**实验一 Python基础**

**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。 以及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、掌握判断结构的语法**

**2、掌握循环结构的语法**

**3、掌握函数编程**

**4、掌握变量与字符串**

**5、掌握列表与列表推导式**

**6、掌握元组与生成器推导式**

**7、掌握切片的使用**

**8、掌握字典的用法**

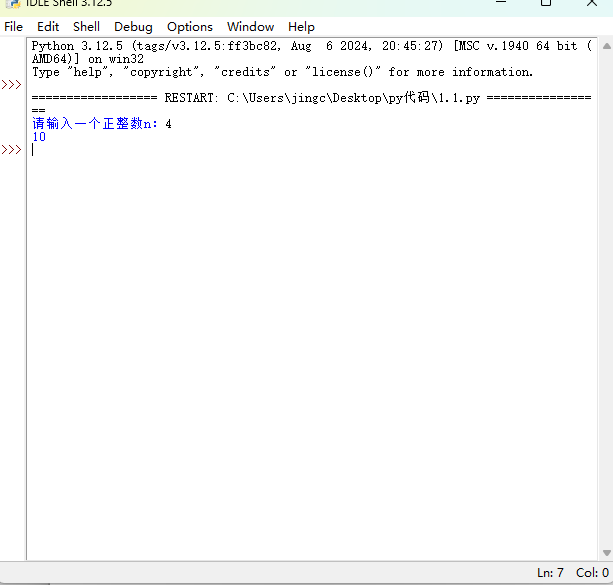
**9、掌握集合的用法**

**(二)实验环境 Python 3**

**(三) 实验内容**

1、用一行代码求和。 问题描述：输入一个正整数n，计算从1到n各数字的和，即计算1+2+3+...+n，要 求用一行代码实现要求（一行代码包含输入，计算，输出）。

print(sum(range(1, int(input("请输入一个正整数n：")) + 1)))



2、编写代码：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正整数，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。

import random

def guess\_the\_number():

# 设置随机数生成的范围

lower\_bound = 1

upper\_bound = 100

# 生成一个随机正整数

secret\_number = random.randint(lower\_bound, upper\_bound)

# 设置最大猜测次数

max\_attempts = 7

attempts = 0

print(f"欢迎来到猜数字游戏！")

print(f"我已经选择了一个在 {lower\_bound} 到 {upper\_bound} 之间的随机正整数。")

print(f"你有 {max\_attempts} 次猜测机会。")

while attempts < max\_attempts:

# 获取用户输入

guess = input(f"请输入你的猜测（第 {attempts + 1} 次猜测）：")

# 检查用户输入是否为数字

if not guess.isdigit():

print("请输入一个有效的正整数！")

continue

guess = int(guess)

attempts += 1

# 判断用户猜测的结果

if guess == secret\_number:

print("恭喜你！猜对了！")

break

elif guess < secret\_number:

print("太小了，请再试一次。")

else:

print("太大了，请再试一次。")

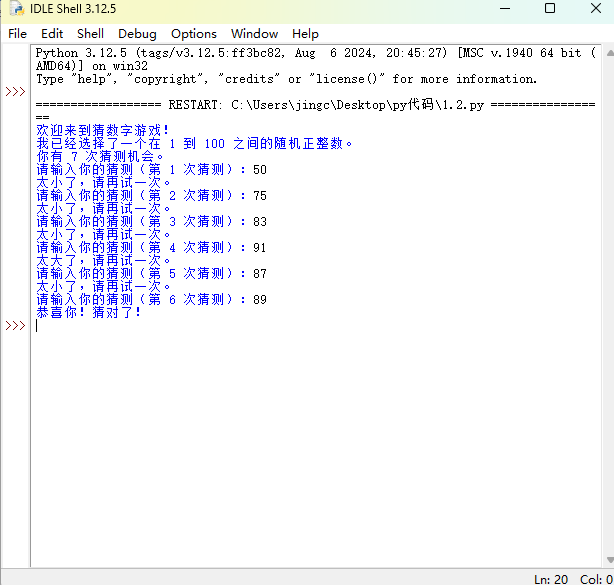
# 如果超过最大尝试次数仍未猜对

if attempts == max\_attempts:

print(f"游戏结束！正确答案是 {secret\_number}。")

# 运行游戏

guess\_the\_number()



3、有1、2、3、4个数字，能组成多少个互不相同且无重复数字的三位数？都是多 少？import itertools

# 所有可用的数字

digits = [1, 2, 3, 4]

# 生成所有互不相同且无重复数字的三位数

three\_digit\_numbers = list(itertools.permutations(digits, 3))

# 显示所有三位数

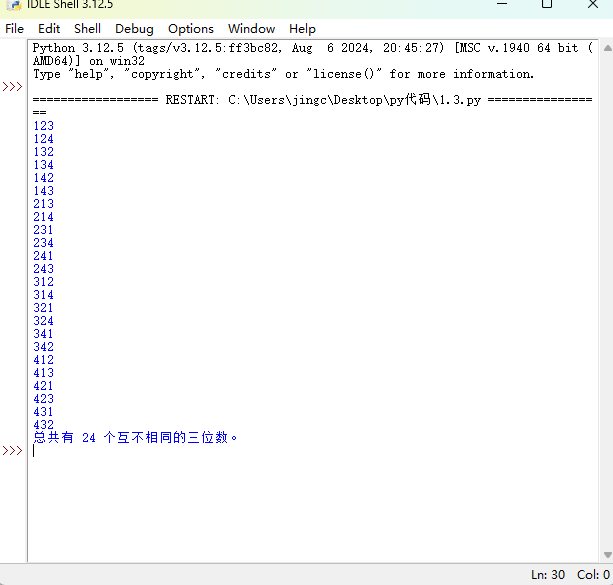
for number in three\_digit\_numbers:

print(''.join(map(str, number)))

# 统计总数

total\_count = len(three\_digit\_numbers)

print(f"总共有 {total\_count} 个互不相同的三位数。")



4、一个数如果恰好等于它的因子之和，这个数就称为"完数"。例如6=1＋2＋3，编 程找出1000以内的所有完数。

def is\_perfect\_number(n):

# 计算n的因子之和

if n < 2:

return False

factor\_sum = 1 # 从1开始，因为1是任何数的因子

for i in range(2, int(n\*\*0.5) + 1):

if n % i == 0:

factor\_sum += i

if i != n // i: # 处理平方数的情况

factor\_sum += n // i

return factor\_sum == n

def find\_perfect\_numbers(limit):

perfect\_numbers = []

for i in range(2, limit):

if is\_perfect\_number(i):

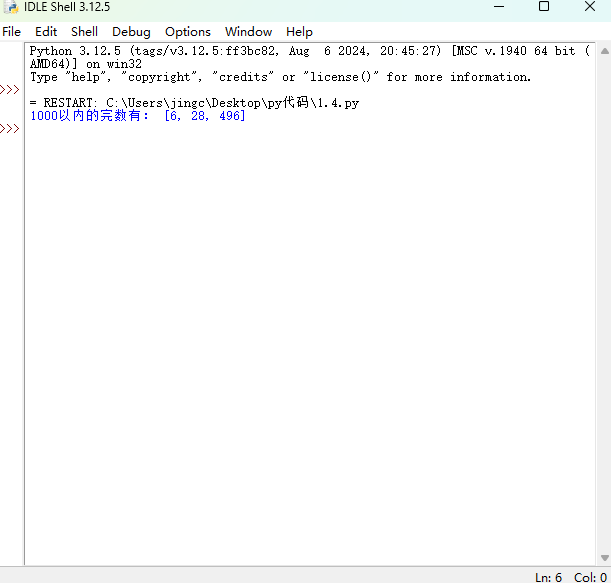
perfect\_numbers.append(i)

return perfect\_numbers

# 查找1000以内的完数

perfect\_numbers = find\_perfect\_numbers(1000)

print("1000以内的完数有：", perfect\_numbers)



5、编写程序，先生成一个包含50个随机整数的列表，然后删除其中所有奇数（提 示：从后向前删除）

import random

# 生成一个包含50个随机整数的列表，范围是1到100

random\_numbers = [random.randint(1, 100) for \_ in range(50)]

print("生成的随机整数列表：")

print(random\_numbers)

