得分：98
评语：作业完成认真，代码正确

python第7-11章作业

第七章作业

题目一:

创建银联信用卡类

```
class UnionPayCreditCard:
    def __init__(self, customer_name, credit_limit, current_limit,
single_transaction_limit):
        self.customer_name = customer_name
        self.credit_limit = credit_limit
        self.current_limit = current_limit
        self.single_transaction_limit = single_transaction_limit

    def get_customer_name(self):
        return self.customer_name

    def get_credit_limit(self):
        return self.credit_limit

    def get_current_limit(self):
        return self.current_limit

    def get_single_transaction_limit(self):
        return self.single_transaction_limit

    def modify_credit_limit(self, new_limit):
        if new_limit >= self.current_limit:
            self.credit_limit = new_limit
            print("授信额度修改成功。")
        else:
            print("新授信额度不能低于当前额度。")

    def modify_single_transaction_limit(self, new_limit):
        self.single_transaction_limit = new_limit
        print("单次刷卡金额上限修改成功。")
```

使用类创建一个信用卡实例

```
credit_card = UnionPayCreditCard("mcl", 10000, 10000, 5000)
```

获取属性值

```
print(credit_card.get_customer_name())  # 输出:mcl
print(credit_card.get_credit_limit())   # 输出: 10000
print(credit_card.get_current_limit())  # 输出: 10000
print(credit_card.get_single_transaction_limit())  # 输出: 5000
```

修改授信额度

```
credit_card.modify_credit_limit(12000)  # 输出: 授信额度修改成功。
```

修改单次刷卡金额上限

'''python credit_card.modify_single_transaction_limit(6000) # 输出: 单次刷卡金额上限修改成功。

题目二:

创建中国信用卡类

```
class BankOfChinaCreditCard(UnionPayCreditCard):
    def __init__(self, customer_name, credit_limit, current_limit,
single_transaction_limit, points=0):
        super().__init__(customer_name, credit_limit, current_limit,
single_transaction_limit)
        self.points = points
        self.preferred_shops = []

    def get_points(self):
        return self.points

    def set_preferred_shops(self, shops_list):
        self.preferred_shops = shops_list
        print("优惠店铺列表更新成功。")

    def swipe(self, shop_name, amount):
        if shop_name in self.preferred_shops:
            discounted_amount = amount * 0.95
            print(f"在优惠店铺 {shop_name} 消费，享受95折优惠。")
            print(f"原始消费金额: {amount}元，优惠后消费金额: {discounted_amount}
元。")
            amount = discounted_amount
        else:
            print(f"在店铺 {shop_name} 消费，无优惠。")

        # 每消费10元，信用卡积分增加1分
        self.points += amount // 10
        print(f"消费后积分: {self.points}分。")

        # 保留父类刷卡方法的其他功能
```

```
        super().current_limit -= amount  # 假设父类有一个current_limit属性来记录当前
额度

        # 检查是否超过单次交易限额
        if amount > self.single_transaction_limit:
            print("交易失败：超过单次交易限额。")
            return False
        else:
            print("交易成功。")
            return True
```

使用类创建一个中国银行信用卡实例

```
china_bank_card = BankOfChinaCreditCard("mcl", 10000, 10000, 5000)
```

获取属性值

```
print(china_bank_card.get_customer_name())  # 输出: mcl
print(china_bank_card.get_points())        # 输出: 0
```

设置优惠店铺列表

```
china_bank_card.set_preferred_shops(["星巴克", "肯德基"])
```

刷卡消费

```
china_bank_card.swipe("星巴克", 100)  # 星巴克在优惠店铺列表中，应享受95折优惠
china_bank_card.swipe("麦当劳", 100)  # 麦当劳不在优惠店铺列表中，无优惠
```

第八章作业

作业一：

读取文件并统计字符频次

```
import collections

def count_character_frequency(file_name):
    with open(file_name, 'r', encoding='utf-8') as file:
        char_frequency = collections.Counter(file.read())
```

```
    # 将结果保存到新文件
    with open(f'{file_name}_字符统计.txt', 'w', encoding='utf-8') as outfile:
        for char, frequency in char_frequency.items():
            outfile.write(f'{char}:{frequency},')
        # 去掉最后一个逗号
        outfile.seek(0)
        content = outfile.read()[:-1]  # 删除最后一个逗号
        outfile.truncate()
        outfile.write(content + '\n')
```

调用函数

```
count_character_frequency('笑傲江湖.txt')
```

作业二：

异常处理

```
def get_number_from_user():
    try:
        number = eval(input("请输入一个数字："))
        if not isinstance(number, (int, float)):
            raise ValueError("输入的不是数字")
        return number
    except ValueError as e:
        print(f"输入错误：{e}")
    except Exception as e:
        print(f"发生未知错误：{e}")
```

调用函数

```
get_number_from_user()
```

作业三：

1.创建screening_prime.py

```
def sieve_of_eratosthenes(n):
    """返回所有小于或等于n的素数"""
    primes = []
    sieve = [True] * (n + 1)
    for p in range(2, n + 1):
```

```
        if sieve[p]:
            primes.append(p)
            for i in range(p * p, n + 1, p):
                sieve[i] = False
    return primes
```

