应用

（二）实验环境

Python 3.6 及以上 or PyCharm

（三）实验内容

实验内容包含变量、字符串、列表、列表推导式、切片、元组、生成器推导式、字典、集合的应用。

（1） 列表推导式与字典的应用

问题描述：编写程序，先生成包含 1000 个随机字符的字符串，然后统计每

个字符出现的次数。

要求：查找资料编写至少 2 种不同的求解方法。

代码：

import string

import random

from collections import Counter

str1 = ''**.**join(random**.**choice(string**.**ascii\_lowercase) for \_ in range(1000))

#*1.*

counter1 =Counter(str1)

for char, count in counter1**.**items():

    print(f"{char}: {count} 次")

#*2.*

dict = {}

for char in str1:

    if char in dict:

        dict[char]+=1

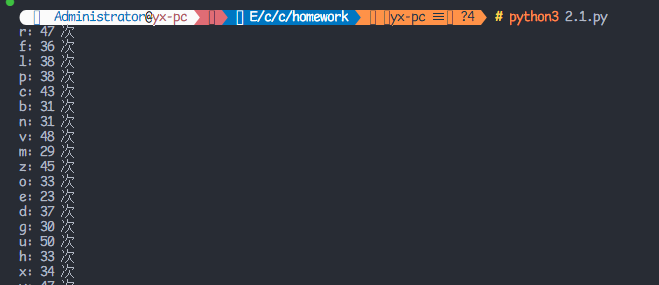
    else:

        dict[char]=1

for char, count in dict**.**items():

    print(f"{char}: {count} 次")

截图：



(2) 集合的应用

问题描述：编写程序，输入两个集合 setA 与 setB，分别输出它们两个交集的交、差、并。

要求：采用系统类与自定义集合类两种方法进行实现

代码：

# *系统类实现*

def system\_set\_operations(setA, setB):

    # *交集*

    intersection = setA**.**intersection(setB)

    # *并集*

    union = setA**.**union(setB)

    # *差集*

    difference = setA**.**difference(setB)

    return intersection, union, difference

# *示例*

setA = {1, 2, 3, 4}

setB = {3, 4, 5, 6}

intersection, union, difference = system\_set\_operations(setA, setB)

print("交集:", intersection)

print("并集:", union)

print("差集 (setA - setB):", difference)

# *自定义集合类*

class CustomSet:

    def \_\_init\_\_(self, elements):

        self**.**elements = elements

    def intersection(self, other):

        result = [item for item in self**.**elements if item in other**.**elements]

        return CustomSet(result)

    def union(self, other):

        result = self**.**elements[:]  # *复制自身元素*

        for item in other**.**elements:

            if item not in result:

                result**.**append(item)

        return CustomSet(result)

    def difference(self, other):

        result = [item for item in self**.**elements if item not in other**.**elements]

        return CustomSet(result)

    def \_\_repr\_\_(self):

        return f"{self**.**elements}"

# *示例*

setA = CustomSet([1, 2, 3, 4])

setB = CustomSet([3, 4, 5, 6])

intersection = setA**.**intersection(setB)

union = setA**.**union(setB)

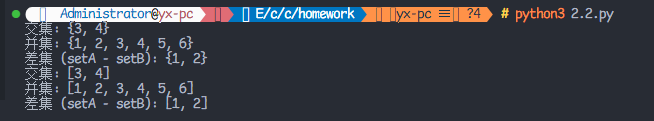
difference = setA**.**difference(setB)

print("交集:", intersection)

print("并集:", union)

print("差集 (setA - setB):", difference)

截图：



（3） 字符串与列表推导式的应用

问题描述： 编写程序，生成含有 n 个元素的嵌套列表，即列表的每个元素仍是列表，要求列表中的元素是长度不超过 m 的数字或字符组成的字符串，并按照字符串长度降序输出结果。

代码：

import random

import string

# *生成随机字符串*

def random\_string(max\_len):

    length = random**.**randint(1, max\_len)  # *随机生成 1 到 max\_len 长度的字符串*

    return ''**.**join(random**.**choice(string**.**ascii\_letters + string**.**digits) for \_ in range(length))

# *生成含有 n 个嵌套列表的结构*

def generate\_nested\_list(n, m):

    nested\_list = []

    for \_ in range(n):

