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:

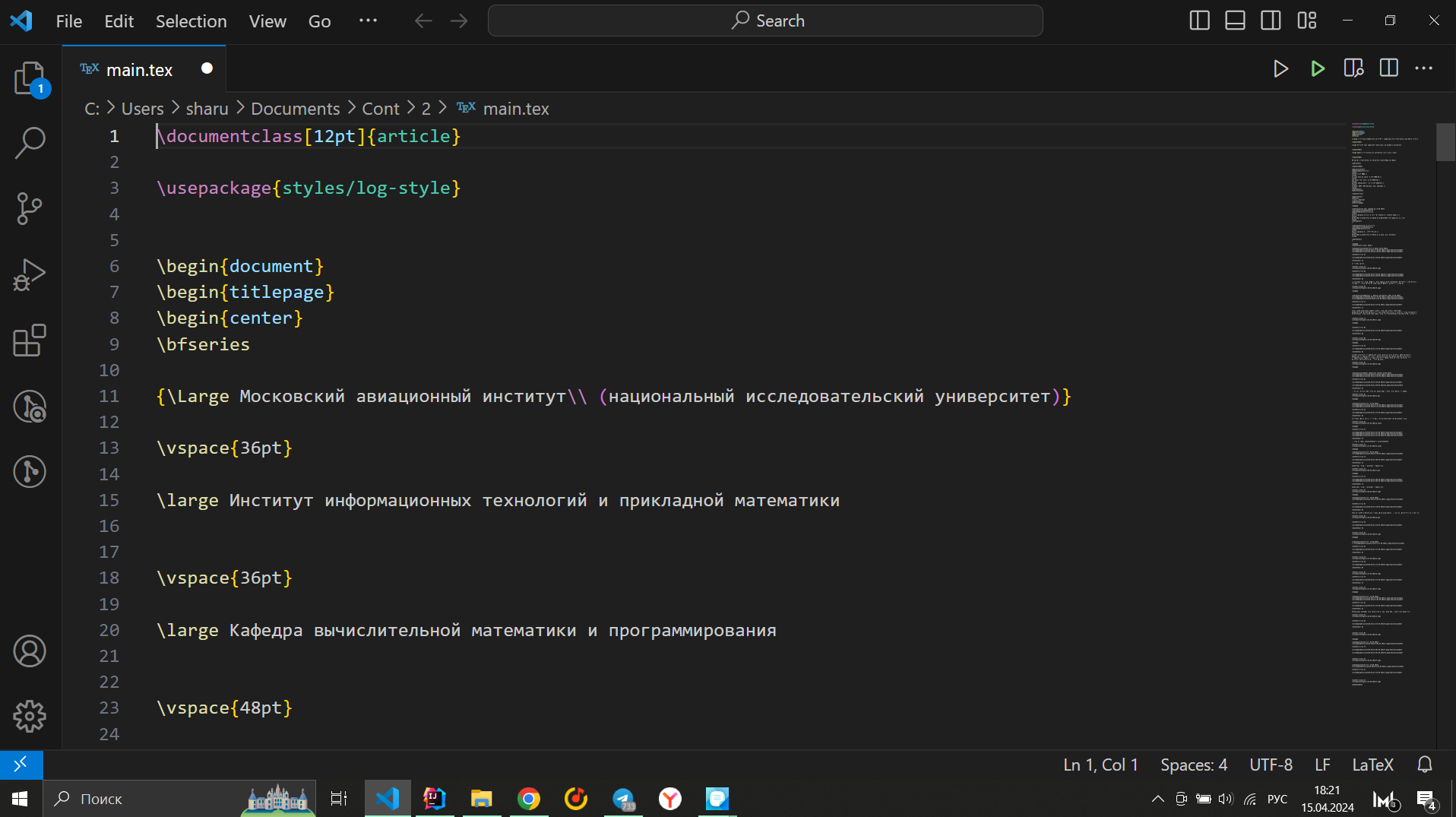
* **Увеличение производительности:** IDE предоставляет разработчику все необходимые инструменты и функции в одной среде, что упрощает рабочий процесс и повышает производительность. Вместо того чтобы использовать отдельные инструменты, разработчик может работать непосредственно в IDE, минимизируя время, 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:

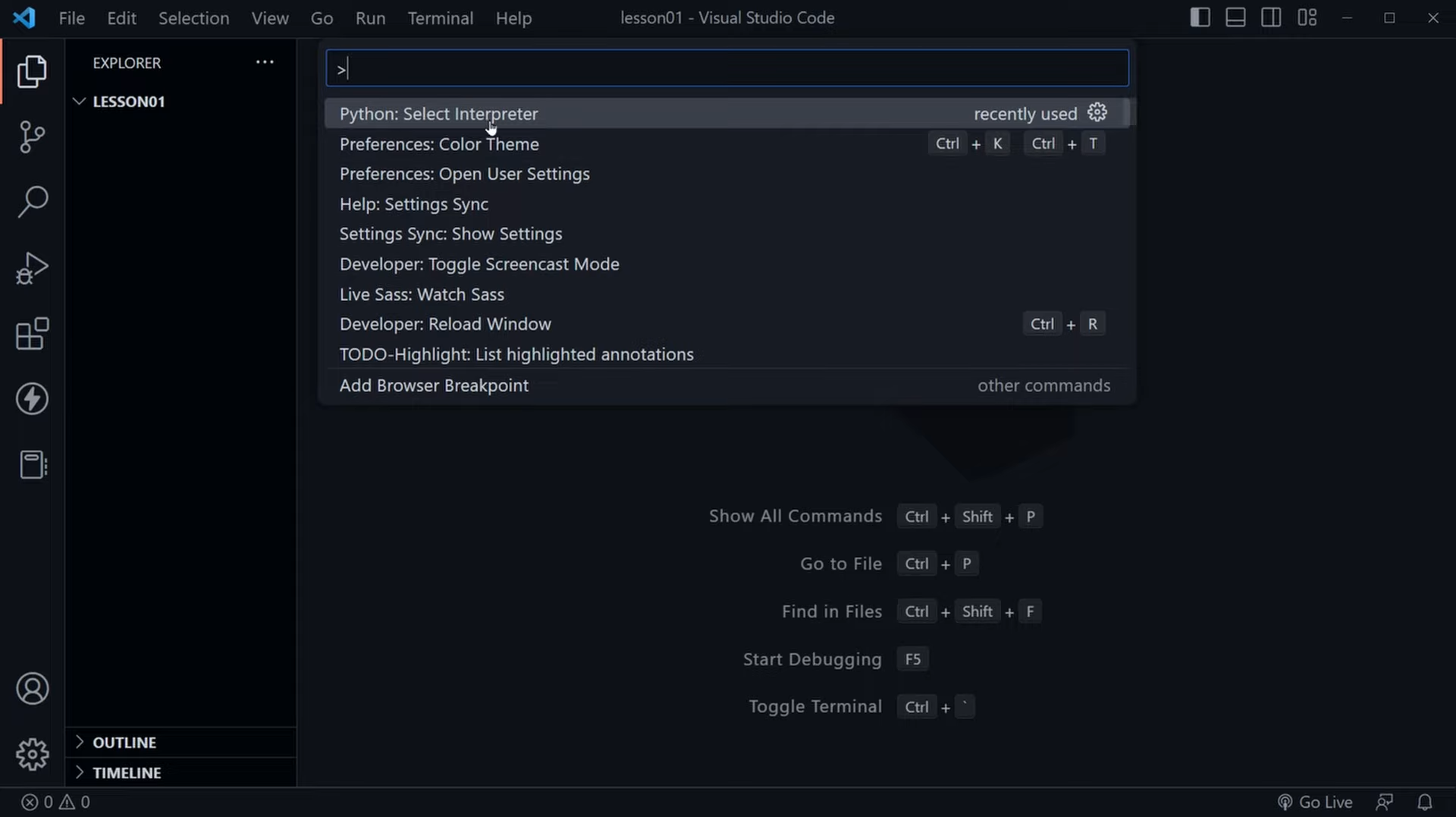
1. **Кроссплатформенность:** Visual Studio Code доступен для операционных систем Windows, macOS и Linux, что делает его доступным для широкого круга разработчиков.
2. **Легковесность и производительность:** В отличие от полноценной IDE, Visual Studio Code является легковесным текстовым редактором с возможностью 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).