# 删除所有奇数，从后向前遍历

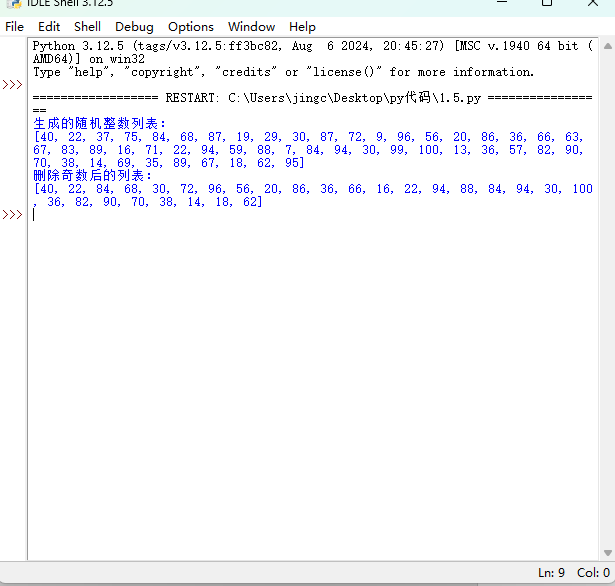
for i in range(len(random\_numbers) - 1, -1, -1):

if random\_numbers[i] % 2 != 0: # 判断是否为奇数

del random\_numbers[i]

print("删除奇数后的列表：")

print(random\_numbers)



6、编写程序，生成一个包含20个随机整数的列表，然后对其中偶数下标的元素进行 降序排列，奇数下标的元素不变。

import random

# 生成一个包含20个随机整数的列表，范围是1到100

random\_numbers = [random.randint(1, 100) for \_ in range(20)]

print("生成的随机整数列表：")

print(random\_numbers)

# 提取偶数下标的元素

even\_indexed\_elements = [random\_numbers[i] for i in range(0, len(random\_numbers), 2)]

# 对偶数下标的元素进行降序排序

even\_indexed\_elements.sort(reverse=True)

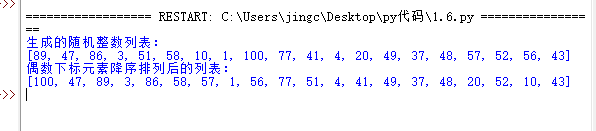
# 将排序后的偶数下标元素放回到原列表中

for i, value in enumerate(even\_indexed\_elements):

random\_numbers[i \* 2] = value

print("偶数下标元素降序排列后的列表：")

print(random\_numbers)



7、编写函数用来计算斐波那契数列中小于参数n的所有值。

def fibonacci\_less\_than(n):

# 定义斐波那契数列的初始化值

fibonacci\_sequence = []

a, b = 0, 1

# 生成斐波那契数列并添加到列表中，直到大于或等于 n

while a < n:

fibonacci\_sequence.append(a)

a, b = b, a + b # 更新斐波那契数列的下一个值

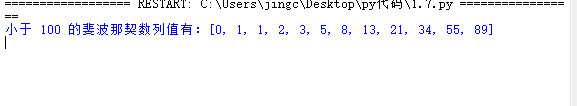
return fibonacci\_sequence

# 示例调用

n = 100

result = fibonacci\_less\_than(n)

print(f"小于 {n} 的斐波那契数列值有：{result}")



8、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。

def sieve\_of\_eratosthenes(n):

# 初始化一个布尔数组，初始假设所有数都是素数

is\_prime = [True] \* n

p = 2 # 从2开始，因为1不是素数

# 0和1不是素数

is\_prime[0] = is\_prime[1] = False

while p \* p < n:

# 如果是素数，则标记其倍数为非素数

if is\_prime[p]:

for i in range(p \* p, n, p):

is\_prime[i] = False

p += 1

# 收集所有素数

prime\_numbers = [i for i in range(n) if is\_prime[i]]

return prime\_numbers

# 获取用户输入

try:

num = int(input("请输入一个大于2的自然数："))

if num > 2:

primes = sieve\_of\_eratosthenes(num)

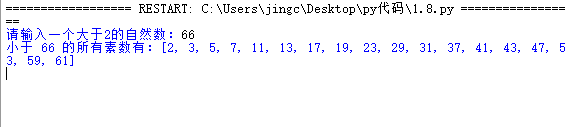
print(f"小于 {num} 的所有素数有：{primes}")

else:

print("请输入一个大于2的自然数。")

except ValueError:

print("请输入一个有效的自然数。")



9、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”

def is\_palindrome(s):

# 移除字符串中的空格并转换为小写

s = s.replace(" ", "").lower()

# 判断字符串是否与其反转相同

return s == s[::-1]

# 示例调用

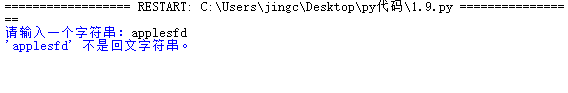
test\_string = input("请输入一个字符串：")

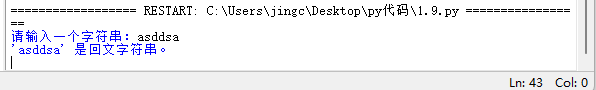
if is\_palindrome(test\_string):

print(f"'{test\_string}' 是回文字符串。")

else:

print(f"'{test\_string}' 不是回文字符串。")





10、对于给定的一个字符串，如果其中包含"apple"（apple不区分大小写)则输出 yes，否则输出no。

def contains\_apple(s):

# 将字符串转换为小写，并检查是否包含 "apple"

return "apple" in s.lower()

# 示例调用

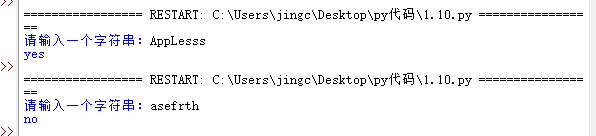
input\_string = input("请输入一个字符串：")

if contains\_apple(input\_string):

print("yes")

else:

print("no")



11、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）

import random

def generate\_random\_list\_and\_average(n):

# 随机生成包含n个整数的列表，范围为1到100

random\_list = [random.randint(1, 100) for \_ in range(n)]

# 打印生成的随机列表

print("生成的随机列表为：", random\_list)

# 计算平均值

average = sum(random\_list) / n

# 过滤出所有大于平均值的整数

above\_average = [x for x in random\_list if x > average]

# 返回元组，包含平均值和大于平均值的整数

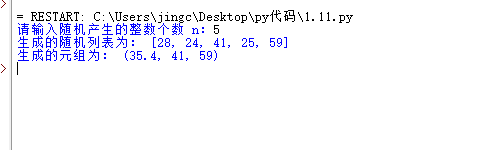
return (average, \*above\_average)

# 示例调用

n = int(input("请输入随机产生的整数个数 n："))

result = generate\_random\_list\_and\_average(n)

print("生成的元组为：", result)



12、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？

def calculate\_effort\_level(target\_effect, work\_days=260, rest\_days=105, rest\_penalty=0.01):

# 计算1%努力的目标水平

target\_level = target\_effect

# 二分法查找努力程度

low, high = 0, 1 # 假设努力程度在0到1之间

while low < high:

mid = (low + high) / 2

work\_level = mid \*\* work\_days # 工作日的努力效果

rest\_level = (1 - rest\_penalty) \*\* rest\_days # 休息日的效果

# 总效果

total\_effect = work\_level \* rest\_level

if total\_effect < target\_level:

low = mid + 1e-6 # 调整低值

else:

high = mid

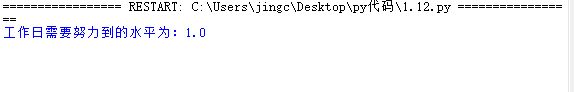
return round(low, 4)

# 计算努力程度

target\_effect = 1.01 \*\* 365 # 目标水平

effort\_level = calculate\_effort\_level(target\_effect)

print(f"工作日需要努力到的水平为：{effort\_level}")



13、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函数 def max\_overlap\_length(s1, s2):

# 计算s1的后缀和s2的前缀的最大重叠部分

return max(len(s1) - s1.rfind(s2[:i]) for i in range(1, len(s2) + 1) if s1.endswith(s2[:i]))

def combine\_strings(s1, s2):

# 使用lambda找出最大重叠长度

overlap = max\_overlap\_length(s1, s2)

# 将两个字符串连接，移除重叠部分

return s1 + s2[overlap:]