2.创建主python文件

```
# main.py

import screening_prime

def main():
    # 获取1000以内的所有素数
    primes = screening_prime.sieve_of_eratosthenes(1000)
    # 打印素数
    print("1000以内的素数有：")
    for prime in primes:
        print(prime, end=' ')
```

第九章作业

作业一：

1.代码如下：

```
def f1(ls=[]):
    ls.append(1)
    return ls

print(f1())  # 第一次调用
print(f1())  # 第二次调用
print(f1())  # 第三次调用
```

2.运行结果

```
[1]
[1, 1]
[1, 1, 1]
```

3.原因：

- 在 Python 中，默认参数只在函数定义时计算一次。因此，ls 参数在第一次调用 f1() 时被初始化为一个空列表 []，然后 1 被添加到这个列表中。
- 在后续的调用中，由于 ls 是一个可变对象（列表），它保持了上一次调用的状态。因此，每次调用 f1() 都是在同一个列表上进行操作，导致每次调用都在原有列表的基础上追加 1

作业二：

1. 列表推导实现素数过滤器

```
def is_prime(n):
    """判断数字n是否为素数"""
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

# 利用列表推导获得100以内的素数列表
prime_numbers = [n for n in range(2, 101) if is_prime(n)]
print(prime_numbers)
```

2.列表推导实现两列表对应元素的幂运算

```
x = [1, 2, 3, 4]
y = [0, 2, 3, 1]

# 利用列表推导获得新列表z
z = [pow(x_i, y_i) for x_i, y_i in zip(x, y)]
print(z)
```

3.使用条件表达式比较两数并赋值

```
x = 5
y = 10

# 使用条件表达式将x，y中的最大值赋值给z
z = x if x > y else y
print(z)
```

题目三：

1. 构造一个生产n以内素数的生成器

```python
def sieve_of_eratosthenes(n):
    primes = []
    sieve = [True] * (n + 1)
    for p in range(2, n + 1):
        if sieve[p]:
            primes.append(p)
            for i in range(p * p, n + 1, p):
                sieve[i] = False
    return primes

def prime_generator(n):
    for prime in sieve_of_eratosthenes(n):
        yield prime

# 验证100以内的素数
n = 100
prime_gen = prime_generator(n)
for _ in range(len(sieve_of_eratosthenes(n))):
    print(next(prime_gen))
```

2. 实现原计费函数charge()

```python
# 商品单价字典
product_prices = {"water": 1.5, "egg": 1, "meat": 15}

def charge(product, quantity):
    # 检查商品是否存在
    if product not in product_prices:
        raise ValueError("商品不存在")

    # 计算价格
    price = product_prices[product] * quantity
    if 6 <= quantity <= 10:
        price *= 0.95
    elif quantity > 10:
        price *= 0.9

    return price

# 增加中秋节优惠
def special_offer_charge(product, quantity):
    print("中秋节快乐!")
    original_price = charge(product, quantity)
    discounted_price = original_price * 0.8
    return discounted_price

# 测试charge函数
print(charge("water", 3))  # 不打折
print(charge("egg", 8))    # 95折
print(charge("meat", 11))  # 9折
```

题目一：

1.三门问题

```
import random

def monty_hall_simulation(num_trials):
    win_by_sticking = 0
    win_by_switching = 0

    for _ in range(num_trials):
        # 随机决定汽车在哪扇门后面
        car_behind = random.randint(0, 2)

        # 参赛者最初选择
        contestant_choice = random.randint(0, 2)

        if contestant_choice == car_behind:
            host_opens = random.choice([i for i in range(3) if i != 
contestant_choice])
        else:
            # 主持人打开一扇有山羊的门
            host_opens = [i for i in range(3) if i != contestant_choice and i != 
car_behind][0]

        # 参赛者不换门
        if contestant_choice == car_behind:
            win_by_sticking += 1

        # 参赛者换门
        new_contestant_choice = [i for i in range(3) if i != contestant_choice and 
i != host_opens][0]
        if new_contestant_choice == car_behind:
            win_by_switching += 1

    return win_by_sticking, win_by_switching

num_trials = 10000
wins_sticking, wins_switching = monty_hall_simulation(num_trials)
print(f"不换门的获胜概率: {wins_sticking / num_trials}")
print(f"换门的获胜概率: {wins_switching / num_trials}")
```

题目二：

2.24点问题

```
from itertools import permutations, product
```

```
def calculate(expression):
    try:
        return eval(expression)
    except Exception:
        return None

def is_close_to_24(num):
    return abs(num - 24) < 1e-6

def solve_24(nums):
    for nums_perm in permutations(nums):
        for ops in product('+-*/', repeat=3):
            expression = f"{nums_perm[0]}{ops[0]}{nums_perm[1]}{ops[1]}
{nums_perm[2]}{ops[2]}{nums_perm[3]}"
            result = calculate(expression)
            if is_close_to_24(result):
                return expression
    return None

nums = [4, 3, 2, 6]
solution = solve_24(nums)
if solution:
    print(f"24点问题的解: {solution} = 24")
else:
    print("无解")
```