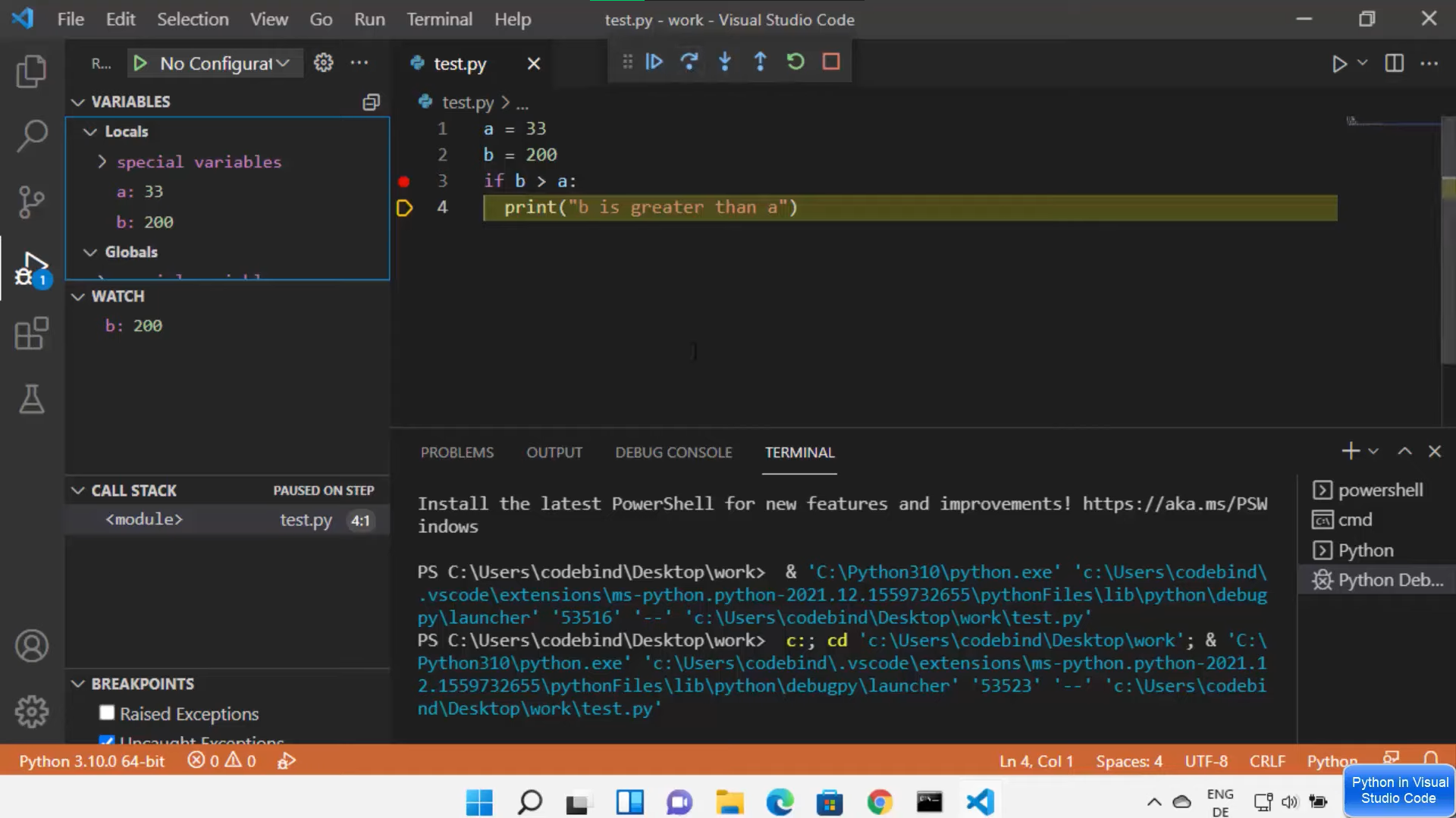
**Устанавливаем необходимые расширения:**



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**.**

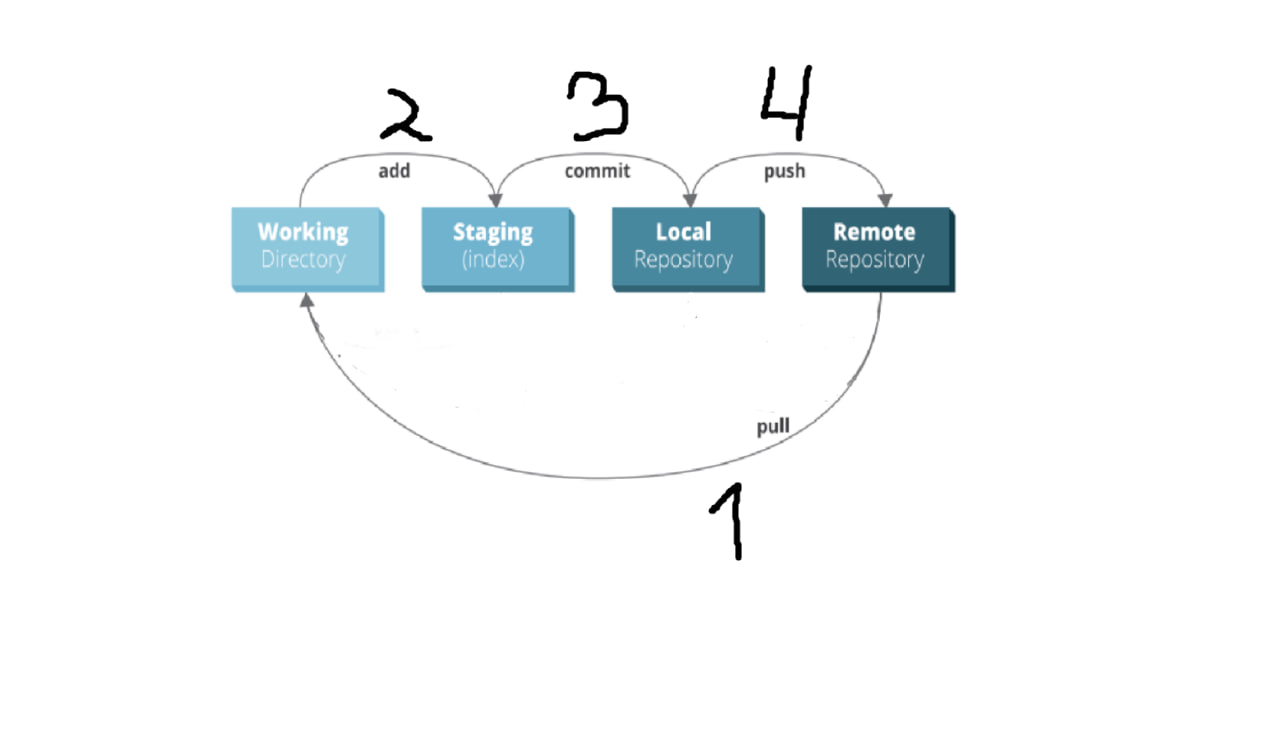
Выбираем интерпретатор:

**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:

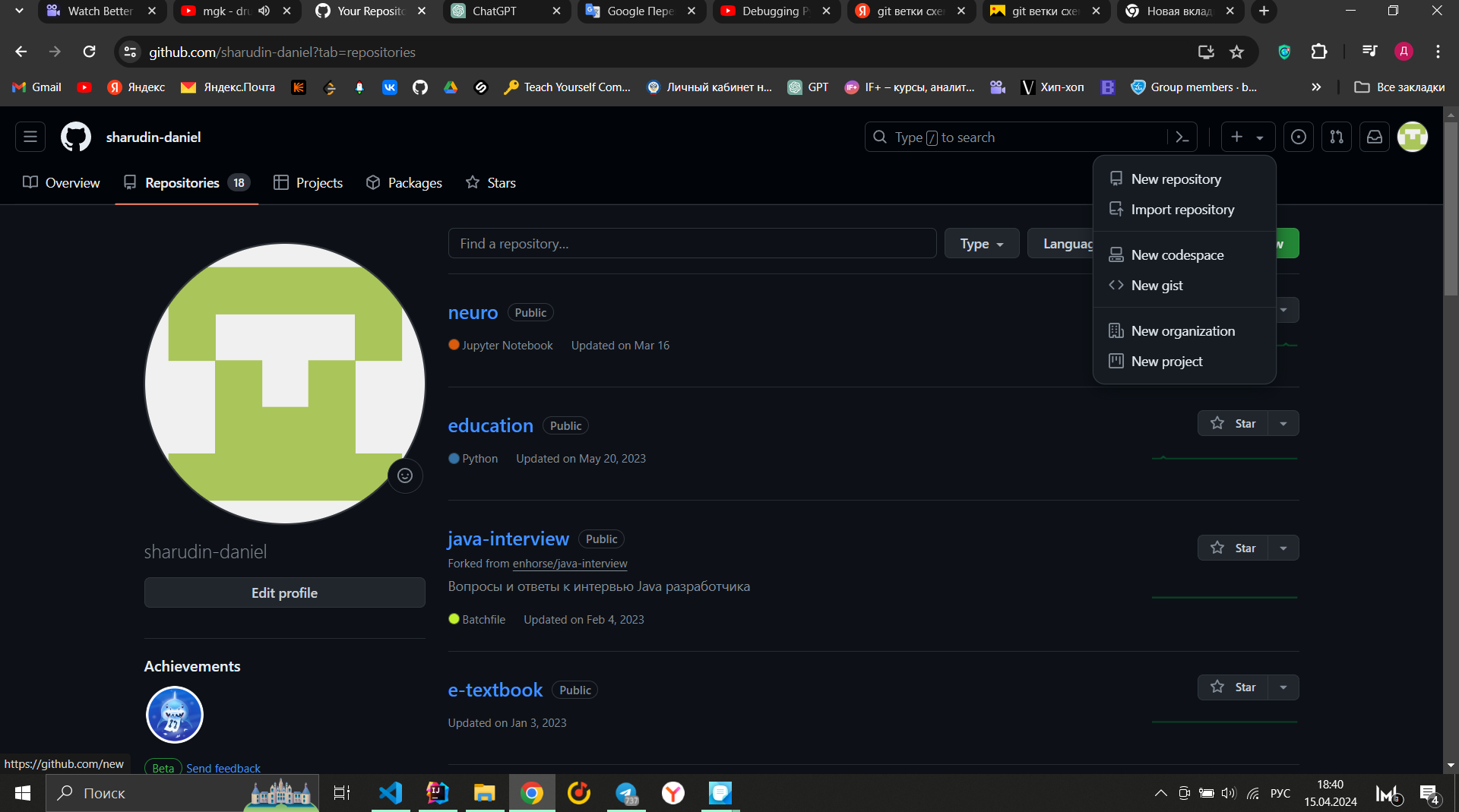
* **Клонирование (Clone):** Эта команда используется для создания локальной копии существующего удаленного репозитория. Она загружает все файлы и историю версий с удаленного репозитория на локальную машину.

git clone <URL\_удаленного\_репозитория>

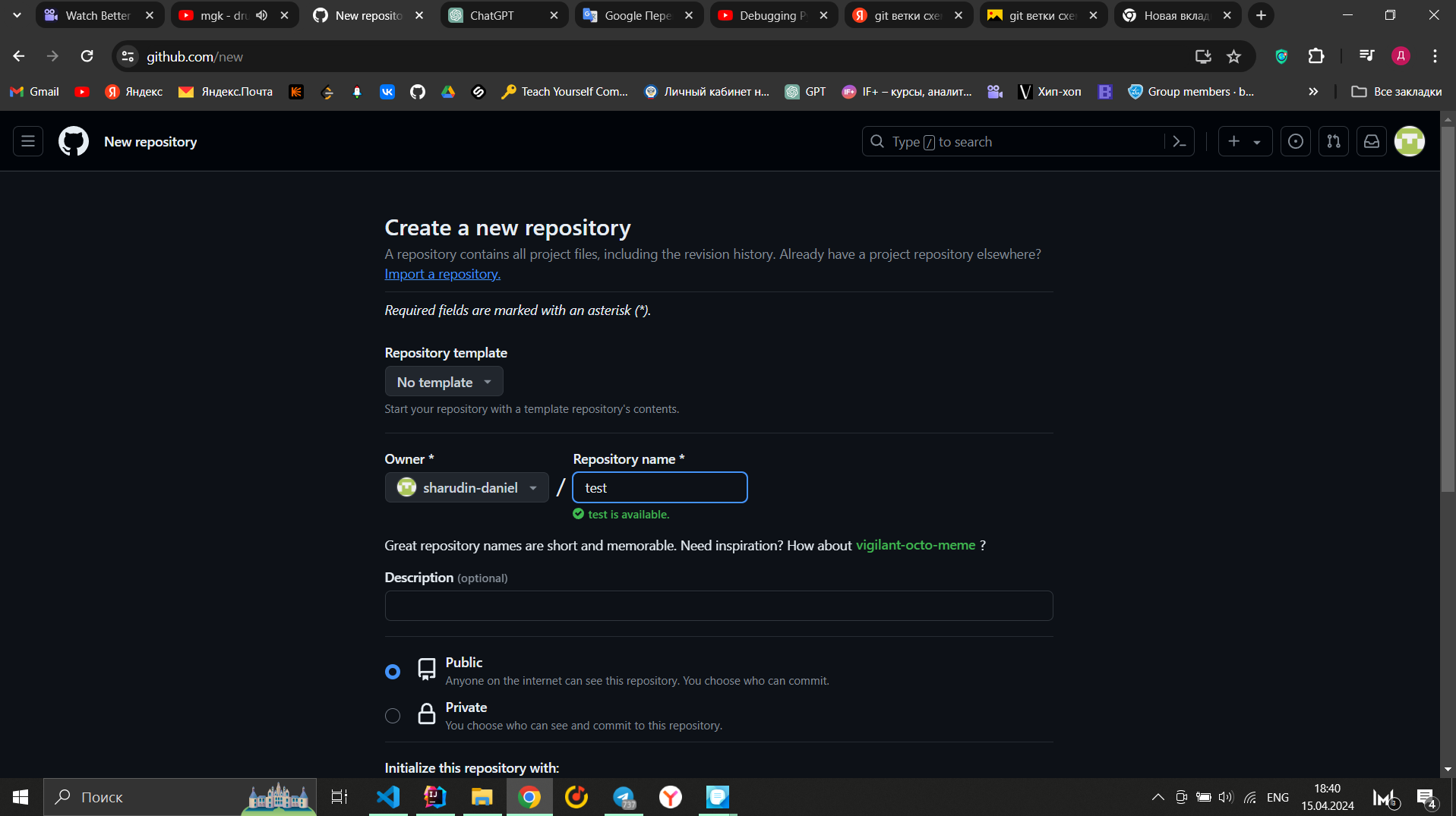
* **Инициализация (Init) и Добавление удаленного репозитория (Remote):** Эти команды используются в случае, если у вас уже есть локальный проект, и вы хотите привязать его к существующему удаленному репозиторию.

