Московский авиационный институт

(Национальный исследовательский университет)

**Лабораторные работы 1 - 10**

**Архитектура программной среды**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Шарудин Д.В./

(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_/Семенов АС./

(Подпись)

Дата : \_\_ 2024

**Лабораторная 1.**  Интегрированные среды разработки и их базовые отличия

1. Интегрированная среда разработки (Integrated Development Environment, IDE) представляет собой программное обеспечение, которое объединяет в себе различные инструменты, необходимые для разработки программного обеспечения. Обычно IDE включает в себя текстовый редактор, компилятор или интерпретатор, отладчик, инструменты для автоматизации сборки и управления проектами, а также другие инструменты, упрощающие процесс разработки.
2. Интегрированная среда разработки (IDE) обычно включает в себя следующие компоненты:

* **Текстовый редактор:** Позволяет разработчику создавать, редактировать и просматривать исходный код программы.
* **Компилятор/Интерпретатор:** Компилятор используется для преобразования исходного кода в машинный код, который может быть выполнен компьютером. Интерпретатор выполняет исходный код построчно без предварительной компиляции.
* **Отладчик:** Инструмент, который помогает разработчику идентифицировать и исправлять ошибки в программе.
* **Управление проектом:** Включает инструменты для создания, управления и организации файлов и ресурсов проекта.
* **Автоматизация сборки:** Позволяет автоматизировать процесс компиляции и сборки программы, что упрощает развертывание и обновление приложения.

1. Интегрированная среда разработки (IDE) и текстовый редактор кода имеют схожие цели - обеспечить удобную среду для написания программного кода. Однако существуют существенные различия между ними:

* **Функциональность:**
  1. **IDE:** Предоставляет полноценную интегрированную среду для разработки, включая компиляторы, отладчики, системы управления проектами, автоматизацию сборки и другие инструменты.
  2. **Редактор кода:** Обычно ограничивается функциями редактирования текста и может иметь некоторые расширения или плагины, но не обладает полной функциональностью IDE.
* **Интеграция инструментов:**
  1. **IDE:** Интегрирует в себя все необходимые инструменты для разработки, что упрощает рабочий процесс разработчика.
  2. **Редактор кода:** Обычно ориентирован на базовые функции редактирования текста и может требовать установки дополнительных инструментов для полноценной разработки.
* **Уровень абстракции:**
  1. **IDE:** Предоставляет высокий уровень абстракции и обычно ориентирован на конкретные языки программирования.
  2. **Редактор кода:** Обычно имеет более низкий уровень абстракции и ориентирован на работу с текстовыми файлами.
* **Сложность использования:**
  1. **IDE:** Имеет более сложный интерфейс и может потребовать времени для освоения.
  2. **Редактор кода:** Обычно более прост в использовании и может быть предпочтителен для начинающих разработчиков или для выполнения простых задач.
* **Ресурсы:**
  1. **IDE:** Возможно, потребуется больше ресурсов компьютера для работы из-за интеграции множества инструментов.
  2. **Редактор кода:** Обычно имеет более низкие требования к ресурсам компьютера.

1. Плагины представляют собой расширения, которые позволяют настраивать функционал интегрированных сред разработки (IDE) под конкретные проекты, задачи или предпочтения разработчика. Вот несколько способов, как плагины могут быть полезны:

* **Дополнительные инструменты:** Плагины могут добавлять новые инструменты и функции, которых нет в стандартной поставке IDE. Например, плагин для анализа кода или интеграции с внешними сервисами.
* **Поддержка новых языков:** Если IDE не поддерживает определенный язык программирования по умолчанию, можно установить плагин, добавляющий поддержку этого языка.
* **Интеграция с внешними инструментами:** Плагины могут обеспечивать интеграцию с системами контроля версий, средствами непрерывной интеграции, системами управления задачами и другими внешними инструментами, упрощая рабочий процесс разработчика.
* **Настройка интерфейса:** Плагины могут изменять внешний вид и поведение IDE в соответствии с предпочтениями разработчика, добавляя новые темы оформления, расширенные возможности настройки интерфейса и т. д.
* **Улучшение производительности:** Некоторые плагины направлены на повышение производительности разработки, например, путем автоматизации рутинных задач или улучшения инструментов отладки и анализа кода.

1. Основная цель использования интегрированной среды разработки (IDE) заключается в обеспечении удобной и эффективной среды для написания, отладки и управления программным кодом. Вот основные цели использования IDE:

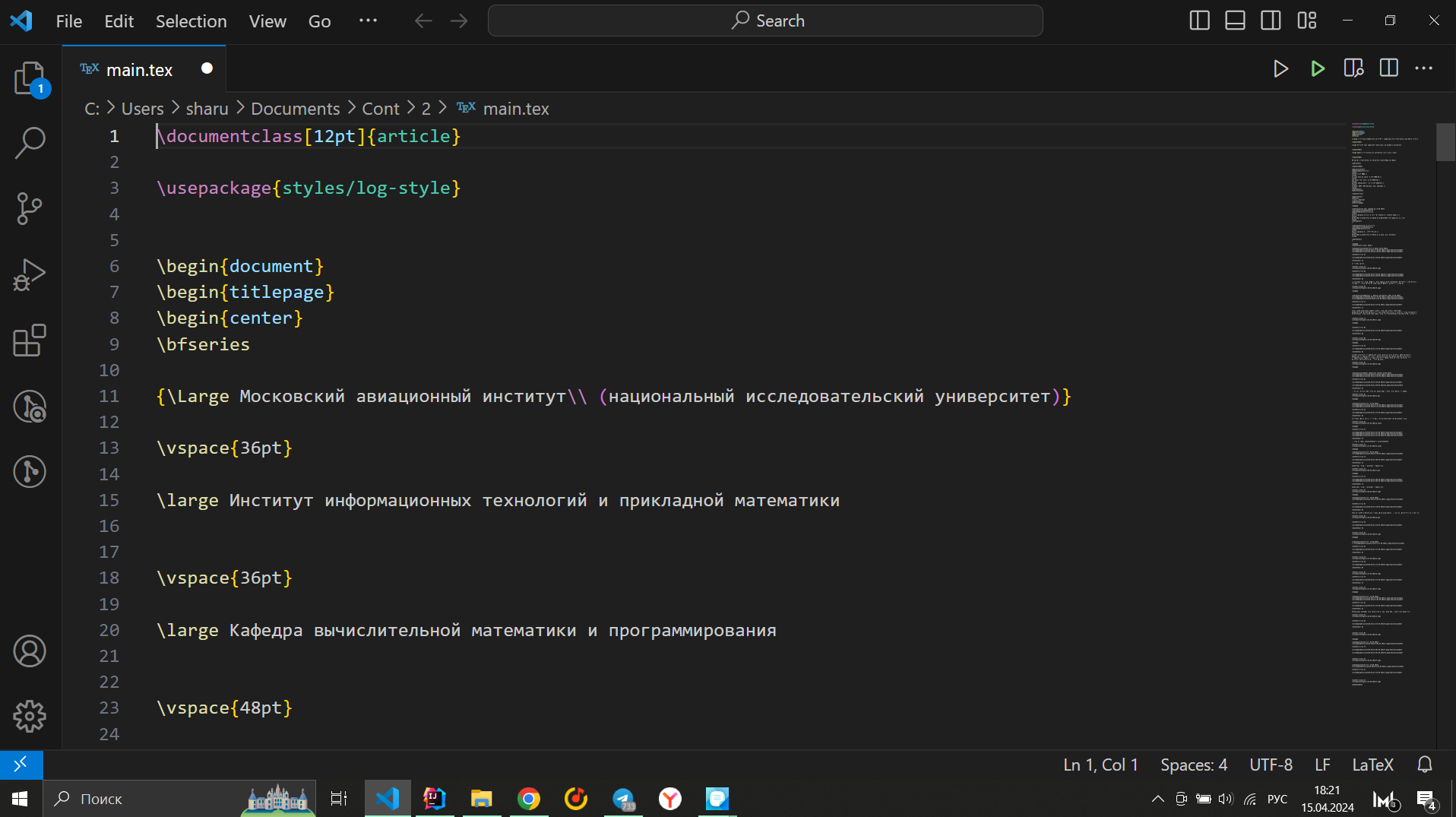
* **Увеличение производительности:** IDE предоставляет разработчику все необходимые инструменты и функции в одной среде, что упрощает рабочий процесс и повышает производительность. Вместо того чтобы использовать отдельные инструменты, разработчик может работать непосредственно в IDE, минимизируя время, затраченное на переключение между приложениями.
* **Упрощение разработки:** IDE предоставляет интегрированные инструменты для автоматизации рутинных задач, таких как сборка и отладка программы, управление зависимостями и версионирование кода. Это упрощает процесс разработки и позволяет сосредоточиться на создании функциональности приложения.
* **Улучшение качества кода:** IDE обеспечивает доступ к инструментам анализа кода, отладке и проверке синтаксиса, что помогает выявлять и исправлять ошибки на ранних стадиях разработки. Это способствует улучшению качества программного продукта и уменьшению количества ошибок.

**Задача 1.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Языки программирования | Свойства среды (автоматизация программирования) | Поддержка методологии разработки | Неучтенные факторы |
| VSCode | C, C#, Java, JavaScript, Ruby, PHP и др. | Интегрированный терминал, автозаполнение кода, отладчик, Git интеграция, IntelliSense, рефакторинг | Водопадная модель  Гибкая методология  DevOps  Lean | VSCode, или Visual Studio Code, это бесплатный и открытый исходный код редактор кода, разработанный компанией Microsoft. Он доступен для различных операционных систем, включая Windows, macOS и Linux. |
| PyCharm | Python | Интегрированный терминал, автозаполнение кода, отладчик, Git интеграция, IntelliSense, рефакторинг | V-модель  Водопадная модель | PyCharm - это интегрированная среда разработки (IDE) для языка программирования Python, разработанная компанией JetBrains. Она предоставляет множество инструментов и функций, облегчающих разработку, отладку и тестирование приложений на Python. |
| Intellij Idea | Java, Kotlin | Интегрированный терминал, автозаполнение кода, отладчик, Git интеграция, IntelliSense, рефакторинг | DevOps  Водопадная модель | IntelliJ IDEA - это интегрированная среда разработки (IDE) для различных языков программирования, таких как Java, Kotlin, Scala, Groovy и других. Она создана компанией JetBrains и широко используется разработчиками для создания профессиональных приложений. |
| Anaconda | Python, R | Интегрированный терминал, Git интеграция | Lean | Anaconda - это платформа и среда разработки для анализа данных и научных вычислений на языке программирования Python. Она включает в себя инструменты для работы с большими объемами данных, различные библиотеки и фреймворки, а также среду разработки Jupyter Notebook. |
| Visual Studio | C#, C++,PHP, Visual Basic .NET, Python и др. | Интегрированный терминал, автозаполнение кода, отладчик, Git интеграция, IntelliSense, рефакторинг | Водопадная модель  Гибкая методология | Visual Studio – это интегрированная среда разработки (IDE) разработанная компанией Microsoft. Она предоставляет разработчикам широкий спектр инструментов для создания приложений для различных платформ, включая Windows, macOS, Android и iOS. |

**Задача 2.**

Visual Studio Code (VS Code) - это бесплатный и открытый исходный код текстовый редактор, разработанный компанией Microsoft.



Вот несколько ключевых особенностей и характеристик Visual Studio Code:

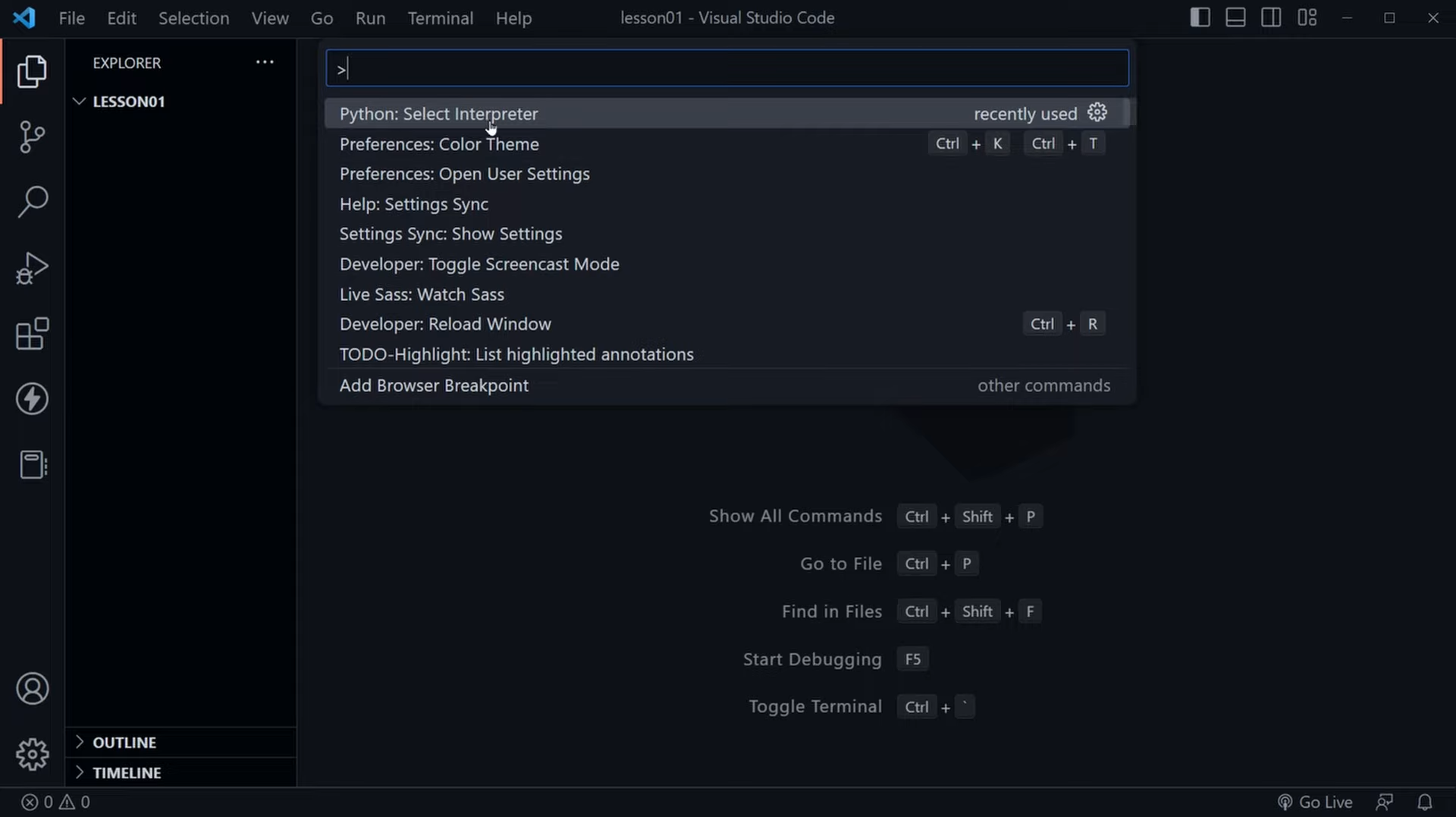
1. **Кроссплатформенность:** Visual Studio Code доступен для операционных систем Windows, macOS и Linux, что делает его доступным для широкого круга разработчиков.
2. **Легковесность и производительность:** В отличие от полноценной IDE, Visual Studio Code является легковесным текстовым редактором с возможностью расширения функционала с помощью плагинов. Это позволяет сохранить высокую производительность и меньшие требования к ресурсам компьютера.
3. **Интеграция с Git:** Visual Studio Code обладает встроенной поддержкой системы контроля версий Git, что позволяет разработчикам управлять своими проектами и делать коммиты прямо из редактора.
4. **Расширяемость:** С помощью расширений и плагинов Visual Studio Code можно расширить его функциональность для работы с различными языками программирования, интеграции с различными сервисами и инструментами, а также настроить его под собственные потребности.
5. **Интеллектуальные функции редактора:** Visual Studio Code предоставляет различные интеллектуальные функции, такие как автозавершение кода, подсветка синтаксиса, быстрая навигация по коду и многие другие, что улучшает процесс написания и редактирования кода.
6. **Интегрированная отладка:** VS Code предоставляет возможности для отладки кода прямо из редактора, что позволяет быстро находить и исправлять ошибки в программном коде.
7. **Широкое сообщество и поддержка:** Visual Studio Code имеет активное сообщество пользователей и разработчиков, что обеспечивает постоянное развитие и поддержку редактора, а также доступ к множеству полезных расширений и ресурсов.

Ниже представлен пример кода на языке программирования Python с учётом предустановленных расширений (Python).

