**Сравнение подходов к реализации асинхронного программирования с использованием asyncio, threading и multiprocessing.**

Оглавление

[Введение 2](#_Toc184675357)

[Обоснование выбора темы: 2](#_Toc184675358)

[Определение цели и задач исследования: 2](#_Toc184675359)

[Основные понятия и определения: 3](#_Toc184675360)

[Методы и подходы к работе: 4](#_Toc184675361)

[Обзор и анализ подходов к асинхронному программированию: 5](#_Toc184675362)

[Asyncio: 5](#_Toc184675363)

[Threading: 6](#_Toc184675364)

[Multiprocessing: 7](#_Toc184675365)

[Проектирование эксперимента 8](#_Toc184675366)

[Постановка задачи: 8](#_Toc184675367)

[Тема выше озвученных задач 8](#_Toc184675368)

[Определение критериев сравнения 9](#_Toc184675369)

[Программная реализация: 10](#_Toc184675370)

[Сценарии кода для каждого подхода: 10](#_Toc184675371)

[Анализ и интерпретация результатов. 10](#_Toc184675372)

[Сравнительный анализ производительности: 10](#_Toc184675373)

[Уместность использования различных методов и рекомендация. 14](#_Toc184675374)

[Заключение 15](#_Toc184675375)

[Приложение 1. Пример файловой структуры проекта. 16](#_Toc184675376)

[Приложение 2. Список необходимых библиотек 17](#_Toc184675377)

[Приложение 3. 18](#_Toc184675378)

[Приложение 4. 21](#_Toc184675379)

[Приложение 5. 24](#_Toc184675380)

[Приложение 6. 28](#_Toc184675381)

Введение**.**

## Обоснование выбора темы:

Асинхронное программирование играет всё более важную роль в разработке современного софта (программного обеспечения), особенно в случаях, где ключевыми являются производительность и возможность масштабирования. Изучение и сравнение подходов, таких как asyncio, threading и multiprocessing, помогает понять, какой из них наиболее эффективен для конкретных задач. Эти знания ценны для разработчиков, которые стремятся сделать свои решения более оптимизированными и производительными.

## Определение цели и задач исследования:

Цель исследования:

Произвести оценку производительности и уместности использования трех ключевых подходов к асинхронному программированию, а именно asyncio, threading и multiprocessing.

Задачи исследования:

1. Изучить и изложить основные концепции каждого подхода.
2. Реализовать тест-задачи с использованием всех трех подходов.
3. Провести анализ результатов и сравнить производительность.
4. Определить, в каких сценариях подходы более уместны.

# Основные понятия и определения:

*Асинхронное программирование* - является подходом в разработке программного обеспечения, который позволяет одновременно выполнять несколько задач без блокирования основного потока выполнения.

*Threading* - это модуль для работы с потоками, которые позволяют выполнять несколько задач одновременно в одном процессе. Потоки делят общий адрес памяти и ресурсы, что делает их удобными для задач, где требуется одновременная обработка, например, загрузка данных, обработка ввода/вывода или управление сетью.

*Multiprocessing* - это модуль, который позволяет запускать несколько процессов параллельно, каждый с собственной памятью и интерпретатором Python. Это особенно полезно для задач, требующих интенсивных вычислений, где стандартный threading не может эффективно использовать несколько ядер из-за *GIL (Global Interpreter Lock)***.**

*Asyncio -* это модуль для реализации асинхронного программирования, который позволяет писать код, выполняющийся параллельно, но в пределах одного потока и процесса. Вместо создания множества потоков или процессов, как в threading или multiprocessing, asyncio использует концепцию событийного цикла (event loop), который управляет задачами и выполняет их по очереди, но при этом не блокирует выполнение.

*GIL (Global Interpreter Lock)* - это механизм синхронизации в интерпретаторе Python, который ограничивает выполнение байт-кода Python одновременно в одном процессе. Это значит, что в один момент времени только один поток может исполнять Python-код, даже если у тебя есть несколько потоков

# Методы и подходы к работе:

1. **Реализация задач с использованием Asyncio:**
   1. ***Ввод/вывод (I/O)****:* запросы к базе данных, взаимодействие с веб-серверами или файловыми системами
   2. ***Множество параллельных операций****:* когда нужно выполнить множество независимых задач, которые в основном ждут ввода/вывода (например, работа с API).
2. **Реализация задачи с использованием Threading:**
   1. ***Задачи ввода/вывода (I/O):*** Чтение файлов, работа с сетью, запросы к API. Также задачи, требующие больших вычислительных ресурсов.
3. **Реализация задачи с использованием Multiprocessing:**
   1. ***Задачи с разделяемым состоянием:*** задачи, где при использовании одного набора данных, должны работать несколько потоков.

# Обзор и анализ подходов к асинхронному программированию:

## Asyncio:

**Архитектура:**

Данный подход использует модель событийного цикла, при которой задачи выполняются поочередно в рамках одного потока. Это означает, что в любой момент времени активна только одна задача, но она может быстро переключаться на другие задачи, ожидающие завершения операций ввода-вывода.

**Плюсы использования:**

Высокая эффективность ввода-вывода:этот подход позволяет эффективно управлять задачами, которые связанны с сетевым взаимодействием, вводом-выводом и работе с файлами, без блокировки основного потока.

Простота кода:синтаксис позволяет писать асинхронный код, который легко читается и воспринимается как синхронный, что значительно упрощает восприятие кода и дальнейшую разработку.

**Минусы использования:**

Не эффективность в вычислительных задачах: Asyncio не предназначен для задач, связанных с интенсивными вычислениями, т.к. выполнение задач данного типа будет блокировать событийный цикл.

Необходимость использования асинхронных версий библиотек: чтобы получить максимальную отдачу от использования данного подхода, необходимо наличие асинхронных версий библиотек, что может в последствии ограничить выбор инструментов и усложнить интеграцию.

## Threading:

**Архитектура:**

Данный подход предназначен для выполнения нескольких задач одновременно в одном процессе. Потоки делят общую память и ресурсы, что делает их удобными для задач, где требуется одновременная обработка.

**Плюсы использования:**

Совместное использование памяти: все потоки в рамках одного процесса имеют доступ к общим переменным и объектам.

Одновременное выполнение нескольких задач: например, один поток загружает данные или ведет вычисления, когда другой поток уже может начать обрабатывать полученные данные.

Легкость: потоки создаются быстрее, чем процессы

**Минусы использования:**

Сложности синхронизации и состояние гонки: потоки делят общую память, что может приводить к состоянию гонки или другим ошибкам, связанным с многозадачностью, если доступ к ресурсам не синхронизирован. Для предотвращения таких проблем приходится использовать механизмы блокировок, что усложняет код и может снизить производительность из-за блокирования потоков.

Ограничение GIL (Global Interpreter Lock): из-за GIL в Python потоки не могут выполнять код одновременно. Это делает threading менее эффективным для задач с интенсивными вычислениями, так как потоки вынуждены ждать своей очереди, вместо того чтобы параллельно использовать несколько ядер процессора.

## Multiprocessing:

**Архитектура:**

Данный подход позволяет запускать несколько процессов параллельно, каждый с собственной памятью и интерпретатором Python, что в свою очередь даёт возможность запускать несколько процессов одновременно и они будут независимы от других.

**Плюсы использования:**

Параллельность:каждый процесс работает независимо, что позволяет использовать все ядра процессора.

Изолированность памяти: каждый процесс имеет собственную область памяти, что исключает проблемы с многозадачностью, характерные для потоков (гонка данных).

**Минусы использования:**

Высокие требования к ресурсам: при использовании нескольких процессов одновременно, может возникнуть проблема с ростом нагрузки на систему.

Сложное меж процессное взаимодействие: т.к. каждый процесс имеет собственную память, то передача данных может быть затруднительной. Для оптимизации этого, необходимо усложнять код путем добавления очередей или каналов.