git init

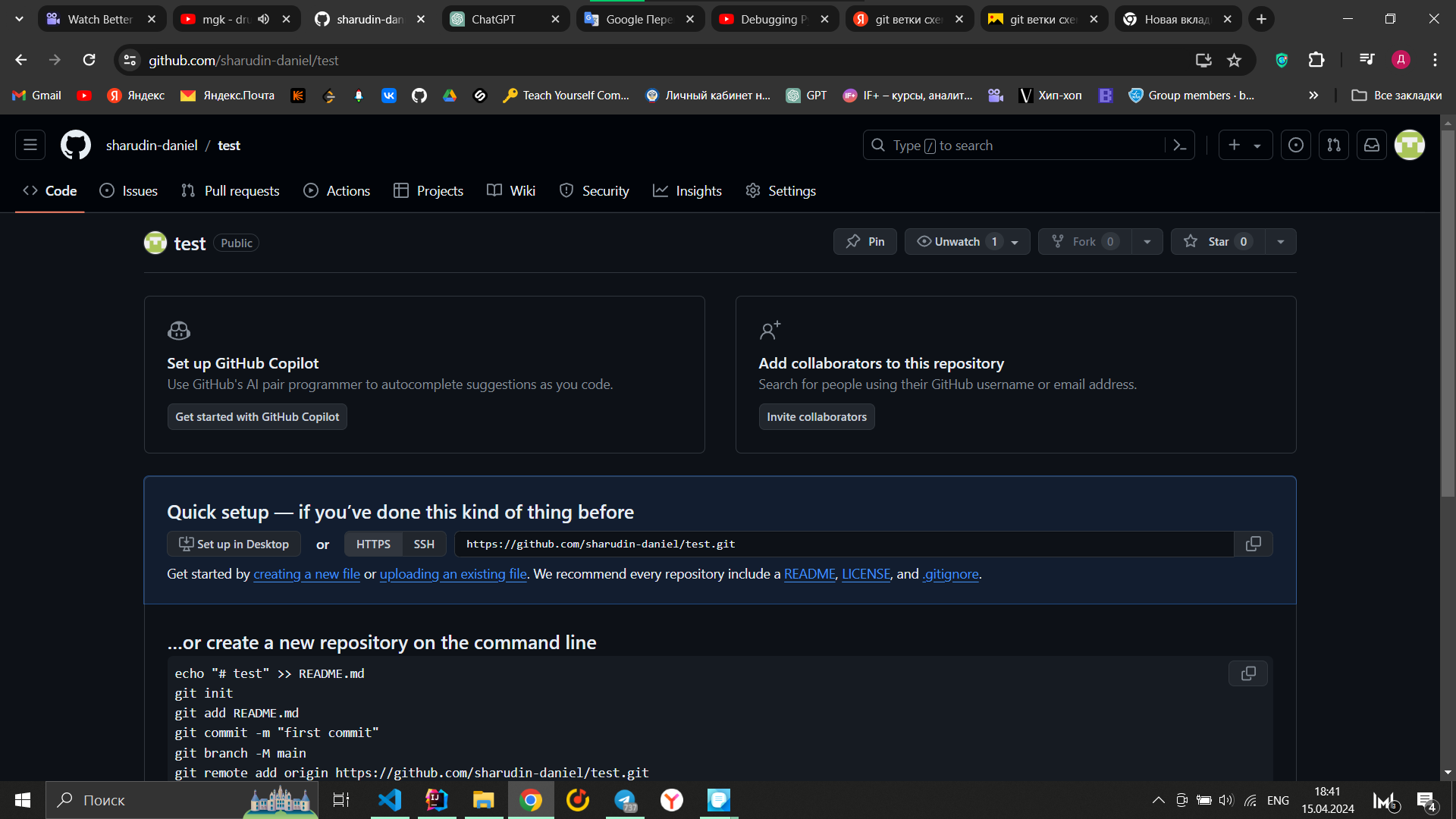
5. На главной странице нажать на кнопку создания репозитория.