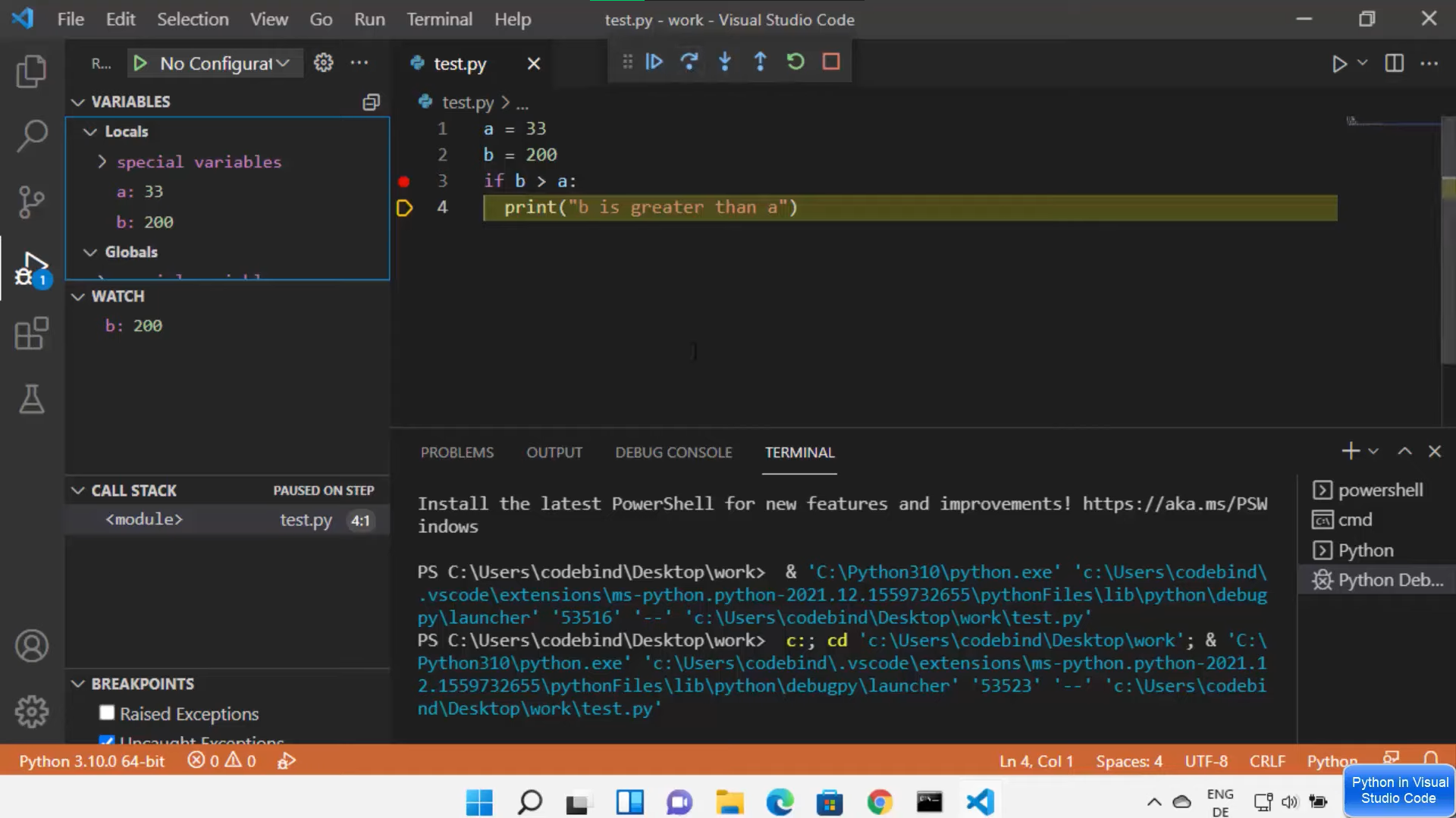
**Устанавливаем необходимые расширения:**



Расширение Visual Studio Code с широкой поддержкой языка Python (для всех активно поддерживаемых версий языка: >=3.7), включая такие функции, как IntelliSense (Pylance), анализ, отладку (Python Debugger), навигацию по коду, форматирование кода, рефакторинг, обозреватель переменных, обозреватель тестов и многое другое**.**

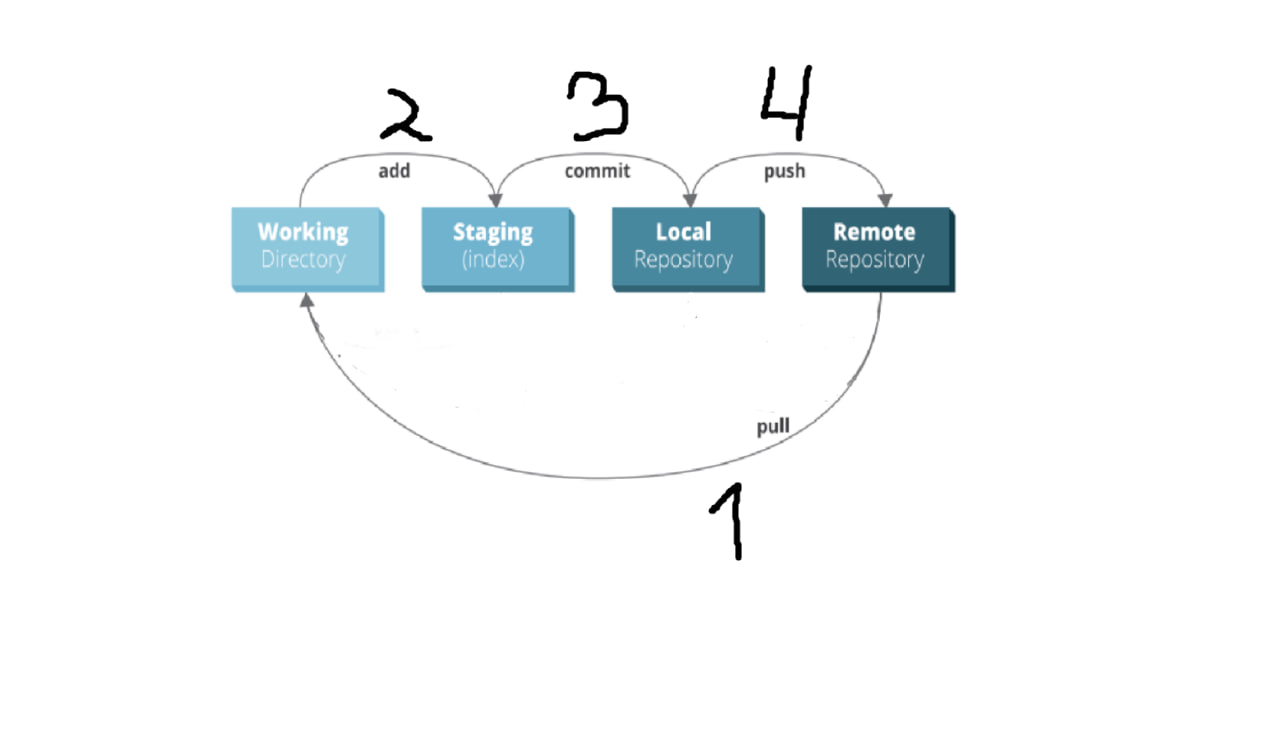
Выбираем интерпретатор:

**Пишем простейшую программу на Python и запускаем её в режиме дебага с помощью средств VSCode:**



**Лабораторная 2.** Потоковая диаграмма GitHub и Архитектура среды.

Ниже представлена общая потоковая диаграмма при работе над проектом как в локальном, так и в удаленном репозитории.

1. 

Потоковая диаграмма GitHub



Архитектура среды

1. Использование веток способствует коллективной разработке, предотвращает внезапные ошибки в коде и делает процесс разработки более прозрачным.



1. Основные принципы, лежащие в основе архитектуры GitHub, включают:

* **Репозитории (Repositories):** GitHub основан на концепции репозиториев, которые являются центральным элементом для хранения и управления исходным кодом и другими файлами проекта.
* **Ветвление (Branching):** GitHub поощряет использование ветвления в разработке, что позволяет разработчикам работать над отдельными фичами или исправлениями ошибок параллельно без влияния на основную ветку кода.
* **Слияние (Merging):** Слияние веток представляет собой важный аспект архитектуры GitHub, который позволяет объединять изменения из различных веток, обеспечивая интеграцию нового кода в основную разработку.
* **Pull-запросы (Pull Requests):** GitHub использует механизм pull-запросов, который позволяет разработчикам предложить изменения из своей ветки кода в основную ветку проекта для обсуждения и слияния.
* **Отслеживание задач (Issue Tracking):** GitHub предоставляет функциональность для отслеживания задач и обсуждения проблем в проекте с помощью системы issue tracking, что способствует управлению проектом и коммуникации в команде.
* **Коллаборация и обсуждение (Collaboration and Discussions):** GitHub облегчает сотрудничество и обсуждение проектов с помощью комментариев к коду, обсуждений в issue и pull-запросах, а также возможности работы в команде над проектом.

4.Локальный репозиторий создается на основе существующего с помощью команд:

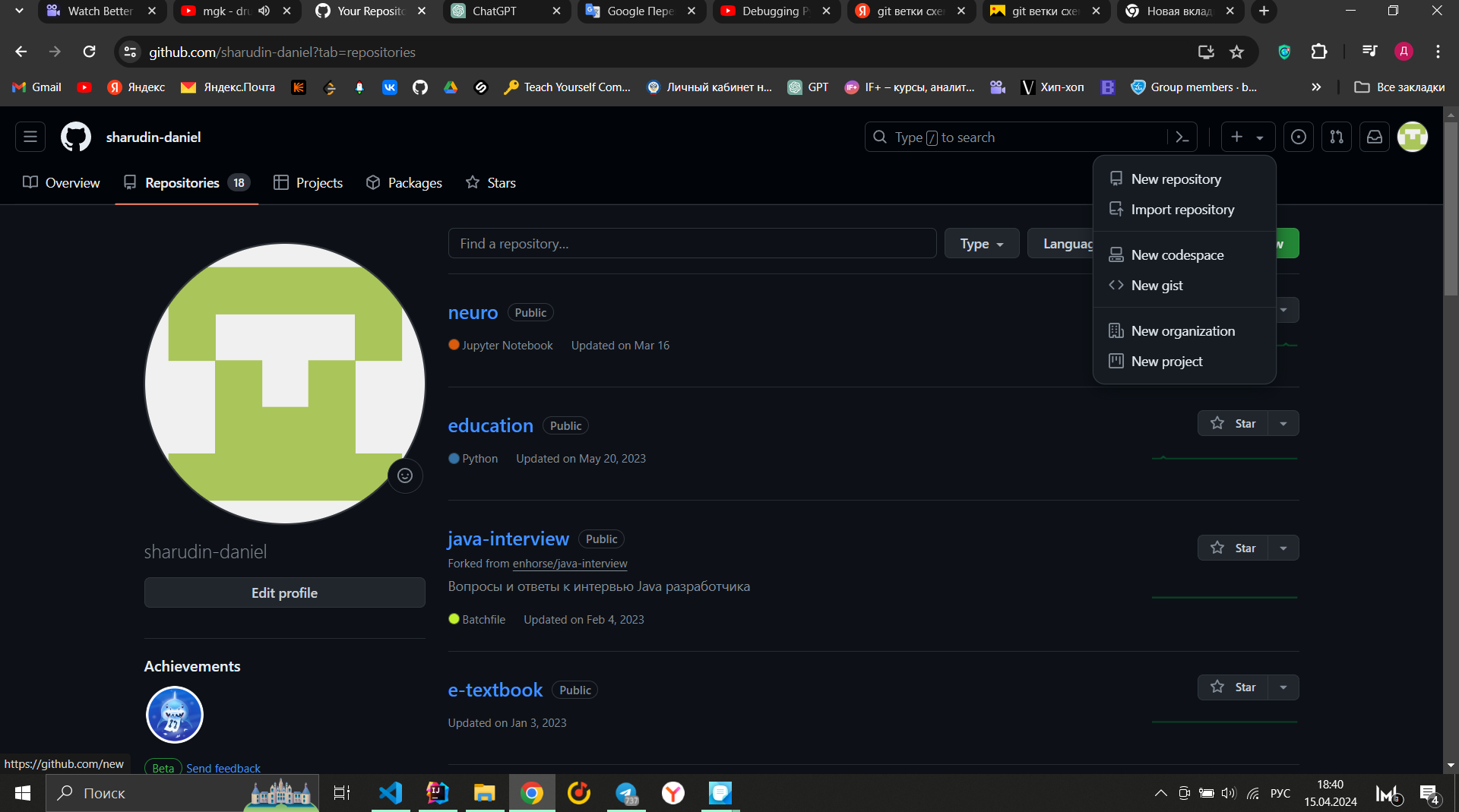
* **Клонирование (Clone):** Эта команда используется для создания локальной копии существующего удаленного репозитория. Она загружает все файлы и историю версий с удаленного репозитория на локальную машину.

git clone <URL\_удаленного\_репозитория>

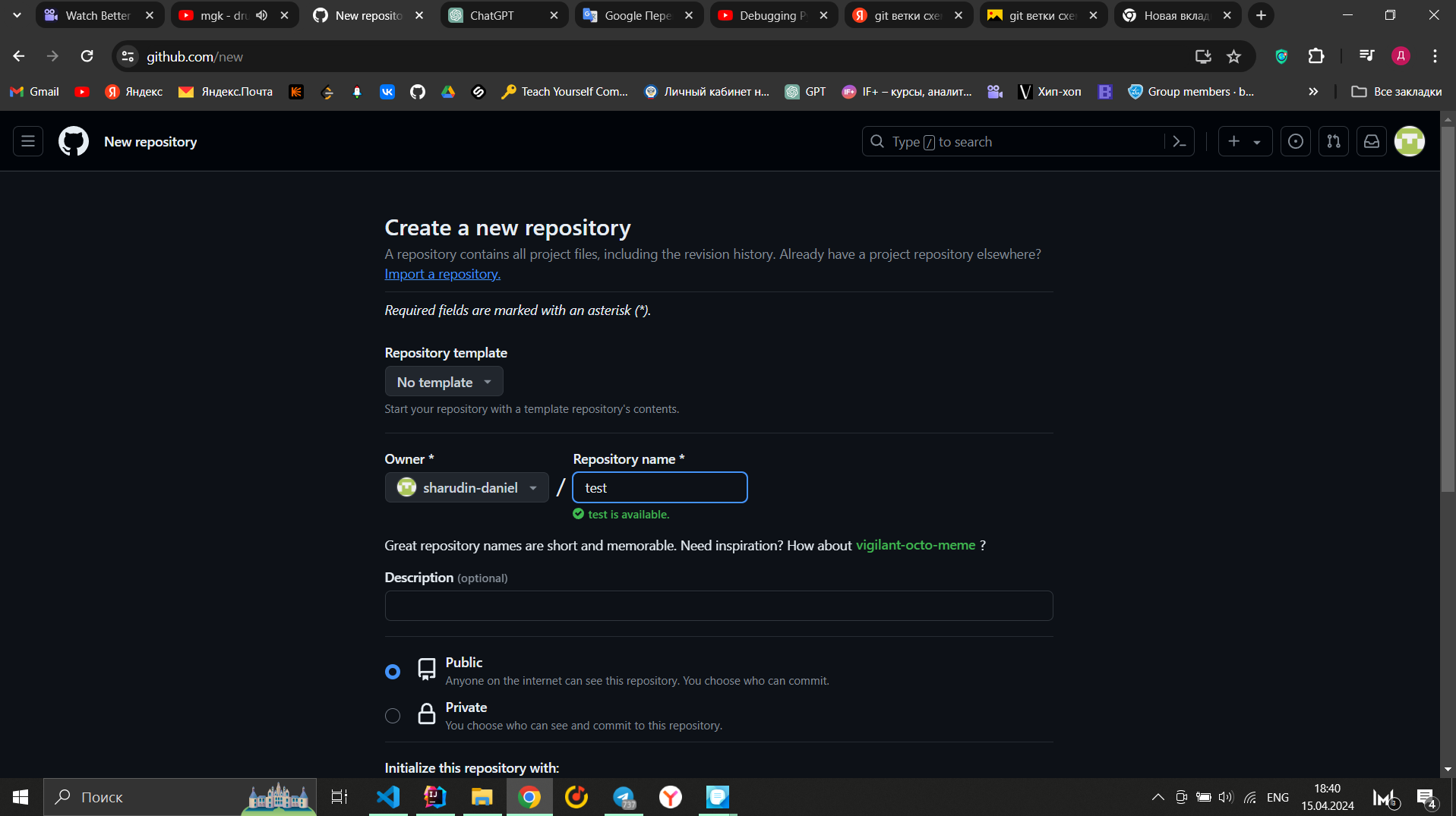
* **Инициализация (Init) и Добавление удаленного репозитория (Remote):** Эти команды используются в случае, если у вас уже есть локальный проект, и вы хотите привязать его к существующему удаленному репозиторию.

