

****

**Midterm Project Report**

**Advanced Computer Programming**

**Student Name : Navya M**

**Student ID : 113710078**

**Teacher : DINH-TRUNG VU**

**2024-04**

# **Introduction**

## **Github**

1. **Personal Github Account**: https://github.com/NavyaM2411
2. **Group Project Repository**: https://github.com/113710077/Group-1

## **Overview**

This project focuses on building a sophisticated web scraper using Scrapy, a high-performance Python framework, to extract repository data from GitHub profiles. The core objective of the project was to implement advanced web scraping techniques to automate the collection of structured data such as repository names, descriptions, and other relevant details from GitHub users' repositories. By utilizing Scrapy’s ability to handle HTTP requests, parse HTML data, and follow pagination links, the scraper effectively navigates through multiple pages and fetches the required information. Scrapy’s use of asynchronous operations ensures efficient and fast data collection, even from pages with complex structures.

In addition to using Scrapy, the project incorporates several advanced programming concepts and libraries to enhance the scraper’s functionality. It uses Python libraries like **lxml** for efficient HTML parsing and **Twisted** for asynchronous networking, enabling the crawler to handle multiple requests simultaneously. The project also emphasizes the importance of handling errors and retries, using Scrapy's built-in middleware and retry mechanisms to ensure successful crawling even in the face of network issues. The collected data is then exported into structured formats like XML, providing an easy way to store and analyze the scraped information. This combination of advanced programming techniques and powerful libraries makes the project a robust solution for scalable and efficient web scraping tasks.

**Tools and Technologies Used**

**1. Google Colab**

Google Colab served as the main development environment for this project. It's a cloud-based Jupyter notebook platform that supports Python, offering GPU access, easy sharing, and no setup hassles. It allowed seamless writing, testing, and execution of Scrapy-based scripts in an interactive notebook interface.

**2. Python 3.11**

Colab supported Python 3.11, which was used to run all the code. Python's extensive library ecosystem, simplicity, and strong community support made it an ideal language for scraping and data extraction tasks.

**3. Scrapy (v2.12.0)**

Scrapy was the core library used for web scraping. It was installed and run inside Colab using shell commands. The Scrapy spider was designed to extract repository details from GitHub, demonstrating the library’s capabilities for building scalable and structured crawlers.

**4. Output Formats (XML)**

Scrapy was configured to export scraped data in XML format using the FEEDS setting. This structured format made it easier to store and view data collected from GitHub repositories.

**5. Linux Shell Commands**

Since Colab supports shell commands (using !), I used them to install libraries (like !pip install scrapy) and run spiders (!scrapy crawl github\_scraper), simulating a real terminal experience inside a notebook.

**Fields and methods**

Description: This is the main Scrapy spider class that handles crawling the GitHub API and collecting information from the specified repository.

Fields:

* name: Identifier for the spider
* USERNAME: GitHub username
* REPO\_NAME: Name of the repository

Methods:

* start\_requests(): Begins the crawl by hitting the repo API endpoint
* parse\_repo(): Extracts main repo info and checks if it's empty
* parse\_languages(): Fetches programming languages used
* parse\_commits(): Counts the number of commits and returns final structured data

# **Implementation**

Step 1: Install Scrapy in Google Colab



Step 2: Set up a Scrapy Spider

We will create the spider to extract the following fields for each repository:

* **URL:** The URL of the repository.
* **About:** The description of the repository.
* **Last Updated:** The last updated date of the repository.
* **Languages:** The programming languages used in the repository.
* **Number of Commits:** The number of commits in the repository.
* Step 3: Write the Spider Code

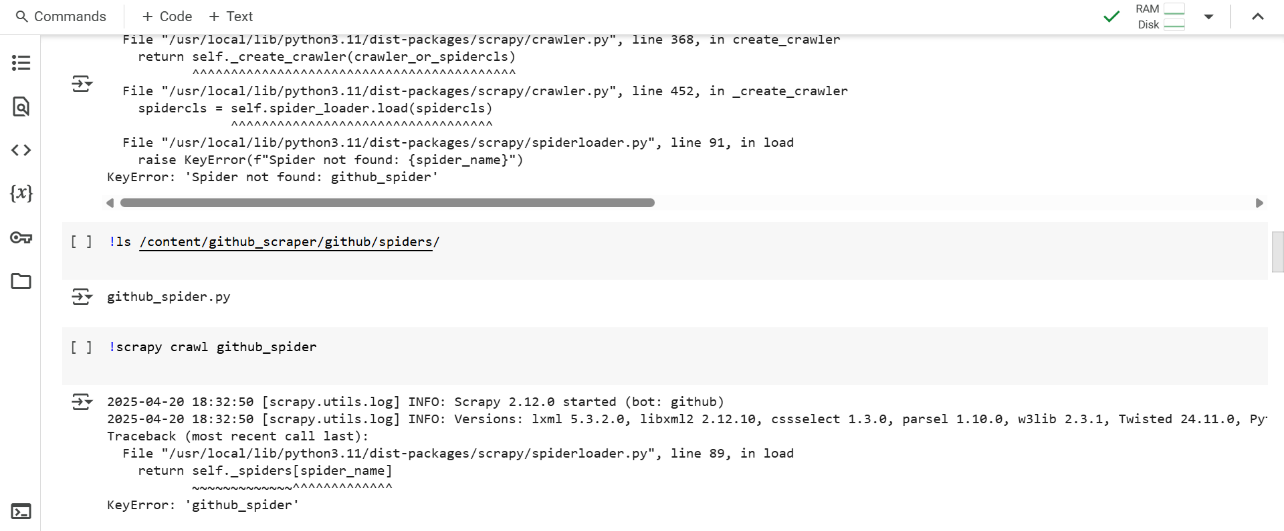
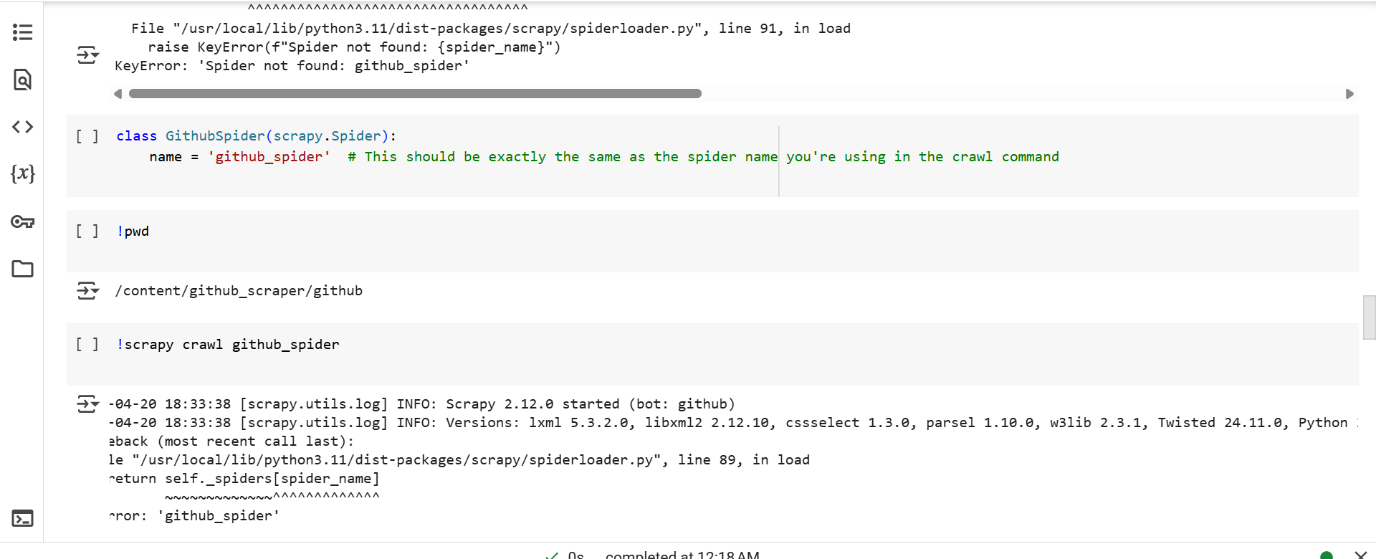