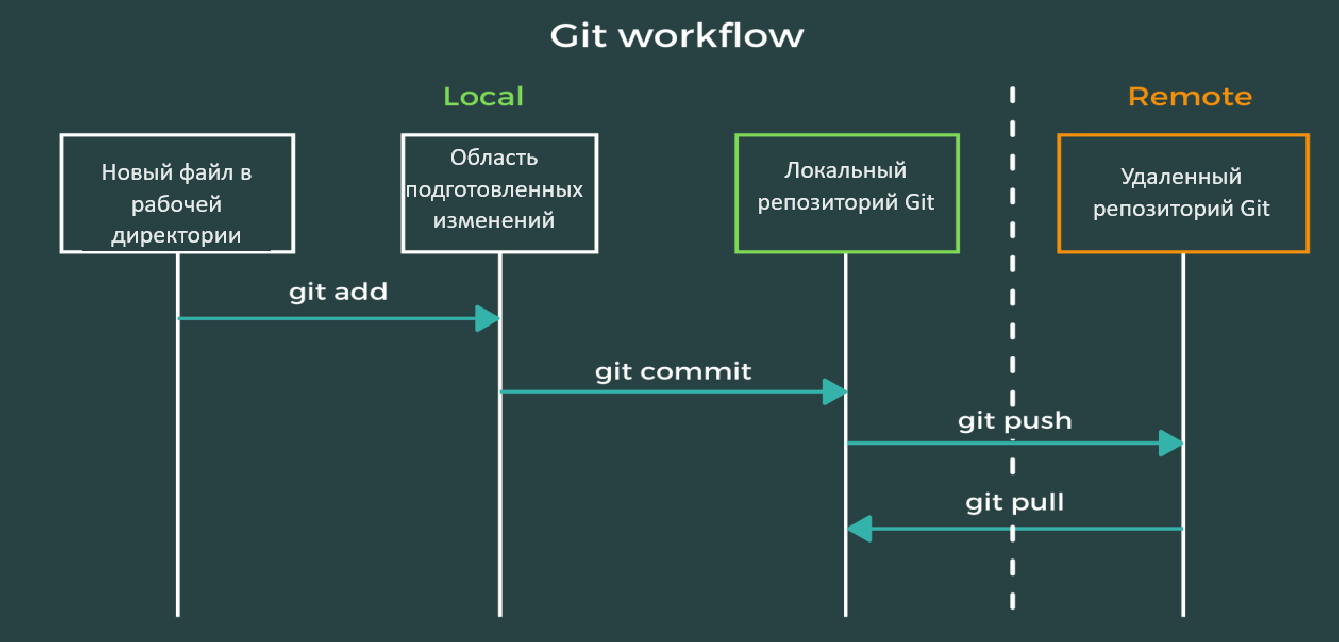


Задать нужные параметры



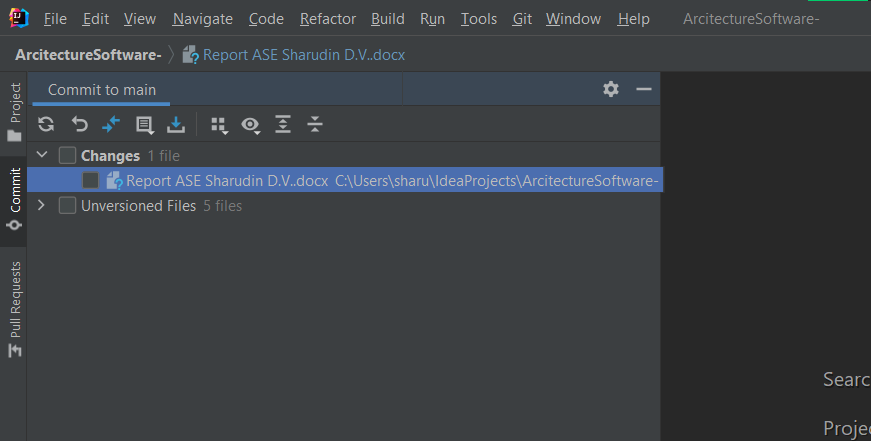
Репозиторий создан



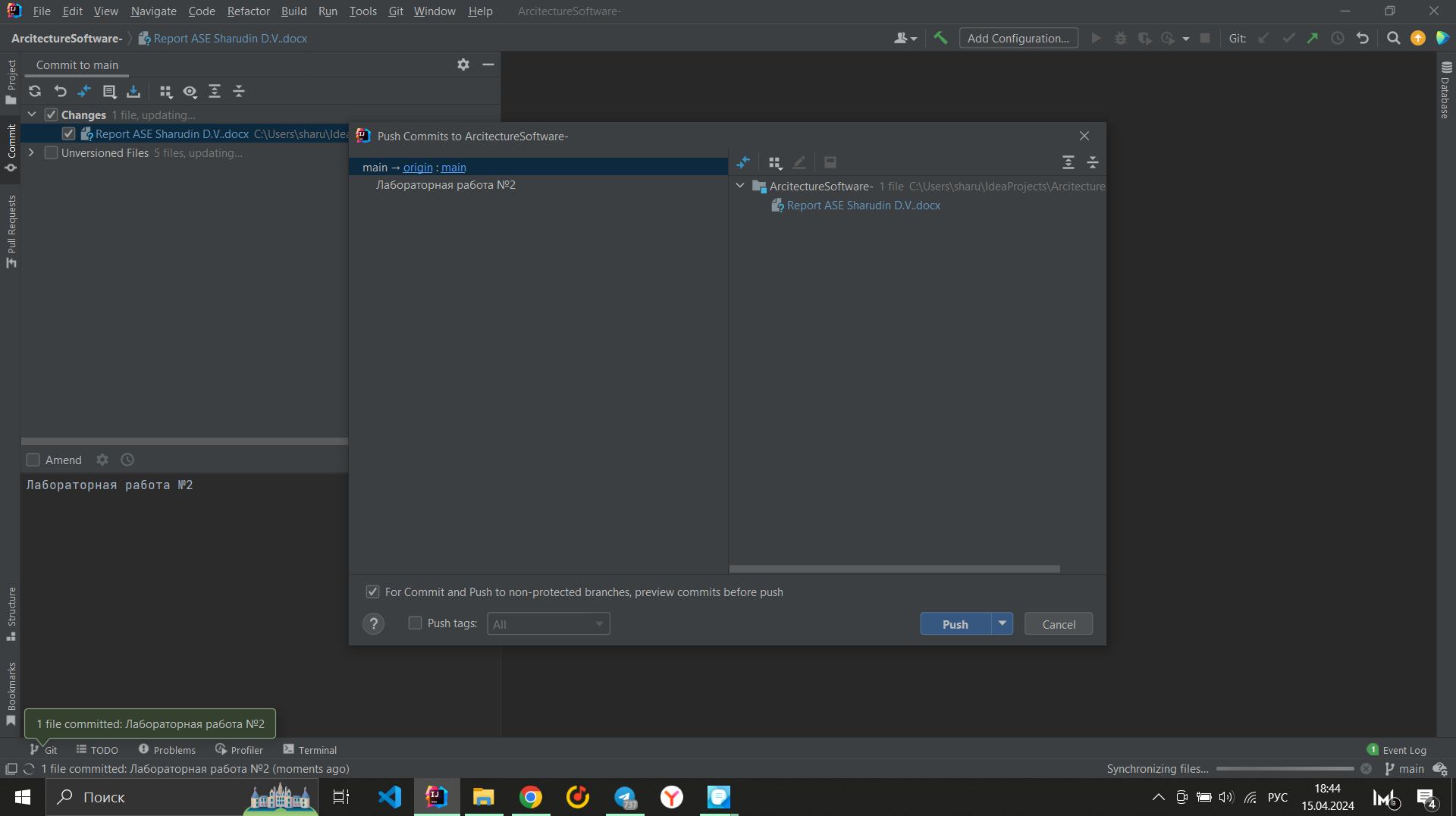
**Задание 1**

**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:

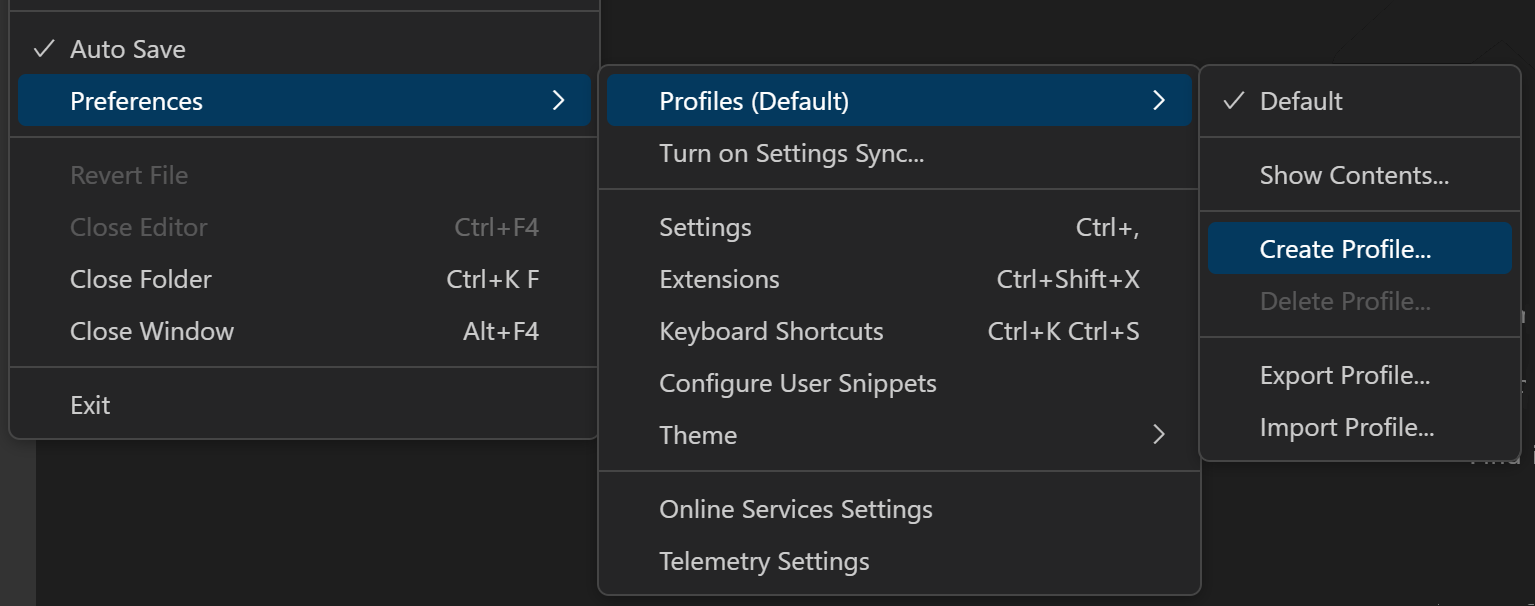
* **Настройки редактора (Editor Settings):** В этом разделе можно определить параметры отображения и поведения редактора, такие как размер отступов, используемый шрифт, цветовая схема и другие настройки визуального оформления.
* **Настройки Python (Python Settings):** Этот раздел содержит параметры, 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.

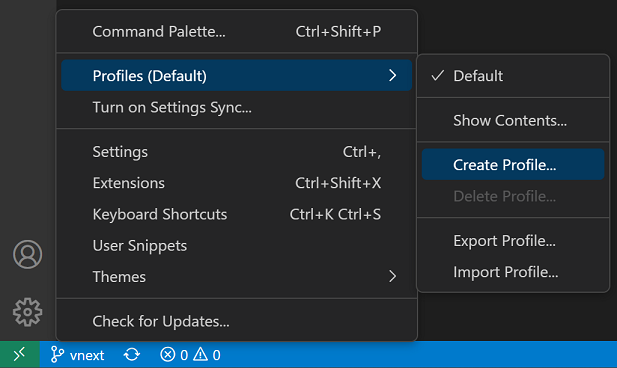
5. Для оптимизации профиля в Visual Studio Code (VS Code) и повышения производительности и удобства при разработке на Python, вы можете принять следующие меры:

* **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.

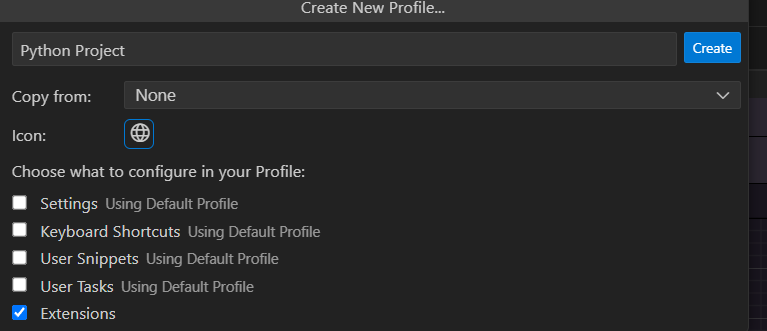
**Задание 1**

Задаем название для создаваемого профилю и выбираем корректируемые параметры.





Задаем название для создаваемого профилю и выбираем корректируемые параметры.