git init

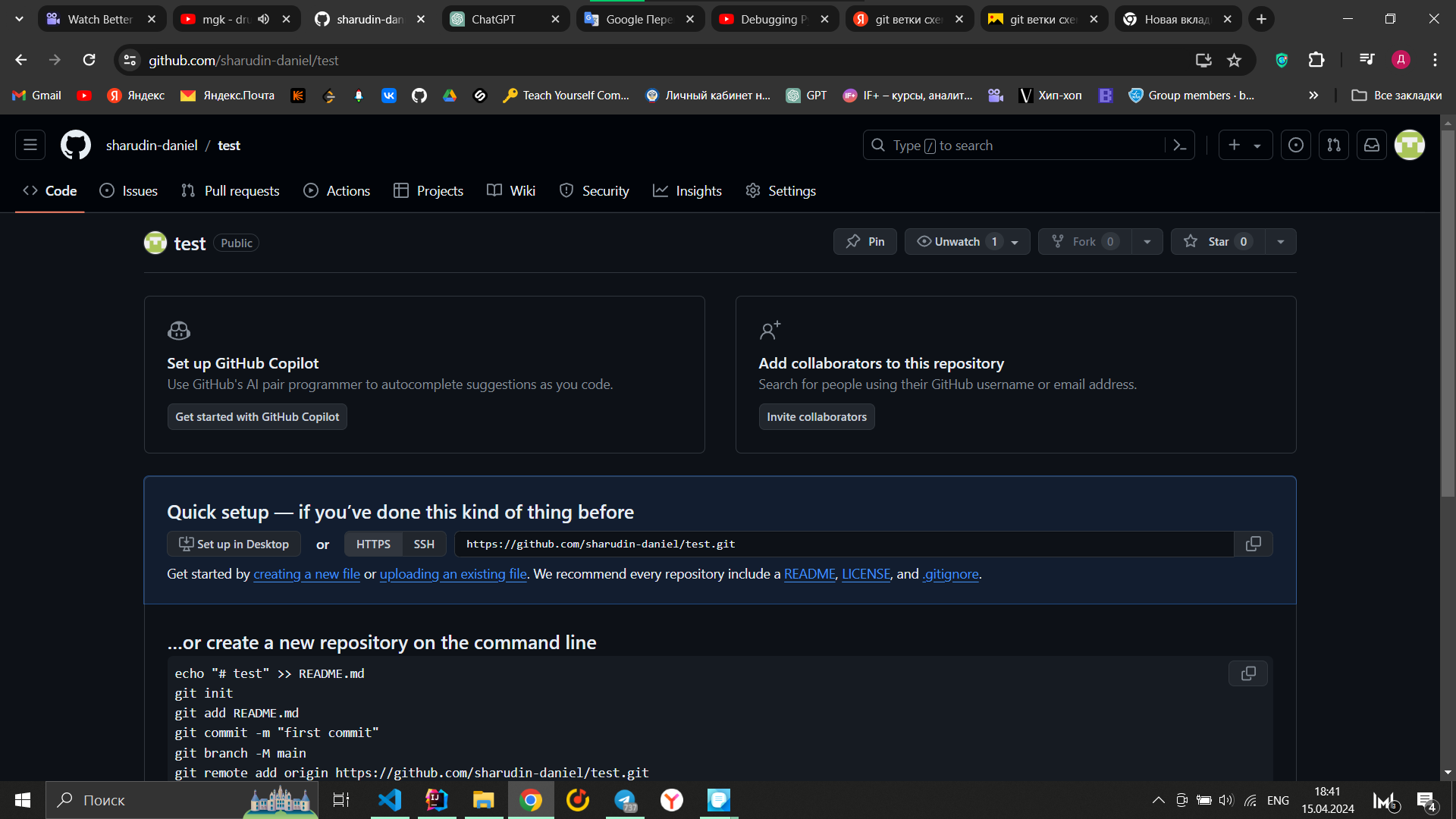
5. На главной странице нажать на кнопку создания репозитория.

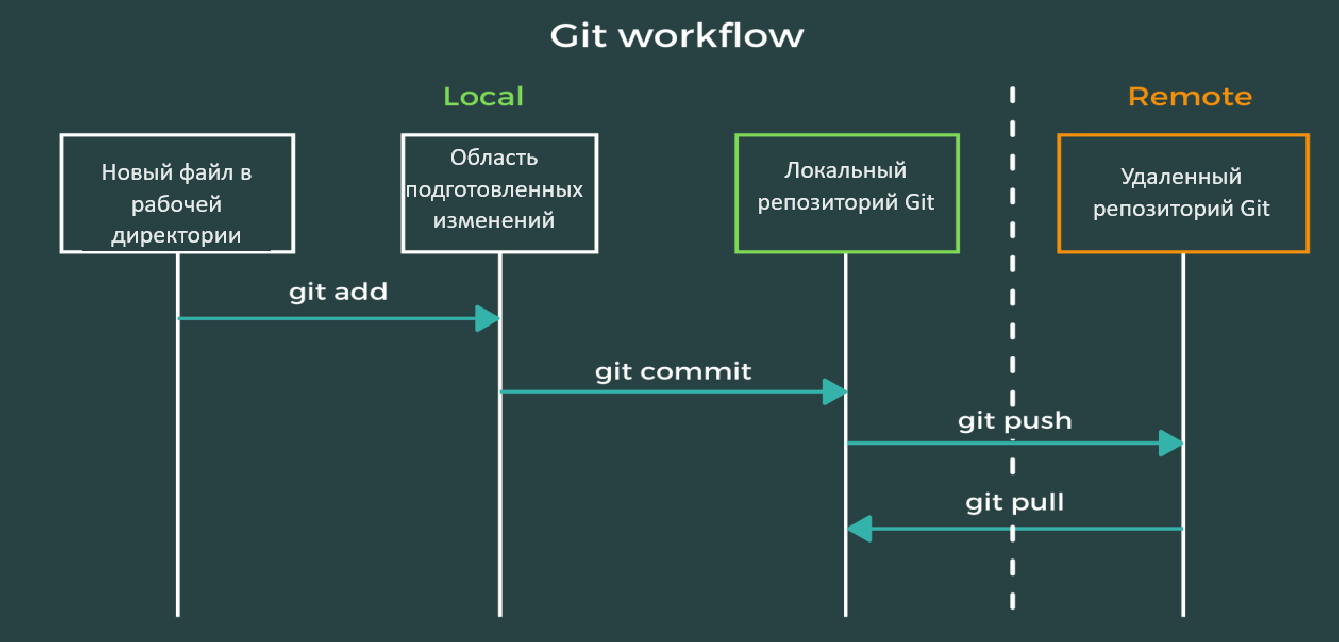


Задать нужные параметры



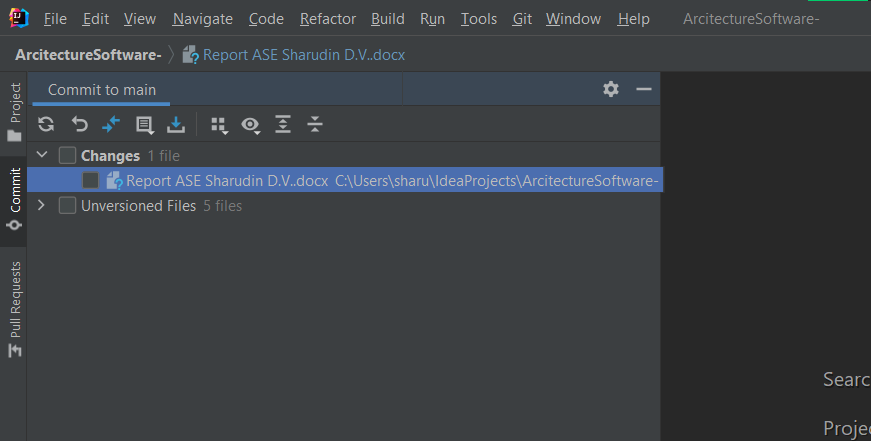
Репозиторий создан



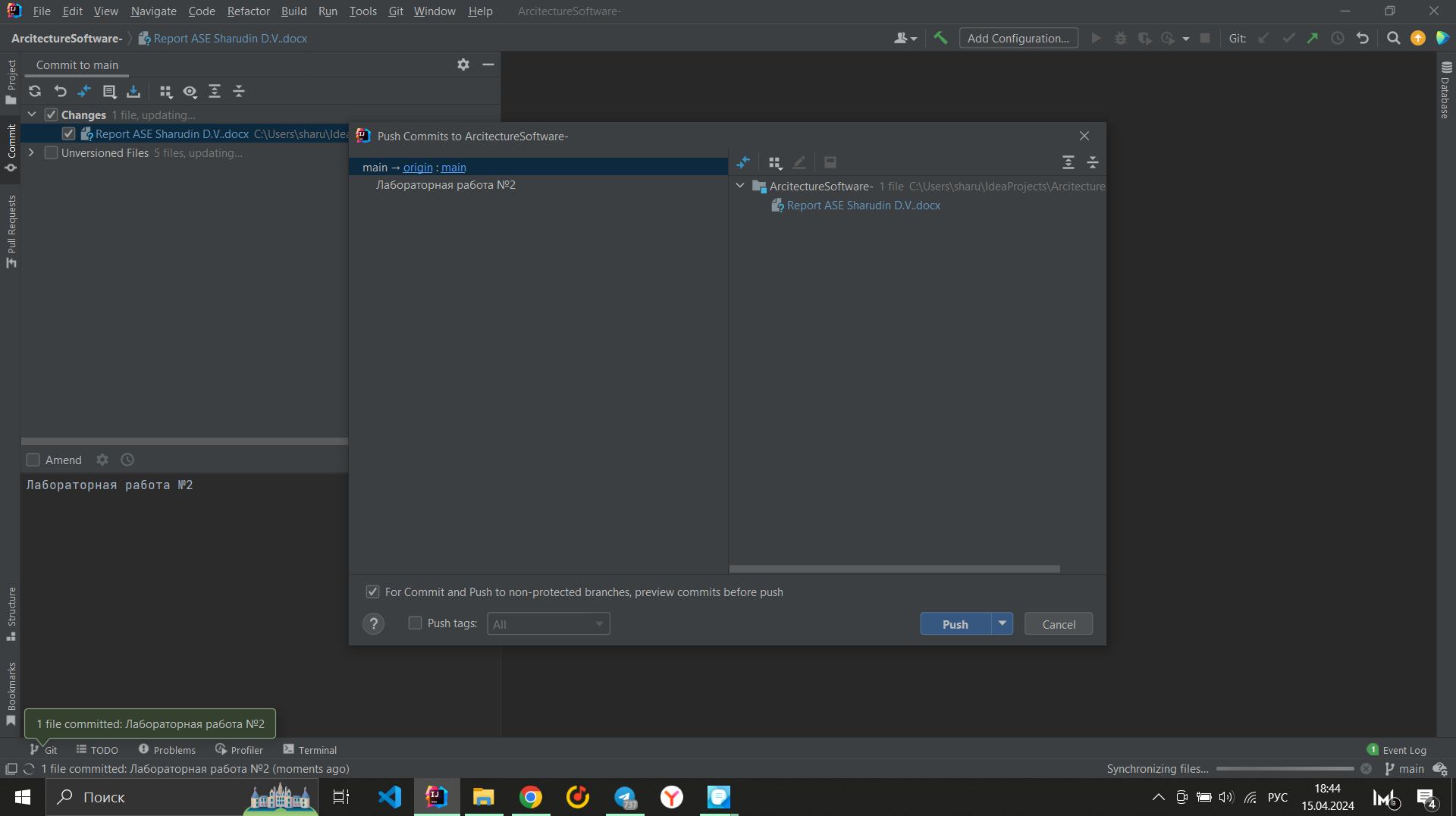
**Задание 1**

**Задание 2**

Ниже будет показан пример прохождения вышеуказанных этапов. Сначала был склонирован удаленный репозиторий. Создали текущий документ и внесли в него изменения.



Мы видим, что наш файл находится в состоянии untracked. Сначала необходимо зафиксировать локальное состояние (commit)



После этого необходимо загрузить данное состояние в удаленный репозиторий с помощью команды git push.

**Лабораторная 3.** Разработка потоковых диаграмм их реализация в среде VS Code. Виды профайлов. Их использование при распределенной разработки. Структура профайла.

1. В Visual Studio Code (VS Code) профили представляют собой возможность настройки различных конфигураций среды разработки для удобства работы в различных сценариях или с разными проектами. Вот несколько основных аспектов профилей в VS Code:

* **Настройка рабочей области:** Каждый профиль может иметь свои собственные настройки, включая расширения, темы оформления, параметры редактора и другие параметры. Это позволяет создавать индивидуальные рабочие области для разных проектов или задач.
* **Переключение между профилями:** VS Code позволяет легко переключаться между различными профилями, что удобно при работе с несколькими проектами или при переключении между режимами разработки.
* **Разделение настроек:** Профили могут иметь собственные файлы настроек (например, settings.json), что позволяет сохранять разные конфигурации для каждого профиля без конфликтов.
* **Группировка расширений:** Вы можете устанавливать различные расширения для каждого профиля, что позволяет подключать только необходимые инструменты в зависимости от проекта или задачи.

2. В Visual Studio Code (VS Code) есть два базовых профиля для работы с Python:

* **Python:** Этот профиль предназначен для общего использования Python и включает в себя базовые инструменты, необходимые для разработки на этом языке программирования. Он подходит для обычных задач, связанных с написанием и отладкой Python-кода. Включает расширения : autoDoc string, Black Formatter, Docker, Python, Python Environment Manager, Remote Development extension pack
* **Python Data Science:** Этот профиль ориентирован на разработку в области науки о данных с использованием Python. Он включает в себя расширения и инструменты, специально предназначенные для работы с данными, анализа данных, машинного обучения и визуализации. Включает расширения : Data Wrangler, Black Formatter, Jupyter, Python , Dev Containers, GitHub Copilot - Your AI pair programmer.

Эти профили предоставляют предустановленные настройки и набор расширений, которые оптимизированы для работы с Python в соответствующих областях разработки. Они позволяют быстро настроить среду разработки для работы с Python и начать работу над проектами без необходимости ручной установки и настройки инструментов.

3. Использование предустановленных профилей в VS Code обеспечивает единые инструменты для программистов при распределенной разработке. Это играет важную роль в обеспечении единообразия рабочей среды и снижении возможных проблем при совместной работе над проектом.

4. Типичная структура профиля в Visual Studio Code (VS Code) для работы с Python обычно включает в себя следующие компоненты:

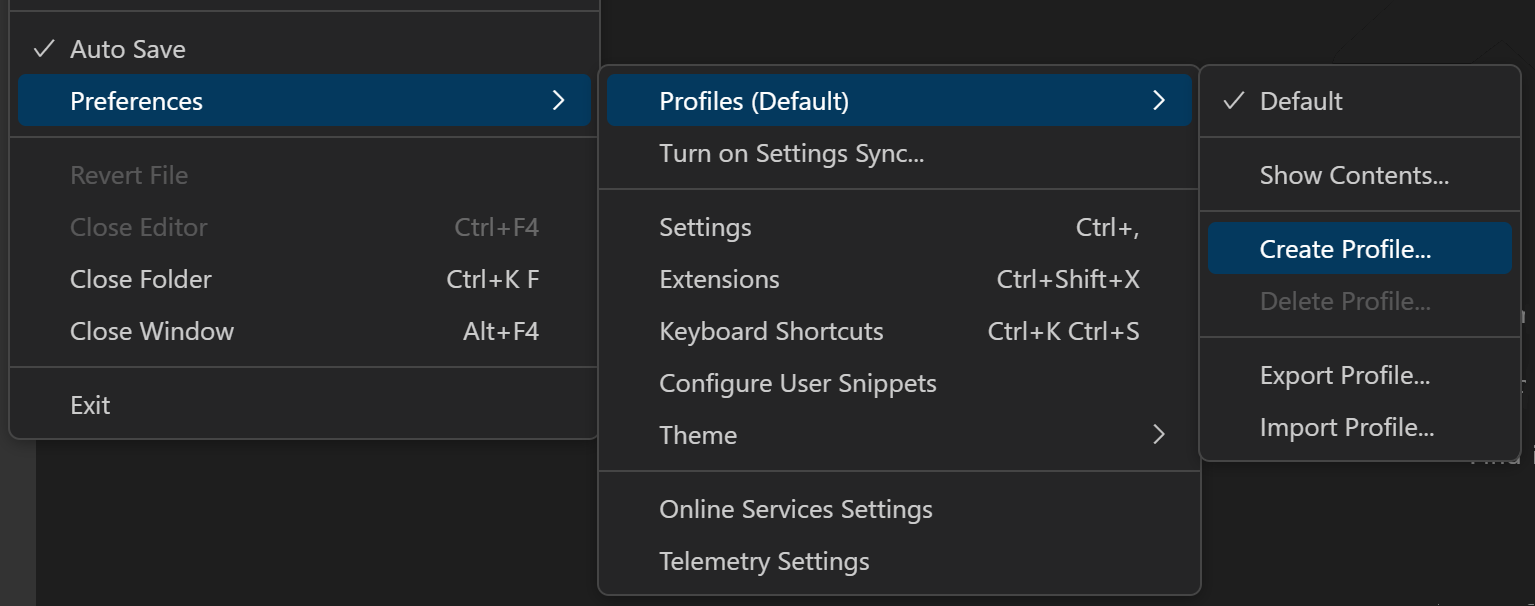
* **Настройки редактора (Editor Settings):** В этом разделе можно определить параметры отображения и поведения редактора, такие как размер отступов, используемый шрифт, цветовая схема и другие настройки визуального оформления.
* **Настройки Python (Python Settings):** Этот раздел содержит параметры, специфические для Python, такие как путь к интерпретатору Python, настройки автодополнения, установленные расширения и другие параметры, связанные с языком программирования Python.
* **Установленные расширения (Installed Extensions):** Здесь отображаются все расширения, установленные в рамках профиля для работы с Python. Это могут быть расширения для автодополнения кода, инструментов анализа кода, средств отладки, интеграции с системами контроля версий и другие полезные инструменты.
* **Настройки файлов (File Settings):** В этом разделе можно указать параметры, относящиеся к файлам и проектам Python, такие как расширения файлов, исключения файлов из индексации или поиск и другие параметры, связанные с файловой структурой проекта.
* **Настройки отладчика (Debugger Settings):** Здесь определяются параметры отладчика, такие как путь к файлу конфигурации отладки, настройки точек останова и другие параметры, необходимые для настройки среды отладки Python.

