Министерство науки и высшего образования Российской Федерации федеральное государственное автономное образовательное учреждение

высшего образования

«Санкт-Петербургский политехнический университет Петра Великого» (ФГАОУ ВО СПбПУ)

Физико-механический институт

Высшая школа теоретической механики и математической физики

# КУРСОВАЯ РАБОТА

По дисциплине «Языки программирования»

Планер с оптимальным распределением задач

(семестр 4)

|  |  |
| --- | --- |
| Студент  группы  5030103/20003 |  |
| подпись, дата | инициалы и фамилия |

Оценка выполненной студентом работы:

|  |  |
| --- | --- |
| Преподаватель,  должность, уч. степ., уч. зв. |  |
| подпись, дата | инициалы и фамилия |

Санкт-Петербург – 2024

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc167898617)

[Определение основных свойств задачи 4](#_Toc167898618)

[Модульная структура программы 5](#_Toc167898619)

[Принцип сортировки и распределения 8](#_Toc167898620)

[Анализ результатов 10](#_Toc167898621)

[Заключение 11](#_Toc167898622)

[Список использованных источников 12](#_Toc167898623)

# Введение

В современном мире управление задачами и планирование времени стали неотъемлемой частью повседневной жизни. Эффективное распределение времени позволяет не только повысить продуктивность, но и избежать стрессовых ситуаций, связанных с нехваткой времени на выполнение важных дел. Несмотря на обилие готовых решений для планирования задач, часто возникают ситуации, когда стандартные инструменты не удовлетворяют уникальные потребности пользователя. В таких случаях на помощь приходит возможность создания собственного алгоритма, адаптированного под индивидуальные требования.

Настоящая курсовая работа посвящена разработке планера задач на языке программирования Python, предназначенного для личного использования. Программа была создана с учетом потребностей автора в управлении своим временем и задачами. Основной целью разработки данного планера является облегчение процесса планирования на некоторый промежуток времени, а также подбор оптимального расписания для выполнения всех задач в срок и эффективного использования свободного времени.

Необходимость в создании такого планера возникла из-за желания иметь инструмент, который бы учитывал не только дедлайны задач, но и их приоритетность, обязательность выполнения, а также индивидуальные интересы. Коммерческие решения часто предлагают общие функции, которые не всегда подходят для специфических сценариев использования. В данном случае, все свойства задач, такие как дедлайн, приоритет и трудоемкость, были определены исходя из собственных потребностей.

# Определение основных свойств задачи

Для создания эффективной программы, важно правильно определить и учитывать основные свойства задач. Рассмотрим задачу "Подготовиться к контрольной по дифференциальным уравнениям" как пример для объяснения выбора этих свойств.

Пример задачи

Название задачи: Подготовиться к контрольной по дифференциальным уравнениям

Deadline: 02.01.2024

Interest: 3

Work Hours: 4

Must Do: True

*name*: название задачи позволяет легко идентифицировать и различать задачи. Это свойство необходимо для пользователя, чтобы понимать, о какой задаче идет речь без необходимости вспоминать детали.

*deadline:* срок выполнения определяет временные рамки, в которые задача должна быть завершена. Это свойство критично для правильного распределения задач по времени и определения приоритетов.

*interest:* свойство позволяет выбрать задачу для выполнения в рамках свободного времени.

*work\_hours:* тег времени, необходимого для выполнения задачи, позволяет правильно распределить рабочее время и избежать перегрузки или просрочки.

*must\_do:* флаг обязательности выполнения помогает выделить критические задачи, которые необходимо выполнить в первую очередь, независимо от других факторов.

# Модульная структура программы

Программа состоит из нескольких модулей и функций, которые реализуют различные аспекты сортировки задач по приоритетам и их распределения по календарю. Разделим программу на несколько ключевых модулей:

* Функция *smart\_sort\_v1(tasks, calendar) -> {int: int}:*

Основная функция, выполняющая умную сортировку задач с учётом дедлайнов и приоритетов.

* Вложенные функции:

*calc\_mustdo\_coef(must\_do: bool)*: рассчитывает коэффициент важности задачи.

*create\_id\_hours\_dict(local\_tasks)*: создаёт словарь, связывающий идентификаторы задач с количеством часов до их дедлайнов.

*substract\_hours(id\_hours\_dict: {}, hrs: int):* вычитает указанное количество часов из словаря.

*chck\_upcoming\_ddlns(hrs\_ddln\_dct: {int: int}):* проверяет наличие предстоящих дедлайнов (в этой функции используется рекурсия).

* Функция *allocate\_work\_hours(work\_hours\_list, calendar):*

Функция распределяет рабочие часы по календарю на основе списка рабочих часов для задач.