        # *每个子列表包含 1 到 n 个随机字符串*

        num\_of\_strings = random**.**randint(1, n)

        sublist = [random\_string(m) for \_ in range(num\_of\_strings)]

        nested\_list**.**append(sublist)

    return nested\_list

# *按照字符串长度降序排序*

def sort\_nested\_list(nested\_list):

    for sublist in nested\_list:

        sublist**.**sort(key=len, reverse=True)  # *对每个子列表按照长度降序排序*

    return nested\_list

n = int(input("请输入列表个数 n: "))

m = int(input("请输入字符串最大长度 m: "))

 # *生成嵌套列表*

nested\_list = generate\_nested\_list(n, m)

print("\n生成的嵌套列表:")

for sublist in nested\_list:

    print(sublist)

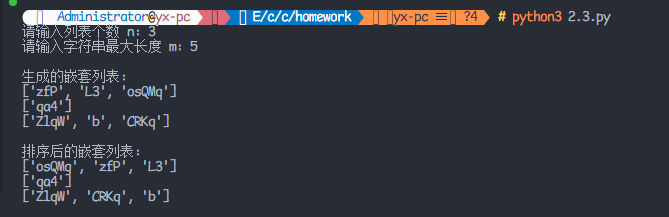
sorted\_nested\_list = sort\_nested\_list(nested\_list)

print("\n排序后的嵌套列表:")

for sublist in sorted\_nested\_list:

    print(sublist)

截图：



（4） 列表与切片的应用

问题描述：编写程序，生成一个整型列表，输出包含原列表中所有元素的新列表、包含原列表中所有元素的逆序列表，以及输出具有偶数位置的元素列表。

代码：

list = [1,2,3,4,5]

list2 =list+list[::-1]+list[1::2]

print(list2)

截图：



（5） 元组的应用

问题描述：编写程序，利用生成器推导式生成包含 n 个整数元素的元组，每个元素值不大于 m，并过滤掉偶数整数，并输出。

代码：

import random

result\_tuple = tuple(x for x in (random**.**randint(1, 5) for \_ in range(3)) if x % 2 != 0)

print(result\_tuple)

截图：



（6） 字典的应用

问题描述：编写程序，输入任意长度的字符串，统计每个单词出现的次数并存储到字典进行输出。

例如：输入：“I love China”，

输出：I：1

love: 1

China: 1

代码：

def count\_word\_frequency(input\_string):

    words = input\_string**.**split()

    word\_count = {}

    for word in words:

        word = word**.**strip()**.**lower()

        if word in word\_count:

            word\_count[word] += 1

        else:

            word\_count[word] = 1

    # *返回字典*

    return word\_count

input\_string =input()

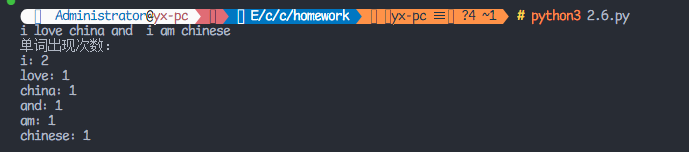
word\_count = count\_word\_frequency(input\_string)

print("单词出现次数：")

for word, count in word\_count**.**items():

    print(f"{word}: {count}")

截图：



（7）正则表达式的应用

问题描述：用户输入一段英文，然后输出这段英文中所有长度为 3 个字母的单词。（提示：可以调用 findall 函数，也可以先调用 split 函数将字符串进行分隔，再搜索长度为 3 的单词。）

代码：

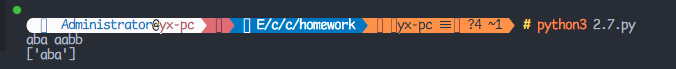
import re

input\_string =input()

three\_letter\_words = re**.**findall(r'\b\w{3}\b', input\_string)

print(three\_letter\_words)

截图：



实验三：Python 基础——面向对象、文件

处理

Python 从设计之初就已经是一门面向对象的语言，面向对象程序设计的思想主要针对大型软件设计而提出，使得软件设计更加灵活，能够很好地支持代码复用和设计复用，代码具有更好的可读性和可扩展性。

(一) 实验目的

1. 掌握面向对象的编程

2. 掌握 python 文件读写

(二) 实验环境

Python 3.6 及以上 or PyCharm

(三) 实验内容

1. 面向对象

2. 文件处理

(四) 实验步骤

1. 类的定义与使用

（1） 编程设计学生信息类，实例属性包括：学号、姓名、性别，年龄、n 门课程成绩，

要求：

1) 利用文件读取，创建一个包含 n 个学生的班级;