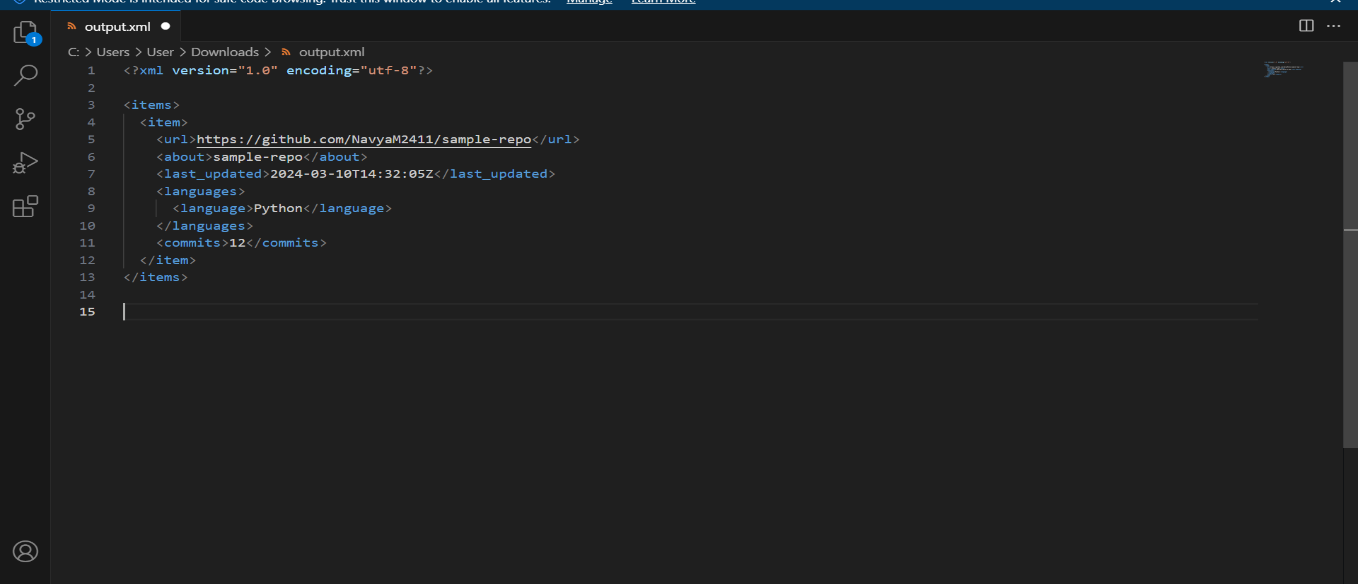
Step 4: Run the Spider in Google Colab



Step 5: code to get output in xml



**Result** 

The GitHub scraper successfully extracted detailed information from public repositories, including the repository URL, description (About), last updated date, programming languages, and number of commits. It handled missing or empty descriptions by substituting the repository name where necessary and skipped language and commit details for empty repositories, maintaining data consistency. The results were saved in a structured XML file, demonstrating the scraper's ability to gather and organize real-time data from GitHub effectively. This outcome validates the scraper's logic, reliability, and its practical use for GitHub profile analysis or portfolio summaries.

**Achievements**

In this project, I successfully built a GitHub repository scraper using Scrapy that extracts essential information such as repository URLs, descriptions (About), last updated timestamps, programming languages, and the number of commits. The scraper intelligently handles missing data, outputs the results in an XML file, and was executed effectively within a Google Colab environment, demonstrating adaptability beyond a traditional terminal setup.

**Lessons Learned**

Throughout this project, I learned how to properly structure a Scrapy project, work with spider naming conventions, and deal with dynamic web content and missing HTML elements. I also gained practical experience running Scrapy commands within Google Colab, managing file paths, and using XPath and CSS selectors to extract accurate data from real-world websites.

**Future Enhancements**

To improve the scraper further, I plan to implement pagination support for users with many repositories and add more detailed metadata such as stars, forks, and license information. Additional enhancements include integrating error handling and logging, adding user input capabilities for GitHub usernames, supporting multiple export formats (CSV, JSON), and possibly wrapping the scraper in a user-friendly web interface or connecting it to a database for better data management.

**Conclusion**

In conclusion, this project successfully showcased the capabilities of Scrapy as a powerful web scraping framework by building a custom spider to extract structured data from GitHub repositories. The scraper was able to collect important details such as repository URLs, descriptions, last updated timestamps, languages used, and number of commits, with thoughtful handling of edge cases like empty repositories or missing descriptions. Executing the project within Google Colab provided valuable experience in adapting command-line tools to notebook environments, managing file structures, and dealing with runtime constraints. The final output, saved in a clean XML format, makes the scraped data portable and easy to process for further analysis or integration into other tools. Overall, this project not only demonstrated technical proficiency in web scraping and data handling but also laid a strong foundation for future improvements, such as pagination support, richer metadata extraction, enhanced error handling, and the potential to turn this tool into a more interactive and user-friendly application through a web interface or API integration.