* Функция *validate\_allocation(calendar, tasks):*

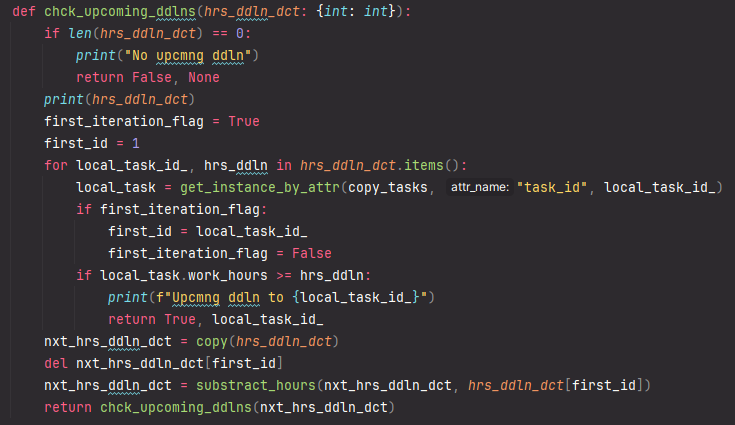
Функция валидирует распределение задач по календарю, возвращая идентификаторы задач, которые не удалось завершить до дедлайна.

* Функция *contains\_must\_do(failed\_tasks):*

Функция проверяет, содержатся ли среди невыполненных задач обязательные к выполнению.

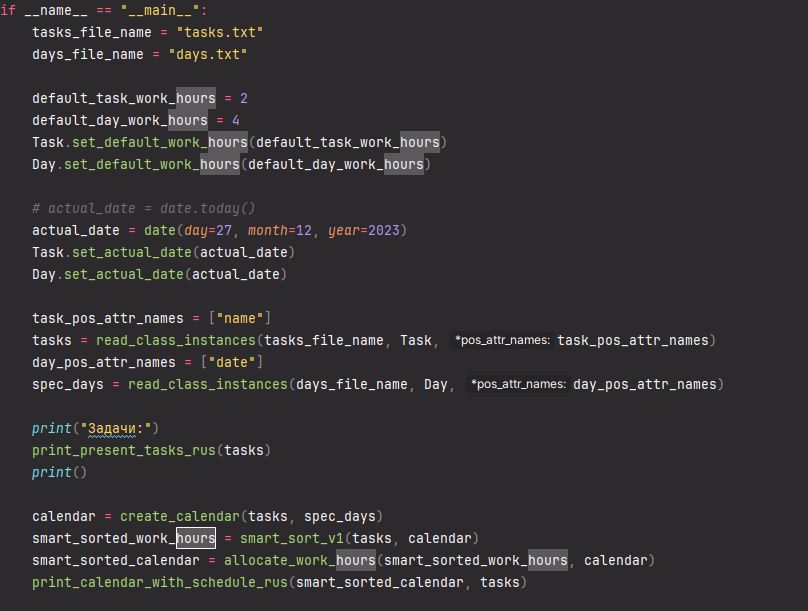
* Функция *chck\_upcoming\_ddlns(hrs\_ddln\_dct: {int: int}):*

Эта функция проверяет наличие задач с предстоящими дедлайнами и рекурсивно вызывает саму себя для обработки оставшихся задач.



*Рисунок 1.Часть кода алгоритма.*

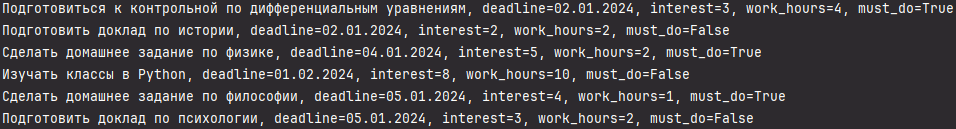
Основной поток программы начинается с вызова *smart\_sort\_v1(tasks, calendar)*, которая формирует список рабочих часов для задач *work\_hours\_list.* Далее, *work\_hours\_list* передаётся в *allocate\_work\_hours(work\_hours\_list, calendar)* для распределения по календарю. После распределения, результат проверяется функцией *validate\_allocation(calendar, tasks)* на наличие задач, не завершённых в срок. Дополнительные функции *(print\_calendar\_with\_schedule\_rus, print\_present\_tasks\_rus)* используются для вывода информации и визуализации данных.



*Рисунок 2. Главная вызывающая функция.*

# Принцип сортировки и распределения

Рассмотрим пример с шестью задачами и поясним, как алгоритм сортировки и распределения задач работает на этом наборе данных.