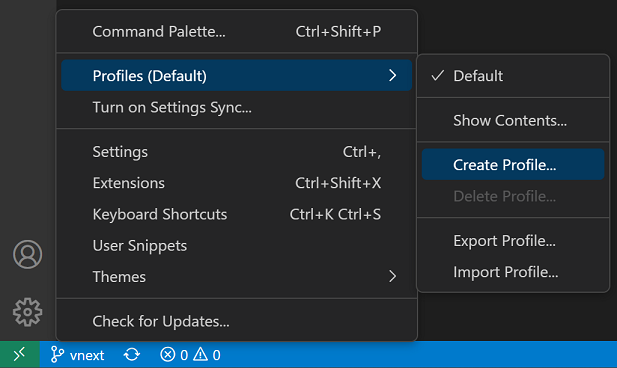
5. Для оптимизации профиля в Visual Studio Code (VS Code) и повышения производительности и удобства при разработке на Python, вы можете принять следующие меры:

* **Использование расширений для Python:** Установите и настройте расширения, специально предназначенные для работы с Python, такие как "Python", "Pylance", "Python Docstring Generator", "Python Test Explorer" и другие. Эти расширения предоставляют инструменты для автодополнения кода, анализа кода, генерации документации и управления тестами, что повышает эффективность разработки.
* **Настройка автоматического форматирования:** Включите автоматическое форматирование кода при сохранении файлов. Это позволяет поддерживать единый стиль кодирования и улучшает читаемость кода.
* **Использование виртуальных сред:** Создайте виртуальное окружение для каждого проекта с помощью инструментов типа virtualenv или venv. Это изолирует зависимости каждого проекта и обеспечивает чистоту окружения.
* **Настройка интегрированного терминала:** Настройте интегрированный терминал в VS Code для запуска команд Python, установки зависимостей и выполнения других задач без необходимости переключения во внешний терминал.
* **Использование сниппетов и шаблонов:** Создайте собственные сниппеты и шаблоны кода для повторяющихся задач, таких как создание функций, классов или тестов. Это ускоряет процесс написания кода и снижает вероятность ошибок.

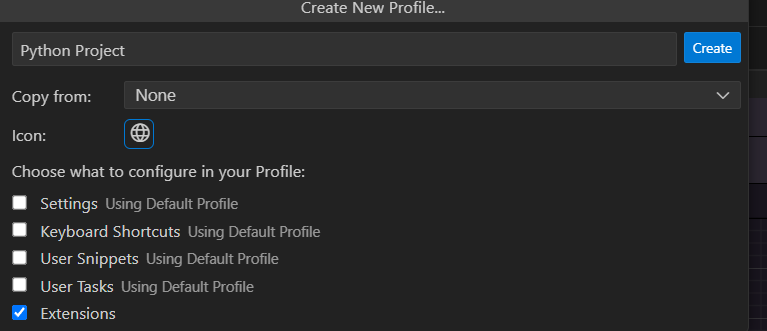
**Задание 1**

Задаем название для создаваемого профилю и выбираем корректируемые параметры.





Задаем название для создаваемого профилю и выбираем корректируемые параметры.

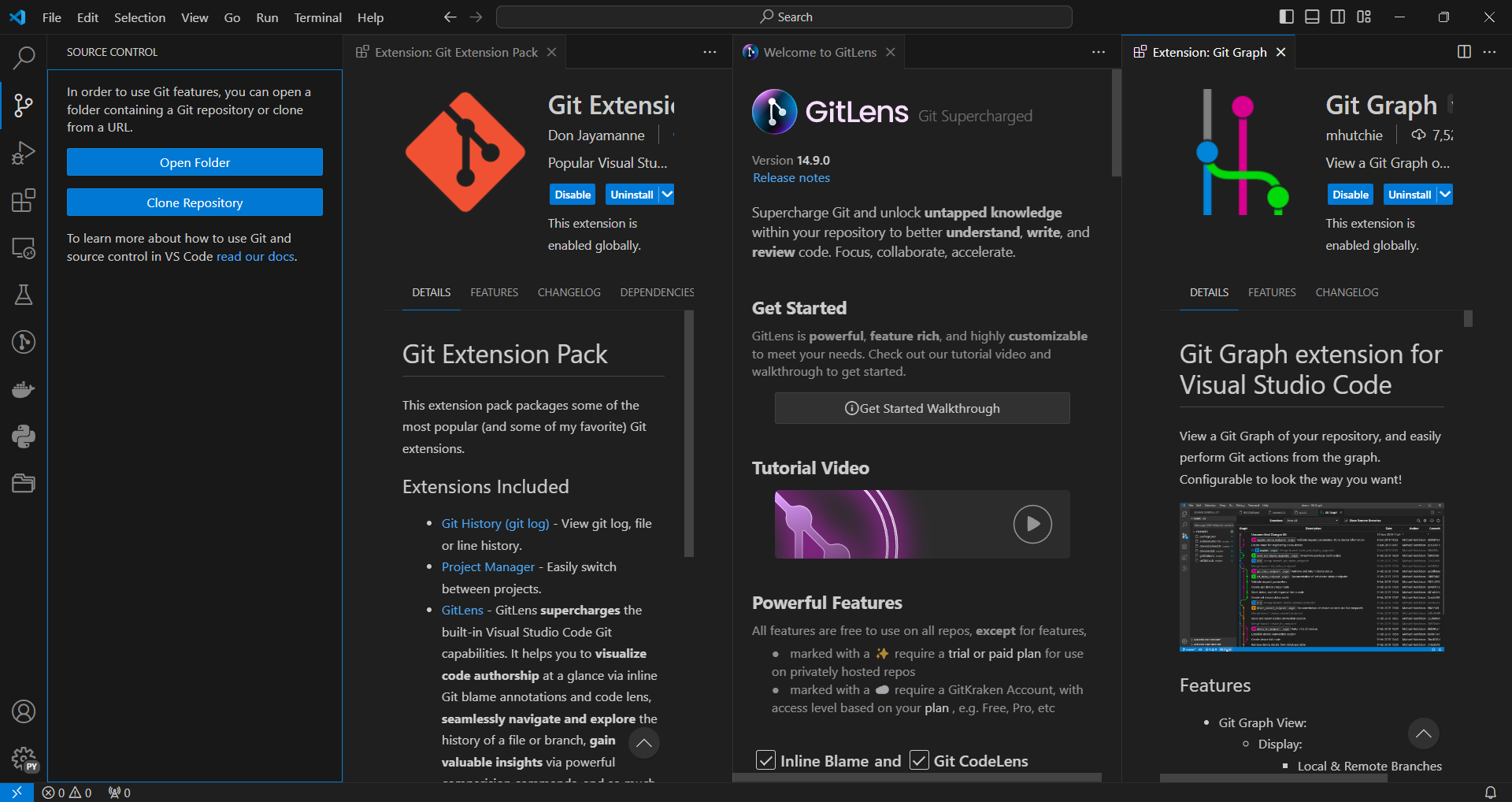


После этого видим, что в созданном профиле есть базовая часть и настройка расширений.



Устанавливаем расширения для работы с Python проектом

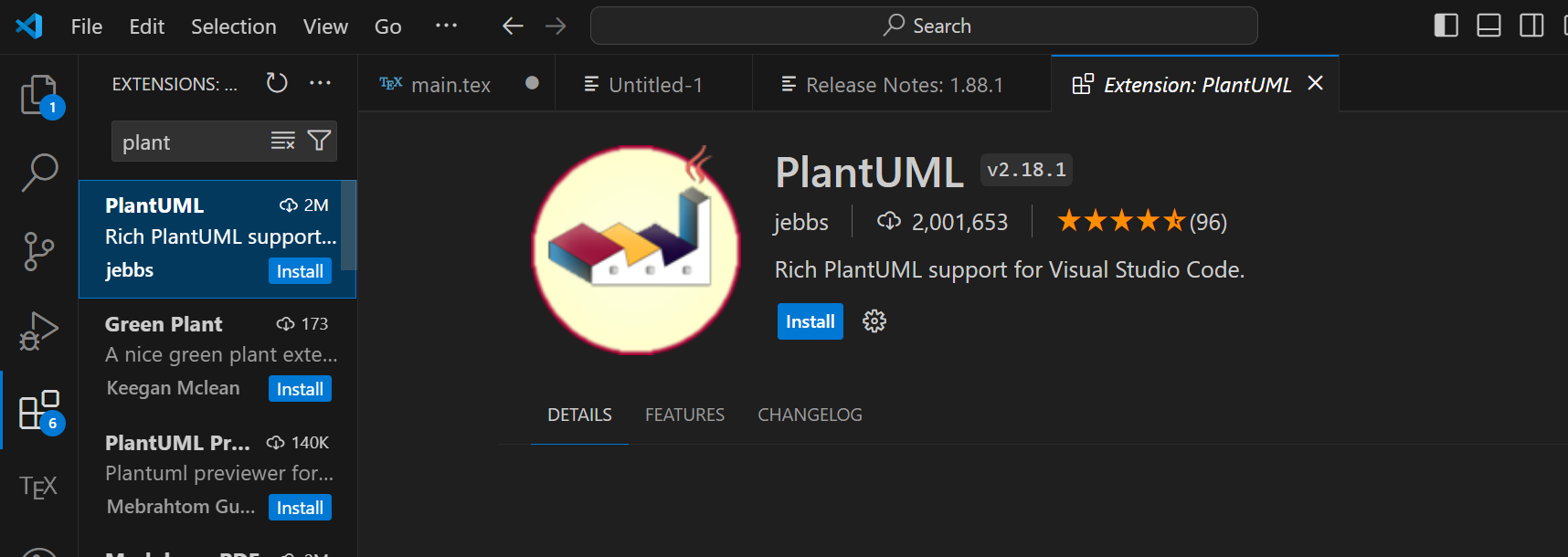
**Задание 2**



**Лабораторная 4.** Создание профайла VS code для работы в среде Python. Виды профайлов. Их использование при распределенной разработки. Структура профайла. GitHub copilot. UML (Unified Model Language) - диаграмма взаимодействий.

1. Data Flow Diagram (DFD), или диаграмма потоков данных, является графическим методом моделирования и анализа систем, который используется для описания потоков данных, процессов и хранилищ данных в информационной системе или бизнес-процессе. Вот основные элементы и концепции DFD:

* **Процессы (Processes):** Процессы представляют собой действия или функции, которые выполняются в системе. Они отображаются на диаграмме в виде прямоугольников и обозначаются номерами или именами.
* **Потоки данных (Data Flows):** Потоки данных представляют передачу данных между процессами, внешними агентами и хранилищами данных. Они изображаются на диаграмме в виде стрелок, указывающих направление передачи данных.
* **Хранилища данных (Data Stores):** Хранилища данных представляют места, где данные сохраняются в системе. Они могут быть файлами, базами данных или другими хранилищами. Хранилища данных обычно изображаются на диаграмме в виде параллелограммов.
* **Внешние агенты (External Entities):** Внешние агенты представляют внешние сущности или системы, с которыми взаимодействует система. Они могут быть пользователями, другими системами или внешними сервисами. Внешние агенты обычно изображаются на диаграмме в виде прямоугольников с наружными стрелками.

1. Использование потоковой диаграммы способствует повышению прозрачности и видимости процесса. Она позволяет немедленно выявить архитектурные проблемы и отследить все взаимодействия между модулями, что значительно облегчает процесс разработки.
2. Для создания потоковой диаграммы в VS Code необходимо установить расширение :
3. Unified Modeling Language (UML) - это стандартизированный язык моделирования, используемый в области разработки программного обеспечения для визуализации, спецификации, построения и документирования системных архитектур, процессов и структурных элементов программного обеспечения. UML предоставляет набор графических нотаций и правил для создания моделей, которые помогают разработчикам понять, проектировать и коммуницировать о сложных системах.

Основные концепции UML включают в себя:

* **Диаграммы классов (Class Diagrams):** Используются для визуализации структуры классов в системе, их атрибутов, методов и взаимосвязей.
* **Диаграммы вариантов использования (Use Case Diagrams):** Показывают функциональные требования к системе и взаимодействие между акторами и случаями использования.
* **Диаграммы последовательностей (Sequence Diagrams):** Описывают взаимодействие между объектами во времени и показывают порядок вызовов методов.
* **Диаграммы состояний (State Diagrams):** Используются для моделирования поведения объекта или системы в различных состояниях и переходах между ними.
* **Диаграммы деятельности (Activity Diagrams):** Описывают последовательность действий или процессов в системе.
* **Диаграммы компонентов (Component Diagrams):** Показывают физические компоненты системы и их взаимосвязи.
* **Диаграммы развертывания (Deployment Diagrams):** Используются для описания физической архитектуры системы и размещения компонентов на вычислительных устройствах.

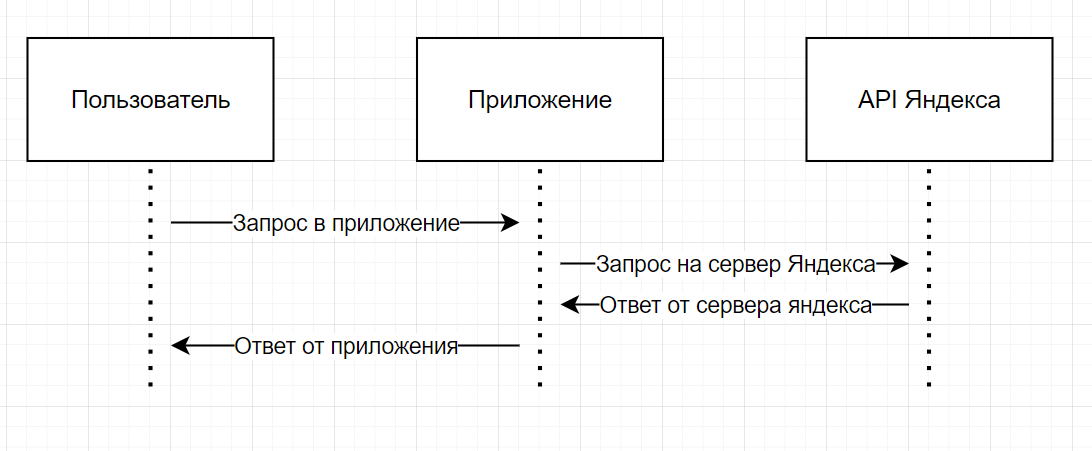
UML предоставляет стандартизированный и универсальный способ моделирования программных систем, что позволяет разработчикам лучше понимать требования, архитектуру и процессы в рамках проекта, а также облегчает коммуникацию между участниками команды разработки и заинтересованными сторонами.

1. UML-диаграммы представляют собой мощный инструмент, который приносит множество преимуществ и полезных возможностей в процессе разработки программного обеспечения. Вот несколько основных способов, как они могут быть полезны:

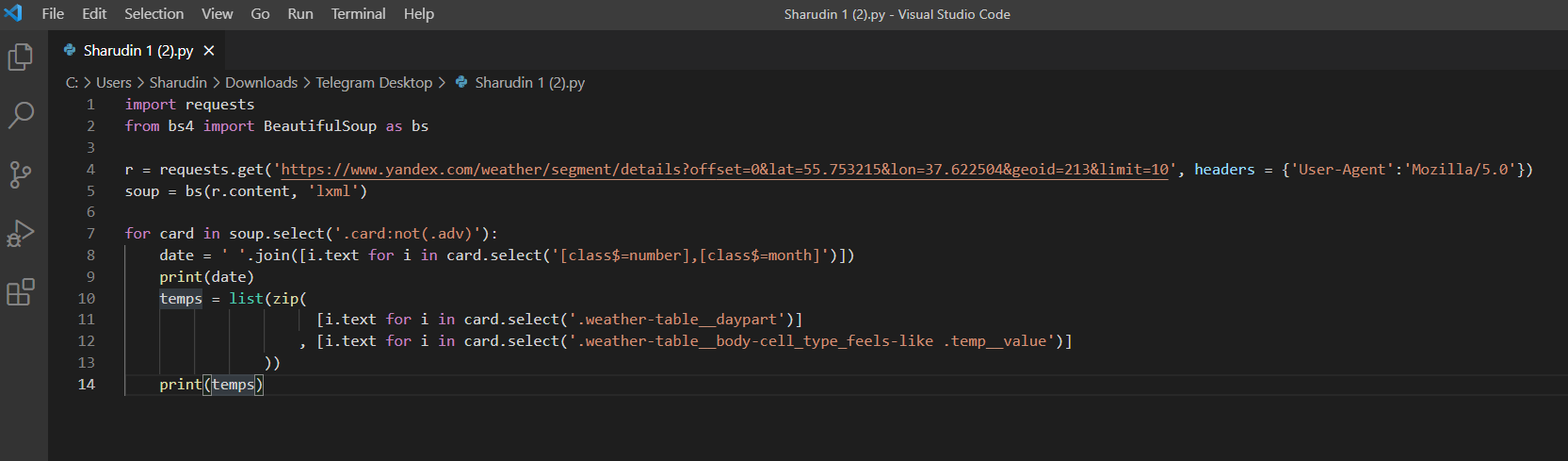
* **Визуализация концепций и структур:** Диаграммы классов, диаграммы компонентов и другие UML-диаграммы позволяют визуализировать концепции, структуры и компоненты системы. Это помогает разработчикам лучше понять архитектуру и дизайн программы.
* **Коммуникация:** UML-диаграммы служат универсальным языком для коммуникации между членами команды разработки и заинтересованными сторонами. Они предоставляют ясное и наглядное представление о системе, что облегчает обсуждение требований, проектирование и принятие решений.
* **Документация:** UML-диаграммы могут служить в качестве документации для системы. Они предоставляют наглядное описание структуры, функциональности и взаимодействия компонентов системы, что полезно для понимания и поддержания проекта в будущем.
* **Анализ и проектирование:** UML-диаграммы предоставляют инструменты для анализа требований, проектирования архитектуры и моделирования поведения системы. Они позволяют выявлять потенциальные проблемы, оценивать альтернативы и принимать обоснованные решения.
* **Тестирование:** UML-диаграммы могут использоваться для планирования и создания тестовых случаев. Они помогают определить, какие части системы должны быть протестированы, и какие результаты ожидать от тестирования.

