亞洲大學
ASIA UNIVERSITY

Midterm Project Report

Advanced Computer Programming

Student Name : Yesuijin Batmunkh
Student ID : 113021192

Teacher : DINH-TRUNG VU

2025-04

Chapter 1 Introduction

1.1 Github

1) **Personal Github Account**: https://github.com/113021192
2) **Group Project Repository**: https://github.com/Jantsagdorj/ACP-AU-1132

1.2 Overview

In this project, I used advanced programming techniques and libraries to develop a web scraper using Scrapy. The primary goal was to extract detailed information from GitHub repositories and output this data into an XML file. The key features and libraries used in this project include:

Scrapy: A powerful and flexible web scraping framework that allows for efficient data extraction from websites.

CSS Selectors: Used to navigate and extract specific elements from the HTML structure of the GitHub pages.

Regular Expressions: Used to accurately extract numerical data, such as the number of commits, from text.

The program is designed to scrape the following information from each repository on a GitHub page:

URL: The link to the repository.

About: A brief description of the repository. If the "About" section is empty, the repository name is used instead.

Last Updated: The date and time when the repository was last updated.

Languages: The programming languages used in the repository.

Number of Commits: The total number of commits made to the repository.

Chapter 2 Implementation

1.1 Setup and Environment

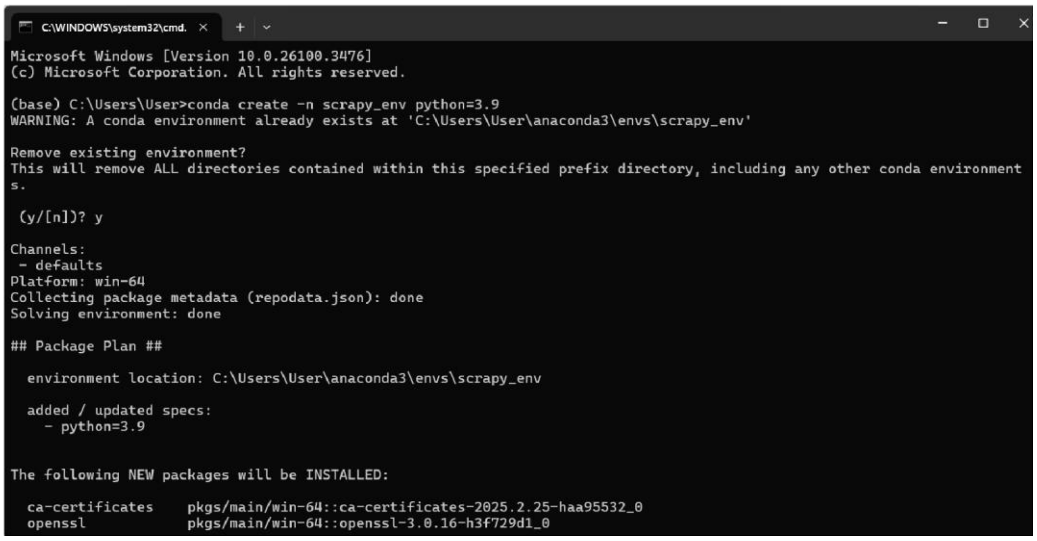
1.1.1 Environment Information

For this project, I used the following setup and environment:

- Anaconda: A distribution of Python and R for scientific computing and data science. It simplifies package management and deployment.
- Anaconda Prompt: The command-line interface provided by Anaconda for managing environments and running Python scripts.
- Python Version: 3.9
- Scrapy Version: 2.12
- Environment: Created a dedicated environment in Anaconda for this project to ensure all dependencies are managed and isolated.

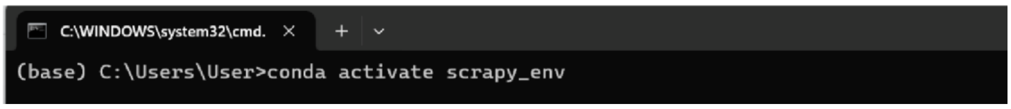
1.1.2 Installing Anaconda and creating the project