2) n 门课程成绩利用字典存储，支持成绩录入、修改;

3) 求解每个学生的三门成绩的平均值，及其平均值排名。并按照成绩平均成绩排

名正序输出功能：学号、姓名、性别、年龄，三门课程成绩，三门课程平均值，排

名。

代码：

import csv

class Student:

    def \_\_init\_\_(self, student\_id, name, gender, age, scores):

        self**.**student\_id = student\_id

        self**.**name = name

        self**.**gender = gender

        self**.**age = age

        self**.**scores = scores

    def average\_score(self):

        """计算三门课程的平均成绩"""

        if len(self**.**scores) == 0:

            return 0

        return sum(self**.**scores**.**values()) / len(self**.**scores)

    def \_\_str\_\_(self):

        return (f"{self**.**student\_id}, {self**.**name}, {self**.**gender}, {self**.**age}, "

                f"{self**.**scores}, 平均成绩: {self**.**average\_score():.2f}")

class Classroom:

    def \_\_init\_\_(self):

        """初始化班级列表，用于存储学生对象"""

        self**.**students = []

    def add\_student(self, student):

        """添加学生到班级列表"""

        self**.**students**.**append(student)

    def load\_students\_from\_file(self, file\_name):

        try:

            with open(file\_name, 'r') as file:

                reader = csv**.**reader(file)

                for row in reader:

                    student\_id, name, gender, age = row[:4]

                    age = int(age)

                    # *将课程成绩转换为字典*

                    scores = {}

                    for item in row[4:]:

                        course, score = item**.**split(':')

                        scores[course] = float(score)

                    # *创建学生对象并添加到班级*

                    student = Student(student\_id, name, gender, age, scores)

                    self**.**add\_student(student)

        except FileNotFoundError:

            print(f"未找到文件 {file\_name}")

    def update\_score(self, student\_id, course, score):

        for student in self**.**students:

            if student**.**student\_id == student\_id:

                student**.**scores[course] = score

                print(f"学号 {student\_id} 的课程 {course} 成绩更新为 {score}")

                return

        print(f"未找到学号为 {student\_id} 的学生")

    def calculate\_ranking(self):

        ranked\_students = sorted(self**.**students, key=lambda x: x**.**average\_score())

        print("班级学生按平均成绩排名：")

        for rank, student in enumerate(ranked\_students, start=1):

            print(f"排名 {rank} - 学号: {student**.**student\_id}, 姓名: {student**.**name}, "

                  f"性别: {student**.**gender}, 年龄: {student**.**age}, 成绩: {student**.**scores}, "

                  f"平均成绩: {student**.**average\_score():.2f}")

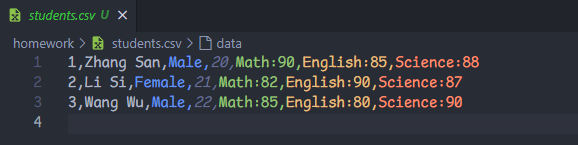
classroom = Classroom()

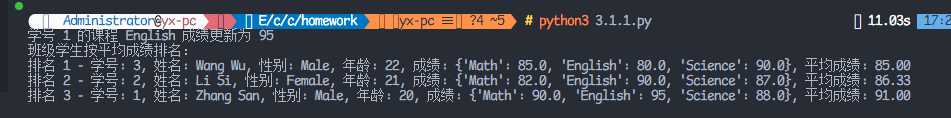
classroom**.**load\_students\_from\_file('students.csv')

classroom**.**update\_score('1', 'English', 95)

classroom**.**calculate\_ranking()

截图：





（2）编程设计一个雇员基类 Employee，包括姓名，编号，月薪三个实例属性，月薪计

算 pay()和信息显示 show()两个函数成员；派生两个子类 manager 类和 salesman 类，重

载相应的 2 个函数成员。

要求：根据以上描述设计类，并在主函数创建两个子类的实例化对象，分别调用其成员

方法。

代码：

class Employee:

    def \_\_init\_\_(self,name,num,basemoney):

        self**.**name = name

        self**.**num = num

        self**.**money = basemoney

    def pay(self):

        return self**.**base\_salary

    def show(self):

        print(f"编号: {self**.**num}, 姓名: {self**.**name}, 月薪: {self**.**pay():.2f}")

class Manager(Employee):

    def \_\_init\_\_(self,name,num,basemoney,addmoney):