# Проектирование эксперимента

## Постановка задачи:

**Цель эксперимента –** Провести сравнение трех подходов к реализации асинхронного программирования (asyncio, threading, multiprocessing) для задач разного типа. Определить их производительность и уместность.

**Задача с вводом-выводом –** Асинхронная загрузка данных, получаема с внешнего ресурса (веб-сервера), при которой имеется значительное время ожидание отклика.

**Задача с высокой параллельностью –** Асинхронная обработка исторических данных с последующей укомплектацией и выводом.

**Задача с потоковой загрузкой файлов –** Одновременная выгрузка n-количества файлов, для последующей обработки.

**Задача формата CPU-bound –** Сложная требующая больших вычислений, задача.

## Тема выше озвученных задач

Для реализации озвученных задач, мною была выбрана сфера финансового рынка и её компонентов. В связи с личным интересом, а также мотивацией реализовать полноценное приложение и последующего развития его в будущем.

**Задача 1** – Получение соотношение валютных пар USD (доллар) к другим валютам, популярным на рынке.

**Задача 2** – Выгрузка и укомплектация исторических данных, по таким параметрам как Дата открытия торгов, начальная цена, конечная цена, а также при наличии личных вложений, количество купленных акций.

**Задача 3** – Скачивание исторических данных, популярных акций, для последующей реализации в зависимости от запроса.

**Задача 4–** Вычислениепростых чисел в указанном диапазоне (1, 5000000).

# Определение критериев сравнения

**Время выполнения:**

Время выполнения - показатель, который даст представление насколько быстро каждый подход в рамках поставленной задачи может справляться.

**Методы измерения:**

1. Установить счетчик от начала выполнения до окончания. Провести вычисления, получить итог.
2. Сравнение времени выполнения между различными подходами.

**Использование ресурсов процессора:**

Определяет, как эффективно каждый метод задействует вычислительные мощности процессора. Это важно для оценки распределения нагрузки и общей производительности.

**Методы измерения:**

1. Отслеживание загрузки процессора во время выполнения задач.
2. Применение профилирующих инструментов для анализа работы процессора.

**Использование памяти:**

Показывает объем оперативной памяти, необходимой для выполнения задач. Этот параметр критически важен для понимания масштабируемости и эффективного использования доступных ресурсов.

**Методы измерения:**

1. Измерение потребления памяти до и после выполнения задач.
2. Использование специализированных инструментов для мониторинга памяти.

# Программная реализация:

## Сценарии кода для каждого подхода:

**Задача 1.**

Приложение 3. Asyncio, Multiprocessing, Threading.

**Задача 2.**

Приложение 4. Asyncio, Multiprocessing, Threading.

**Задача 3.**

Приложение 5. Asyncio, Multiprocessing, Threading.

**Задача 4.**

Приложение 6. Asyncio, Multiprocessing, Threading

**Сбор и Анализ данных**

Сбор полученных данных для их последующего анализа.

# Анализ и интерпретация результатов.

## Сравнительный анализ производительности:

**Задача 1 –** Вывод соотношения USD к другим валютам, популярным на рынке. В целях успешной реализации задачи, было снижено количество запросов до 4 по каждой программе, в связи с ограничением ключа API к хосту по количеству запросов.

**Результаты и обсуждение:**

Asyncio – 0,72 секунд

Threading – 0,74 секунд

Multiprocessing – 12,53 секунд

**Asyncio:**

1. Отлично подходит для многозадачного HTTP-запроса, особенно если нужно быстро получить данные с разных серверов.
2. Эффективен, так как работает асинхронно и экономит ресурсы.

**Threading:**

1. Легко реализуется, но из-за GIL не дал прироста производительности при большом количестве запросов.

**Multiprocessing:**

1. Хорош для случаев, когда нужно параллельно делать много запросов, но требует больше памяти и процессорных ресурсов.

**Итог:** для парсинга валют с API, где операции — это в основном I/O (сетевые запросы), asyncio будет наиболее эффективным, однако если сервис возвращает данные медленно, threading или multiprocessing тоже подойдут, но asyncio будет предпочтительнее.

**Задача 2 –** Обработка полученных данных и упаковка их в читаемый вид таблицы, с четко разделенными столбцами и значениями.

**Результаты и обсуждение:**

Asyncio – 13,90 секунд

Threading – 12,10 секунд

Multiprocessing – 2,57 секунд

**Asyncio:**

1. Asyncio отлично подходит, если данные нужно получать из сети, так как HTTP-запросы не блокируют цикл событий.

**Threading:**

1. Данный подход также подойдёт, если нужно комбинировать вычисления и I/O, но может быть ограничен из-за GIL.

**Multiprocessing:**

1. Т.к. данный подход лучше подходит для многозадачных и вычислительно интенсивных задач, показал самый лучший результат, ведь для реализации использует несколько процессов и может использовать все ядра процессора.

**Итог:** С текущей задачей лучше всего справился подход с использованием Multiprocessing, благодаря возможности работы с несколькими процессами и большей вычислительной мощностью, хоть и энергозатратной.

**Задача 3 –** Скачивание с внешнего хостинга исторических данных по акциям компаний, с целью выявить самый подходящий способ реализации этой задачи.

**Результаты и обсуждение:**

Asyncio – 0,69 секунд

Threading – 0,67 секунд

Multiprocessing – 1,54 секунд

**Asyncio:**

1. Показал себя чуть медленнее, потому что чтение и запись в файл, в asyncio требуют синхронных операций.

**Threading:**

1. Показал самый лучший результат, так как он хорошо справляется с I/O-задержками.

**Multiprocessing:**

1. Оказался самым медленным, так как на создание процессов уходит дополнительное время.

**Итог:** в сравнении с другими методами, в данном примере лучше всех себя показал Threading, т.к. он хорошо справляется с I/O-задержками.

**Задача 4** – CPU-bound задача. Вычисление простых чисел в диапазоне от 1 до 5000000.

**Результаты и обсуждение:**

Asyncio – 45,17 секунд

Threading – 42,20 секунд

Multiprocessing – 37,29 секунд

**Asyncio:**

1. Показывает себя хуже, т.к. для CPU-bound задач менее эффективен, из-за работы в одном потоке.

**Threading:**

1. Ограничен из-за GIL: реальная параллельная работа потоков невозможна, что делает его неэффективным для интенсивных вычислений.

**Multiprocessing:**

1. Лидер для CPU-bound задач. Каждый процесс работает независимо, используя своё ядро.
2. Устраняет проблему GIL и масштабируется на многопроцессорных системах.

**Итог:** для CPU-bound задач, подобных поиску простых чисел, multiprocessing является оптимальным выбором. Однако для I/O-bound задач или при необходимости асинхронности лучше рассматривать asyncio или threading.

# Уместность использования различных методов и рекомендация.

**Asyncio:**

Данный метод реализации асинхронного программирования отлично подойдет для задач, где важна работа с сетью, базами данных, файловыми операциями. Однако является не лучшим выбором для вычислительно интенсивных задач

**Threading:**

Данный метод реализации асинхронного программирования удобен для задач с лёгкой многозадачностью, но менее эффективен для вычислительных задач из-за ограничений GIL. Прост в реализации, но требует аккуратной работы с синхронизацией данных.

**Multiprocessing:**

Данный метод реализации асинхронного программирования эффективен для вычислительно сложных задач, где требуется многопроцессорная обработка. Однако не очень подходит для задач с вводом-выводом из-за накладных расходов на создание процессов и обмен данными.

# Заключение

На примере проделанной работы, мы можем понять, что каждый метод реализации асинхронного программирования уникален и имеет свои сильные и слабые стороны, что позволяет сохранять маневренность при использовании в задачах, а также реализации комбинирования этих трех методов, для формирования более функционального и сложного ПО.

Использование данных методов в реализации взаимодействия с фондовым рынком, позволит улучшать взаимодействие обычных пользователей, ускорять работу и возможно в будущем станет маленькими кирпичиком в улучшении финансовой грамотности в стране и мире!

