План проекта: «Консольный планировщик задач на Python»

1. Планирование структуры (около 1 часа)

- **Классы (Python)**
  - **Task (Класс Задачи)**
    - **Поля:**
      - **task_id (тип int): Уникальный идентификатор задачи.**
      - **name (тип str): Название задачи (аналогично title из документа Plan.pdf).**
      - **priority (тип int): Приоритет задачи (целое число от 1 до 5).**
      - **status (тип str): Статус задачи, значения "не выполнено" или "выполнено" (аналогично isDone в документе Plan.pdf).**
    - **Методы:**
      - **__init__(self, task_id, name, priority, status="не выполнено"): Конструктор для инициализации полей задачи (аналогично конструктору с параметрами в документе Plan.pdf).**
      - **__str__(self): Возвращает строковое представление задачи для форматированного вывода (аналогично toString() в документе Plan.pdf).**
      - **update_status(self, new_status): Метод для изменения статуса задачи.**
  - **TaskScheduler (Класс Менеджера Задач)**
    - **Поля:**
      - **tasks (тип list): Список, хранящий объекты Task (аналогично vector<Task> в документе Plan.pdf).**
      - **next_id (тип int): Счетчик для генерации следующего уникального ID задачи (аналогично nextId в документе Plan.pdf).**
    - **Методы:**

- **add_task(self): Добавляет новую задачу. Запрашивает у пользователя название и приоритет. ID генерируется автоматически, статус по умолчанию "не выполнено" (функционально соответствует addTask(string title, int priority) из документа Plan.pdf).**
- **delete_task(self): Удаляет задачу по ID, который вводит пользователь (функционально соответствует removeTask(int id) из документа Plan.pdf).**
- **change_task_status(self): Изменяет статус существующей задачи по ID (функционально соответствует changeStatus(int id, bool status) из документа Plan.pdf).**
- **show_tasks(self, tasks_to_show=None, header=...): Отображает список задач. Может отображать все задачи или отфильтрованный список (функционально соответствует showAll() из документа Plan.pdf).**
- **filter_tasks_by_status(self): Фильтрует задачи по статусу (выполненные/невыполненные) и отображает их (функционально соответствует showByStatus(bool status) из документа Plan.pdf).**
- **sort_tasks_by_priority(self): Сортирует задачи по приоритету (по убыванию) и отображает их (использует Python sorted(), что аналогично std::sort из документа Plan.pdf).**
- **_find_task_by_id(self, task_id): Вспомогательный метод для поиска задачи по ID (внутренняя реализация).**

2. Программирование (около 5 часов)

- **Основные блоки кода (Python)**
  - **Консольное меню (реализовано в функции main_menu):**
    - **В цикле while True пользователю отображается список доступных команд: 1. Добавить, 2. Показать все, 3. Удалить, 4. Изменить статус, 5. Фильтр по статусу, 6. Сортировка по приоритету, 0. Выход.**

- **Считывается ввод пользователя (input()) и в зависимости от выбора вызываются соответствующие методы объекта TaskScheduler.**

- **Добавление задачи (метод add_task в TaskScheduler):**
  - **Запрашивается ввод названия задачи и ее приоритета.**
  - **Выполняется проверка корректности введенного приоритета (должен быть числом от 1 до 5).**
  - **Создается новый объект Task с использованием self.next_id для ID, который затем инкрементируется (self.next_id += 1). Статус по умолчанию устанавливается как "не выполнено".**
  - **Задача добавляется в список self.tasks.**
- **Удаление задачи (метод delete_task в TaskScheduler):**
  - **Запрашивается ID задачи для удаления.**
  - **Осуществляется поиск задачи по ID в списке self.tasks.**
  - **Если найдена, задача удаляется из списка (self.tasks.remove()).**
- **Изменение статуса (метод change_task_status в TaskScheduler):**
  - **Запрашивается ID задачи для изменения статуса.**
  - **Осуществляется поиск задачи по ID.**
  - **Если найдена, пользователю предлагается выбрать новый статус ("выполнено" или "не выполнено"), который затем присваивается атрибуту status найденной задачи.**
- **Показ задач (методы show_tasks, filter_tasks_by_status, sort_tasks_by_priority в TaskScheduler):**
  - **Все задачи: Метод show_tasks() без дополнительных параметров выводит все задачи из списка self.tasks.**
  - **Фильтрация по статусу: Метод filter_tasks_by_status() запрашивает у пользователя, какие задачи показать (выполненные или невыполненные), фильтрует список self.tasks и передает результат в show_tasks().**

        - **Сортировка по приоритету: Метод sort_tasks_by_priority() создает новый отсортированный список задач из self.tasks с помощью функции sorted() с ключом по полю priority (по убыванию) и передает результат в show_tasks().**

3. Тестирование и оформление (около 2 часов)

- **Проверка всех функций по чек-листу изначального задания (добавление, удаление, смена статуса, фильтрация, сортировка).**
- **Создание README.docx (или README.txt):**
    - **Краткое описание функционала приложения.**
    - **Инструкции по запуску: как сохранить код в .py файл и запустить его командой python <имя_файла>.py.**
    - **Пример работы: текстовый вывод из консоли, демонстрирующий основные операции.**