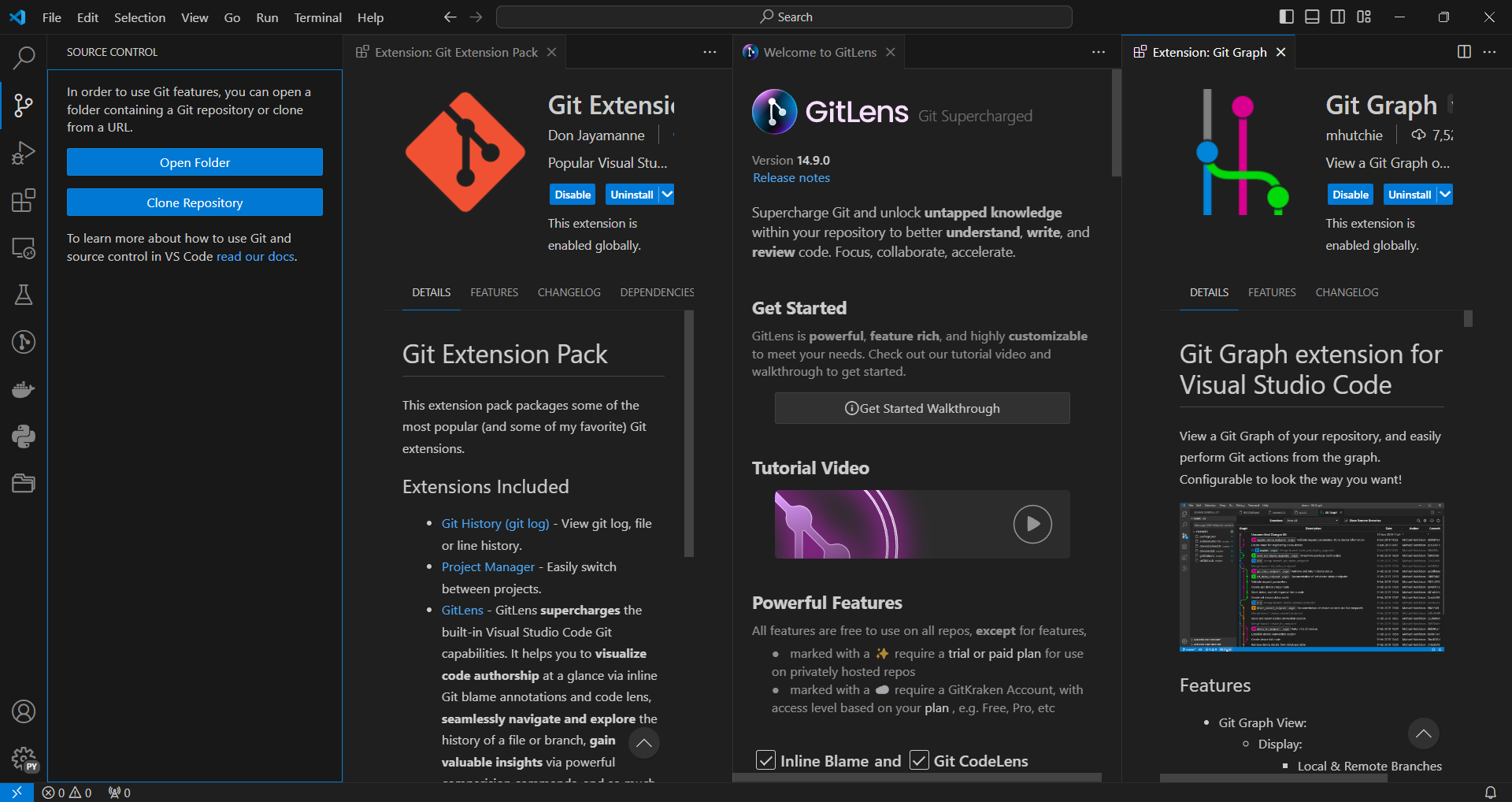


После этого видим, что в созданном профиле есть базовая часть и настройка расширений.



Устанавливаем расширения для работы с Python проектом

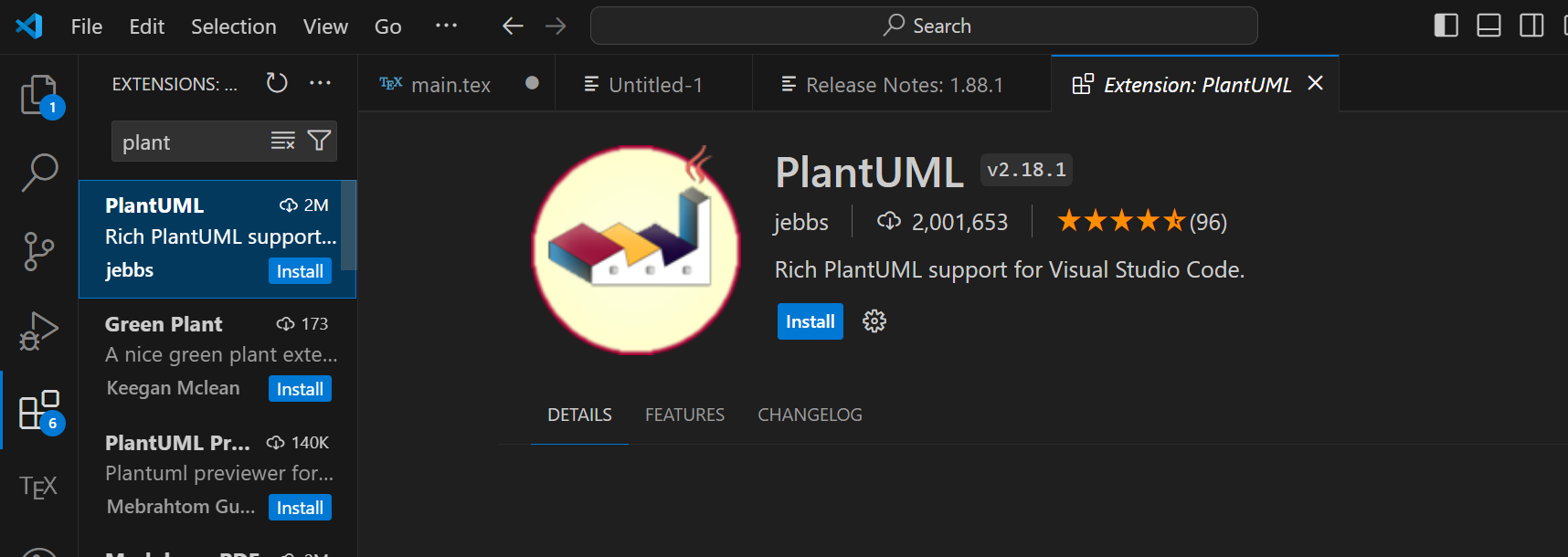
**Задание 2**



**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.

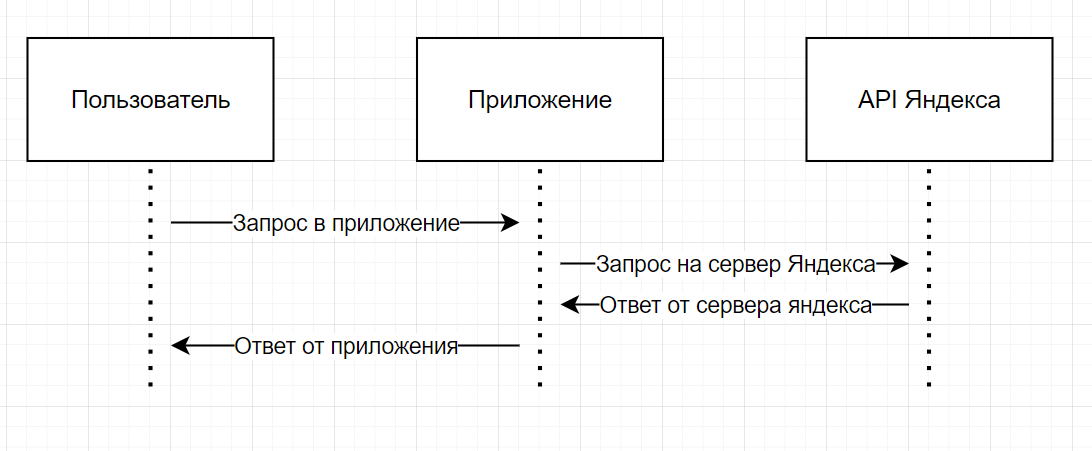
UML предоставляет стандартизированный и универсальный способ моделирования программных систем, что позволяет разработчикам лучше понимать требования, архитектуру и процессы в рамках проекта, а также облегчает коммуникацию между участниками команды разработки и заинтересованными сторонами.