В целом, UML-диаграммы являются важным инструментом в процессе разработки программного обеспечения, который способствует пониманию, коммуникации и управлению сложностью в разработке систем.

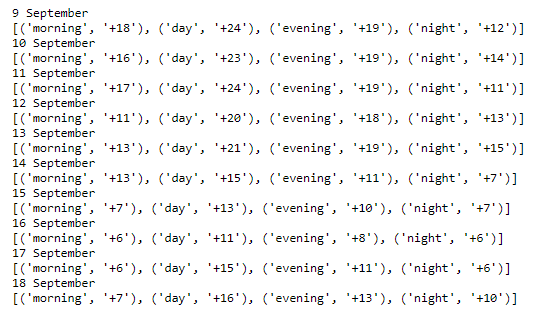
**Задание 1,2**



UML диаграмма программы



Исходная программа

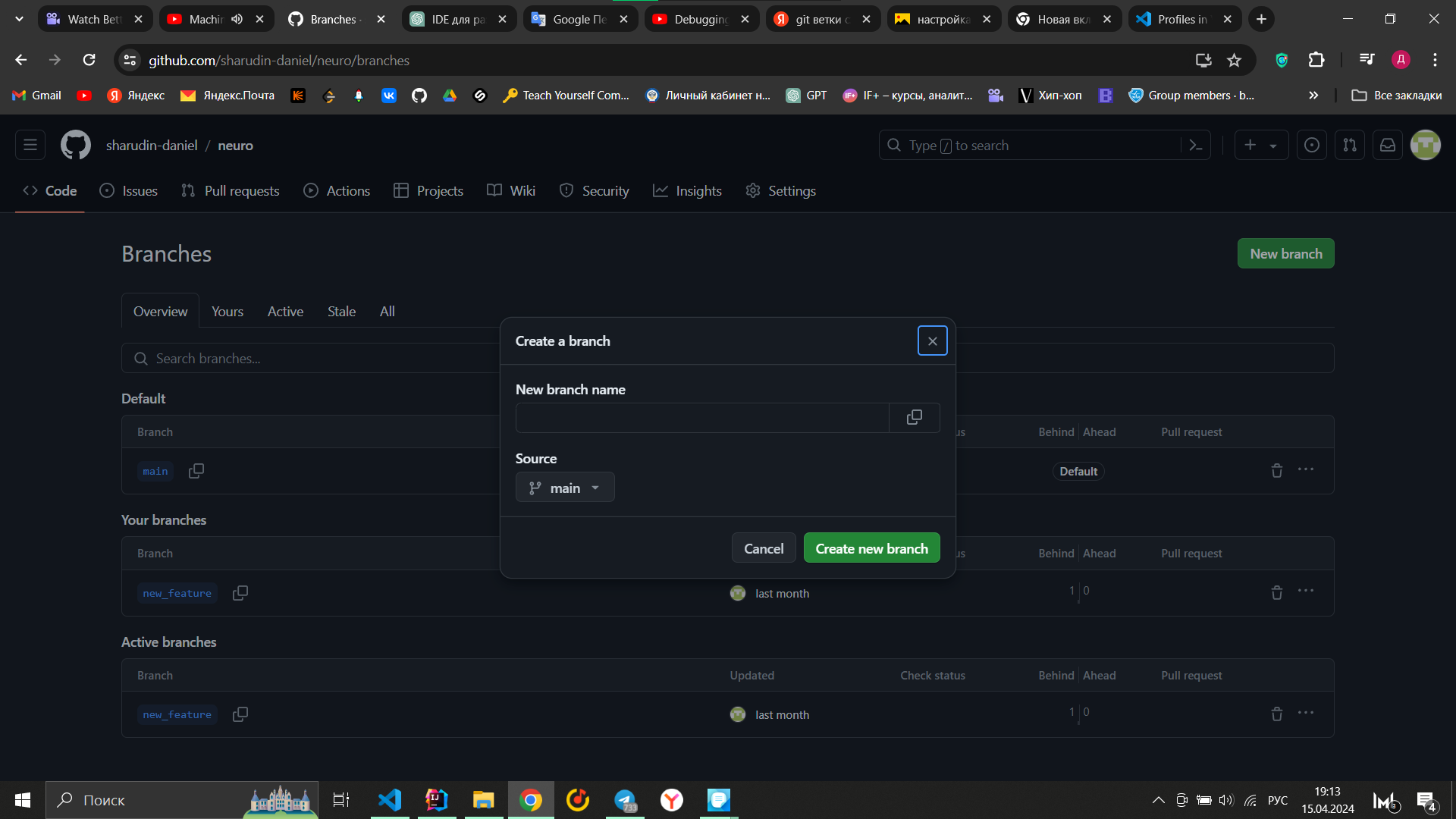


Результат работы

Как видим расширение работает, программа запустилась. Таким образом, можно сделать вывод, что создание профайлов это по сути дела гибкая настройка IDE для конкретного проекта или направления. Т.е можно создать проект для веб разработки на Go, либо профиль для разработки нативных приложений. При переходе на свой профиль разработчик будет чётко понимать, что у него есть всё необходимое для выполнения задачи.

**Лабораторная 5**. Создание ветвей в GitHub для распределенной разработки и их слияние из среды VS Code.

1. Создание ветки через web-интерфейс.



2. При работе с ветками в Git следует придерживаться нескольких основных принципов, чтобы обеспечить эффективное и безопасное управление версиями проекта. Вот некоторые из них:

* **Основная ветка (Main/Branch):** Основная ветка (обычно называется main или master) должна отражать стабильное состояние проекта. Все изменения должны быть интегрированы в основную ветку только после тщательного тестирования и проверки качества кода.
* **Создание отдельных веток для функциональности:** Для каждой новой функциональности или задачи создайте отдельную ветку. Это позволит изолировать изменения и избежать влияния на другие части проекта.
* **Регулярное обновление веток:** Периодически обновляйте локальные ветки основной веткой для интеграции последних изменений. Это помогает избежать конфликтов при слиянии и обеспечивает актуальность кода.
* **Описание изменений:** При создании коммитов и слиянии веток всегда прикрепляйте описания изменений. Это помогает другим разработчикам понять суть изменений и их влияние на проект.

3. Использовать команду git merge

4. Конфликты слияния (merge conflicts) возникают, когда Git не может автоматически объединить изменения из разных веток во время операции слияния. Это происходит, когда две ветки вносят изменения в одно и то же место в файле или когда одна из веток удаляет файл, который другая ветка пытается изменить.

Когда Git обнаруживает конфликт слияния, он помечает файлы, в которых произошел конфликт, специальными маркерами (обычно <<<<<<<, =======, >>>>>>>), чтобы показать, где начинаются и заканчиваются изменения из разных веток. Это помогает разработчику вручную разрешить конфликт, выбрав нужные изменения или комбинируя их вручную.

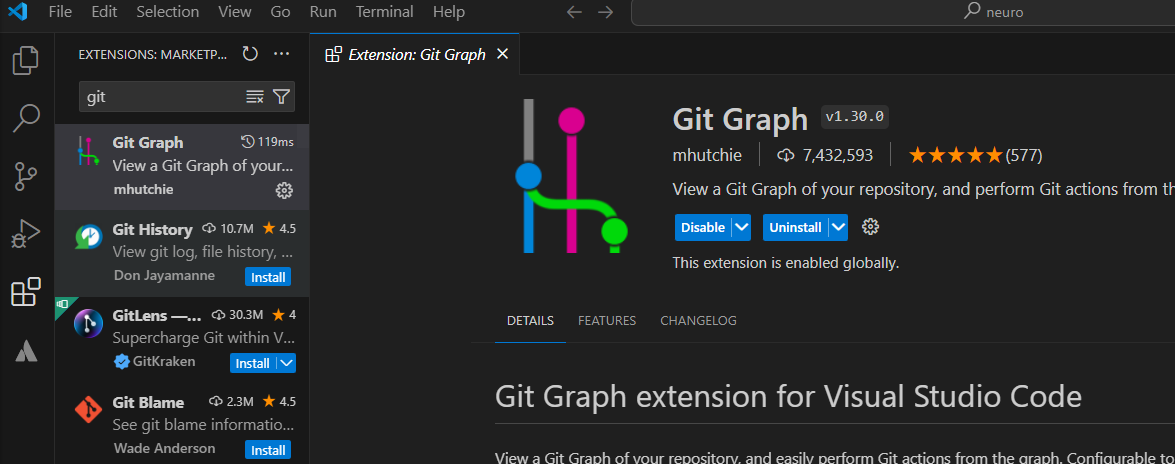
Разрешение конфликта слияния включает в себя следующие шаги:

* **Идентификация конфликтов:** После попытки слияния Git сообщает о наличии конфликтов. Разработчик должен найти эти места в файлах, где произошли конфликты.
* **Разрешение конфликтов:** Разработчик должен вручную выбрать, какие изменения должны быть сохранены в итоговом файле. Это может включать в себя сохранение изменений из одной из веток, объединение изменений или редактирование содержимого файла.
* **Пометка конфликтов как разрешенных:** После решения конфликтов файлы должны быть помечены как разрешенные для завершения слияния. Это обычно делается с помощью команды git add для каждого файла, содержащего конфликты.
* **Завершение слияния:** После разрешения всех конфликтов разработчик завершает слияние с помощью команды git merge --continue или git commit, в зависимости от процесса слияния.

Разрешение конфликтов слияния является нормальной частью работы с Git, особенно в командах с большим количеством разработчиков и сложными ветвями разработки. Важно внимательно разбираться в изменениях и тщательно разрешать конфликты, чтобы сохранить целостность и качество кода.

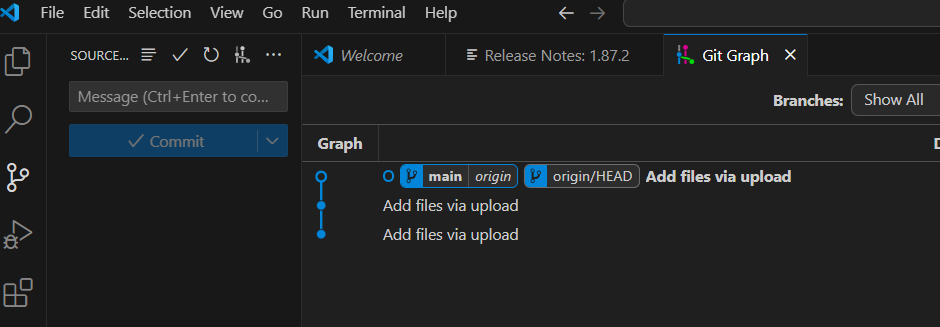
5. Расширение для удобного управления ветками.

Установим расширение Git Graph для комфортной работы с ветками Git:

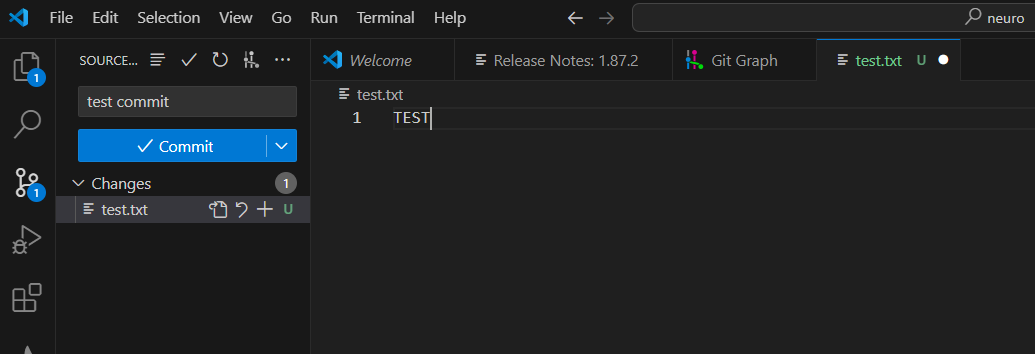


**Задание 1**

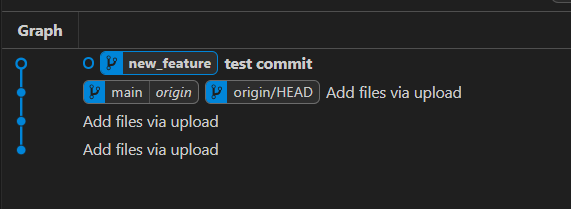
Изначально наш граф выглядит следующим образом:



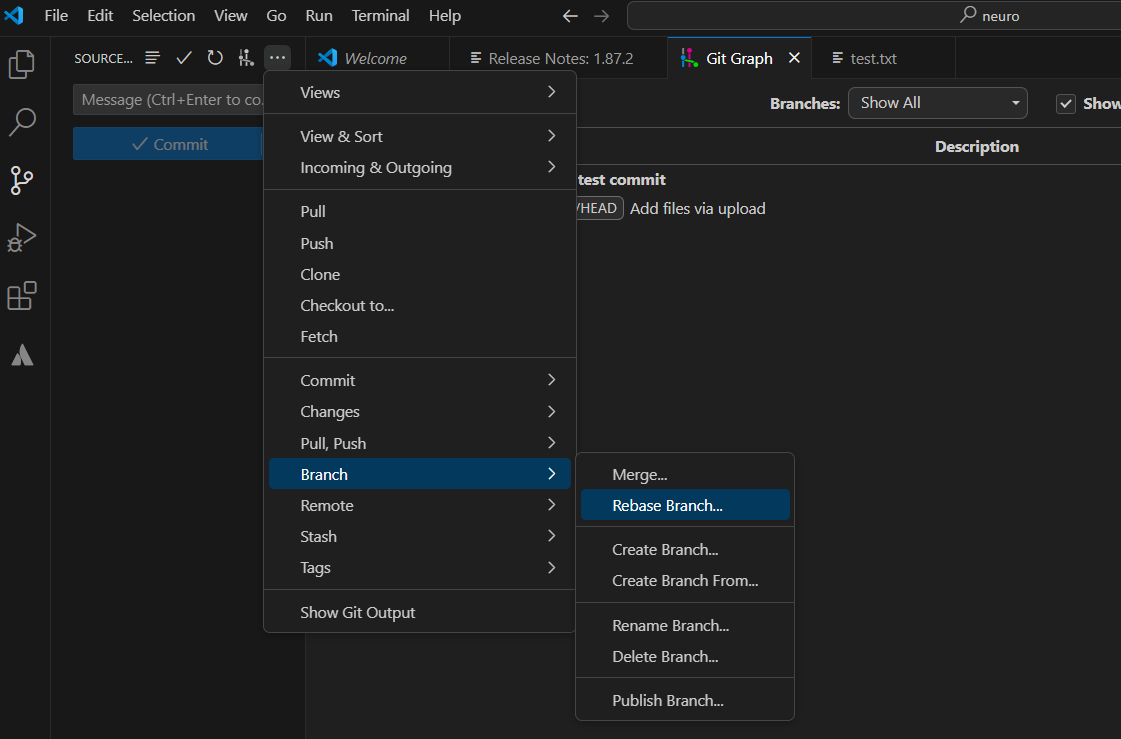
Создадим новую ветку, затем создадим новый файл и закомитим его:



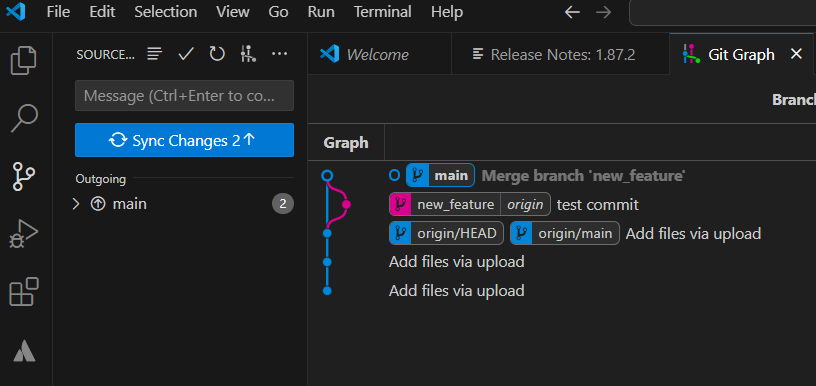
Наша новая ветка выглядит так:



Сделаем рибейз новой ветки на master (на случай, если ктото успел внести свои изменения):



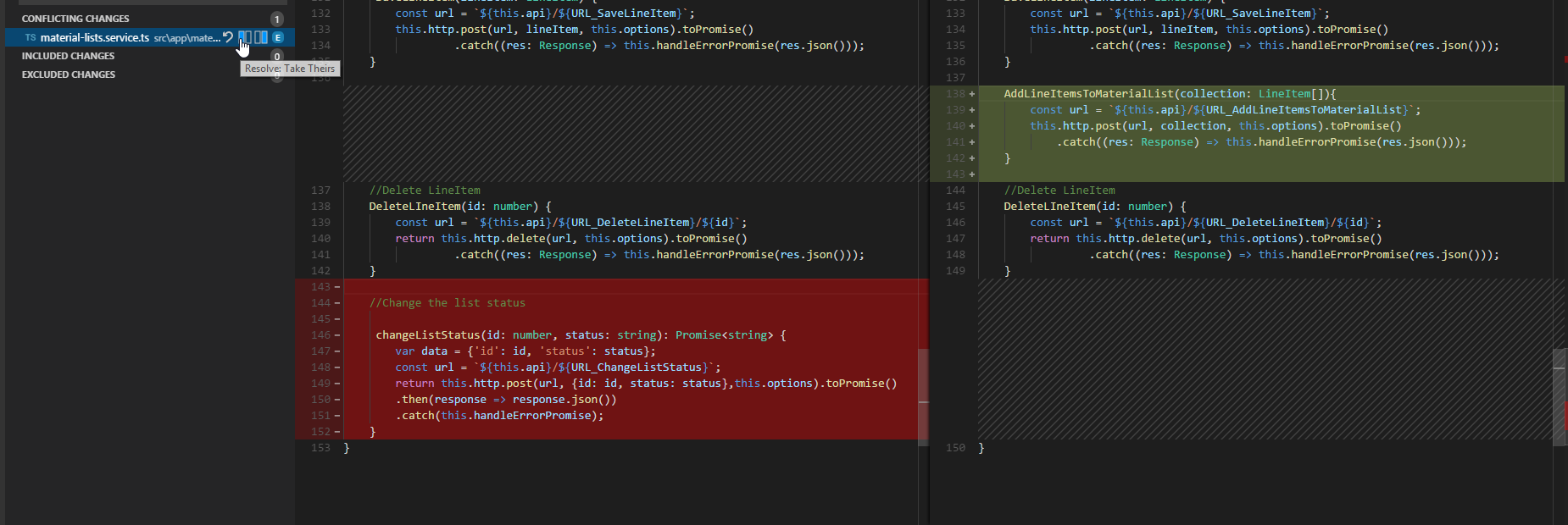
Вольем новую ветку в ветку master:



В результате, наши новые изменения, сделанные в новой ветке, попали в ветку master.

**Задание 2**

Разрешение конфликтов:



**Лабораторная** 6. Файлы конфигурации. Создание User Task. Настройка редактора VSCOde.

1. В Visual Studio Code файл settings.json используется для хранения пользовательских настроек, которые применяются к вашей среде разработки. Файл settings.override.json позволяет вам определить настройки, которые должны переопределять настройки из файла settings.json в рабочей области или проекте.

Вот как это работает:

* settings.json: Этот файл содержит основные пользовательские настройки, которые применяются к вашей среде разработки VS Code.
* settings.override.json: Этот файл позволяет вам определить настройки, которые должны переопределять настройки из settings.json в рабочей области или проекте. Это полезно, когда вам нужно изменить настройки только для конкретного проекта, не затрагивая основные настройки.

Например, если вам нужно переопределить значение настройки только для одного проекта, вы можете использовать settings.override.json, чтобы определить это значение, не изменяя settings.json.

2. Пользовательская задача в Visual Studio Code - это способ настройки и запуска пользовательских команд или скриптов прямо из редактора.

Обычно пользовательские задачи используются для автоматизации часто используемых задач в процессе разработки, таких как компиляция кода, запуск тестов, сборка проекта и т.д.

Чтобы создать пользовательскую задачу, вы создаете или редактируете файл tasks.json, который содержит описание ваших задач. Этот файл может находиться в корне вашего проекта или в папке .vscode в корне проекта.

Чтобы создать новую пользовательскую задачу в Visual Studio Code, следуйте этим шагам:

* Откройте файл tasks.json, нажав Ctrl+Shift+P (в Windows/Linux) или Cmd+Shift+P (на macOS), чтобы открыть палитру команд, и введите "Tasks: Configure Task" (или выберите Terminal > Configure Tasks... из меню).
* В открывшемся списке выберите "Create tasks.json file from template", а затем "Others".
* Создайте свою задачу, следуя примеру ниже:



Этот пример создаст задачу, которая запустит скрипт Python app.py. Вы можете адаптировать этот пример под вашу конкретную задачу, заменив python app.py на вашу команду.

* Сохраните файл tasks.json.
* Теперь вы можете выполнить эту задачу, выбрав Terminal > Run Task... (или нажав Ctrl+Shift+B (в Windows/Linux) или Cmd+Shift+B (на macOS)) и выбрав вашу задачу из списка.

3. Вот несколько настроек редактора Visual Studio Code, которые можно настроить для повышения производительности при работе с файлами конфигурации:

* **Уменьшение задержки при открытии больших файлов**:
  + files.largeFileOptimizations: Установите это значение в true, чтобы включить оптимизацию для больших файлов. При установке в true редактор не будет анализировать большие файлы для выявления фрагментов кода.

Пример настройки:

"files.largeFileOptimizations": true

* **Отключение автоматической проверки файлов**:
  + files.autoSave: Отключите автоматическое сохранение файлов или установите оптимальное значение для вашей работы.

Пример настройки (для отключения автоматического сохранения):

"files.autoSave": "off"

* **Отключение анализа файлов**:
  + files.exclude: Исключите большие файлы или файлы, которые не нужно анализировать.

Пример настройки (исключение файлов с расширением .log):

"files.exclude": { "\*\*/\*.log": true }

* **Ограничение числа открытых файлов**:
  + files.maxMemory: Установите эту настройку для ограничения количества памяти, используемой редактором для хранения открытых файлов.

Пример настройки (ограничение до 4 ГБ):

"files.maxMemory": 4096

* **Отключение ненужных расширений для файлов конфигурации**:
  + files.associations: Измените ассоциации файлов, чтобы отключить функции, которые не нужны для файлов конфигурации.

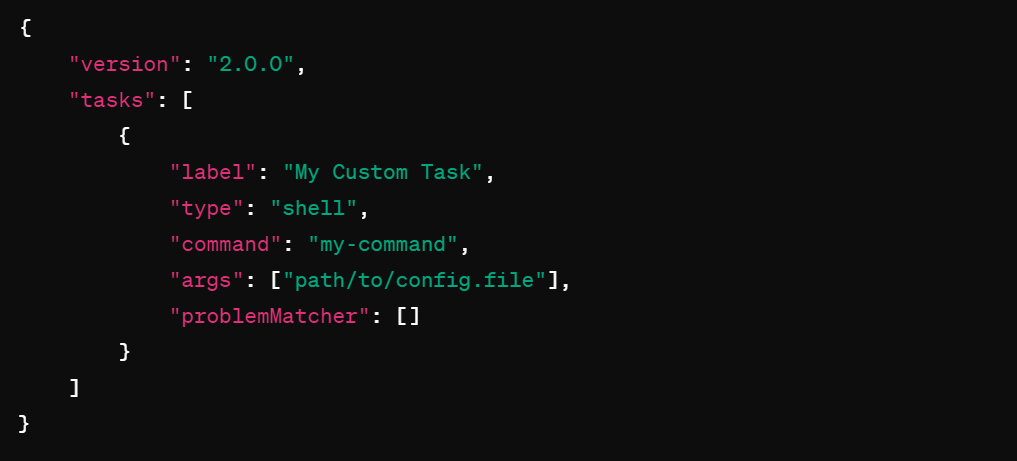
Пример настройки (отключение синтаксического выделения для файлов с расширением .config):

"files.associations": { "\*.config": "plaintext" }

Эти настройки помогут улучшить производительность редактора при работе с файлами конфигурации. Вам также следует экспериментировать с различными значениями, чтобы найти оптимальные настройки для вашей рабочей среды.

4. Чтобы указать путь к файлу конфигурации для пользовательской задачи в Visual Studio Code, вы можете использовать атрибут "args" в вашем файле tasks.json.

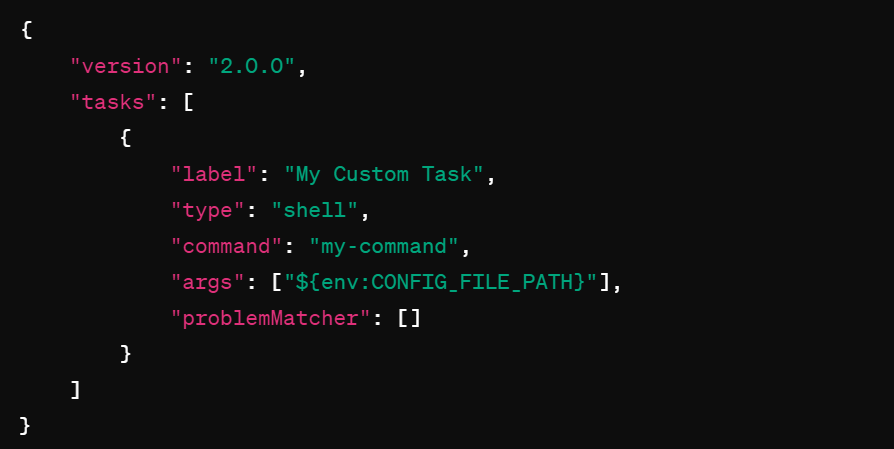
Вот пример того, как вы можете указать путь к файлу конфигурации для вашей пользовательской задачи:

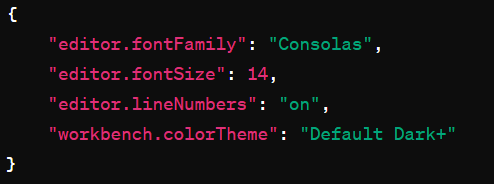


В этом примере "args" содержит массив аргументов, которые будут переданы в команду "my-command". Замените "my-command" на команду, которую вы хотите выполнить, и "path/to/config.file" на путь к вашему файлу конфигурации.

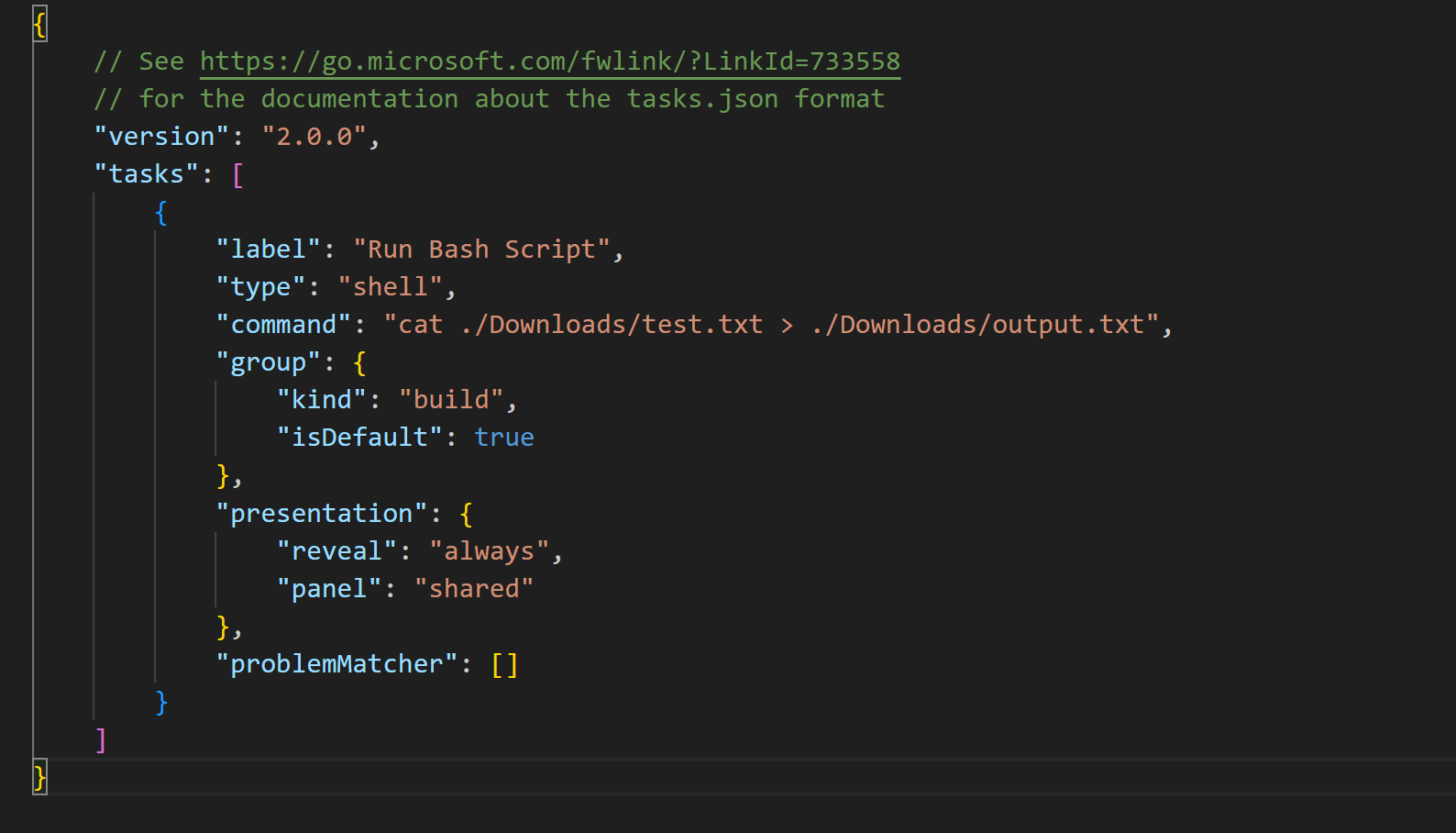
После этого, когда вы выполните эту пользовательскую задачу, она будет использовать указанный файл конфигурации вместе с вашей командой.

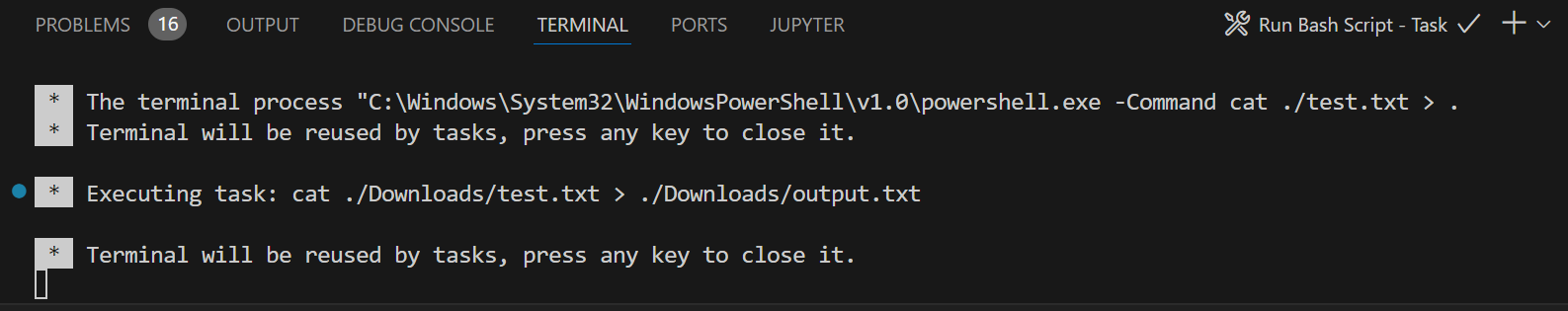
5. Вы можете использовать переменные среды в файлах конфигурации Visual Studio Code, включая файле tasks.json. Для этого вы можете использовать синтаксис ${env:ИМЯ\_ПЕРЕМЕННОЙ\_СРЕДЫ}. Пример:

  
В этом примере переменная среды CONFIG\_FILE\_PATH будет использоваться в качестве аргумента для вашей пользовательской задачи.Пожалуйста, убедитесь, что переменная среды CONFIG\_FILE\_PATH установлена в вашей операционной системе.  
 **Задача 1**



**Задача 2**





**Лабораторная** 7. Создание и работа в профиле для формирования документации

1. При работе с документацией в Visual Studio Code (VS Code) полезны следующие расширения:

* **Markdown All in One**: Это расширение обеспечивает расширенную поддержку языка разметки Markdown, включая подсветку синтаксиса, автодополнение, создание таблиц, превью и другие функции, делая работу с документацией на Markdown более удобной.
* **Markdown Preview Enhanced**: Это расширение предоставляет мощные инструменты для просмотра и редактирования документации в формате Markdown. Оно поддерживает расширенные функции превью, включая многоколоночный просмотр, вставку математических формул и даже поддержку схем JSON.
* **vscode-pandoc**: Это расширение интегрирует VS Code с Pandoc, инструментом конвертации документов. Оно позволяет легко конвертировать документы из одного формата в другой, что особенно полезно при работе с различными форматами документации.
* **Code Spell Checker**: Это расширение проверяет правописание в вашем коде и документации, помогая избежать опечаток и ошибок. Оно может быть полезно при написании и редактировании документации, чтобы убедиться, что текст не содержит орфографических ошибок.
* **Document This**: Это расширение автоматически создает комментарии для функций и методов на основе их сигнатуры. Это может быть полезно при документировании кода, поскольку оно автоматически создает заголовки с описанием параметров и возвращаемого значения.
* **TODO Highlight**: Это расширение выделяет ключевые слова TODO, FIXME и другие в вашем коде и документации, делая их более заметными и помогая отслеживать незавершенные задачи и замечания в документации.