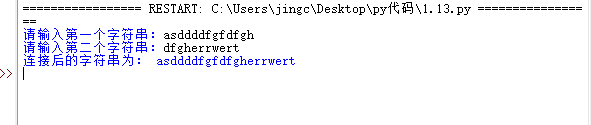
# 示例调用

str1 = input("请输入第一个字符串：")

str2 = input("请输入第二个字符串：")

result = combine\_strings(str1, str2)

print("连接后的字符串为：", result)



14、写一个删除列表中重复元素的函数，要求去重后元素相对位置保持不变。

def remove\_duplicates(lst):

seen = set() # 用于记录已经遇到的元素

unique\_list = [] # 存储去重后的元素

for item in lst:

if item not in seen:

seen.add(item) # 记录该元素

unique\_list.append(item) # 保留该元素

return unique\_list

# 示例调用，允许用户输入元素

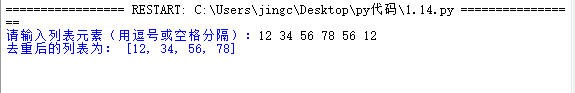
input\_str = input("请输入列表元素（用逗号或空格分隔）：")

# 将输入的字符串分割成列表并转换为整数

input\_list = [int(x.strip()) for x in input\_str.replace(',', ' ').split()]

result = remove\_duplicates(input\_list)

print("去重后的列表为：", result)



15、编写函数，计算传入函数的字符串中，数字、字母、空格以及其他字符的个数， 并返回结果。

def count\_character\_types(text):

digit\_count = 0 # 计算数字个数

letter\_count = 0 # 计算字母个数

space\_count = 0 # 计算空格个数

other\_count = 0 # 计算其他字符个数

for char in text:

if char.isdigit():

digit\_count += 1

elif char.isalpha():

letter\_count += 1

elif char.isspace():

space\_count += 1

else:

other\_count += 1

return {

'digits': digit\_count,

'letters': letter\_count,

'spaces': space\_count,

'others': other\_count

}

# 示例调用

input\_string = input("请输入一个字符串：")

result = count\_character\_types(input\_string)

print("统计结果：", result)



16、列表推导式与字典的应用 （1）.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。； （2）.问题描述：编写一个程序，接受逗号分隔的单词序列作为输入，按字母顺序排 序后，按逗号分隔的序列打印单词。假设向程序提供以下输入: without,hello,bag,world 则输出为: bag,hello,without,world

import random

import string

from collections import Counter

# 生成包含1000个随机字符的字符串

random\_string = ''.join(random.choice(string.ascii\_letters + string.digits) for \_ in range(1000))

# 统计每个字符出现的次数

char\_count = Counter(random\_string)

# 打印结果

print("生成的随机字符串：", random\_string)

print("每个字符出现的次数：", dict(char\_count))

# 接受逗号分隔的单词序列

input\_words = input("请输入逗号分隔的单词序列：")

# 使用列表推导式分割并排序

sorted\_words = sorted(word.strip() for word in input\_words.split(','))

# 按逗号分隔的序列打印单词

output = ','.join(sorted\_words)

print("排序后的单词序列：", output)



17、编写一个程序，接受一系列空格分隔的单词作为输入，然后删除所有重复的单词 并按字母数字排序后打印这些单词。

def process\_words(input\_string):

# 按空格分割单词，并使用set去重

words = set(input\_string.split())

# 将单词按字母顺序排序

sorted\_words = sorted(words)

# 返回排序后的单词

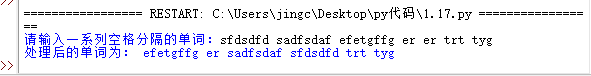
return ' '.join(sorted\_words)

# 接受用户输入

input\_string = input("请输入一系列空格分隔的单词：")

result = process\_words(input\_string)

print("处理后的单词为：", result)



18、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。

import random

import string

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

nested\_list = []

for \_ in range(n):

# 生成一个随机长度1到m的字符串

length = random.randint(1, m)

random\_string = ''.join(random.choices(string.ascii\_letters + string.digits, k=length))

nested\_list.append([random\_string]) # 每个元素是一个列表，包含这个字符串

# 按照字符串长度降序排序

nested\_list.sort(key=lambda x: len(x[0]), reverse=True)

return nested\_list

# 示例调用

n = int(input("请输入嵌套列表的元素个数 n："))

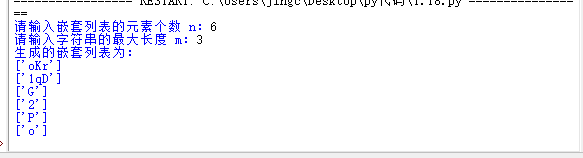
m = int(input("请输入字符串的最大长度 m："))

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

print("生成的嵌套列表为：")

for item in result:

print(item)



19、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。

import random

def generate\_integer\_list(length):

# 生成一个包含随机整数的列表

return [random.randint(1, 100) for \_ in range(length)]

def process\_list(lst):

# 输出原列表

print("原列表：", lst)

# 输出逆序列表

reversed\_list = lst[::-1]

print("逆序列表：", reversed\_list)

# 输出偶数位置的元素列表

even\_position\_elements = lst[::2]

print("偶数位置的元素列表：", even\_position\_elements)

# 示例调用

length = int(input("请输入列表的长度："))

integer\_list = generate\_integer\_list(length)

process\_list(integer\_list)

20、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。

import random

def generate\_tuple(n, m):

# 使用生成器推导式生成不大于 m 的随机整数，并过滤掉偶数

result\_tuple = tuple(num for num in (random.randint(1, m) for \_ in range(n)) if num % 2 != 0)

return result\_tuple

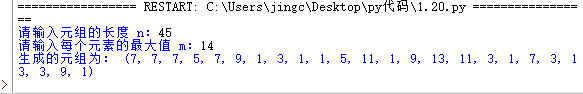
# 示例调用

n = int(input("请输入元组的长度 n："))

m = int(input("请输入每个元素的最大值 m："))

result = generate\_tuple(n, m)

print("生成的元组为：", result)



21、字典的应用 问题描述：编写程序，输入一段英文，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 apple apple”，输出：I：1 love: 1 apple: 2

def word\_count(text):

# 将输入文本转换为小写并分割成单词

words = text.split()

# 创建一个空字典用于存储单词及其计数

count\_dict = {}

# 遍历每个单词并统计出现次数

for word in words:

# 对单词进行处理，去掉标点符号（如果有的话）

word = word.strip().lower() # 转换为小写以避免大小写影响计数

if word in count\_dict:

count\_dict[word] += 1 # 如果单词已存在，则计数加1

else:

count\_dict[word] = 1 # 新单词，初始化为1

return count\_dict

# 示例调用

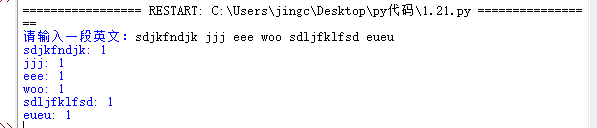
input\_text = input("请输入一段英文：")

result = word\_count(input\_text)

# 输出结果

for word, count in result.items():

print(f"{word}: {count}")



22、用户输入一段英文，使用正则表达式查找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。）

import re

def find\_three\_letter\_words(text):

# 使用正则表达式查找所有长度为3的单词

pattern = r'\b[a-zA-Z]{3}\b' # 匹配长度为3的单词（字母组成）

three\_letter\_words = re.findall(pattern, text)

return three\_letter\_words

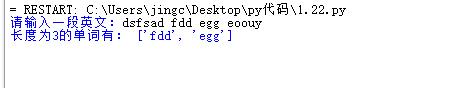
# 用户输入

input\_text = input("请输入一段英文：")

result = find\_three\_letter\_words(input\_text)

# 输出结果

print("长度为3的单词有：", result)



23、生成长度为n的字母与数字随机组成的字符串，使用正则表达式查找字符串中最 长的字母子串。 例如：3423ab45defgD字符串中的最长字符串为defgD

import random

import string

import re

def generate\_random\_string(n):

# 生成包含字母和数字的随机字符串

characters = string.ascii\_letters + string.digits # 包含大小写字母和数字

return ''.join(random.choice(characters) for \_ in range(n))

def find\_longest\_alpha\_substring(s):

# 使用正则表达式查找所有字母子串

pattern = r'[a-zA-Z]+' # 匹配字母组成的子串

alpha\_substrings = re.findall(pattern, s)

