Prime Number Generator
Project Documentation

Midaas Telesoft Private Limited

Created By: Priya Debrani

Table of Contents

1.Introduction

This project consists of two main parts: a Prime Number Generator and a Prime Number Generator Server.

The Prime Number Generator is a Python script that efficiently generates all prime numbers in a range provided by the user. It offers different prime number generation strategies with various performance optimizations. The generator is usable via the command-line, allowing the user to select the generation strategy. It includes unit tests to validate the correctness of the program.

The Prime Number Generator Server is a Fast API application served by a Uvicorn server. It uses the Prime Number Generator as its engine and provides users with the ability to use the generator over a REST API over HTTP. Each execution is recorded in a database, including timestamp, range, time elapsed, algorithm chosen, and the number of primes returned. For simplicity of implementation, an in-memory database is used.

The code is documented clearly and concisely, with the assumption that it will be maintained in the future by people who won't be able to contact the original author for explanations. The implementation is organized to make the intentions clear and, if necessary, comments are added to make it explicit.

Please note that this assignment is about server code, which should run for a potentially long time. This has been considered when thinking about errors (i.e., exceptions) and resource (i.e., database connections) management.

This documentation will provide a detailed overview of both parts of the project, including code structure, key functions/classes, API endpoints, and examples of how to use them.

2. Python Code Documentation

Overview

The Python script is designed to generate prime numbers within a given range. It offers four different strategies for generating these prime numbers: the brute force method, the trial division method, the Miller-Rabin method, and the Sieve of Eratosthenes method.

Code Structure

The script is structured into four main functions, each implementing a different prime number generation strategy:

1. **brute_force_method(start, end):** This function uses the brute force method to find prime numbers in a given range. It checks each number one by one to see if it has any factors other than 1 and itself. If a number has no other factors, it's considered prime. This method is simple but slow, especially for large ranges, because it checks every single number.
2. **trial_division_method(start, end):** This function uses the trial division method to find prime numbers in a given range. It improves on the brute force method by skipping even numbers and multiples of 3. It checks each remaining number to see if it has any factors other than 1 and itself. If a number has no other factors, it's considered prime. This method is faster than the brute force method, but still relatively slow for large ranges.

3. **miller_rabin_method(start, end, k=5):** This function uses the Miller-Rabin method to find prime numbers in a given range. This is a probabilistic test that's much faster for large numbers. It picks a random number 'a' and checks if it satisfies certain conditions. If 'a'

does, then the number is probably prime. If ‘a’ doesn’t, then the number is composite. The more random numbers ‘a’ we check, the more confident we can be in the result. However, this method might occasionally guess wrong.

4. **sieve_of_eratosthenes_method(start, end):** This function uses the Sieve of Eratosthenes method to find prime numbers in a given range. It starts by assuming all numbers are prime, then progressively marks the multiples of each number as composite (not prime). At the end, the numbers that are still marked as prime are the prime numbers. This method is very fast and always correct, but it needs to make a list of all the numbers up front, which can use a lot of memory for large ranges.

The **main()** function is the entry point of the script. It checks the command-line arguments, validates them, and calls the appropriate prime number generation function based on the method name provided. It also records and prints the time taken by the prime number generation.

Key Functions/Classes

1. **brute_force_method(start, end):** This function generates prime numbers in the given range using the brute force method. It returns a list of prime numbers.

2. **trial_division_method(start, end):** This function generates prime numbers in the given range using the trial division method. It returns a list of prime numbers.
3. **miller_rabin_method(start, end, k=5):** This function generates prime numbers in the given range using the Miller-Rabin method. It returns a list of prime numbers.
4. **sieve_of_eratosthenes_method(start, end):** This function generates prime numbers in the given range using the Sieve of Eratosthenes method. It returns a list of prime numbers.
5. **main():** This is the main function that runs when the script is executed. It checks the command-line arguments, validates them, calls the appropriate prime number generation function, and prints the prime numbers and the time taken.