Эти расширения помогут вам улучшить процесс создания, редактирования и просмотра документации в Visual Studio Code, делая его более удобным и эффективным.

1. Visual Studio Code предоставляет ряд функций, которые значительно облегчают процесс написания и форматирования текста в документации. Вот некоторые из них:

* **Автодополнение и IntelliSense**: VS Code предлагает автодополнение кода и контекстно-зависимые подсказки (IntelliSense) для различных языков разметки, таких как Markdown и HTML. Это ускоряет процесс написания, предлагая доступные теги, атрибуты и другие элементы.
* **Форматирование кода**: VS Code имеет встроенную функцию форматирования кода, которая автоматически выравнивает и структурирует текст в соответствии с выбранным стилем. Это полезно для поддержания единообразного форматирования в документации.
* **Просмотр предварительного просмотра Markdown**: Расширение для Markdown в VS Code позволяет просматривать результаты форматирования в реальном времени. Это помогает видеть, как будет выглядеть текст после применения разметки Markdown.
* **Быстрое создание заголовков и списков**: В VS Code есть горячие клавиши для быстрого создания заголовков разных уровней и списков различных типов (нумерованных и маркированных). Это значительно ускоряет процесс написания структурированных документов.
* **Проверка орфографии**: Некоторые расширения для VS Code предоставляют функции проверки орфографии в реальном времени, помогая обнаруживать и исправлять опечатки и грамматические ошибки в документации.
* **Сохранение и синхронизация документов**: VS Code автоматически сохраняет изменения в документах, что предотвращает потерю данных в случае сбоя или выключения. Также вы можете использовать интеграцию с облачными хранилищами, такими как GitHub или OneDrive, для синхронизации документации между разными устройствами.
* **Live Share**: Если вам нужно совместно работать над документацией с коллегами, вы можете использовать расширение Live Share для VS Code, которое позволяет нескольким пользователям редактировать документы одновременно в реальном времени.

Эти функции значительно облегчают процесс написания и форматирования текста в документации, делая его более удобным и эффективным.

1. Настройка среды разработки в Visual Studio Code (VS Code) для работы с различными форматами документации, такими как Markdown, reStructuredText и HTML, включает несколько шагов:

* **Установка расширений**:
  1. Для работы с Markdown: Установите расширение "Markdown All in One" для поддержки языка разметки Markdown.
  2. Для работы с reStructuredText: Установите расширение "reStructuredText" или "reStructuredText (reST) Snippets".
  3. Для работы с HTML: В VS Code уже встроена поддержка HTML, но вы также можете установить дополнительные расширения для улучшения работы с HTML, например, "HTML CSS Support" для автодополнения CSS в HTML файле.
* **Просмотр превью**:
  1. Для Markdown: В VS Code есть встроенный просмотр Markdown. Вы можете открыть файл Markdown и нажать Ctrl+Shift+V (или Cmd+Shift+V на macOS) для просмотра превью.
  2. Для reStructuredText и HTML: Вам может потребоваться установить дополнительные расширения для просмотра превью, такие как "reStructuredText" или "HTML Preview".
* **Автодополнение и подсказки**:
  1. Расширения Markdown и reStructuredText предоставляют автодополнение и контекстно-зависимые подсказки для ускорения процесса написания кода.
  2. Для HTML: В VS Code уже есть встроенная поддержка автодополнения для HTML, которая поможет вам быстрее писать код.
* **Форматирование**:
  1. В VS Code есть встроенная функция форматирования, которая может быть использована для автоматического выравнивания и структурирования кода в файлах Markdown, reStructuredText и HTML. Просто используйте горячие клавиши Shift+Alt+F (или Shift+Option+F на macOS) для форматирования выделенного текста или всего файла.
* **Интеграция с другими инструментами**:
  1. Многие расширения предлагают интеграцию с другими инструментами для улучшения процесса разработки. Например, вы можете использовать расширение "Markdown All in One" для автоматической генерации таблиц содержания, ссылок и многое другое в файлах Markdown.

Следуя этим шагам, вы сможете настроить среду разработки в VS Code для работы с различными форматами документации, обеспечивая удобство и эффективность при написании и форматировании текста.

1. Visual Studio Code (VS Code) предоставляет несколько инструментов и функций для проверки правописания и стиля текста в документах. Вот некоторые из них:

* **Встроенная проверка орфографии**: VS Code включает в себя базовый механизм проверки орфографии, который автоматически выделяет слова с ошибками в тексте. Ошибочные слова обычно подчеркиваются красным или другим цветом, в зависимости от используемой темы оформления.
* **Расширения для проверки орфографии и стиля**:
  1. **Code Spell Checker**: Это расширение обеспечивает проверку орфографии в вашем коде и текстовых файлах. Оно выделяет опечатки и грамматические ошибки, помогая вам их исправить.
  2. **LanguageTool**: Расширение LanguageTool предоставляет проверку грамматики и стиля для текста на разных языках. Оно помогает обнаруживать и исправлять структурные ошибки, повторяющиеся слова, неправильное использование пунктуации и многое другое.
  3. **Grammarly**: Это расширение предлагает расширенную проверку грамматики и стиля текста с использованием сервиса Grammarly. Оно обнаруживает и исправляет различные виды ошибок, включая орфографические, грамматические и стилистические.
* **Форматирование кода и текста**: VS Code включает в себя функцию автоматического форматирования кода и текста, которая помогает поддерживать единообразный стиль в документах. Просто используйте команду "Format Document" (обычно доступна через горячие клавиши Shift+Alt+F или Shift+Option+F на macOS) для автоматического выравнивания и структурирования вашего кода и текста.
* **Расширения для стиля кода и текста**: Существуют расширения, которые предлагают расширенные функции для проверки и поддержания стиля кода и текста в соответствии с выбранными стандартами и рекомендациями. Например, расширение "Prettier" обеспечивает автоматическое форматирование кода с использованием различных стилей и настроек.

Эти инструменты и функции помогают поддерживать высокий уровень правописания и стиля текста в документах, делая процесс написания и редактирования более удобным и профессиональным.

1. Visual Studio Code (VS Code) предоставляет несколько возможностей для совместной работы над документацией и отслеживания изменений:

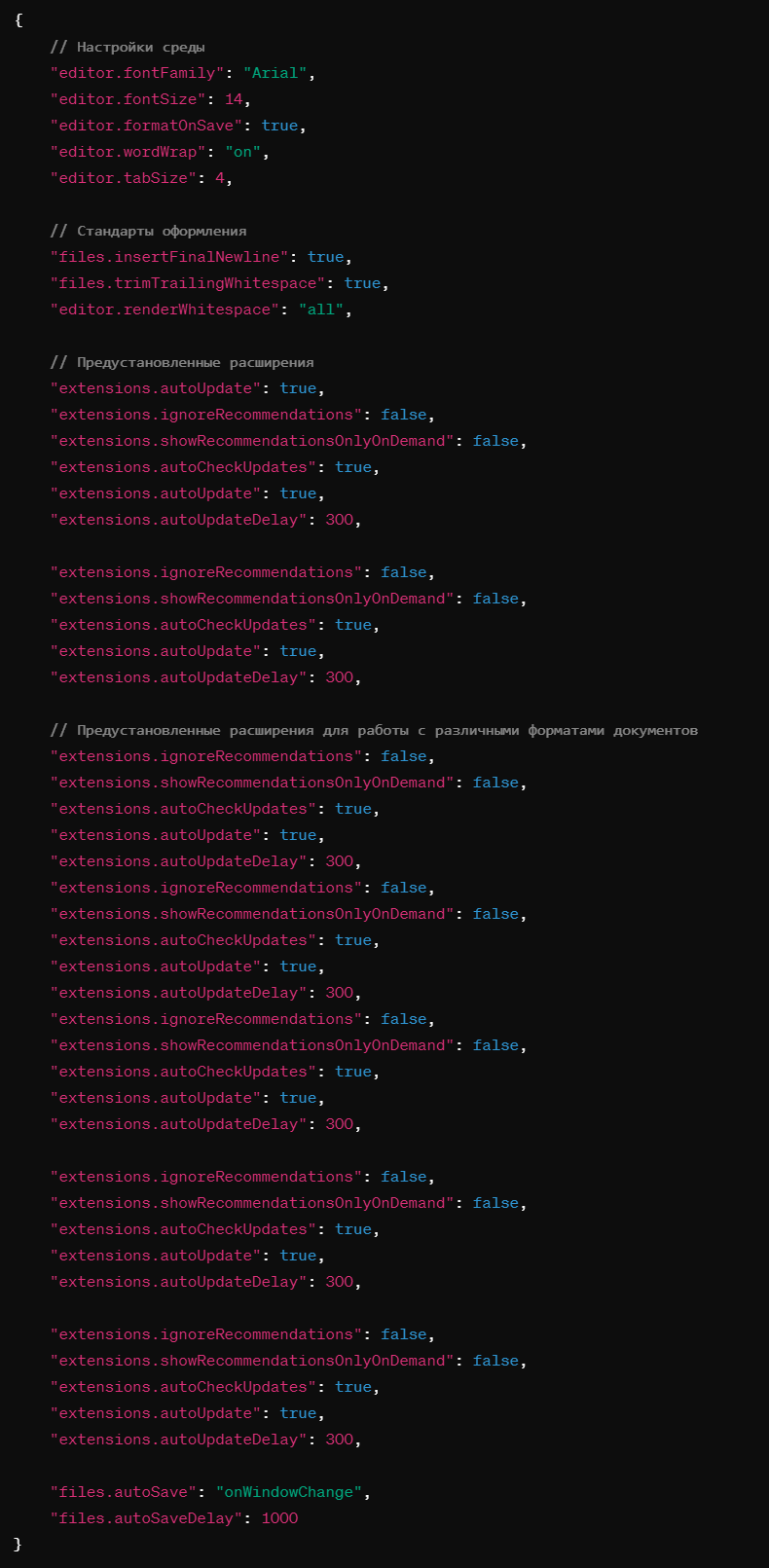
* **Интеграция с системами контроля версий (VCS)**: VS Code имеет встроенную поддержку популярных систем контроля версий, таких как Git. Это позволяет команде легко отслеживать изменения в документации, делать коммиты, ветвиться и сливать изменения, а также работать параллельно над различными ветками проекта.
* **Расширение Live Share**: Live Share позволяет нескольким разработчикам работать в реальном времени над одним и тем же файлом документации. Это позволяет совместно редактировать документы, обмениваться комментариями и идеями, а также проводить код-ревью, не покидая среду VS Code.
* **Интеграция с облачными сервисами**: VS Code поддерживает интеграцию с облачными хранилищами, такими как GitHub, GitLab и Bitbucket. Это позволяет команде легко совместно работать над документацией, загружать изменения, создавать запросы на объединение и вести отслеживание задач прямо из VS Code.
* **Расширение Markdown All in One**: Расширение Markdown All in One предоставляет дополнительные функции для работы с Markdown-документацией, включая превью, автоматическое создание таблиц содержания, переходы по заголовкам и многое другое. Это делает совместную работу над Markdown-файлами более удобной и эффективной.
* **Интеграция с коммуникационными инструментами**: Многие коммуникационные платформы, такие как Slack, Microsoft Teams и другие, предоставляют расширения для VS Code, которые позволяют вам обмениваться ссылками на файлы документации, комментариями и другой информацией прямо из редактора кода.

Эти возможности делают совместную работу над документацией в VS Code более удобной и эффективной, обеспечивая высокий уровень синхронизации и координации в команде.

**Задача 1**

Этот шаблон включает:

* Настройки среды для шрифта, размера шрифта, автоматического форматирования при сохранении и другие.
* Стандарты оформления, такие как вставка финальной пустой строки, удаление завершающих пробелов и отображение пробелов в тексте.
* Предустановленные расширения, которые помогут вам работать с различными форматами документов.



**Задача 2**

Чтобы использовать это расширение:

1. Откройте документ в Visual Studio Code.
2. Выберите разделы и добавьте заголовки с использованием символов #.
3. Нажмите Ctrl+Shift+P, выберите "Generate Table of Contents" и нажмите Enter.
4. Содержание будет автоматически сгенерировано в начале документа.



**Лабораторная 8**. VS Code для Data Science Profile Template

1. При работе в области Data Science, некоторые из наиболее полезных расширений для Visual Studio Code включают:

* **Python**: Расширение Python обеспечивает поддержку для языка Python в VS Code, включая функции подсказок, автодополнения, отладки и многое другое. Это важно для многих задач в Data Science, так как Python часто используется в этой области.
* **Jupyter**: Расширение Jupyter обеспечивает поддержку интерактивных блокнотов Jupyter внутри VS Code. Оно позволяет запускать, редактировать и отлаживать блокноты Jupyter прямо в редакторе кода.
* **Git**: Расширение Git интегрирует функции управления версиями Git непосредственно в VS Code. Это полезно для отслеживания изменений в коде и сотрудничества с другими разработчиками.
* **Markdown All in One**: Это расширение обеспечивает поддержку языка разметки Markdown в VS Code, что полезно для написания документации, отчетов и комментариев к коду.
* **Pylance**: Расширение Pylance предоставляет более быструю и точную поддержку для языка Python в VS Code. Оно использует Language Server Protocol для обеспечения функций, таких как подсказки, автодополнение и анализ кода.
* **Docker**: Расширение Docker облегчает работу с контейнерами Docker внутри VS Code. Это может быть полезно для создания изолированных окружений для разработки и тестирования приложений Data Science.
* **Visual Studio IntelliCode**: Это расширение использует машинное обучение для предложения умных подсказок и автодополнений кода на основе контекста вашей работы. Это может значительно ускорить процесс написания кода.

Эти расширения помогут улучшить процесс разработки и увеличить производительность при работе в области Data Science в Visual Studio Code.

2. Настройка среды разработки в Visual Studio Code для работы с Jupyter Notebook включает несколько шагов. Вот пошаговая инструкция:

* **Установка Visual Studio Code**: Если у вас еще нет Visual Studio Code, загрузите и установите его с [официального сайта](https://code.visualstudio.com/).
* **Установка расширения Python**: Откройте Visual Studio Code, перейдите во вкладку Extensions (иконка в боковой панели слева или нажмите Ctrl+Shift+X), введите "Python" в строке поиска и установите расширение Python, разработанное Microsoft.
* **Установка расширения Jupyter**: Теперь установите расширение Jupyter. Воспользуйтесь тем же способом, что и в предыдущем шаге, но вместо "Python" введите "Jupyter" в строке поиска и установите расширение "Jupyter" от Microsoft.
* **Установка Jupyter и ipykernel**: Убедитесь, что у вас установлен Jupyter и ipykernel в вашей среде Python. Если нет, выполните следующие команды в терминале или командной строке:

pip install jupyter pip install ipykernel

* **Настройка ядра (Kernel)**: После установки Jupyter и ipykernel, выполните следующую команду в терминале или командной строке, чтобы добавить вашу среду Python в качестве ядра Jupyter:

python -m ipykernel install --user --name=myenv --display-name "Python (myenv)"

Где myenv - это название вашей среды Python. Замените его на фактическое название вашей среды.

* **Открытие Jupyter Notebook в VS Code**: Теперь вы можете открыть или создать файл Jupyter Notebook прямо в Visual Studio Code. Просто откройте командное окно (нажмите Ctrl+Shift+P), введите "Jupyter: Create New Blank Notebook" для создания нового блокнота или "Jupyter: Open Notebook" для открытия существующего.

После выполнения этих шагов вы будете готовы использовать Jupyter Notebook в Visual Studio Code. Вы сможете выполнять код, добавлять новые ячейки, использовать интерактивные функции и многое другое, прямо в редакторе кода.