# 找到最长的字母子串

longest\_substring = max(alpha\_substrings, key=len) if alpha\_substrings else ''

return longest\_substring

# 示例调用

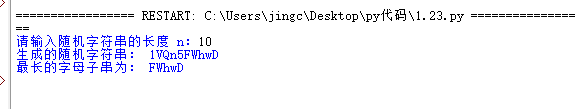
n = int(input("请输入随机字符串的长度 n："))

random\_string = generate\_random\_string(n)

print("生成的随机字符串：", random\_string)

longest\_alpha\_substring = find\_longest\_alpha\_substring(random\_string)

print("最长的字母子串为：", longest\_alpha\_substring)



24、将字符串："a:1\*a1:2\*a2:3\*a3:4"，处理成 python 字典：{'a':'1', 'a1':'2', 'a2':'3','a3':'4' }

def process\_string\_to\_dict(s):

# 首先通过'\*'分割字符串

pairs = s.split('\*')

# 使用字典推导式构建字典，': '进行分割

result\_dict = {key\_value.split(':')[0]: key\_value.split(':')[1] for key\_value in pairs}

return result\_dict

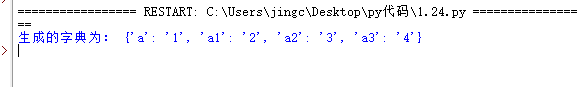
# 示例字符串

input\_string = "a:1\*a1:2\*a2:3\*a3:4"

result\_dict = process\_string\_to\_dict(input\_string)

# 输出结果

print("生成的字典为：", result\_dict)



25、用 Python 实现一个二分查找的函数。

def binary\_search(arr, target):

left, right = 0, len(arr) - 1

while left <= right:

mid = (left + right) // 2 # 计算中间索引

if arr[mid] == target: # 找到目标元素

return mid

elif arr[mid] < target: # 目标在右侧

left = mid + 1

else: # 目标在左侧

right = mid - 1

return -1 # 未找到目标元素

# 示例调用

sorted\_array = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

target\_value = int(input("请输入要查找的数字："))

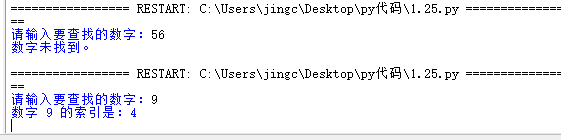
result = binary\_search(sorted\_array, target\_value)

if result != -1:

print(f"数字 {target\_value} 的索引是：{result}")

else:

print("数字未找到。")



26、一个列表中保存了一些整数，请输出列表中重复的数字，以及重复的次数。

def find\_duplicates(numbers):

count\_dict = {} # 用于存储数字及其出现的次数

# 统计每个数字的出现次数

for number in numbers:

if number in count\_dict:

count\_dict[number] += 1

else:

count\_dict[number] = 1

# 筛选出重复的数字

duplicates = {num: count for num, count in count\_dict.items() if count > 1}

return duplicates

# 示例调用

input\_numbers = [1, 2, 3, 4, 2, 3, 5, 1, 6, 1, 3]

result = find\_duplicates(input\_numbers)

# 输出结果

if result:

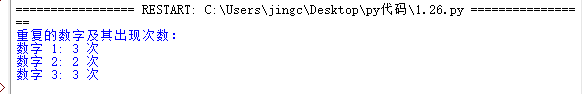
print("重复的数字及其出现次数：")

for num, count in result.items():

print(f"数字 {num}: {count} 次")

else:

print("没有重复的数字。")



27、输入一个不多于5位的正整数，请输出它的位数，以及逆序输出各位数字。

def process\_number(num):

# 确保输入是一个字符串，以便处理

num\_str = str(num)

# 获取位数

digit\_count = len(num\_str)

# 逆序输出各位数字

reversed\_digits = num\_str[::-1]

return digit\_count, reversed\_digits

# 用户输入

input\_number = int(input("请输入一个不多于5位的正整数："))

if 1 <= input\_number <= 99999: # 确保输入在有效范围内

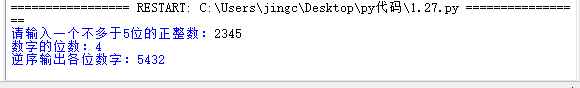
digit\_count, reversed\_digits = process\_number(input\_number)

print(f"数字的位数：{digit\_count}")

print(f"逆序输出各位数字：{reversed\_digits}")

else:

print("输入无效，请确保输入的是一个不多于5位的正整数。")



28、计算s=a+aa+aaa+aaaa+aa…a 的值，其中a是一个整数(1-9之间)。例如： s=2+22+222+2222+22222（此时共有5个数相加），几个数相加由用户输入。 例如，输入： a=2 n=5 表示计算 2+22+222+2222+22222

def calculate\_sum(a, n):

total\_sum = 0

for i in range(1, n + 1):

# 生成数字 aa..a (i个a)

current\_number = int(str(a) \* i) # 将a转换为字符串并重复i次，然后转换为整数

total\_sum += current\_number

return total\_sum

# 用户输入

a = int(input("请输入一个整数（1-9之间）："))

n = int(input("请输入要相加的数字个数："))

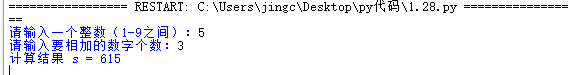
if 1 <= a <= 9 and n > 0:

result = calculate\_sum(a, n)

print(f"计算结果 s = {result}")

else:

print("输入无效，请确保 a 是 1-9 之间的整数，n 是大于0的整数。")



**实验二 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 3. 了解科学计算与可视化**

**(二) 实验环境 Python 3**

**(三) 实验内容**

29. 问题描述： 编写关于学生信息的类，实例属性包括：学号、姓名、性别，班级、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; 文件存储格式以空格分隔例如： 202001 李四 1班 高数:89 英语:28 计算机:77

2) n门课程成绩利用字典存储，支持成绩录入、修改与删除;

3) 按照学号或姓名查找，修订对应人员的课程成绩

4) 求解每个学生的n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：学号、姓名、性别、n门课程成绩，n门课程平均值，排名。

import operator

class Student:

def \_\_init\_\_(self, student\_id, name, gender, class\_name, grades):

self.student\_id = student\_id

self.name = name

self.gender = gender

self.class\_name = class\_name

self.grades = grades # 字典，存储课程成绩

def average\_grade(self):

return sum(self.grades.values()) / len(self.grades) if self.grades else 0

def update\_grade(self, course, score):

self.grades[course] = score

def delete\_grade(self, course):

if course in self.grades:

del self.grades[course]

def \_\_str\_\_(self):

avg = self.average\_grade()

grades\_str = ', '.join(f'{course}: {score}' for course, score in self.grades.items())

return f'{self.student\_id} {self.name} {self.gender} {self.class\_name} {grades\_str} 平均成绩: {avg:.2f}'

def read\_students\_from\_file(file\_path):

students = []

with open(file\_path, 'r', encoding='utf-8') as file:

for line in file:

parts = line.strip().split()

student\_id = parts[0]

name = parts[1]

class\_name = parts[2]

gender = parts[3]

grades = {}

for grade\_part in parts[4:]:

course, score = grade\_part.split(':')

grades[course] = int(score)

student = Student(student\_id, name, gender, class\_name, grades)

students.append(student)

return students

def find\_student(students, identifier):

for student in students:

if student.student\_id == identifier or student.name == identifier:

return student

return None

def display\_students(students):

for student in students:

print(student)

def display\_ranking(students):

students\_sorted = sorted(students, key=lambda x: x.average\_grade())

print("\n按平均成绩排名：")

for rank, student in enumerate(students\_sorted, start=1):

print(f'排名: {rank}, {student}')

def add\_student(students):

student\_id = input("请输入学号：")

name = input("请输入姓名：")

gender = input("请输入性别：")

class\_name = input("请输入班级：")

grades = {}

while True:

course = input("请输入课程名称（输入'结束'停止输入）：")

if course == '结束':

break

score = int(input(f"请输入 {course} 的成绩："))

grades[course] = score

student = Student(student\_id, name, gender, class\_name, grades)

students.append(student)

print("学生信息添加成功！")

def main():

file\_path = 'students.txt' # 请将此路径修改为你的文件路径

students = read\_students\_from\_file(file\_path)

while True:

print("\n菜单选项:")

print("1. 显示所有学生信息")

print("2. 按平均成绩排名")

print("3. 查找并修改学生成绩")

print("4. 添加学生信息")

print("5. 删除学生成绩")

print("0. 退出")

choice = input("请选择一个操作（0-5）：")

if choice == '1':

display\_students(students)

elif choice == '2':

display\_ranking(students)

elif choice == '3':

identifier = input("请输入要查找的学号或姓名：")

student = find\_student(students, identifier)

if student:

print("找到学生信息：", student)

course = input("请输入修改的课程名称：")

new\_score = int(input("请输入新的成绩："))

student.update\_grade(course, new\_score)

print("修改后的学生信息：", student)

else:

print("未找到该学生。")

elif choice == '4':

add\_student(students) # 调用添加学生信息的函数

elif choice == '5':

identifier = input("请输入要删除成绩的学生学号或姓名：")

student = find\_student(students, identifier)

if student:

course = input("请输入要删除的课程名称：")

student.delete\_grade(course)

print("删除成绩后的学生信息：", student)

else:

print("未找到该学生。")

elif choice == '0':

print("退出程序。")

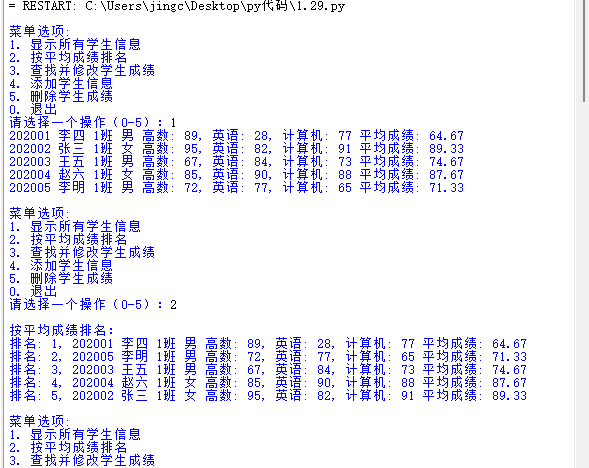
break

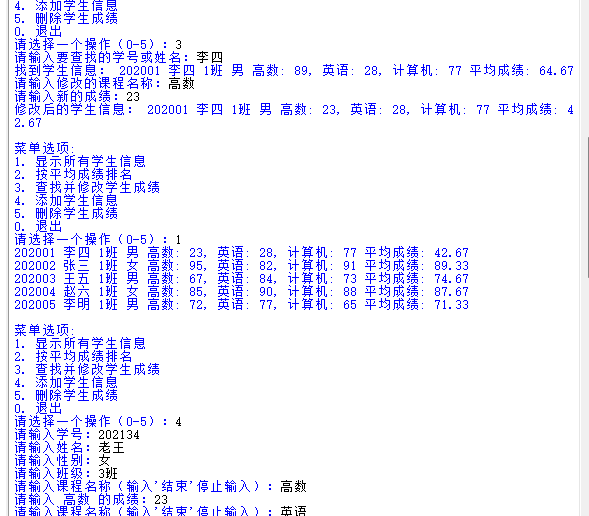
else:

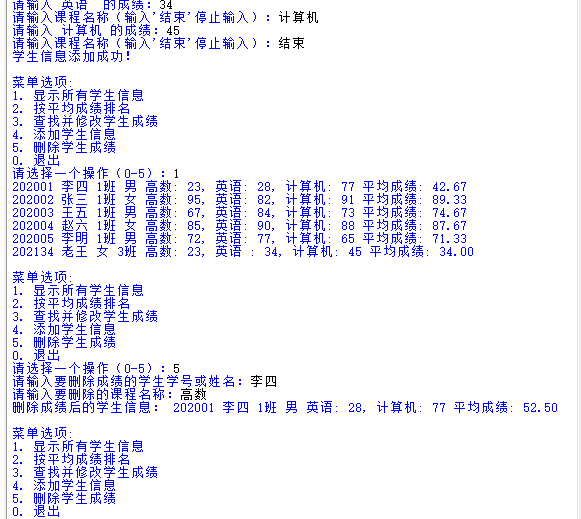
print("无效选择，请重新输入。")

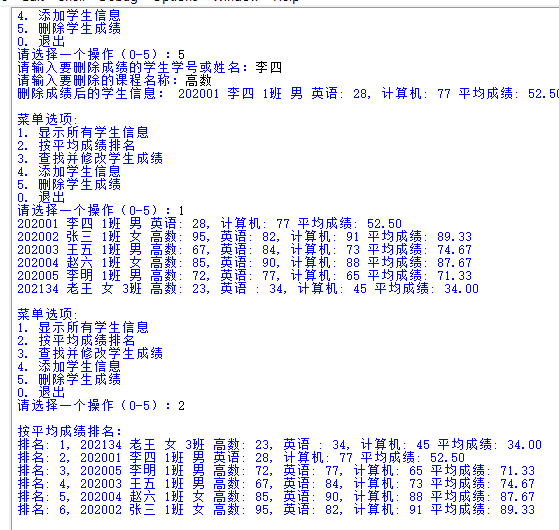
if \_\_name\_\_ == '\_\_main\_\_':

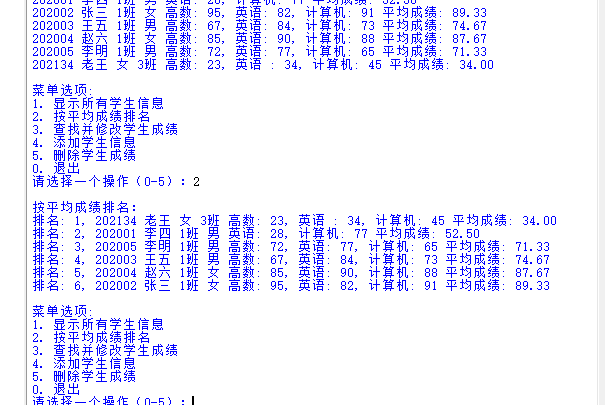
main()











30. 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。

class Employee:

def \_\_init\_\_(self, name, employee\_id, monthly\_salary):

self.name = name

self.employee\_id = employee\_id

self.monthly\_salary = monthly\_salary

def pay(self):

# 计算薪水

return self.monthly\_salary

def show(self):

# 显示员工信息

return f"姓名: {self.name}, 编号: {self.employee\_id}, 月薪: {self.pay()}"

class Manager(Employee):

def \_\_init\_\_(self, name, employee\_id, monthly\_salary, bonus):

super().\_\_init\_\_(name, employee\_id, monthly\_salary)

self.bonus = bonus # 经理的额外奖金

def pay(self):

# 经理薪水 = 月薪 + 奖金

return self.monthly\_salary + self.bonus

def show(self):

return f"经理 - {super().show()}, 奖金: {self.bonus}"

class Salesman(Employee):

def \_\_init\_\_(self, name, employee\_id, monthly\_salary, commission):

super().\_\_init\_\_(name, employee\_id, monthly\_salary)

self.commission = commission # 销售员的佣金

def pay(self):

# 销售员薪水 = 月薪 + 佣金

return self.monthly\_salary + self.commission

def show(self):

return f"销售员 - {super().show()}, 佣金: {self.commission}"

def main():

# 创建经理对象

manager = Manager("张经理", "001", 10000, 3000)

print(manager.show()) # 显示经理信息

print(f"经理薪水: {manager.pay()}") # 显示经理薪水

print() # 换行

# 创建销售员对象

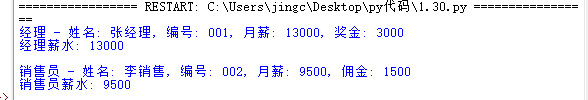
salesman = Salesman("李销售", "002", 8000, 1500)

print(salesman.show()) # 显示销售员信息

print(f"销售员薪水: {salesman.pay()}") # 显示销售员薪水

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

main()



31. 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、可修改的属性。

class Vehicle:

def \_\_init\_\_(self, max\_speed, weight):

self.\_\_max\_speed = max\_speed # 最大速度（私有属性）

self.\_\_weight = weight # 重量（私有属性）

@property

def max\_speed(self):

return self.\_\_max\_speed

@property

def weight(self):

return self.\_\_weight

class Bicycle(Vehicle):

def \_\_init\_\_(self, max\_speed, weight, height):

super().\_\_init\_\_(max\_speed, weight) # 调用父类构造方法

self.\_\_height = height # 高度（私有属性）

@property

def height(self):

return self.\_\_height

@height.setter

def height(self, height):

self.\_\_height = height

def set\_max\_speed(self, max\_speed):

self.\_Vehicle\_\_max\_speed = max\_speed # 直接访问私有属性

def main():

# 创建自行车对象

my\_bicycle = Bicycle(max\_speed=20, weight=15.5, height=1.1)

# 设置新的最大速度

my\_bicycle.set\_max\_speed(25)

# 输出属性值

print(f"最大速度: {my\_bicycle.max\_speed} km/h")

print(f"重量: {my\_bicycle.weight} kg")

print(f"高度: {my\_bicycle.height} m")

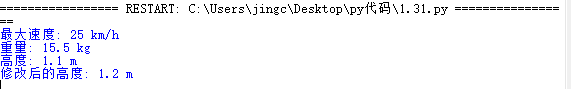
# 修改高度

my\_bicycle.height = 1.2

print(f"修改后的高度: {my\_bicycle.height} m")

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

main()



32. 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。

class MyQueue:

def \_\_init\_\_(self, size):

self.size = size # 队列的最大长度

self.data = [] # 存储队列元素

self.current = 0 # 当前元素个数

def is\_empty(self):

"""判断队列是否为空"""

return self.current == 0

def is\_full(self):

"""判断队列是否满"""

return self.current == self.size

def front(self):

"""取出队头元素"""

if self.is\_empty():

return "队列为空，无法取出队头元素"

return self.data[0]

def enqueue(self, element):

"""入队，将元素放到队列的尾部"""

if self.is\_full():

return "队列已满，无法入队"

self.data.append(element)

self.current += 1

def dequeue(self):

"""出队，删除当前队头元素"""

if self.is\_empty():

return "队列为空，无法出队"

element = self.data.pop(0) # 删除队头元素

self.current -= 1

return element

def main():

N = int(input("请输入队列的最大长度 N："))

queue = MyQueue(N)

# 测试队列的功能

print("初始化队列。")

# 检查是否为空

print("队列为空吗？", queue.is\_empty())

# 入队一些元素

for i in range(1, N + 1):

print(f"入队元素: {i}")

print(queue.enqueue(i))

# 再次检查是否满

print("队列满吗？", queue.is\_full())

# 取出队头元素

print("队头元素为:", queue.front())

# 出队一些元素

for \_ in range(2):

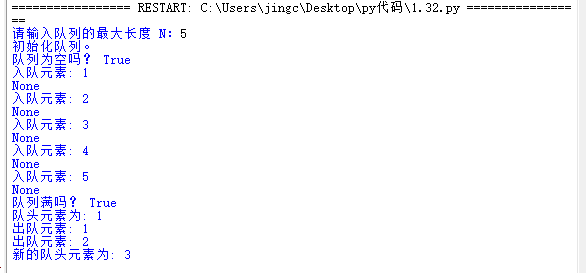
print("出队元素:", queue.dequeue()) # 出队两个元素

# 重新检查队头元素

print("新的队头元素为:", queue.front())

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

main()



33.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。

def write\_strings\_to\_file(file\_path, strings):

"""将字符串列表写入文件"""

with open(file\_path, 'w', encoding='utf-8') as file:

for string in strings:

file.write(string + '\n') # 每个字符串换行写入

def read\_strings\_from\_file(file\_path):

"""从文件中读取字符串，并返回字符串个数"""

with open(file\_path, 'r', encoding='utf-8') as file:

lines = file.readlines() # 读取所有行

return [line.strip() for line in lines] # 去除每行末尾的换行符

def main():

# 生成多个字符串

strings = ["Hello, World!", "Python is great.", "File handling in Python.", "Let's count strings.", "Goodbye!"]

# 文件路径

file\_path = 'strings.txt'

# 写入字符串到文件

write\_strings\_to\_file(file\_path, strings)

print(f"{len(strings)} 个字符串已写入到 {file\_path}。")

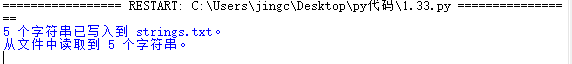
# 从文件读取字符串并统计数量

read\_strings = read\_strings\_from\_file(file\_path)

print(f"从文件中读取到 {len(read\_strings)} 个字符串。")

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

main()

34.假设有一个英文文本文件，编写程序读取其内容，并将其中的大写字母变为小写字母， 小写字母变为大写字母

def convert\_case(input\_file, output\_file):

try:

# 读取文本文件内容

with open(input\_file, 'r', encoding='utf-8') as file:

content = file.read()

# 反转大小写

converted\_content = content.swapcase()

# 将转换后的内容写入输出文件

with open(output\_file, 'w', encoding='utf-8') as file:

file.write(converted\_content)

print(f"成功反转了文件 {input\_file} 中的字母，并将结果写入到 {output\_file}。")

except FileNotFoundError:

print(f"文件 {input\_file} 不存在，请检查文件路径。")

except Exception as e:

print(f"发生错误: {e}")

def main():

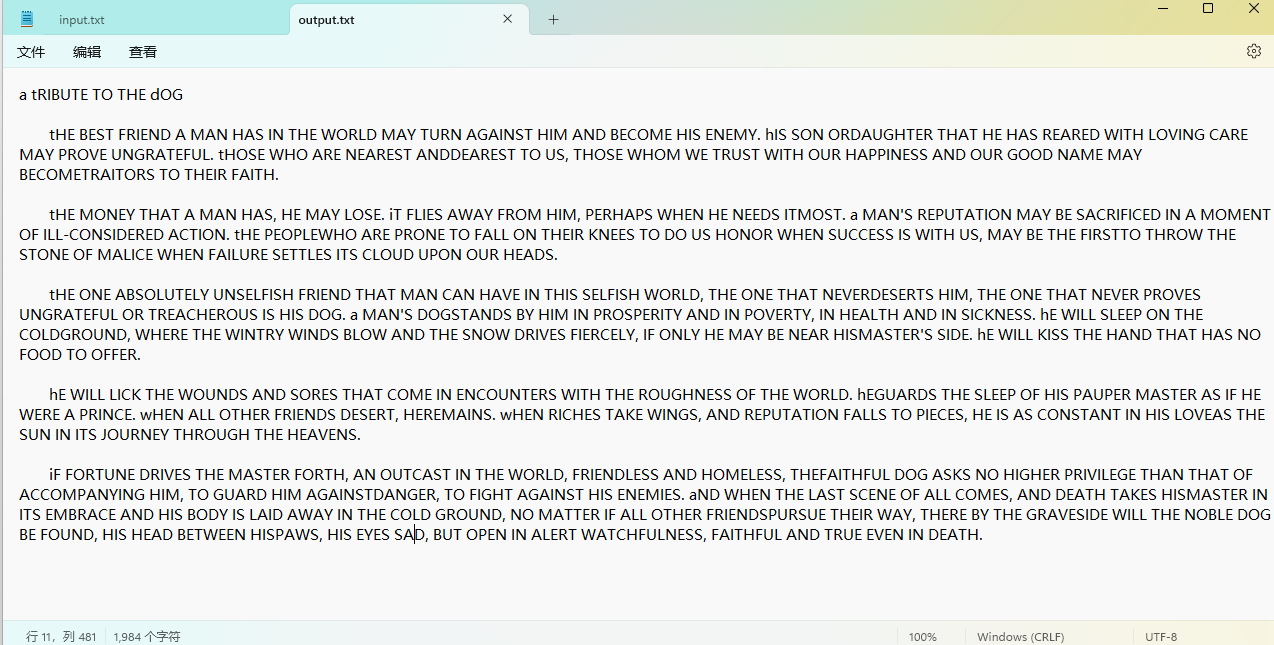
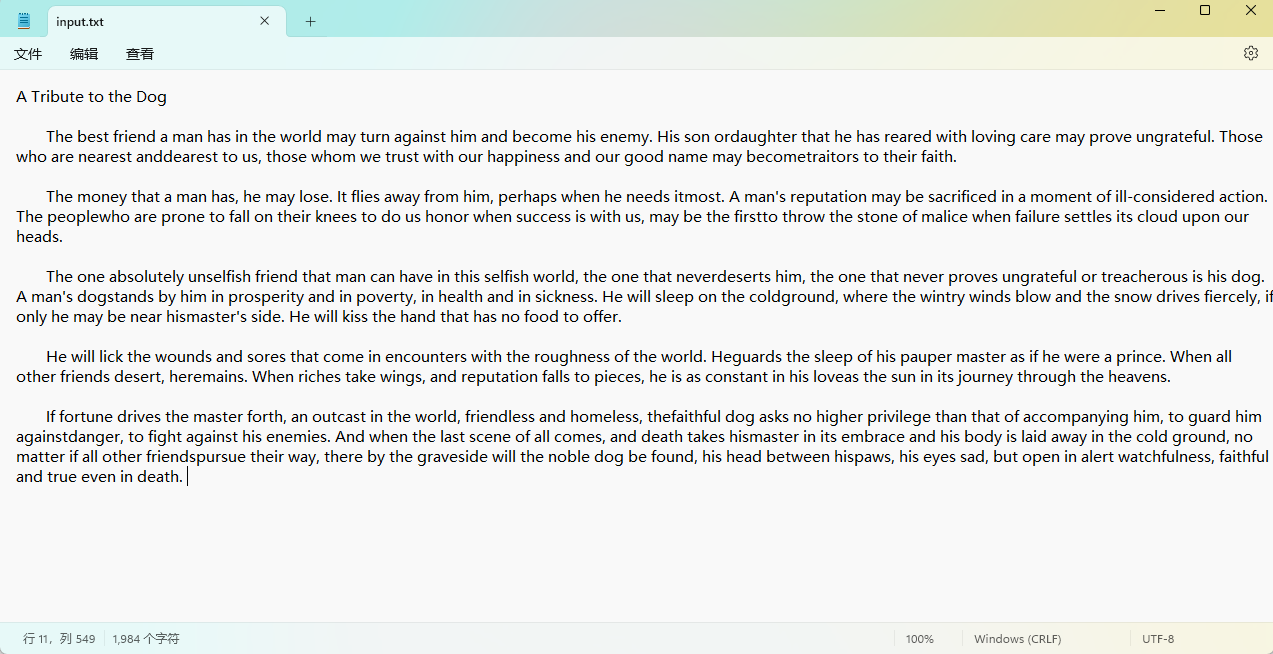
input\_file = 'input.txt' # 输入文件名

output\_file = 'output.txt' # 输出文件名

convert\_case(input\_file, output\_file)

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

main()



35.假设有一个文本文件的编码为 GBK，编写程序读取其内容，将其转成 UTF-8 编码保 存到另一个文件中。

def convert\_encoding(input\_file, output\_file):

try:

# 读取 GBK 编码的文件

with open(input\_file, 'r', encoding='gbk') as file:

content = file.read()

# 写入 UTF-8 编码的文件

with open(output\_file, 'w', encoding='utf-8') as file:

file.write(content)

print(f"成功将文件 {input\_file} 的编码从 GBK 转换为 UTF-8，并保存到 {output\_file}。")

except FileNotFoundError:

print(f"文件 {input\_file} 不存在，请检查文件路径。")

except Exception as e:

print(f"发生错误: {e}")

def main():

input\_file = 'input\_gbk.txt' # 输入文件名，假设为 GBK 编码

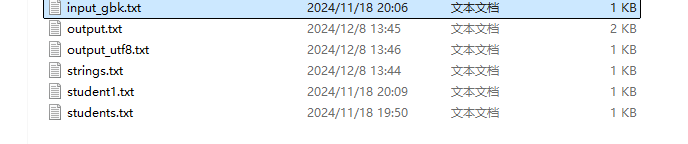
output\_file = 'output\_utf8.txt' # 输出文件名，保存为 UTF-8 编码

convert\_encoding(input\_file, output\_file)

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

main()





36.列出文件目录d:\data下的所有的文本文件（文件扩展名为.txt）

import os

def list\_text\_files(directory):

try:

# 获取目录下的所有文件

files = os.listdir(directory)

# 过滤出扩展名为.txt的文件

txt\_files = [file for file in files if file.endswith('.txt')]

return txt\_files

except FileNotFoundError:

print(f"目录 {directory} 不存在。")

return []

except Exception as e:

print(f"发生错误: {e}")

return []

def main():

directory = r'd:\data' # 指定目录

text\_files = list\_text\_files(directory)

if text\_files:

print("目录下的文本文件有：")

for file in text\_files:

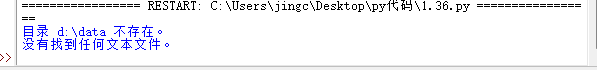
print(file)

else:

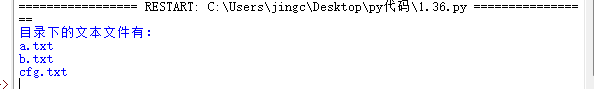
print("没有找到任何文本文件。")

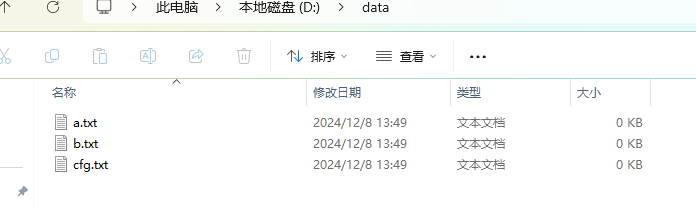
if \_\_name\_\_ == "\_\_main\_\_":

main()



我在D盘data 文件夹下新建了几个文件





37.假设当前目录下有一个文件名为student.txt的文本文件，存放着某班学生的学号（第 1 列）、数学成绩（第2列）和英语成绩（第3列），以空格分割各列数据。一行存放一 个学生的成绩。编写程序完成下列要求： （1）分别求出这个班数学成绩和英语成绩的最高分（保留1位小数）并输出。 （2）找出这个班两门课的平均成绩≥90分的学生，输出这些学生的学号、数学成绩和英 语成绩和平均成绩。

def read\_student\_data(file\_path):

"""读取学生数据，并返回学号、数学成绩和英语成绩的列表"""

students = []

try:

with open(file\_path, 'r', encoding='utf-8') as file:

for line in file:

# 以空格分割行数据

parts = line.strip().split()

if len(parts) == 3:

student\_id = parts[0]

math\_score = float(parts[1])

english\_score = float(parts[2])

students.append((student\_id, math\_score, english\_score))

except FileNotFoundError:

print(f"文件 {file\_path} 不存在，请确认文件路径。")

except Exception as e:

print(f"发生错误: {e}")

return students

def main():

file\_path = 'student1.txt' # 文件路径

# 读取学生数据

students = read\_student\_data(file\_path)

if not students:

print("没有读取到学生数据。")

return

# (1) 计算数学成绩和英语成绩的最高分

math\_scores = [score[1] for score in students]

english\_scores = [score[2] for score in students]

max\_math = max(math\_scores)

max\_english = max(english\_scores)

print(f"数学成绩最高分: {max\_math:.1f}")

print(f"英语成绩最高分: {max\_english:.1f}")

# (2) 找到平均成绩≥90的学生

print("\n平均成绩≥90的学生：")

for student in students:

student\_id, math\_score, english\_score = student

average\_score = (math\_score + english\_score) / 2

if average\_score >= 90:

print(f"学号: {student\_id}, 数学成绩: {math\_score:.1f}, 英语成绩: {english\_score:.1f}, 平均成绩: {average\_score:.1f}")

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

main()



38. 有n个人围成一圈（n的值由用户输入），顺序排号。从第一个人开始报数（从1到 3 报数），凡报到3的人退出圈子，问最后留下的是原来第几号的那位。

def last\_person\_standing(n):

# 创建一个列表，表示所有人的编号，从1到n

people = list(range(1, n + 1))

index = 0 # 从第一个人开始

while len(people) > 1:

# 每数到3就删除该位置的人

index = (index + 2) % len(people) # 这将对报数进行计算

print(f"退出的人是: {people[index]}")

people.pop(index) # 移除报到3的人

return people[0]

def main():

# 用户输入

n = int(input("请输入人数 n："))

if n <= 0:

print("人数必须为正整数。")