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:

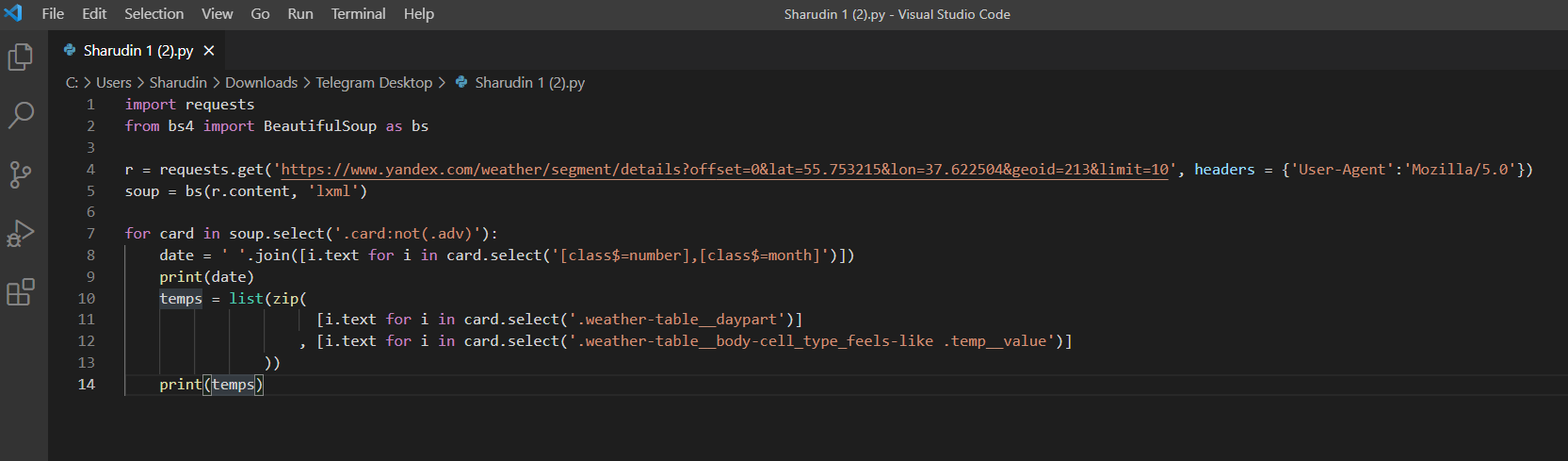
* **Визуализация концепций и структур:** Диаграммы классов, диаграммы компонентов и другие UML-диаграммы позволяют визуализировать концепции, структуры и компоненты системы. Это помогает разработчикам лучше понять архитектуру и дизайн программы.
* **Коммуникация:** UML-диаграммы служат универсальным 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.

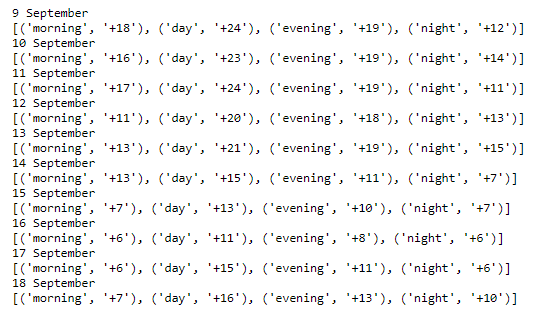
**Задание 1,2**



UML диаграмма программы



Исходная программа

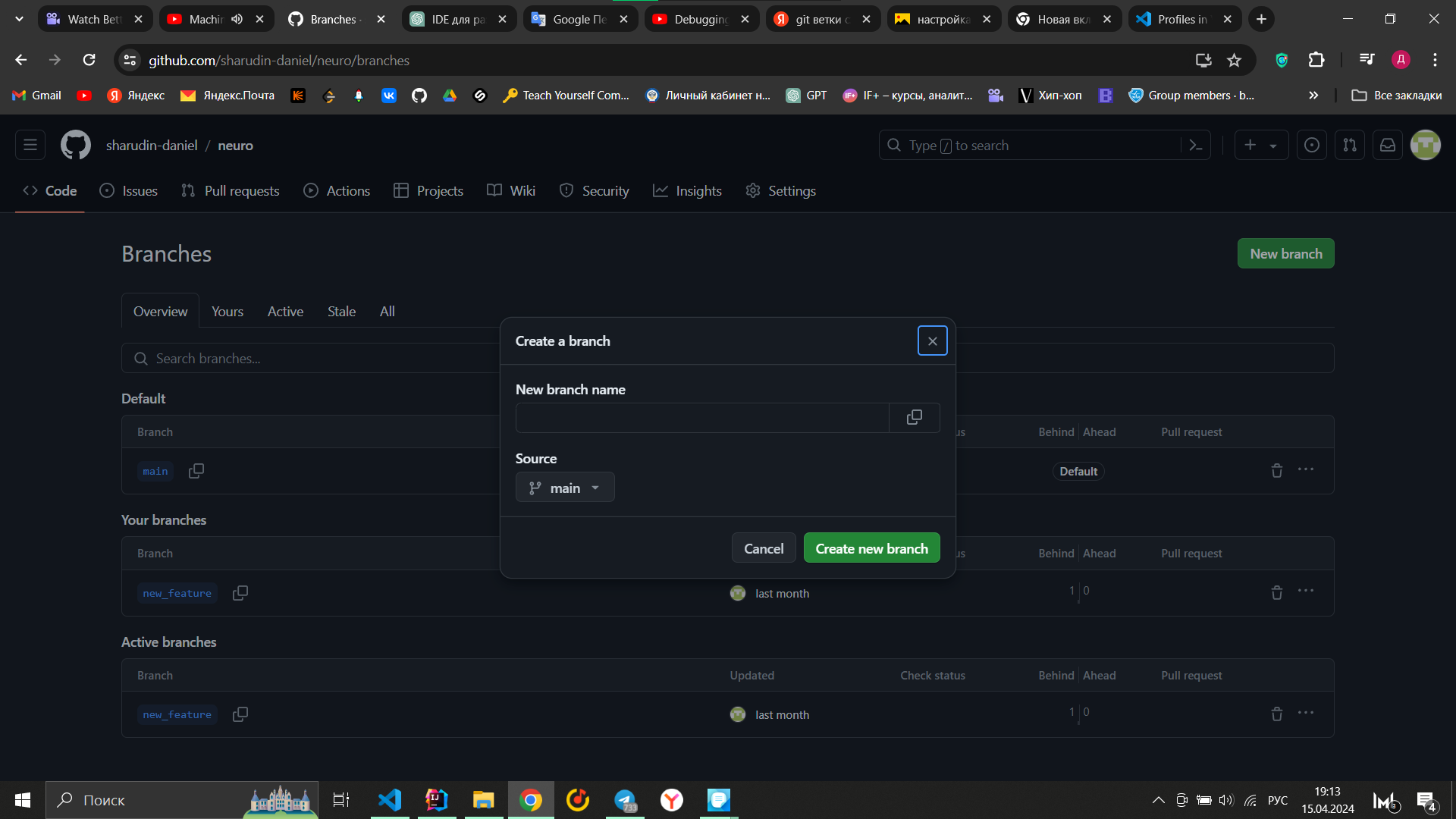


Результат работы

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.