Unit Tests

The script includes a set of unit tests to validate the correctness of the prime number generation functions. The tests are written using the unittest module, which is a built-in Python module for writing and running tests. The tests check that each function returns the correct prime numbers for different ranges. The tests can be run by executing the test script directly.

Examples

Here is an example of how to use the script:

```
python prime_no_generator.py brute_force 1 10
```

This command will generate all prime numbers between 1 and 10 using the brute force method. The prime numbers and the time taken will be printed to the console.

Conclusion

This Python script provides a robust and efficient way to generate prime numbers within a given range. It offers four different strategies for generating prime numbers, allowing users to choose the one that best fits their needs. The script includes comprehensive documentation and unit tests, ensuring that it is easy to understand and maintain.

3. FastAPI Application and Uvicorn Server Documentation

Overview

The FastAPI application serves as a server for the Prime Number Generator. It provides a REST API over HTTP, allowing users to generate

prime numbers within a given range using different generation strategies. The application records each execution in a database, including timestamp, range, time elapsed, algorithm chosen, and the number of primes returned.

Code Structure

The FastAPI application is structured into two main parts: the FastAPI application itself and the routers.

1. **FastAPI Application:** The FastAPI application (app) is created and configured in the main module. It includes routers from the primes and executions modules.
2. **Routers:** The primes and executions routers define the routes for the prime number generation and execution details respectively. The primes router includes a route for generating prime numbers (/primes) and a homepage route (/) that provides a form for prime number generation. The executions router includes a route (/executions) that returns a table of all executions.

Key Functions/Classes

1. **FastAPI Application (app):** This is the main FastAPI application that includes the routers and handles the requests.
2. **Routers (primes and executions):** These routers define the routes for the prime number generation and execution details.

3. **Data Models (PrimeRequest and PrimeResponse):** These Pydantic models validate the incoming request data and ensure that the response data conforms to the specified structure.
4. **Database Functions (get_db):** This function generates database sessions.
5. **Endpoint Functions (generate_primes and home):** These functions handle the requests to the /primes and / endpoints respectively.

Uvicorn Server

The Uvicorn server is an ASGI server implementation that serves the FastAPI application. It runs the application on a specified host and port, making it accessible over the network. The server is run directly from a Python script, which checks if the script is being run directly and not being imported as a module.

Examples

Here is an example of how to use the application:

1. **Generate Prime Numbers:** Send a POST request to the /primes endpoint with the start and end of the range and the generation

method in the request body. The response will include the generated prime numbers and the time elapsed.
2. **View Execution Details:** Send a GET request to the /executions endpoint. The response will be an HTML table of all executions, including timestamp, range, time elapsed, algorithm chosen, and the number of primes returned.
3. **Run the Server:** Run the Python script that starts the Uvicorn server. The server will start running the FastAPI application on the specified host and port.

Conclusion

This FastAPI application provides a robust and efficient way to generate prime numbers within a given range over a REST API. It offers different generation strategies, records each execution in a database, and provides a user-friendly form for prime number generation. The application is served by a Uvicorn server, making it accessible over the network.

4. Conclusion

This project provides a robust and efficient solution for generating prime numbers within a given range. It offers four different strategies for generating these prime numbers, allowing users to choose the one that best

fits their needs. The Python script includes comprehensive documentation and unit tests, ensuring that it is easy to understand and maintain.

The FastAPI application serves as a server for the Prime Number Generator. It provides a REST API over HTTP, allowing users to generate prime numbers within a given range using different generation strategies. The application records each execution in a database, including timestamp, range, time elapsed, algorithm chosen, and the number of primes returned.

The Uvicorn server is an ASGI server implementation that serves the FastAPI application. It runs the application on a specified host and port, making it accessible over the network.

Overall, this project demonstrates a good understanding of prime number generation algorithms, Python programming, unit testing, FastAPI application development, and Uvicorn server management. It’s a great example of how different technologies can be combined to create a useful and efficient service.