# Приложение 1. Пример файловой структуры проекта.

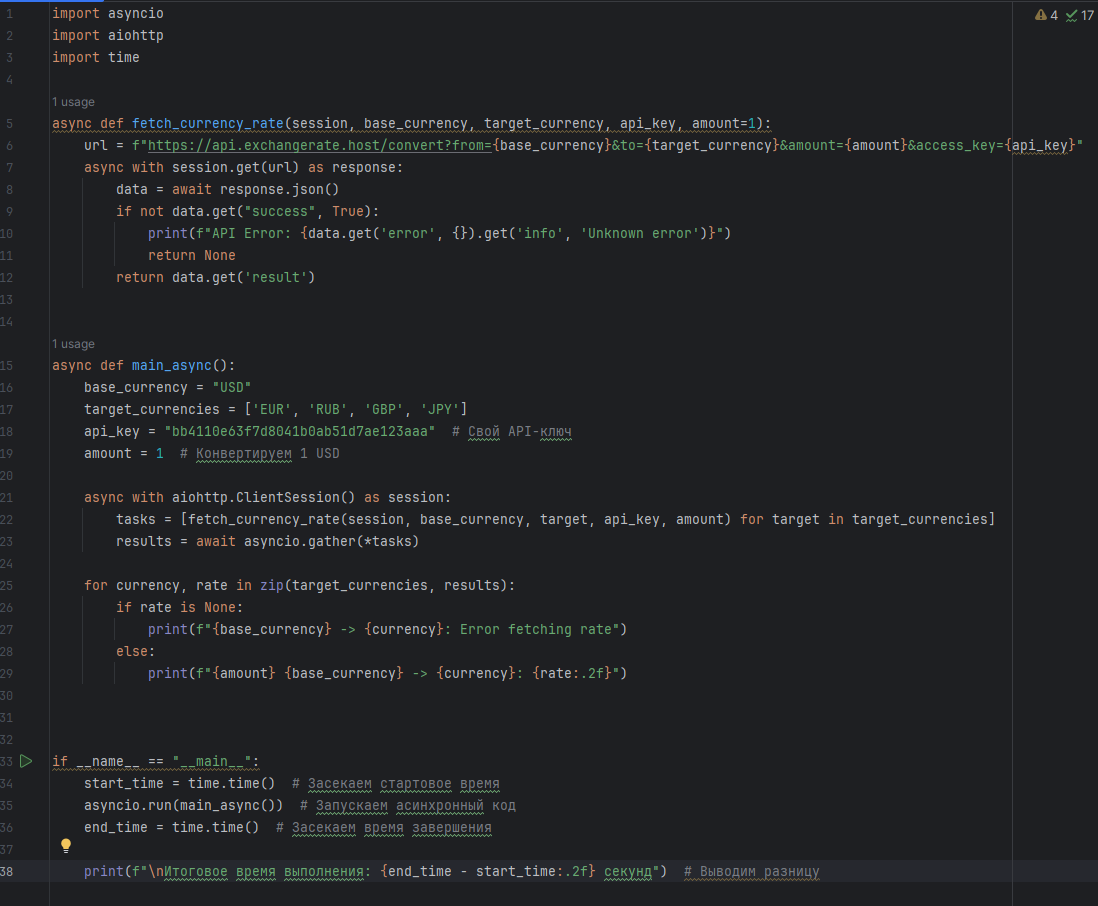
pythonProject  
├── .idea  
├── .venv  
├── Task\_1  
│ ├── Async.py  
│ ├── Multiproces.py  
│ └── Thread.py  
├── Task\_2  
│ ├── Async.py  
│ ├── Multiproces.py  
│ └── Thread.py  
├── Task\_3  
│ ├── Async.py  
│ ├── Multiproces.py  
│ └── Thread.py

├── Task\_4  
│ ├── Async.py  
│ ├── Multiproces.py  
│ └── Thread.py  
├── README.md  
├── folders\_structure.txt  
└── requirements.txt

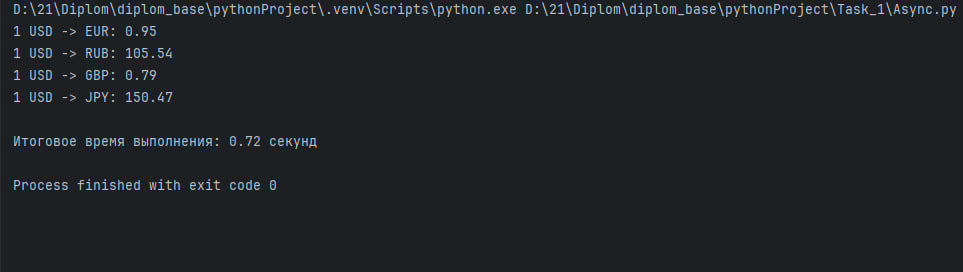
Приложение 2. Список необходимых библиотек.

asyncio==3.4.3  
aiohttp==3.11.9  
requests==2.32.3  
aiofiles==24.1.0  
yfinance==0.2.50  
beautifulsoup4==4.12.3  
aiohttp\_request==0.0.2  
aiosignal==1.3.1  
numpy==2.1.3  
pandas==2.2.3  
pillow==11.0.0

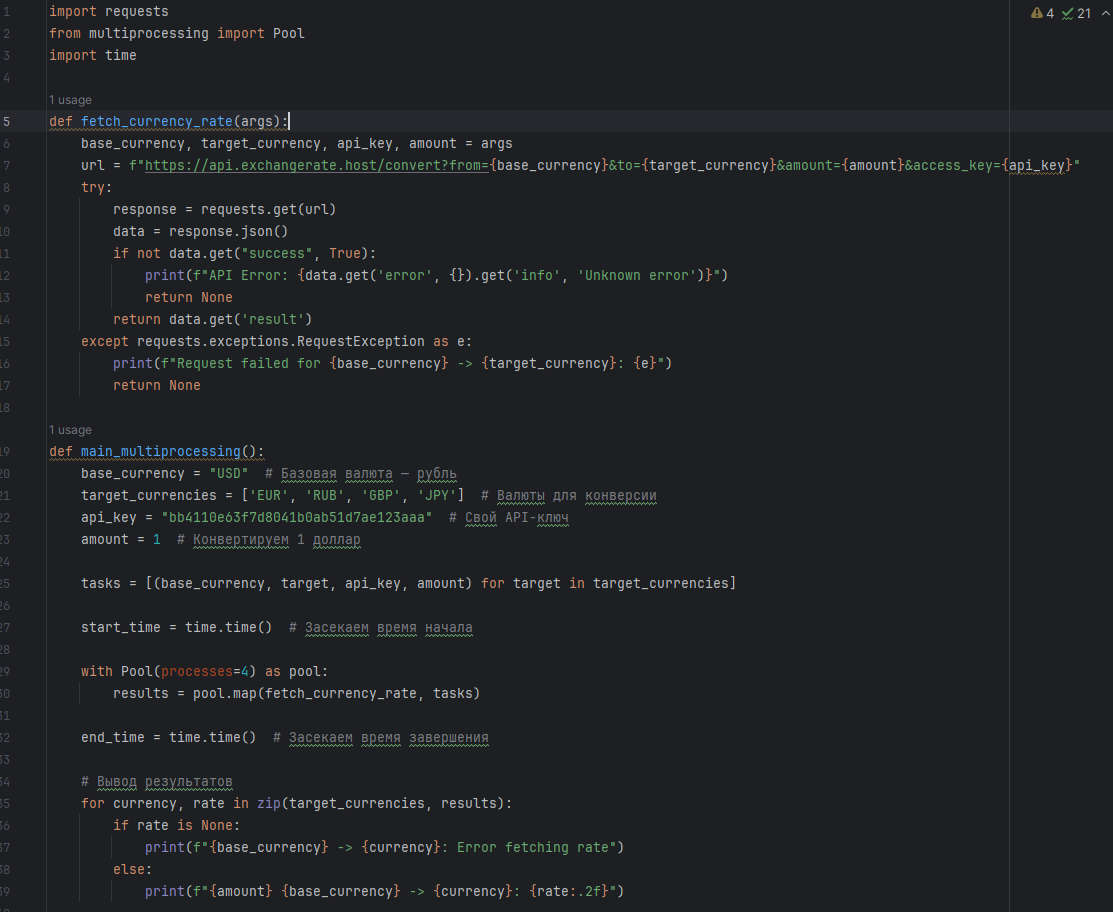
# Приложение 3.

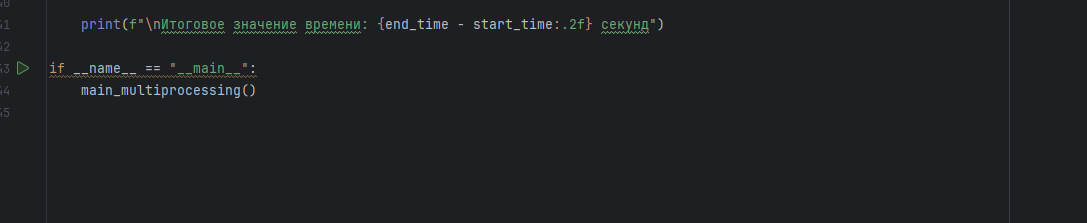


*Задача 1. Реализация с Asyncio.*

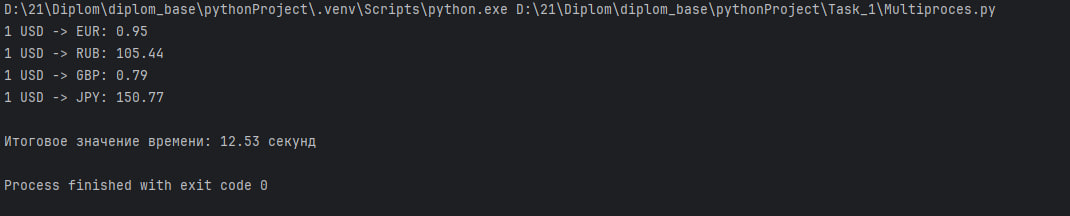


*Задача 1. Результат отработки кода с Asyncio.*

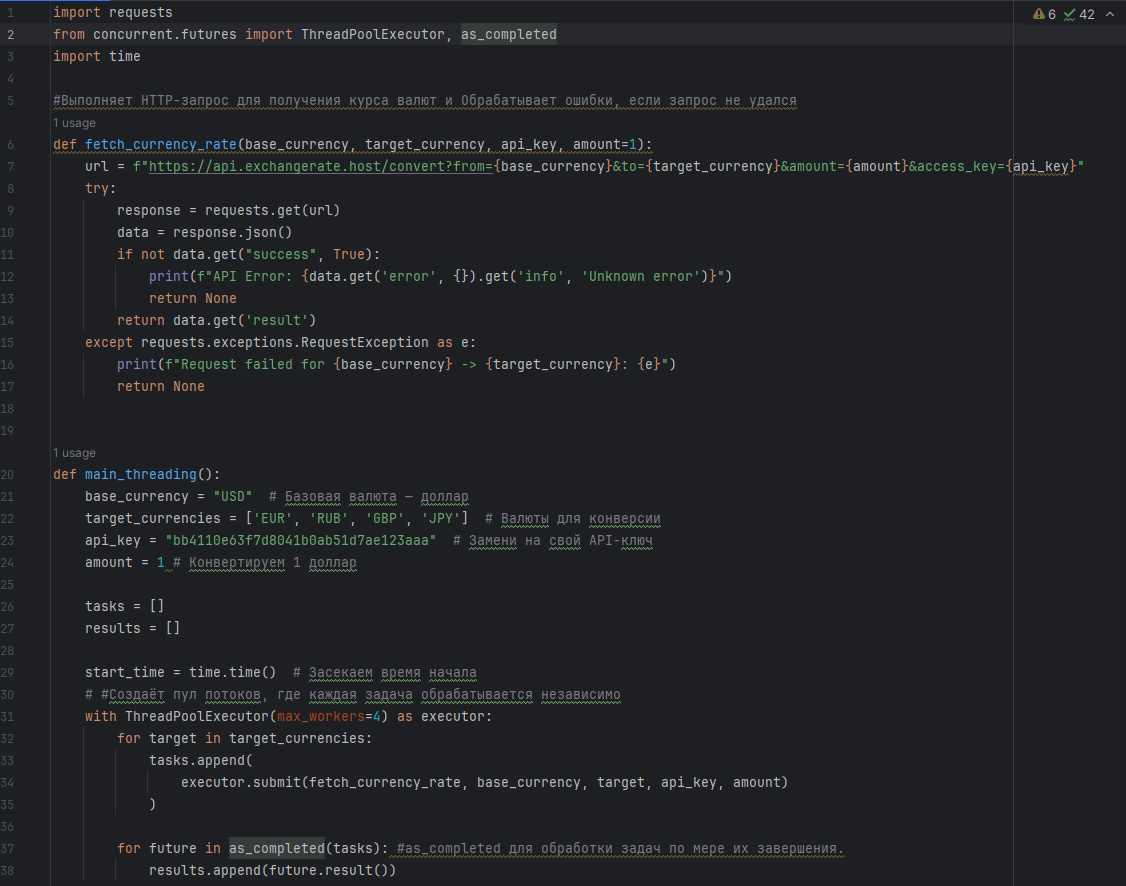
**

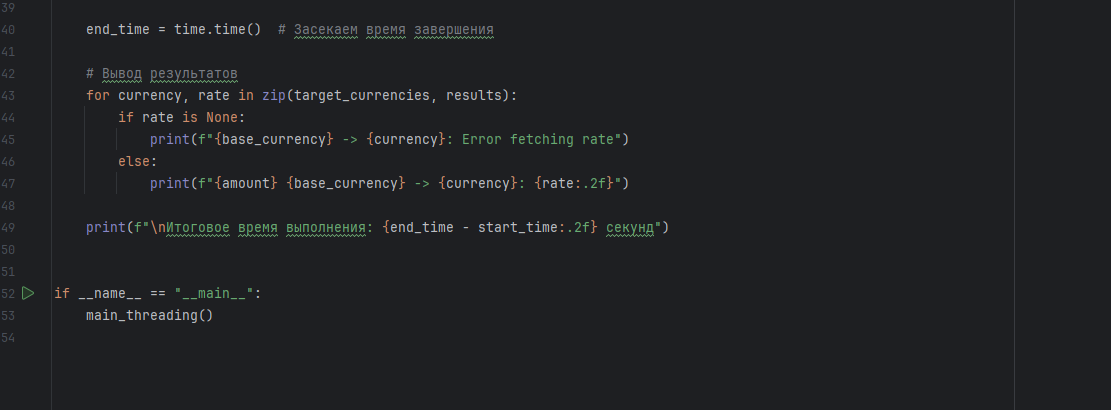
**

*Задача 1. Реализация с Multiprocessing.*

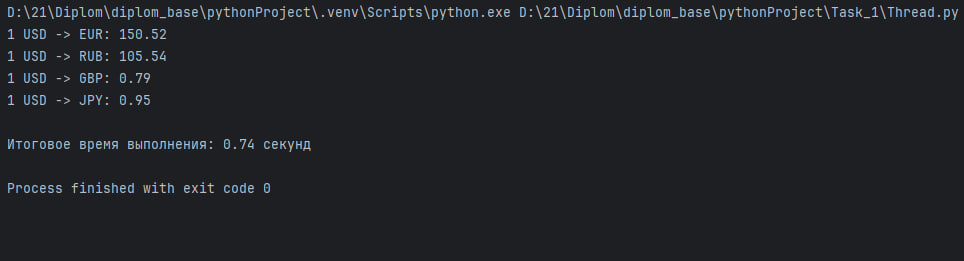


*Задача 1. Результат отработки кода с Multiprocessing.*

**

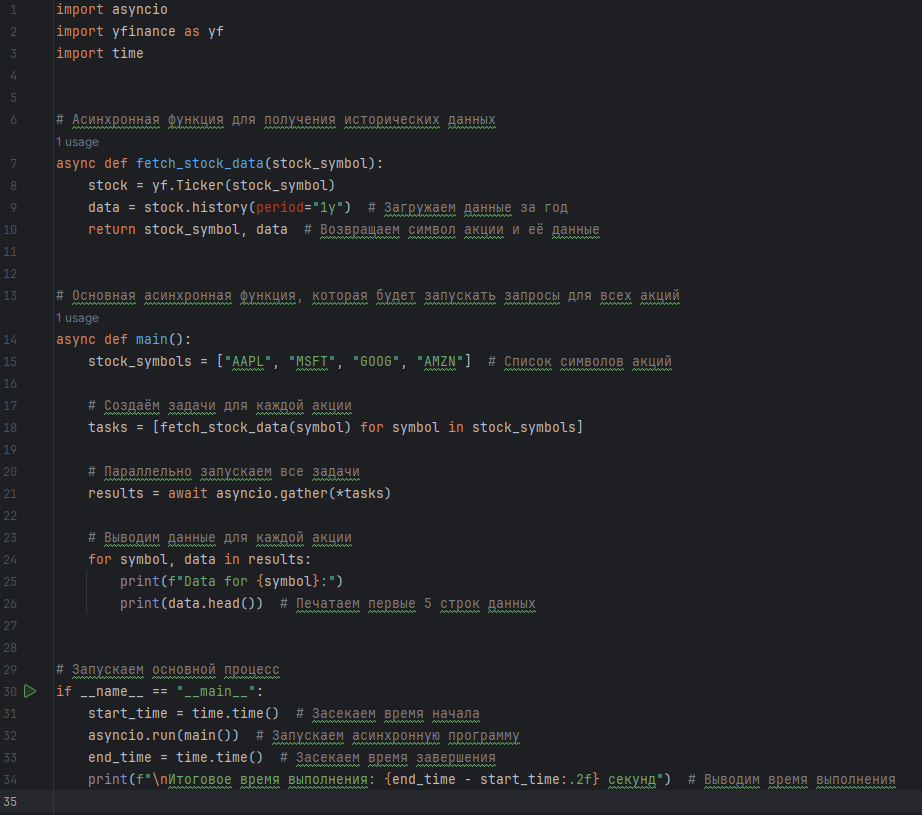
**

*Задача 1. Реализация с Threading.*

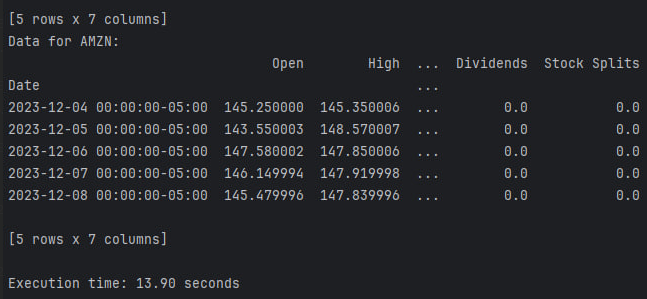


*Задача 1. Результат отработки кода с Threading.*

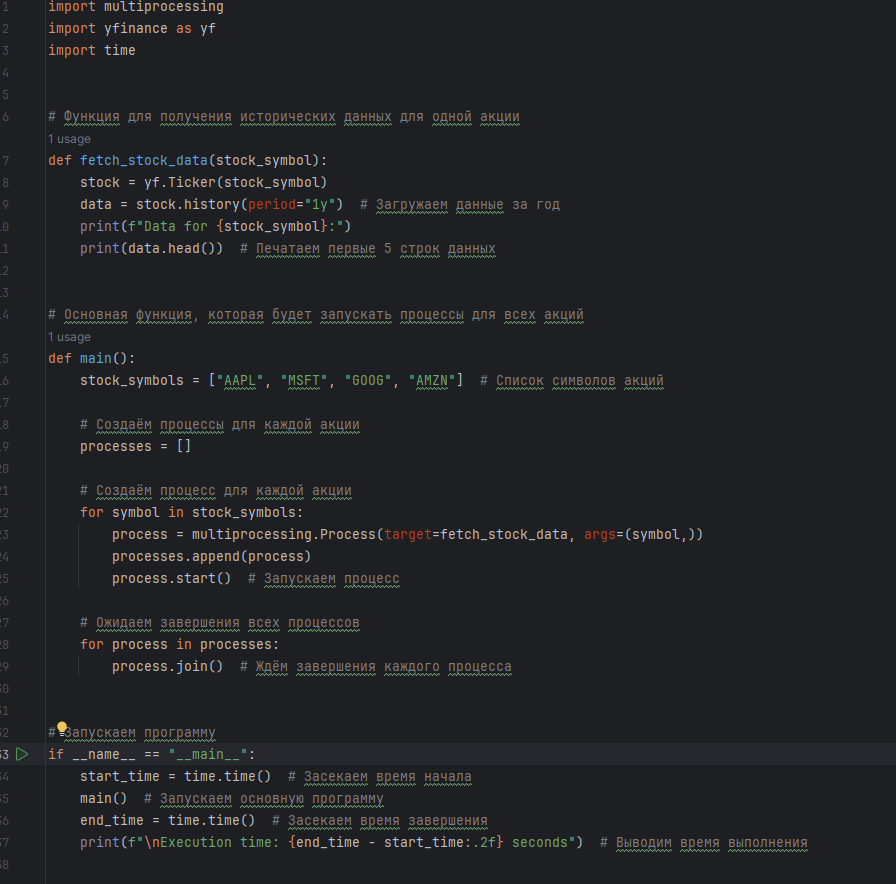
# Приложение 4.



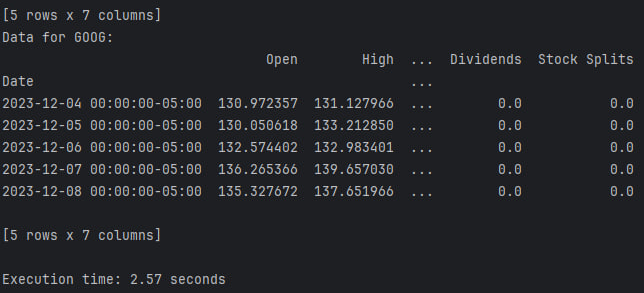
*Задача 2. Реализация с Asyncio.*



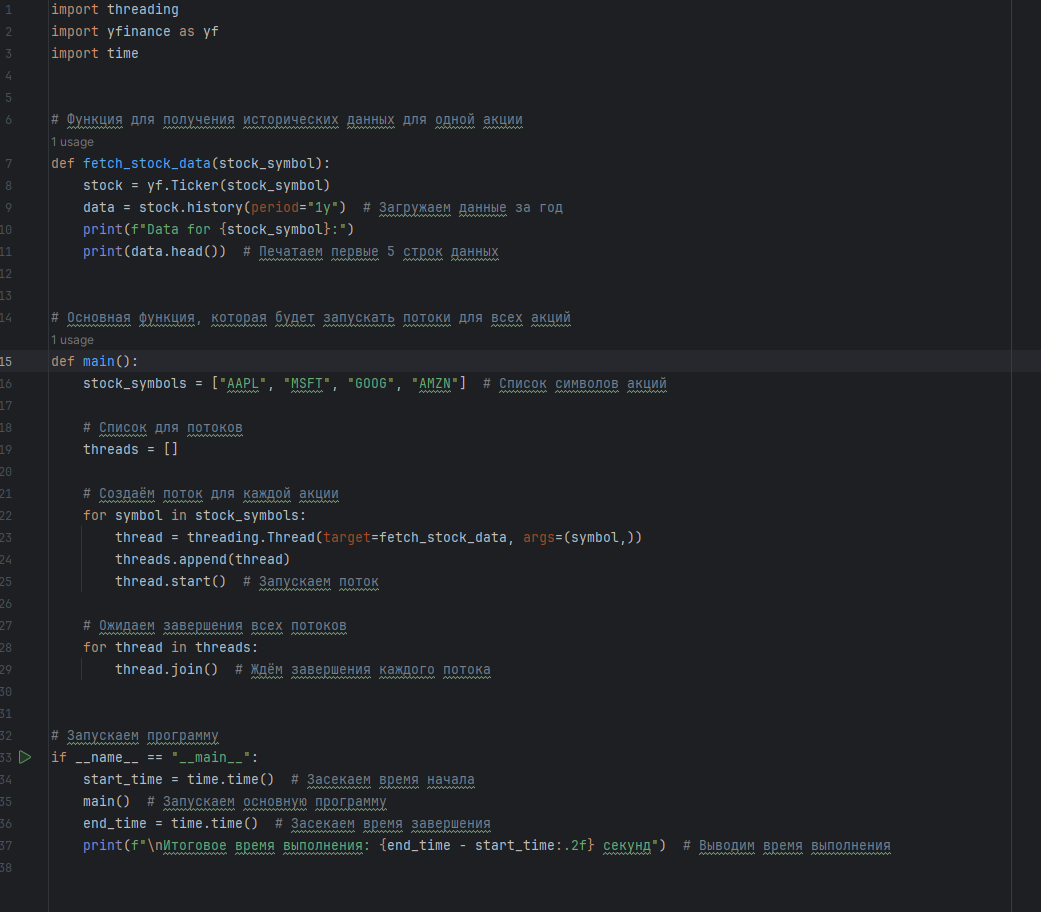
*Задача 2. Результат отработки кода с Asyncio.*

**

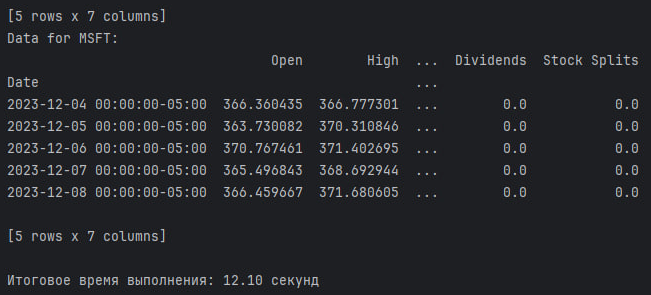
*Задача 2. Реализация с Multiprocessing*

**

*Задача 2. Результат отработки кода с Multiprocessing.*

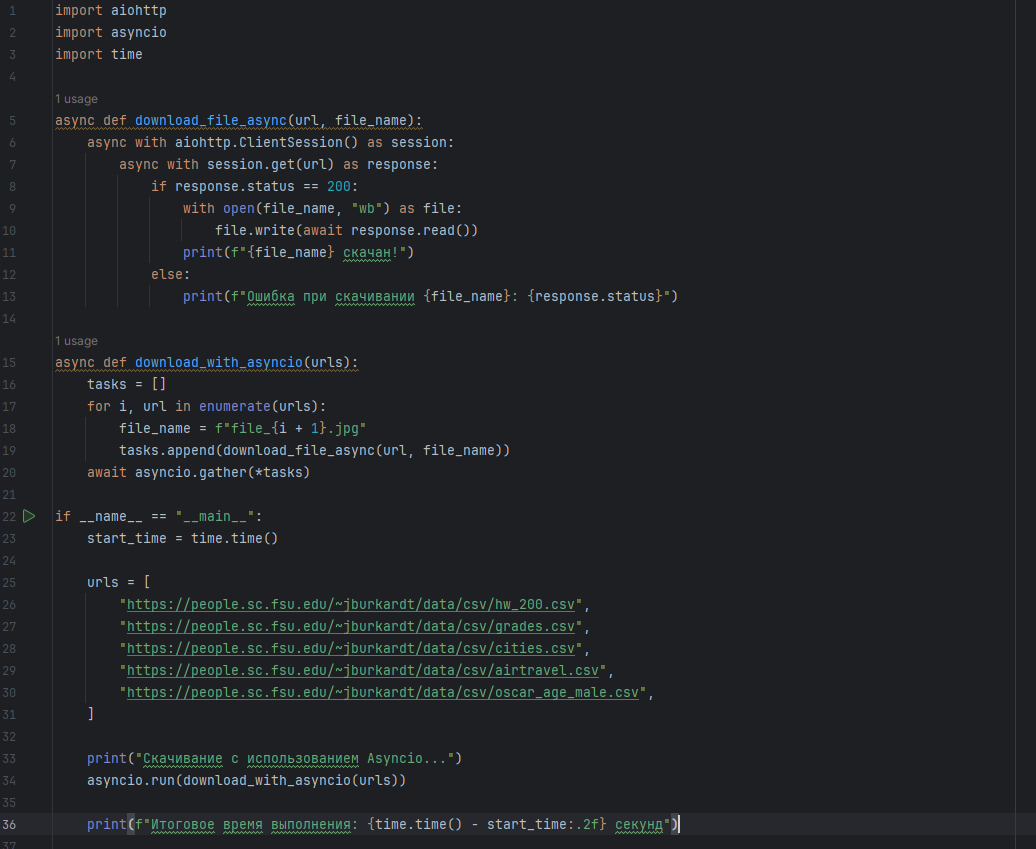
**

*Задача 2. Реализация с Threading.*

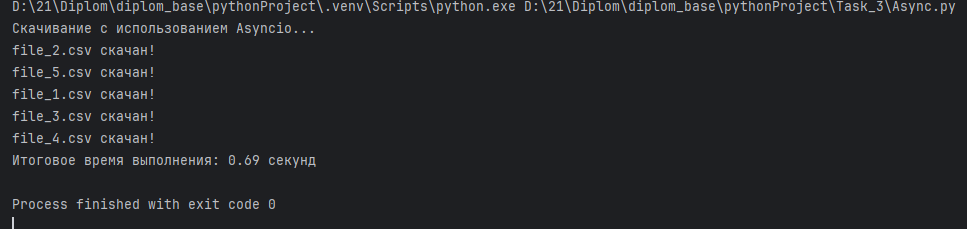
**

*Задача 2. Результат отработки кода с Threading.*

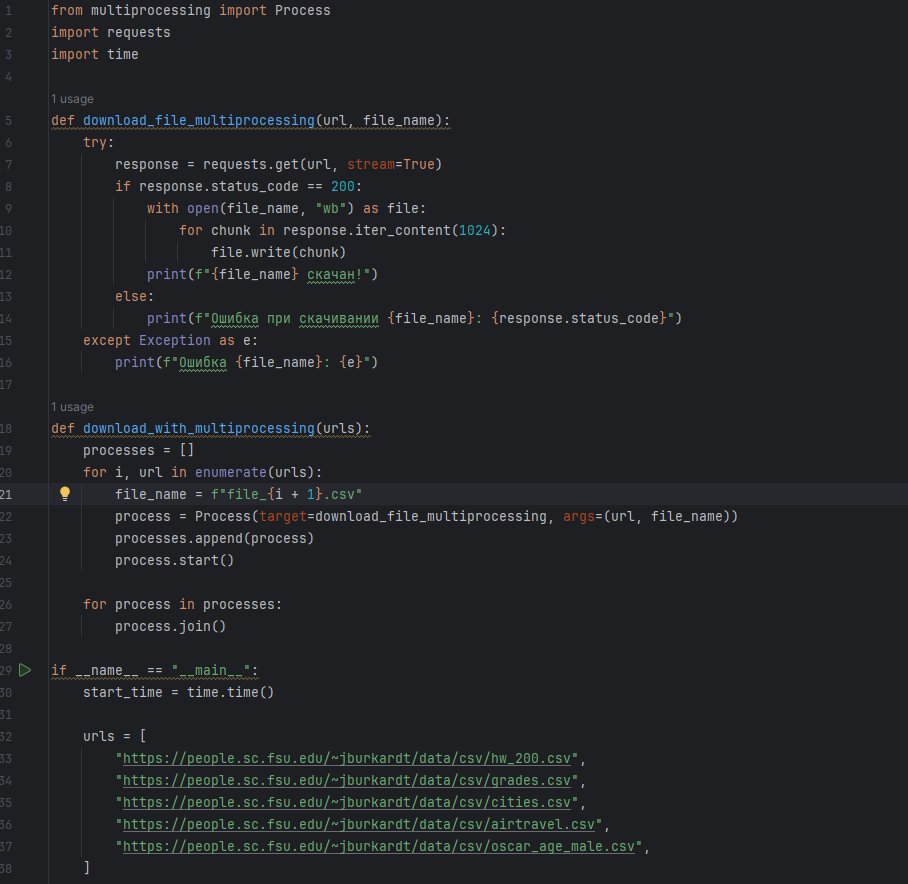
# Приложение 5.

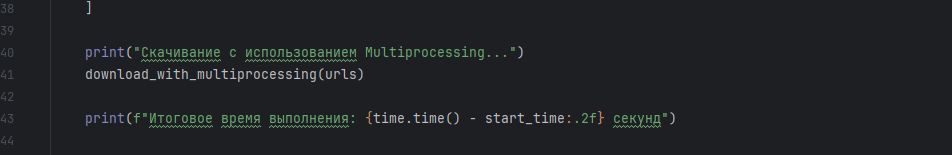
**

*Задача 3. Реализация с Asyncio.*

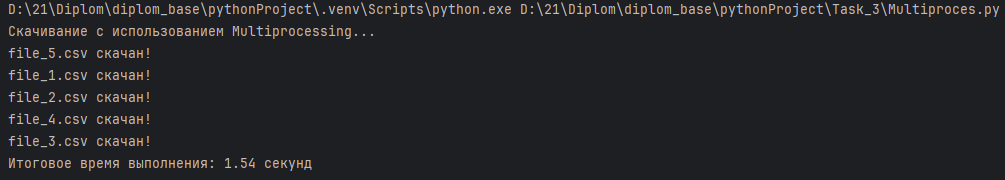
**

*Задача 3. Результат отработки кода с Asyncio.*

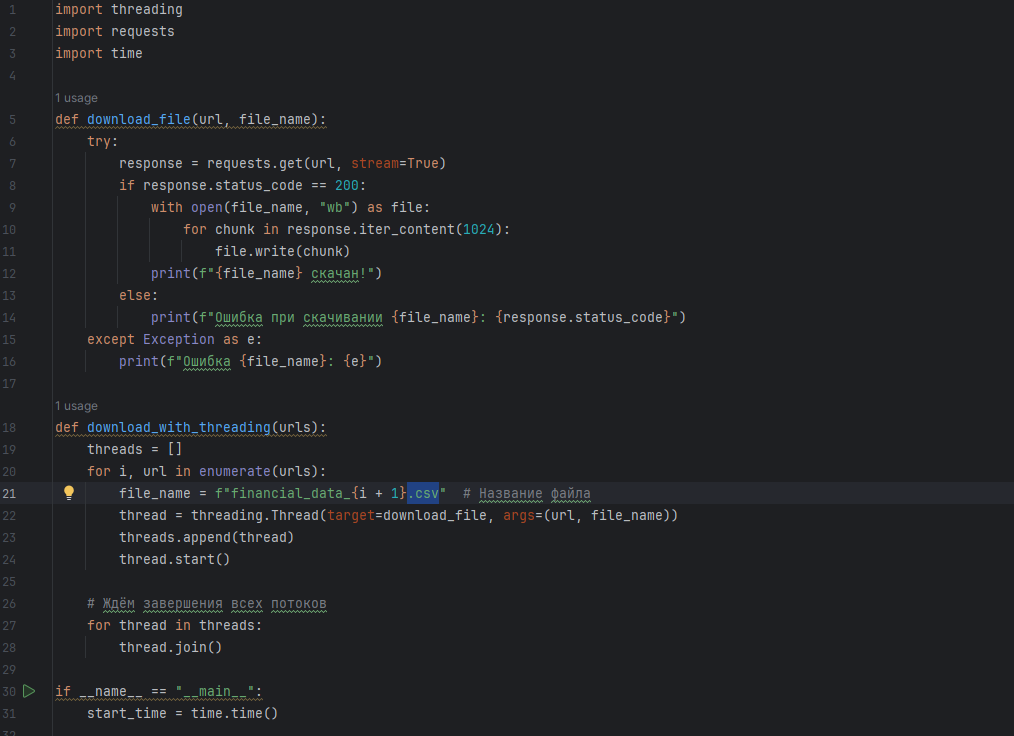
**

**

*Задача 3. Реализация с Multiprocessing*

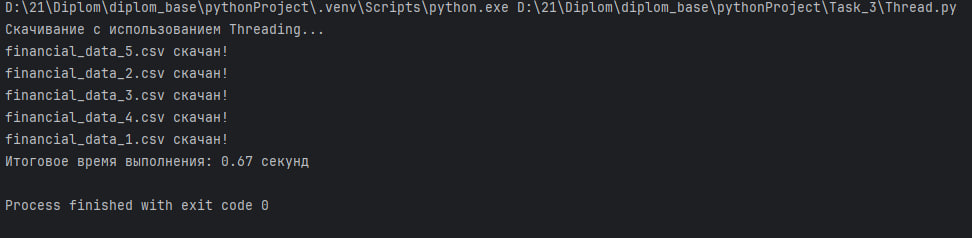
**

*Задача 3. Результат отработки кода с Multiprocessing.*

**

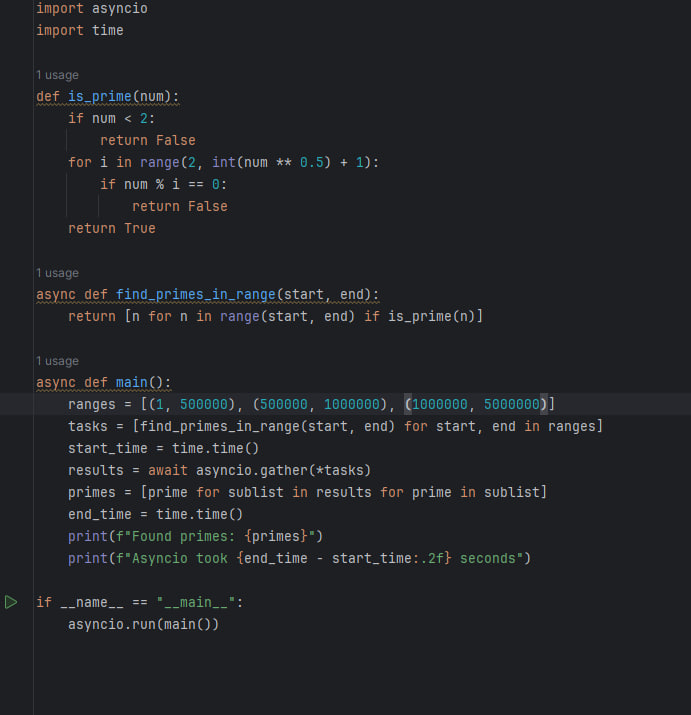
**

*Задача 3. Реализация с Threading.*

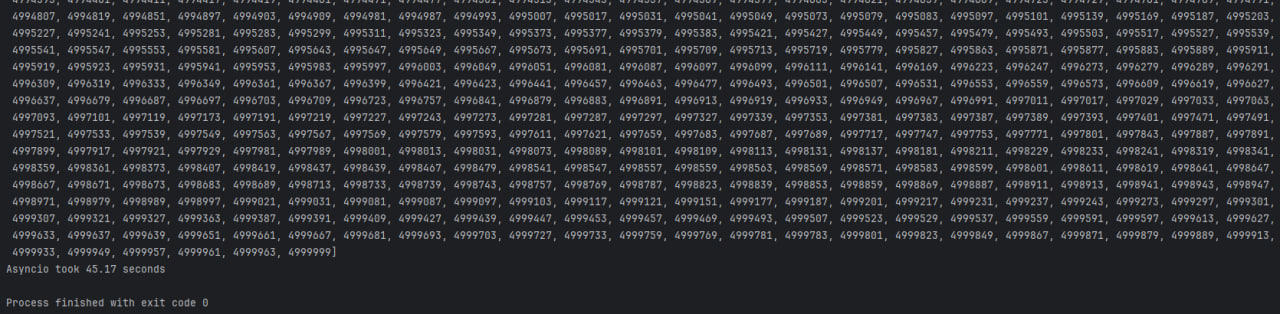


*Задача 3. Результат отработки кода с Threading.*

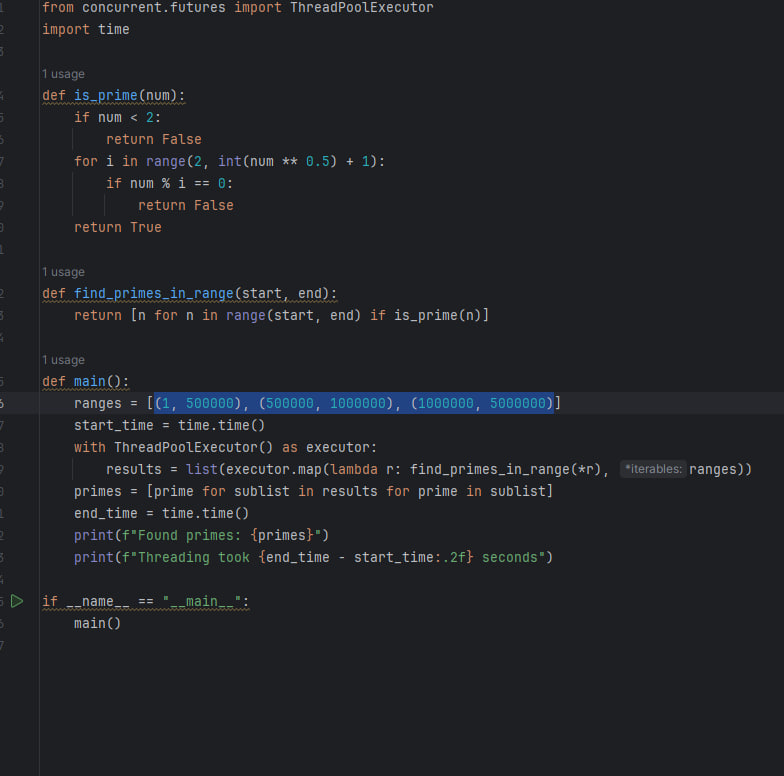
# Приложение 6.



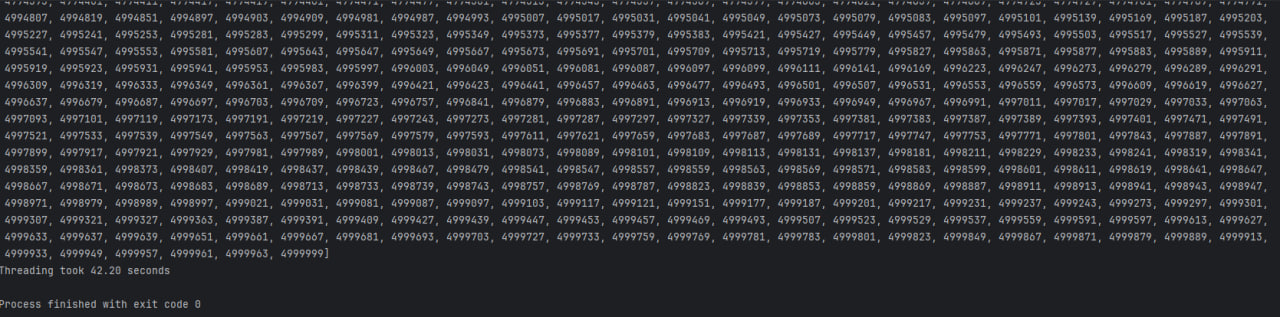
*Задача 4. Реализация с Asyncio.*



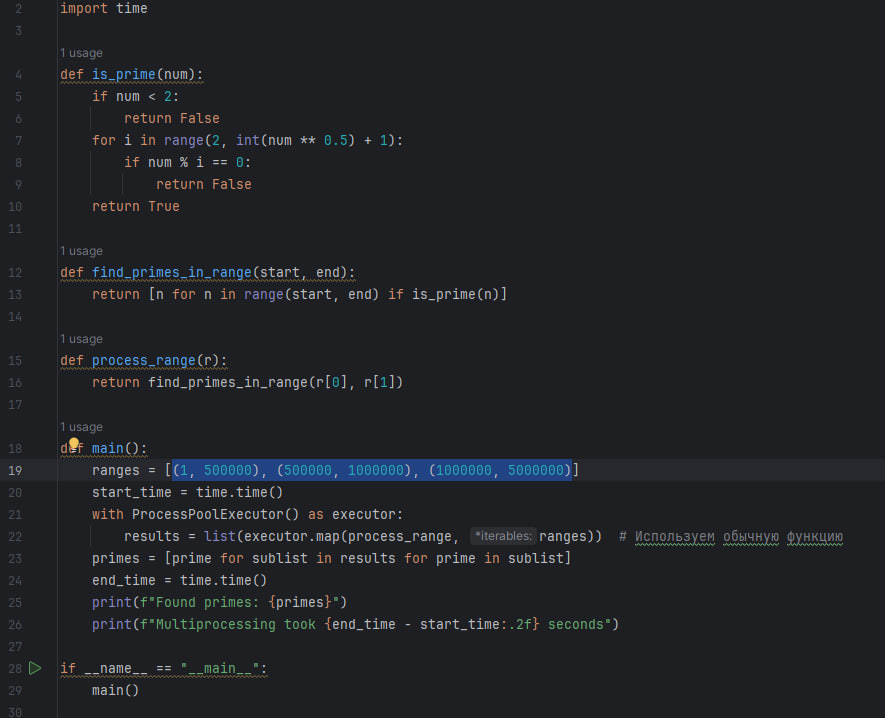
*Задача 4. Результат отработки кода с Asyncio.*



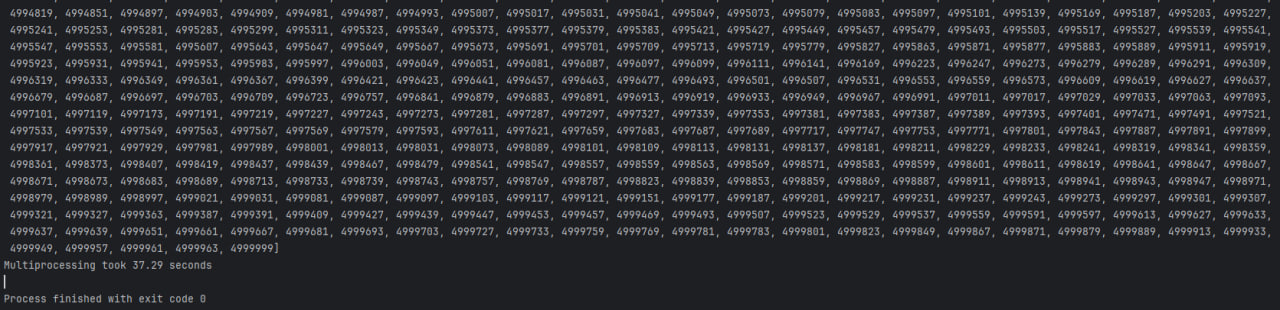
*Задача 4. Реализация с Threading.*



*Задача 4. Результат отработки кода с Threading.*



*Задача 4. Реализация с Multiprocessing*



*Задача 4. Результат отработки кода с Multiprocessing.*