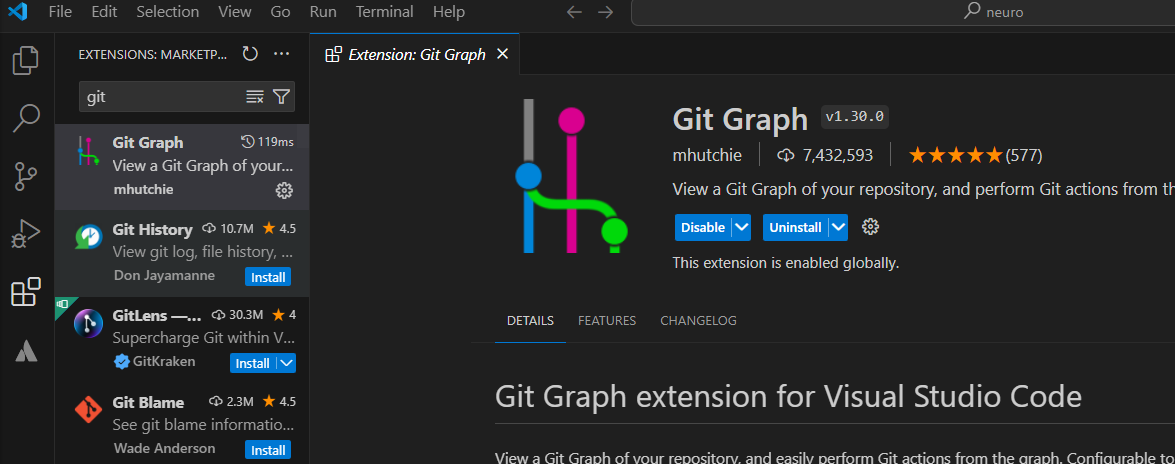
Разрешение конфликта слияния включает в себя следующие шаги:

* **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.

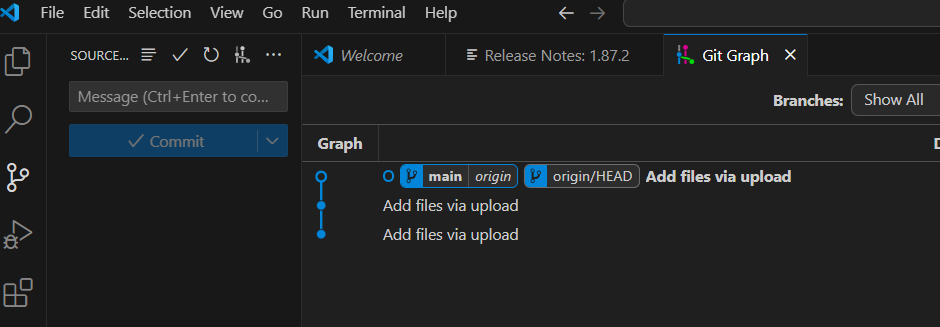
5. Расширение для удобного управления ветками.

Установим расширение Git Graph для комфортной работы с ветками Git:

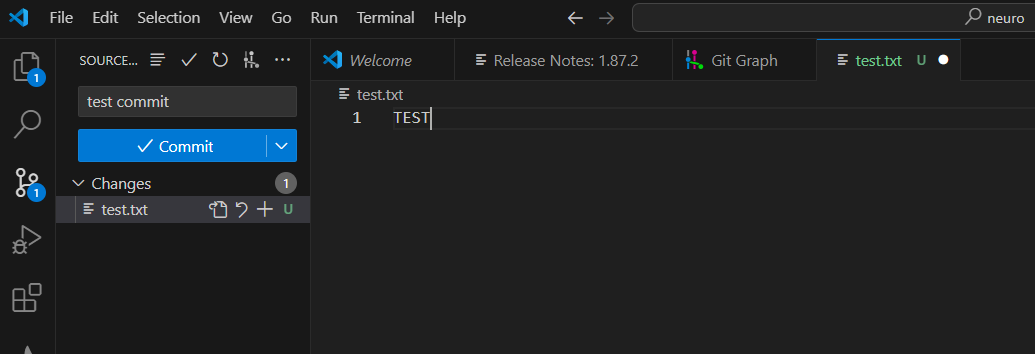


**Задание 1**

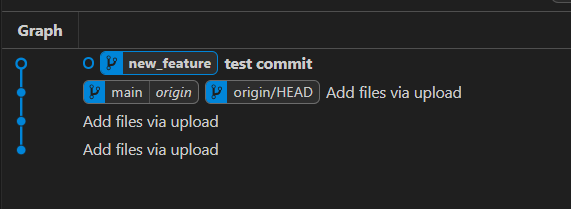
Изначально наш граф выглядит следующим образом:



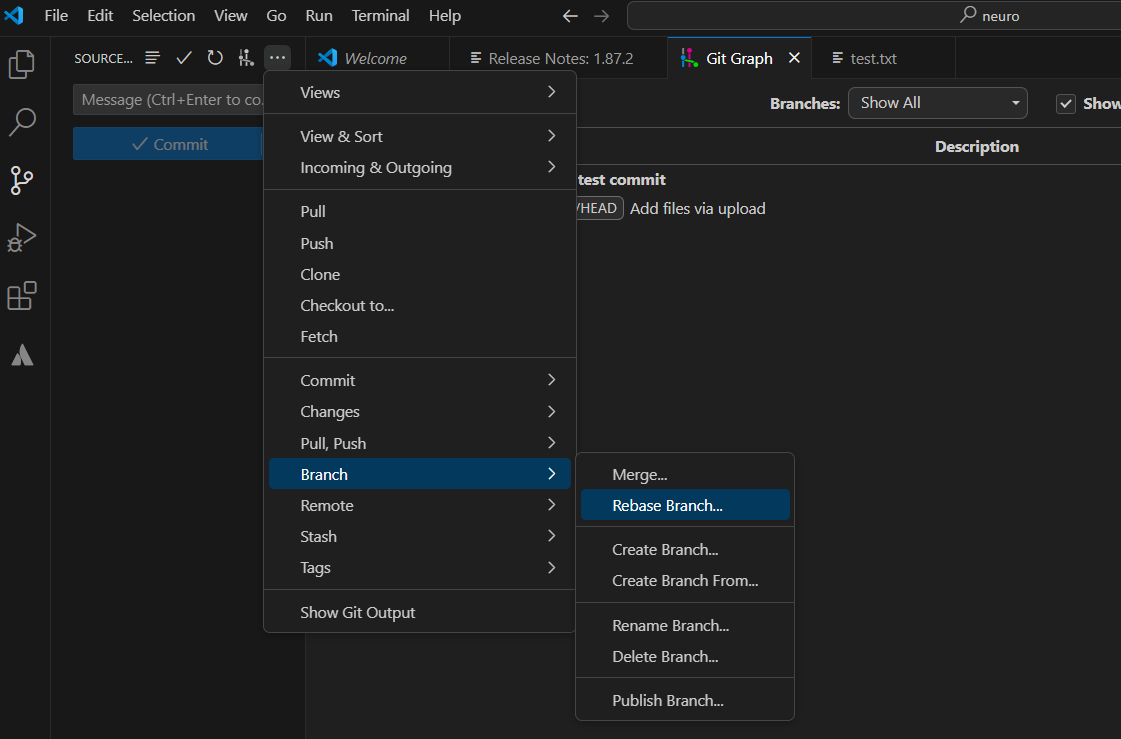
Создадим новую ветку, затем создадим новый файл и закомитим его:



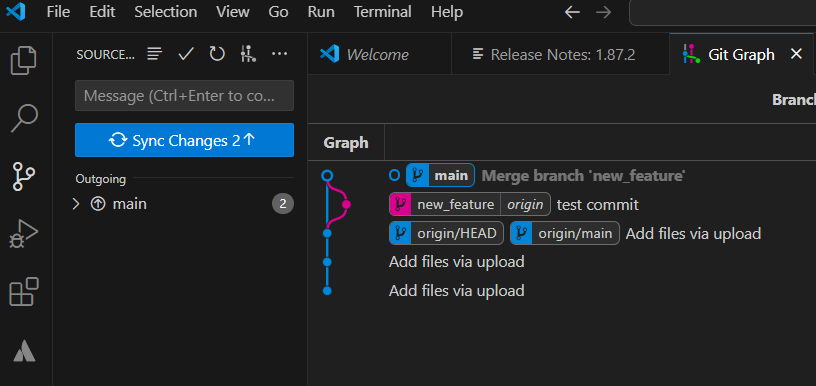
Наша новая ветка выглядит так:



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:

