ГУАП

КАФЕДРА № 41

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассистент		Е.К. Григорьев
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 7

ООП

по курсу: ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4116		Четвергов В.Ю.
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2023

Цель работы: познакомиться с основными способами объявления и использования генераторов в Python. Вариант 7

Напишите класс «Академическая группа», в который можно добавлять (и удалять) экземпляры класса «Студенты». В конструктор «Академической группы» следует передавать параметр максимального и минимального количества студентов в группе. Класс «Студенты» должен содержать параметры ФИО и дата поступления. У класса «Академическая группа» должны быть следующие методы: поиск студента по ФИО, количество студентов в группе. При этом если добавляемый студент превышает максимальное количество студентов в группе, он не добавляется в неё. Требуется реализовать возможность вывода текущих состояний объектов в терминал.

```
class Student:

    def __init__(self, full_name, enrollment_date):

        self.full_name = full_name

        self.enrollment_date = enrollment_date




class AcademicGroup:

    def __init__(self, min_students, max_students):

        self.min_students = min_students

        self.max_students = max_students

        self.students = []


    def add_student(self, student):
```

```
            if len(self.students) < self.max_students:
                self.students.append(student)
                print(f"Студент {student.full_name} добавлен в группу.")
            else:
                print("Достигнуто максимальное количество студентов в группе.")

        def remove_student(self, student):
            if student in self.students:
                self.students.remove(student)
                print(f"Студент {student.full_name} удален из группы.")
            else:
                print("Студент не найден в группе.")

        def find_student_by_name(self, name):
            for student in self.students:
                if student.full_name == name:
                    return student
            return None
```

```
        def count_students(self):
            return len(self.students)


    # основная част
    group = AcademicGroup(1, 5)  # Создание академической группы с минимум 1 студентом и максимум 5 студентами

    student1 = Student("Позднов Иван Чайковский", "01.06.2000")
    student2 = Student("Петров Петр Петрович", "01.02.1988")
    student3 = Student("Сидоров Евгений Сироткин", "01.03.2001")

    group.add_student(student1)  # Добавление студентов в группу
    group.add_student(student2)
    group.add_student(student3)

    print(f"Количество студентов в группе: {group.count_students()}")

    found_student = group.find_student_by_name("Петров Петр Петрович")  # Поиск студента по ФИО
    if found_student:
        print(f"Найден студент: {found_student.full_name}")
```

```
else:
    print("Студент не найден.")

group.remove_student(student2)  # Удаление студента из группы

print(f"Количество студентов в группе: {group.count_students()}")
```

.Напишите класс «Телефон», которому в конструктор передаются следующие данные: модель, мощность батареи и стоимость. Перегрузите у реализованного класса методы сравнения (сравнивать по стоимости), далее напишите класс, который находит телефон с максимальной и минимальной стоимостью из списка, способен вычислять сумму всех моделей телефонов и проводить их сортировку по возрастанию мощности. Список из телефонов можно передавать в конструктор реализуемого класса, также у класса должен быть метод для добавления и удаления одного телефона из списка. Требуется реализовать возможность вывода текущих состояний объектов в терминал

```
class Phone:
    def __init__(self, model, battery_power, cost):
        self.model = model
        self.battery_power = battery_power
        self.cost = cost

    def __eq__(self, other):
        return self.cost == other.cost

    def __lt__(self, other):
        return self.cost < other.cost
```

```
    def __gt__(self, other):
        return self.cost > other.cost

    def __repr__(self):
        return f"Phone(model={self.model}, battery_power={self.battery_power}, cost={self.cost})"


class PhoneCollection:
    def __init__(self, phones=None):
        self.phones = phones if phones is not None else []

    def add_phone(self, phone):
        self.phones.append(phone)
        print(f"Телефон {phone.model} добавлен в коллекцию.")

    def remove_phone(self, phone):
        if phone in self.phones:
            self.phones.remove(phone)
            print(f"Телефон {phone.model} удален из коллекции.")
        else:
            print("Телефон не найден в коллекции.")

    def find_max_cost_phone(self):
        if self.phones:
            max_cost_phone = max(self.phones, key=lambda phone: phone.cost)
```

```
            return max_cost_phone
        else:
            return None

    def find_min_cost_phone(self):
        if self.phones:
            min_cost_phone = min(self.phones, key=lambda phone: phone.cost)
            return min_cost_phone
        else:
            return None

    def calculate_total_cost(self):
        total_cost = sum(phone.cost for phone in self.phones)
        return total_cost

    def sort_by_battery_power(self):
        self.phones.sort(key=lambda phone: phone.battery_power)

    def display_collection(self):
        print("Текущая коллекция телефонов:")
        for phone in self.phones:
            print(phone)


# Пример использования классов
phone1 = Phone("iPhone 12", 3000, 1000)
phone2 = Phone("Samsung Galaxy S21", 4000, 900)
```

```
phone3 = Phone("Google Pixel 5", 3500, 800)

collection = PhoneCollection([phone1, phone2, phone3])

collection.add_phone(Phone("OnePlus 9", 3800, 950))

max_cost_phone = collection.find_max_cost_phone()
min_cost_phone = collection.find_min_cost_phone()

if max_cost_phone:
    print(f"Телефон с максимальной стоимостью: {max_cost_phone}")
else:
    print("Коллекция пуста.")

if min_cost_phone:
    print(f"Телефон с минимальной стоимостью: {min_cost_phone}")
else:
    print("Коллекция пуста.")

total_cost = collection.calculate_total_cost()
print(f"Общая стоимость телефонов в коллекции: {total_cost}")

collection.sort_by_battery_power()
print("Коллекция отсортирована по возрастанию мощности батареи:")
collection.display_collection()

collection.remove_phone(phone2)
```

```
print("Обновленная коллекция:")
collection.display_collection()
```

Вывод: я ознакомлен с основными способами работы в объектно — ориентированном программировании Python