*Рисунок 3. Пример входных данных.*

Предположим, что текущая дата 27.12.2023. Рассчитаем количество часов до дедлайна для каждой задачи:

Задача 1: 144 часа (до 02.01.2024)

Задача 2: 144 часа (до 02.01.2024)

Задача 3: 264 часа (до 04.01.2024)

Задача 4: 792 часа (до 01.02.2024)

Задача 5: 288 часов (до 05.01.2024)

Задача 6: 288 часов (до 05.01.2024)

При помощи *srt\_hours\_to\_deadline\_dict* отсортируем задачи по часам до дедлайна и важности и получим отсортированный список:

Задача 1: (144 часа, True, 3)

Задача 2: (144 часа, False, 2)

Задача 3: (264 часа, True, 5)

Задача 5: (288 часов, True, 4)

Задача 6: (288 часов, False, 3)

Задача 4: (792 часа, False, 8)

Далее отсортируем по интересу и преобразуем в словарь с помощью *srt\_interest\_dict*

Задача 4: (8, False, 792 часа)

Задача 3: (5, True, 264 часа)

Задача 5: (4, True, 288 часов)

Задача 6: (3, False, 288 часов)

Задача 1: (3, True, 144 часа)

Задача 2: (2, False, 144 часа)

Теперь при помощи *srt\_hours\_to\_deadline\_dict* распределим задачи по рабочим часам, начиная с задач с наименьшим количеством часов до дедлайна

Задача 1:

144 часа до дедлайна, 4 рабочих часа.

Она может быть выполнена вовремя.

Добавляем задачу 1 в рабочие часы: work\_hours\_list = [1, 1, 1, 1].

Задача 2:

144 часа до дедлайна, 2 рабочих часа.

Она может быть выполнена вовремя.

Добавляем задачу 2 в рабочие часы: work\_hours\_list = [1, 1, 1, 1, 2, 2].

Задача 3:

264 часа до дедлайна, 2 рабочих часа.

Она может быть выполнена вовремя.

Добавляем задачу 3 в рабочие часы: work\_hours\_list = [1, 1, 1, 1, 2, 2, 3, 3].

Обновляем часы в словарях:

Задача 5:

288 часов до дедлайна, 1 рабочий час.

Она может быть выполнена вовремя.

Добавляем задачу 5 в рабочие часы: work\_hours\_list = [1, 1, 1, 1, 2, 2, 3, 3, 5].

Задача 6:

288 часов до дедлайна, 2 рабочих часа.

Она может быть выполнена вовремя.

Добавляем задачу 6 в рабочие часы: work\_hours\_list = [1, 1, 1, 1, 2, 2, 3, 3, 5, 6,

Задача 4:

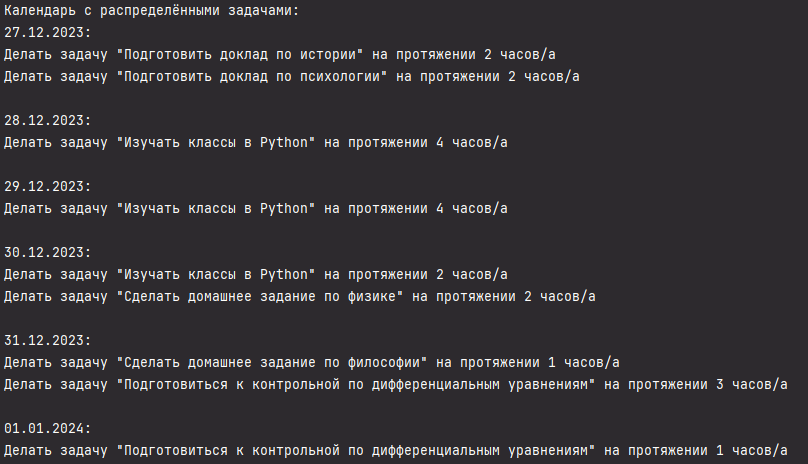
792 часа до дедлайна, 10 рабочих часов.

Она может быть выполнена вовремя.

Добавляем задачу 4 в рабочие часы: work\_hours\_list = [1, 1, 1, 1, 2, 2, 3, 3, 5, 6, 6, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4].

# Анализ результатов

В итоге мы получим ранжированный список задач, который включает оптимальное распределение нагрузки на каждый день. Количество рабочих часов можно изменять, варьируя ежедневную нагрузку.



*Рисунок 4. Результат сортировки задач*

# Заключение

В ходе работы была разработана и протестирована система умного распределения рабочих задач на основе различных критериев, таких как время до срока сдачи, интерес к задаче и её обязательность. Алгоритм сортировки задач позволяет оптимизировать распределение рабочего времени и повышает эффективность работы.

Результаты анализа показали, что использование данной системы позволяет эффективно распределять задачи в порядке их приоритетности, начиная с наиболее срочных и важных задач и заканчивая менее срочными и важными. Это способствует более рациональному использованию времени и повышению производительности труда.

Однако, в ходе исследования были выявлены некоторые ограничения алгоритма, такие как ограничение в учете всех возможных факторов, влияющих на приоритетность задач. Дальнейшее развитие программы будет включать в себя учет дополнительных критериев, а также оптимизацию алгоритма распределения задач в области свободного времени.

Тем не менее, текущие результаты свидетельствуют о потенциале данной программы в повышении эффективности работы и оптимизации процесса управления задачами.

# Список использованных источников

1. "Python. Подробное руководство" Марк Лутц, издательство "Вильямс", 2016 г.
2. "Python documentation", Python Software Foundation. – URL: https://docs.python.org/3/
3. "How to Optimize Python Code", Real Python. – URL: https://realpython.com/python-optimization/