        super()**.**\_\_init\_\_(name,num,basemoney)

        self**.**addmoney = addmoney

    def pay(self):

        return self**.**addmoney +2 \*self**.**money

    def show(self):

        print(f"【主管】编号: {self**.**num}, 姓名: {self**.**name}, 月薪: {self**.**pay():.2f}")

class Salesman(Employee):

    def \_\_init\_\_(self,name,num,basemoney,addmoney):

        super()**.**\_\_init\_\_(name,num,basemoney)

        self**.**addmoney = addmoney

    def pay(self):

        return 0.6 \*self**.**addmoney +self**.**money

    def show(self):

        print(f"【销售员】编号: {self**.**num}, 姓名: {self**.**name}, 月薪: {self**.**pay():.2f}")

m1 = Manager('ayy',1,1000,1000)

m1**.**show()

a1 =Salesman('Yyx',2,1000,1000)

a1**.**show()

截图：



（3）编程设计一个基类汽车类 Vehicle，包含最大速度 MaxSpeed，weight 两个实例私

有属性；设计一个派生子类自行车（Bicycle）类，增加 1 个实例私有属性高度（height）

和 1 个成员函数 SetMaxSpeed 实现给父类的实例属性 MaxSpeed 的赋值。

要求：

1) 根据以上描述设计类，并在主函数中创建子类的实例化对象，并设置对象的

MaxSpeed 值。

2) 利用 property 将 height 设定为可读、可修改、可删除的属性。

代码：

class Vehicle:

    def \_\_init\_\_(self,MaxSpeed,weight):

        self**.**\_\_max\_speed = MaxSpeed

        self**.**\_\_weight = weight

    def get\_max\_speed(self):

        return self**.**\_\_max\_speed

    def set\_max\_speed(self, speed):

        self**.**\_\_max\_speed = speed

    def get\_weight(self):

        return self**.**\_\_weights

class Bicycle(Vehicle):

    def \_\_init\_\_(self,MaxSpeed,weight,height):

        super()**.**\_\_init\_\_(MaxSpeed,weight)

        self**.**\_\_height = height

    def set\_max\_speed(self, speed):

        super()**.**set\_max\_speed(speed)

    def get\_max\_speed(self):

        return super()**.**get\_max\_speed()

    @property

    def height(self):

        return self**.**\_\_height

    @height**.**setter

    def height(self, value):

        self**.**\_\_height = value

    @height**.**deleter

    def height(self):

        del self**.**\_\_height

bike =Bicycle(100,10,2)

print (bike**.**get\_max\_speed())

bike**.**set\_max\_speed(120)

print (bike**.**get\_max\_speed())

截图：



（4） 编程设计一个队列类 Myqueue，主要的类成员包括： 3 个数据成员（队列的最

大长度 size，队列所有数据 data，队列的元素个数 current）和 6 个成员方法如下：

1) 初始化 ：设置队列为空;

2) 判断队列为空：若为空，则返回 TRUE，否则返回 FALSE.

3) 判断队列为满：若为满，则返回 TRUE，否则返回 FALSE.

4) 取队头元素：取出队头元素;

条件：队列不空。

否则，应能明确给出标识，以便程序的处理.

5) 入队：将元素入队，即放到队列的尾部

6) 出队：删除当前队头的元素

要求：根据以上描述设计类，并在主函数中创建类的实例化对象，构建一个长度为 N 的

队列，分别调用上述成员方法。

代码：

class Myqueue:

    def \_\_init\_\_(self, size):

        """

        初始化队列，设置最大容量 size 和当前元素计数 curr。

        """

        self**.**size = size  # *队列的最大大小*

        self**.**data = []    # *队列的数据存储*

        self**.**curr = 0     # *当前队列中的元素数*

    def empty(self) -> bool:

        return self**.**curr == 0

    def full(self) -> bool:

        return self**.**curr == self**.**size

    def top(self):

        if not self**.**empty():

            return self**.**data[0]  # *返回队列的第一个元素*

        else:

            raise IndexError("队列为空，无法获取队首元素")  # *抛出异常*

    def push(self, data):

        if not self**.**full():

            self**.**data**.**append(data)  # *添加元素到队列的末尾*

            self**.**curr += 1          # *更新队列的元素计数*

        else:

            raise OverflowError("队列已满，无法添加元素")  # *抛出异常*

    def pop(self):

        if not self**.**empty():

            self**.**curr -= 1          # *更新队列的元素计数*

            return self**.**data**.**pop(0)  # *移除并返回队列的第一个元素*

        else:

            raise IndexError("队列为空，无法弹出元素")  # *抛出异常*

N = 3

queue = Myqueue(N)

print(queue**.**empty())

print(queue**.**full())

queue**.**push(1)

print(queue**.**top())