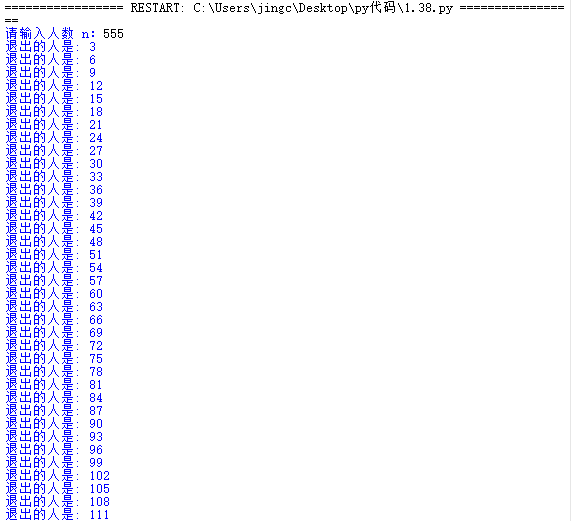
return

last\_person = last\_person\_standing(n)

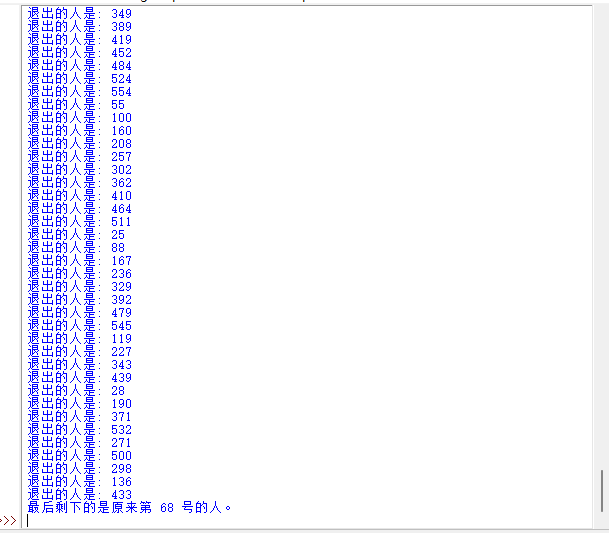
print(f"最后剩下的是原来第 {last\_person} 号的人。")

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

main()



中间太长就不放图片了



39.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

1. [A-Z]之间至少有一个字母

3. [$＃@]中至少有1个字符

4.最短交易密码长度：6

5.交易密码的最大长度：12

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＃@

import re

def is\_valid\_password(password):

if not (6 <= len(password) <= 12): # 检查长度

return False

if not re.search(r'[a-z]', password): # 检查至少一个小写字母

return False

if not re.search(r'[A-Z]', password): # 检查至少一个大写字母

return False

if not re.search(r'[0-9]', password): # 检查至少一个数字

return False

if not re.search(r'[$#@]', password): # 检查至少一个特殊字符

return False

return True # 如果通过所有检查，则返回 True

def main():

# 用户输入

input\_passwords = input("请输入多个逗号分隔的密码：")

passwords = input\_passwords.split(',')

# 检查每个密码的有效性并收集符合条件的密码

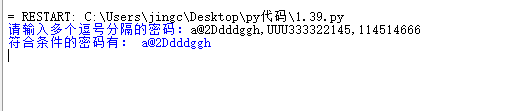
valid\_passwords = [password.strip() for password in passwords if is\_valid\_password(password.strip())]

# 输出符合条件的密码

print("符合条件的密码有：", ', '.join(valid\_passwords))

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

main()



40. 编写程序，接受一系列逗号分隔的4位二进制数作为输入，检查它们是否可被5整 除。 能被5整除的数字将以逗号分隔的顺序打印。

例如， 输入： 0100,0011,1010,1001

输出： 1010

def is\_divisible\_by\_5(binary\_str):

"""检查二进制字符串转换为十进制后是否能被5整除"""

decimal\_value = int(binary\_str, 2) # 将二进制字符串转换为十进制整数

return decimal\_value % 5 == 0 # 检查是否能被5整除

def main():

# 用户输入

input\_binaries = input("请输入一系列逗号分隔的4位二进制数：")

binary\_numbers = input\_binaries.split(',')

# 检查每个二进制数，并收集能被5整除的数

divisible\_by\_5 = [binary for binary in binary\_numbers if is\_divisible\_by\_5(binary.strip())]

# 输出能被5整除的二进制数

if divisible\_by\_5:

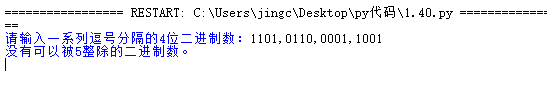
print("可以被5整除的二进制数是：", ', '.join(divisible\_by\_5))

else:

print("没有可以被5整除的二进制数。")

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

main()



41. 在一个坐标系中，绘出两个函数的图形，分别为y=sin(x)和y=cos(x)，x坐标取值范 围为(0, 2ￗ) 。

import numpy as np

import matplotlib.pyplot as plt

# 定义 x 的范围

x = np.linspace(0, 2 \* np.pi, 1000) # 生成 1000 个点

# 计算 y = sin(x) 和 y = cos(x)

y\_sin = np.sin(x)

y\_cos = np.cos(x)

# 创建图形

plt.figure(figsize=(10, 6))

# 绘制 y = sin(x)

plt.plot(x, y\_sin, label='y = sin(x)', color='blue')

# 绘制 y = cos(x)

plt.plot(x, y\_cos, label='y = cos(x)', color='orange')

# 添加图例

plt.legend()

# 添加标题和轴标签

plt.title('Plot of y = sin(x) and y = cos(x)')

plt.xlabel('x (radians)')

plt.ylabel('y')

# 设置 x 轴范围

plt.xticks(np.arange(0, 2 \* np.pi + 0.1, np.pi / 2),

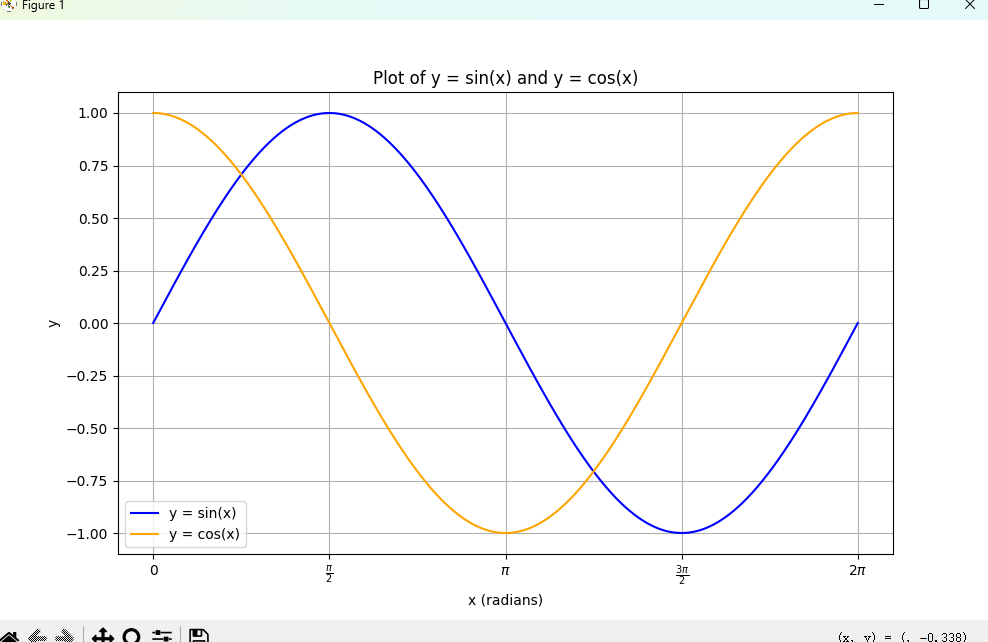
[r'0', r'$\frac{\pi}{2}$', r'$\pi$', r'$\frac{3\pi}{2}$', r'$2\pi$'])

# 添加网格

plt.grid()

# 显示图形

plt.show()



完成这些实验，我获得以下收获：

1. **掌握Python基础**：深入理解Python控制结构和函数设计，提升编程能力。

2. **操作数据结构**：熟练使用列表、元组等，增强数据处理技能。

3. **算法设计与实现**：通过编程问题，加深对算法的理解和应用。

4. **逻辑思维提升**：锻炼解决问题的逻辑思维能力。

5. **代码优化**：学习编写高效、可读的代码，提高代码质量。