

****

**Midterm Project Report**

**Advanced Computer Programming**

**Student Name : Khuslen Gantumur**

**Student ID : 113021189**

**Teacher : DINH-TRUNG VU**

**2025-04-13**

# Introduction

## Github

In this project, we were instructed to use **Scrapy**, a powerful web scraping framework in Python, to extract data from a GitHub user page. The objective was to collect detailed information about all repositories associated with a GitHub account and export the data into an **XML** file. The required information for each repository included the **repository URL**, **About description**, **Last Updated date**, **programming languages used**, and the **number of commits**.

Additionally, we were given two specific instructions to enhance the data extraction process:

1. **About field extraction** – If the "About" section of a repository is empty, we must check whether the repository itself is empty. If the repository is not empty, we should use the repository name as the "About" text.
2. **Languages and number of commits** – These should only be extracted if the repository is not empty. Otherwise, both fields should be set to "None".

To complete these tasks, I used my personal GitHub account with the username **113021189**.

## Overview

This project implements a GitHub repository scraper using Python, with a focus on structured data extraction and conditional logic. The scraper collects comprehensive repository information from a specified GitHub user profile and exports it in an XML format. Key accomplishments of the project include:

* **Successful extraction** of five core data fields: **Repository URL**, **About**, **Last Updated**, **Languages**, and **Number of Commits**
* **Conditional data handling** for incomplete or empty repositories to ensure robustness
* **Automated XML generation** for structured and portable output

**Advanced Features and Techniques Used:**

1. **Data Classes**: Utilized the @dataclass decorator to define a clean and structured Repository model for storing repository attributes.
2. **HTML Parsing with BeautifulSoup**: Implemented BeautifulSoup to parse and navigate GitHub’s HTML structure effectively.
3. **Conditional Logic**: Applied smart fallbacks such as using the repository name as the "About" field when no description is available, and setting Languages and Commits to None for empty repositories.
4. **(Planned) Scrapy Integration**: While the current version uses BeautifulSoup, the project lays a foundation for transitioning to Scrapy for enhanced scalability and built-in features like XML feed exporting.
5. **Custom Data Formatting**: Prepared data for XML output with structured formatting, allowing easy integration with other systems.

# Implementation

## Class RepositoryItem(Data Class)

Description of Repository Item: RepositoryItem is a data class that defines the schema for each GitHub repository’s extracted information. It provides a structured and type-safe way to store repository attributes, making it easy to manage and export data in formats such as XML. The use of the @dataclass decorator reduces boilerplate code by automatically generating initializer and representation methods.

### Fields

A screen shot of a computer program

AI-generated content may be incorrect.

### Methods

Utilizes built-in dataclass methods:

* \_\_init\_\_() - It initializes the object with the values you pass when creating a new instance.
* \_\_repr\_\_() - It makes printed objects more readable by showing their contents.

## GithubSpider (Scrapy Spider)

Description of GithubSpider: GithubSpiderSpider is a custom Scrapy spider that crawls a specified GitHub user's repository page, extracts detailed information about each repository, and exports the data to an XML file. It handles pagination, detects empty repositories, and follows links to gather additional repository details such as programming languages and commit count. The spider uses conditional logic to handle incomplete or empty repositories and ensures robust data extraction with fallbacks.

### Fields

**A computer screen with text

AI-generated content may be incorrect.**

### Methods

**parse(self, response)**

**Purpose:**  
Handles the initial page of repositories, extracts basic information, and detects if repositories are empty.

**Functionality:**

* Extracts repository URL, "About" text, and last updated timestamp.
* Checks for empty repositories using content analysis.
* Applies fallback logic if the "About" section is missing.
* If the repository is not empty, follows the repository URL to get more details.
* Handles pagination to continue crawling across multiple pages.

**parse\_repository\_details(self, response)**

**Purpose:**  
Extracts detailed information from the repository’s individual page.

**Functionality:**

* Extracts programming languages used in the repository using modern CSS selectors.
* Retrieves the number of commits using multiple fallback methods (CSS and XPath).
* Completes the RepositoryItem object and yields it for XML output.

# Results

## Result 1

The spider successfully crawled the GitHub profile and generated structured XML data for each repository. The resulting XML file includes essential metadata such as repository URL, description, last updated timestamp, programming languages, and number of commits.

A sample of the generated XML output:

A computer screen shot of a program

AI-generated content may be incorrect.

.

## Result 2

Extracted Repository Data

A screenshot of a computer

AI-generated content may be incorrect.

**3.3 Result 3**

Summary Statistics

**A black screen with white text

AI-generated content may be incorrect.**

# Conclusions

This project successfully demonstrated the practical application of web scraping techniques to extract structured data from GitHub using Python. The scraper was designed to gather comprehensive information about repositories—such as URLs, descriptions, update timestamps, languages, and commit counts—from a specified GitHub user profile and export this data into a cleanly structured XML file.

Key accomplishments of the project include:

* **Robust Data Extraction:** The scraper accurately handled both complete and incomplete repository entries by implementing conditional logic and fallback strategies. This ensured that fields like "About", "Languages", and "Commits" were handled appropriately, even when certain data was missing or unavailable.
* **Efficient Structure with Data Classes:** The use of a RepositoryItem data class improved the clarity, reusability, and type safety of the code, simplifying data handling and export operations.
* **Effective Parsing and Navigation:** Leveraging libraries such as BeautifulSoup and Scrapy allowed for precise and reliable HTML parsing, even when dealing with pagination and dynamic page content.
* **Automated XML Export:** The final data was exported in a structured XML format suitable for further processing, analysis, or integration into other systems.

1. **Future Enhancements**

While the current version of the scraper is effective for public repository data, several areas of improvement can be explored in future iterations:

* **Scrapy-Only Implementation:** Transitioning the full implementation to Scrapy can provide better scalability, modularity, and performance.
* **Authentication Support:** Adding GitHub authentication will allow scraping private repositories if permitted by the user.
* **Rate Limiting and Error Recovery:** Introducing delay management and retry logic would enhance reliability and ensure compliance with GitHub’s usage policies.
* **Testing Framework Integration:** Incorporating Scrapy's built-in testing tools can improve code quality and help maintain reliability over time.

1. **Final Remarks**

Overall, this project not only fulfilled its technical objectives but also provided a valuable learning experience in data extraction, structured programming, and real-world use of Python web scraping libraries. It lays a solid foundation for future enhancements and larger-scale data collection tools.