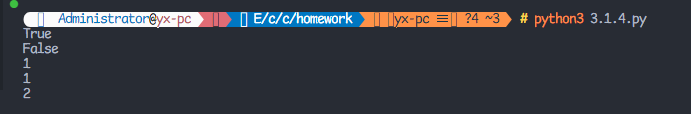
queue**.**push(2)

print(queue**.**top())

queue**.**pop()

print(queue**.**top())

截图：



2. 文件的使用

（1） 问题描述：编写程序，生成多个字符串，将字符串写入文件，同时读取当

前文件，并输出统计字符串的个数。

import random

import string

# *函数：生成随机字符串*

def generate\_random\_string(length=8):

    """生成一个由字母和数字组成的随机字符串"""

    letters\_and\_digits = string**.**ascii\_letters + string**.**digits

    return ''**.**join(random**.**choice(letters\_and\_digits) for i in range(length))

# *函数：将多个字符串写入文件*

def write\_strings\_to\_file(filename, num\_strings=10):

    """生成指定数量的字符串，并写入文件"""

    with open(filename, 'w') as file:

        for \_ in range(num\_strings):

            random\_string = generate\_random\_string()

            file**.**write(random\_string + '\n')

# *函数：读取文件并统计字符串的个数*

def count\_strings\_in\_file(filename):

    """读取文件内容，并统计其中的字符串个数"""

    with open(filename, 'r') as file:

        lines = file**.**readlines()  # *读取所有行*

        string\_count = len(lines)  # *行数等于字符串个数*

        return string\_count

# *主程序*

def main():

    # *文件名*

    filename = "strings.txt"

    # *生成并写入10个随机字符串到文件*

    write\_strings\_to\_file(filename, num\_strings=10)

    # *读取文件并统计字符串的个数*

    string\_count = count\_strings\_in\_file(filename)

    # *输出统计结果*

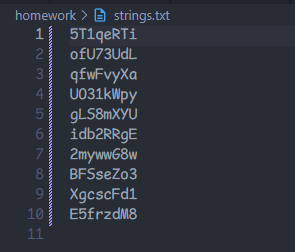
    print(f"文件中字符串的个数是: {string\_count}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

截图：





（2）编写程序以检查用户输入的密码的有效性。

检查密码的标准为：

1） [a-z]之间至少有 1 个字母

2） [0-9]之间至少有 1 个数字

3） [A-Z]之间至少有 1 个字母

4） [$＃@]中至少有 1 个字符

5） 最短交易密码长度：6

6） 交易密码的最大长度：12

问题描述： 程序接受一系列逗号分隔的密码，进行检查。再输出符合条件

的密码，每个密码用逗 号分隔。

例如：程序的输入： abcdEF12＃@,ccword12 程序的输出： abcdEF12

＃@

代码：

import re

# *函数：检查密码是否有效*

def is\_valid\_password(password):

    """检查密码是否符合要求"""

    if (6 <= len(password) <= 12 and                     # *检查密码长度*

        re**.**search("[a-z]", password) and                 # *至少有一个小写字母*

        re**.**search("[A-Z]", password) and                 # *至少有一个大写字母*

        re**.**search("[0-9]", password) and                 # *至少有一个数字*

        re**.**search("[$#@]", password)):                   # *至少有一个特殊字符*

        return True

    return False

# *函数：处理输入和输出有效密码*

def check\_passwords(passwords):

    """检查逗号分隔的密码，并返回符合条件的密码"""

    # *将输入的字符串按逗号分隔成列表*

    password\_list = passwords**.**split(',')

    # *使用列表推导式筛选有效密码*

    valid\_passwords = [password for password in password\_list if is\_valid\_password(password)]

    # *返回符合条件的密码，用逗号分隔*

    return ','**.**join(valid\_passwords)

# *主程序*

def main():

    # *输入一系列逗号分隔的密码*

    input\_passwords = input("请输入一系列逗号分隔的密码： ")

    # *检查并输出符合条件的密码*

    result = check\_passwords(input\_passwords)

    # *打印符合条件的密码*

    print("符合条件的密码为：", result)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

截图：