3. Visual Studio Code (VS Code) предлагает несколько функций и расширений, которые могут значительно упростить работу с данными и их визуализацией. Вот некоторые из них:

* **Расширение Python**: Расширение Python для VS Code обеспечивает мощные инструменты для работы с данными на языке Python. С его помощью вы можете напрямую выполнять анализ данных, обработку и визуализацию в удобной среде разработки.
* **Интеграция с Jupyter Notebook**: Расширение Jupyter позволяет вам работать с блокнотами Jupyter прямо в VS Code. Это значит, что вы можете выполнять ячейки кода, создавать графики и визуализации, а также сохранять результаты работы в удобном формате.
* **Интерактивные окна данных**: Некоторые расширения позволяют создавать интерактивные окна данных прямо в VS Code. Это позволяет вам быстро и удобно исследовать данные, строить графики и выполнять анализ без необходимости использовать сторонние инструменты.
* **Расширения для визуализации данных**: Существует множество расширений для VS Code, предназначенных для визуализации данных. Некоторые из них предлагают широкий набор графических инструментов, который позволяет создавать различные типы графиков и диаграмм для анализа данных.
* **Интеграция с библиотеками визуализации данных**: Вы также можете использовать VS Code с библиотеками визуализации данных, такими как Matplotlib, Seaborn, Plotly и др. Редактор кода обеспечивает удобную среду разработки для создания и настройки графиков и диаграмм с помощью этих библиотек.
* **Поддержка работы с CSV и другими форматами данных**: VS Code предоставляет удобные инструменты для работы с различными форматами данных, включая CSV, JSON, XML и другие. Это позволяет быстро загружать, просматривать и редактировать данные прямо в редакторе кода.
* **Расширения для работы с базами данных**: Если ваши данные хранятся в базе данных, вы можете воспользоваться расширениями для VS Code, которые обеспечивают интеграцию с популярными СУБД, такими как PostgreSQL, MySQL, MongoDB и другими. Это позволяет вам выполнять SQL-запросы, просматривать и редактировать данные напрямую из редактора кода.

Эти функции и расширения помогут вам упростить работу с данными и их визуализацией прямо в среде разработки VS Code, делая процесс анализа данных более удобным и эффективным.

**4.** Visual Studio Code (VS Code) предоставляет несколько возможностей для совместной работы над проектами в области Data Science:

* **Интеграция с системами контроля версий**: VS Code имеет встроенную поддержку систем контроля версий, таких как Git. Это позволяет команде легко отслеживать изменения в коде, сливать их, а также работать параллельно над различными ветками проекта.
* **Live Share**: Это расширение для VS Code, которое позволяет нескольким разработчикам работать в реальном времени над одним и тем же проектом. С помощью Live Share участники могут совместно редактировать код, обмениваться комментариями и отладочными данными прямо в редакторе кода.
* **Обмен кодом и документацией**: В VS Code можно легко обмениваться кодом, блокнотами Jupyter и другой документацией с помощью встроенных инструментов обмена файлами, а также с помощью расширений, таких как Slack, Microsoft Teams и других.
* **Использование контейнеров Docker**: VS Code обеспечивает интеграцию с Docker, что позволяет упаковывать ваш проект Data Science в контейнеры Docker. Это упрощает развертывание и управление окружением разработки, а также обеспечивает консистентность среды между членами команды.
* **Управление зависимостями и окружением**: С помощью VS Code и соответствующих расширений вы можете легко управлять зависимостями Python, создавать виртуальные среды, устанавливать и обновлять пакеты, что помогает поддерживать консистентность окружения между членами команды.
* **Использование облачных сервисов**: Многие облачные сервисы для анализа данных и машинного обучения предоставляют интеграцию с VS Code. Это позволяет командам легко работать с данными, моделями и ресурсами в облаке, совместно используя их в проектах.

Эти возможности позволяют командам Data Science эффективно сотрудничать, обмениваться идеями и ресурсами, а также управлять проектами, обеспечивая высокий уровень совместной работы и производительности.

**5.** Visual Studio Code (VS Code) предоставляет несколько инструментов и расширений для управления зависимостями и виртуальными средами в Python:

* **Установка и управление пакетами с помощью Terminal**: Вы можете управлять зависимостями Python непосредственно из терминала в VS Code. Используйте команды pip для установки, обновления и удаления пакетов.
* **Установка расширения Python**: Установите расширение Python для VS Code, чтобы получить доступ к дополнительным инструментам для работы с Python, включая управление зависимостями. Расширение позволяет устанавливать и обновлять пакеты, создавать виртуальные среды и многое другое прямо из интерфейса VS Code.
* **Использование виртуальных сред с помощью venv или virtualenv**: VS Code интегрируется с инструментами создания виртуальных сред для Python, такими как venv или virtualenv. Вы можете создавать новые виртуальные среды, активировать их и устанавливать в них зависимости, используя интерфейс VS Code или терминал.
* **Использование Conda**: Если вы используете Conda для управления окружением Python, VS Code также обеспечивает интеграцию с этим инструментом. Вы можете создавать, управлять и активировать Conda-среды прямо из интерфейса VS Code.
* **Автоматическое обнаружение виртуальных сред**: VS Code автоматически обнаруживает виртуальные среды Python в вашей системе и предлагает их для использования в проектах. Это делает процесс управления зависимостями более удобным и эффективным.
* **Интеграция с Docker**: Если вы предпочитаете использовать контейнеры Docker для управления окружением Python, VS Code обеспечивает интеграцию с Docker, что позволяет вам создавать и использовать контейнеры для разработки и тестирования.

Эти средства позволяют управлять зависимостями и виртуальными средами Python непосредственно из VS Code, делая процесс разработки более гибким и удобным.

**Лабораторная** 9. Создание и отладка веб-приложений с использованием расширений VSCode для браузера.

1. "Debugger for Chrome" предоставляет возможности для отладки веб-приложений в браузере Google Chrome. Вот основные возможности этого расширения:

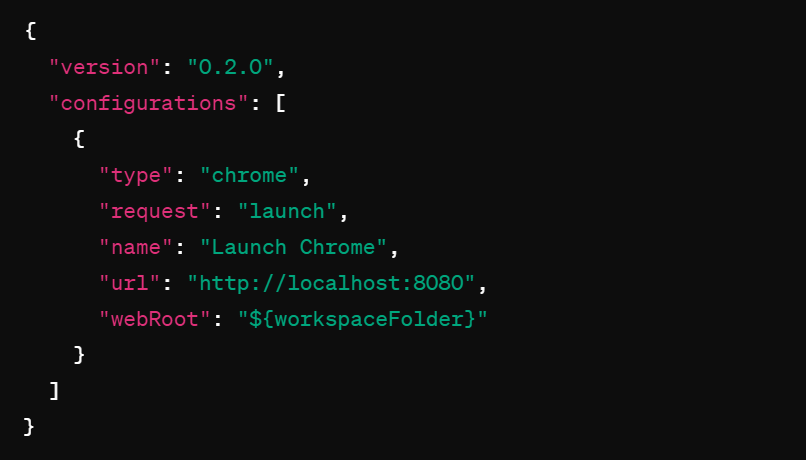
* **Отладка JavaScript и TypeScript**: Расширение позволяет запускать и отлаживать код JavaScript и TypeScript прямо из среды разработки VS Code.
* **Отладка в браузере Google Chrome**: Расширение позволяет подключить VS Code к экземпляру браузера Google Chrome и выполнять отладку вашего веб-приложения прямо в браузере.
* **Поддержка точек останова (breakpoints)**: Вы можете устанавливать точки останова в коде и пошагово выполнять код, отслеживая его выполнение.
* **Просмотр переменных и стека вызовов**: Расширение позволяет просматривать значения переменных и стек вызовов во время выполнения программы.
* **Контроль над выполнением кода**: Вы можете управлять выполнением кода, приостанавливать его, возобновлять выполнение, шагать по коду и многое другое.
* **Конфигурация для разных сценариев отладки**: Вам доступны различные конфигурации для отладки различных типов приложений, таких как приложения на основе Node.js, веб-приложения на локальном сервере и т.д.

Для начала работы с отладкой веб-приложений в Visual Studio Code с помощью расширения "Debugger for Chrome", выполните следующие шаги:

* Установите расширение "Debugger for Chrome".
* Откройте свой проект в VS Code.
* Создайте или откройте файл launch.json.
* Добавьте или настройте конфигурацию для отладки в браузере Chrome.
* Запустите ваше веб-приложение в браузере Chrome.
* Выберите нужную конфигурацию отладки в VS Code и запустите ее.

После этого вы сможете использовать все функции отладчика для отслеживания выполнения вашего кода в браузере Chrome.

1. Для настройки расширения "Debugger for Chrome" в Visual Studio Code для работы с вашим проектом веб-приложения, вам нужно выполнить следующие шаги:

* **Установите расширение "Debugger for Chrome"**:
  1. Откройте Visual Studio Code.
  2. Нажмите Ctrl+Shift+X (в Windows/Linux) или Cmd+Shift+X (на macOS), чтобы открыть панель расширений.
  3. Введите "Debugger for Chrome" в поле поиска.
  4. Установите расширение, нажав кнопку "Install".
* **Откройте ваш проект в Visual Studio Code**:
  1. Откройте папку, содержащую ваш проект в Visual Studio Code.
* **Создайте или настройте файл** launch.json:
  1. Нажмите Ctrl+Shift+D (в Windows/Linux) или Cmd+Shift+D (на macOS), чтобы открыть панель отладки.
  2. Нажмите на значок шестеренки (Configure or Fix 'launch.json').
  3. Выберите "Chrome" из списка предварительно настроенных сред.
  4. Это создаст файл launch.json в папке .vscode вашего проекта и добавит в него настройки для отладки в браузере Chrome.
* **Настройте конфигурацию отладки**:
  1. Отредактируйте файл launch.json согласно вашим потребностям.
  2. Вот пример базовой конфигурации для отладки веб-приложения:
* В этом примере указано, что веб-приложение запущено на http://localhost:8080. Замените это значение на адрес вашего локального сервера.
* **Запустите ваше веб-приложение**:
  + Запустите ваше веб-приложение на локальном сервере.
* **Выберите конфигурацию отладки и запустите ее**:
  + Нажмите F5, чтобы запустить отладку.
  + Выберите конфигурацию отладки "Launch Chrome" из выпадающего списка конфигураций отладки.
  + Отладчик откроется, и ваше веб-приложение будет запущено в браузере Chrome, готовое для отладки.

**3.** Вот основные шаги по отладке веб-приложения с использованием расширений в Visual Studio Code:

* **Установите необходимые расширения**:
  + Убедитесь, что у вас установлены необходимые расширения для работы с вашим веб-приложением. Например, для отладки веб-приложений на JavaScript/TypeScript с использованием браузера Google Chrome, установите расширение "Debugger for Chrome".
* **Откройте ваш проект в Visual Studio Code**:
  + Откройте папку, содержащую ваш веб-проект, в Visual Studio Code.
* **Настройте файл** launch.json:
  + Создайте файл launch.json в папке .vscode вашего проекта (если он ещё не создан) или отредактируйте существующий.
  + Настройте конфигурацию отладки в соответствии с вашим веб-приложением и используемыми технологиями.
* **Запустите локальный сервер**:
  + Убедитесь, что ваше веб-приложение запущено на локальном сервере.
* **Выберите конфигурацию отладки и запустите ее**:
  + Нажмите F5 или выберите Debug > Start Debugging из меню.
  + Выберите нужную конфигурацию отладки из выпадающего списка.
* **Используйте инструменты отладки**:
  + Откройте инструменты отладки, чтобы следить за выполнением вашего кода, устанавливать точки останова и просматривать значения переменных.
  + В Visual Studio Code откройте вкладку "Отладка" (Debug).
* **Отслеживайте выполнение кода**:
  + Используйте точки останова, чтобы приостановить выполнение кода в нужном месте.
  + Шагайте по коду, используя кнопки "Шаг вперёд" и "Шаг назад", чтобы понять, как ваш код выполняется.
* **Анализируйте ошибки и проблемы**:
  + Используйте инструменты отладки для анализа ошибок и проблем в вашем веб-приложении.
  + Просматривайте стек вызовов, значения переменных и другую отладочную информацию.
* **Исправляйте проблемы**:
  + Используйте информацию, полученную в процессе отладки, чтобы исправить ошибки и проблемы в вашем веб-приложении.
* **Тестируйте исправления**:
  + После внесения изменений повторно запустите отладку, чтобы убедиться, что проблемы были исправлены.

**4.**   
Для взаимодействия с элементами DOM и отладки проблем с производительностью в веб-приложениях с использованием Visual Studio Code и его расширений, вы можете воспользоваться следующими инструментами:

* **Использование расширения "Debugger for Chrome"**:
  + Установите расширение "Debugger for Chrome".
  + Настройте файл launch.json для запуска отладки вашего веб-приложения в браузере Google Chrome.
  + Используйте инструменты разработчика браузера Google Chrome для взаимодействия с элементами DOM, а также для анализа производительности вашего приложения.
* **Отладка JavaScript и TypeScript**:
  + Установите расширение "Debugger for Chrome" и "Debugger for Firefox" (если используете Firefox).
  + Убедитесь, что ваши исходные файлы JavaScript и TypeScript настроены для отладки в соответствующем файле launch.json.
  + Используйте точки останова, консоль и другие инструменты отладки для взаимодействия с элементами DOM и анализа производительности кода JavaScript/TypeScript.
* **Использование инструментов разработчика браузера**:
  + Запустите ваше веб-приложение в браузере Google Chrome.
  + Откройте инструменты разработчика, нажав F12 или щелкнув правой кнопкой мыши и выбрав "Исследовать элемент".
  + Взаимодействуйте с элементами DOM, изменяйте их состояние и стили, чтобы тестировать различные сценарии.
  + Используйте вкладку "Performance" для анализа производительности вашего веб-приложения. Запускайте запись, чтобы собирать данные о производительности, а затем анализируйте полученную информацию.
* **Использование расширения "Live Server"**:
  + Установите расширение "Live Server", которое позволяет запустить локальный сервер для вашего веб-приложения.
  + Запустите ваше веб-приложение с помощью "Live Server".
  + Используйте инструменты разработчика браузера Google Chrome для взаимодействия с элементами DOM и анализа производительности.
* **Использование расширения "Quokka.js"**:
  + Установите расширение "Quokka.js", которое позволяет в реальном времени просматривать результаты выполнения вашего кода JavaScript/TypeScript.
  + Используйте "Quokka.js" для быстрой проверки и анализа фрагментов кода JavaScript/TypeScript.

5. Вот несколько полезных расширений для повышения эффективности разработки веб-приложений в Visual Studio Code:

* **ESLint**:
  + Позволяет использовать ESLint в вашем проекте, чтобы автоматически проверять ваш JavaScript/TypeScript на соответствие стандартам кодирования.
  + [ESLint на Visual Studio Code Marketplace](https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint)
* **Prettier - Code formatter**:
  + Позволяет автоматически форматировать ваш код JavaScript, TypeScript, HTML, CSS и другие файлы с помощью Prettier.
  + [Prettier на Visual Studio Code Marketplace](https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode)
* **Auto Rename Tag**:
  + Автоматически переименовывает закрывающий тег HTML, когда вы изменяете имя открывающего тега, и наоборот.
  + [Auto Rename Tag на Visual Studio Code Marketplace](https://marketplace.visualstudio.com/items?itemName=formulahendry.auto-rename-tag)
* **Auto Close Tag**:
  + Автоматически закрывает тег HTML при вводе закрывающего символа >.
  + [Auto Close Tag на Visual Studio Code Marketplace](https://marketplace.visualstudio.com/items?itemName=formulahendry.auto-close-tag)
* **Live Server**:
  + Запускает локальный сервер с автоматической перезагрузкой для разработки веб-приложений.
  + [Live Server на Visual Studio Code Marketplace](https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer)
* **Debugger for Chrome**:
  + Позволяет отлаживать веб-приложения в браузере Google Chrome прямо из Visual Studio Code.
  + [Debugger for Chrome на Visual Studio Code Marketplace](https://marketplace.visualstudio.com/items?itemName=msjsdiag.debugger-for-chrome)
* **Bracket Pair Colorizer**:
  + Подсвечивает парные скобки в вашем коде для лучшей читаемости.
  + [Bracket Pair Colorizer на Visual Studio Code Marketplace](https://marketplace.visualstudio.com/items?itemName=CoenraadS.bracket-pair-colorizer)
* **Path Intellisense**:
  + Автоматически дополняет имена файлов и каталогов при вводе путей в вашем коде.
  + [Path Intellisense на Visual Studio Code Marketplace](https://marketplace.visualstudio.com/items?itemName=christian-kohler.path-intellisense)
* **GitLens — Git supercharged**:
  + Предоставляет расширенные возможности работы с Git, включая аннотации, сравнение, историю, коммиты и многое другое.
  + [GitLens на Visual Studio Code Marketplace](https://marketplace.visualstudio.com/items?itemName=eamodio.gitlens)
* **Material Icon Theme**:
  + Добавляет красивые иконки для файлов и папок в вашем проекте.
  + [Material Icon Theme на Visual Studio Code Marketplace](https://marketplace.visualstudio.com/items?itemName=PKief.material-icon-theme)

Эти расширения помогут вам повысить эффективность разработки веб-приложений в Visual Studio Code, улучшить качество вашего кода и обеспечить более удобный рабочий процесс.