1. Install Anaconda: Download and install Anaconda from the official website.
2. Create a new environment:



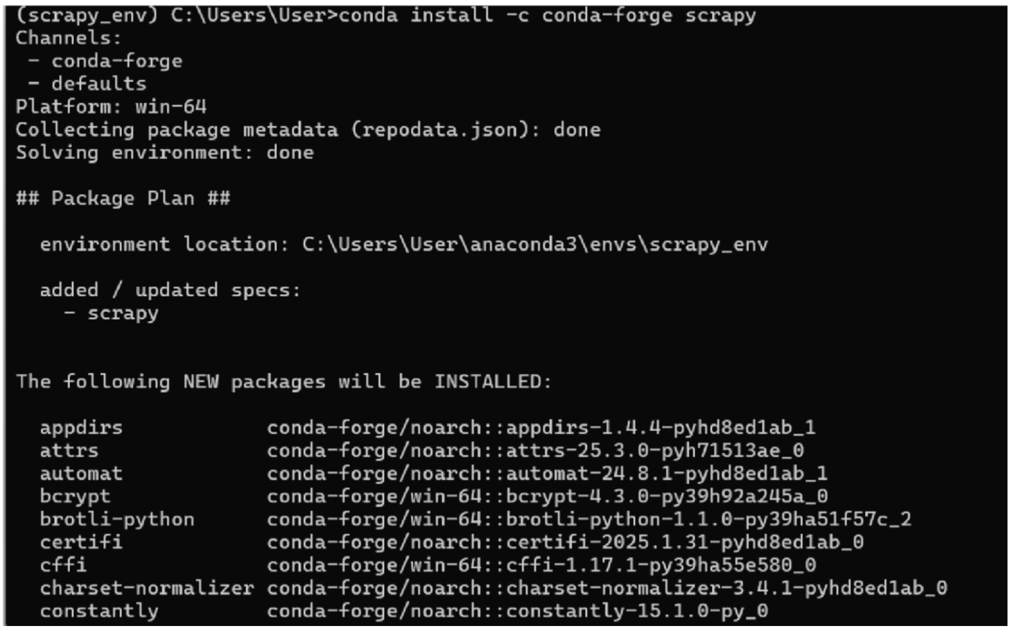
```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

(base) C:\Users\User>conda create -n scrapy_env python=3.9
WARNING: A conda environment already exists at 'C:\Users\User\anaconda3\envs\scrapy_env'

Remove existing environment?
This will remove ALL directories contained within this specified prefix directory, including any other conda environment
s.

 (y/[n])? y

Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\User\anaconda3\envs\scrapy_env

  added / updated specs:
    - python=3.9


The following NEW packages will be INSTALLED:

  ca-certificates    pkgs/main/win-64::ca-certificates-2025.2.25-haa95532_0
  openssl            pkgs/main/win-64::openssl-3.0.16-h3f729d1_0
```

3. Activate the environment:



```
(base) C:\Users\User>conda activate scrapy_env
```

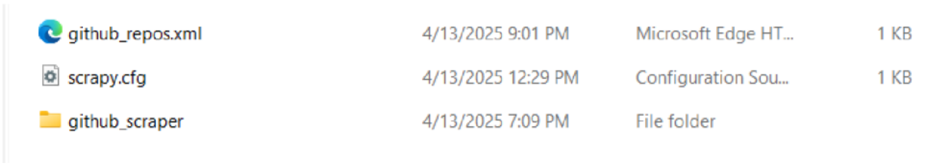
4. Install Scrapy:



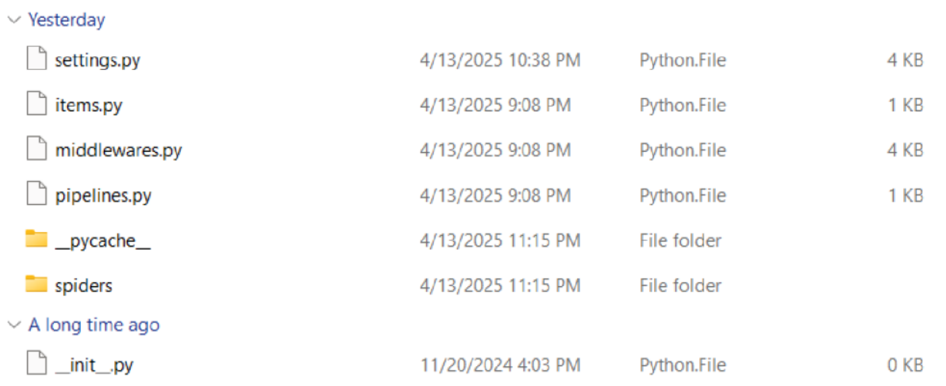
```
(scrapy_env) C:\Users\User>conda install -c conda-forge scrapy
Channels:
 - conda-forge
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\User\anaconda3\envs\scrapy_env

  added / updated specs:
    - scrapy


The following NEW packages will be INSTALLED:

  appdirs            conda-forge/noarch::appdirs-1.4.4-pyhd8ed1ab_1
  attrs              conda-forge/noarch::attrs-25.3.0-pyh71513ae_0
  automat            conda-forge/noarch::automat-24.8.1-pyhd8ed1ab_1
  bcrypt             conda-forge/win-64::bcrypt-4.3.0-py39h92a245a_0
  brotli-python      conda-forge/win-64::brotli-python-1.1.0-py39ha51f57c_2
  certifi            conda-forge/noarch::certifi-2025.1.31-pyhd8ed1ab_0
  cffi               conda-forge/win-64::cffi-1.17.1-py39ha55e580_0
  charset-normalizer conda-forge/noarch::charset-normalizer-3.4.1-pyhd8ed1ab_0
  constantly         conda-forge/noarch::constantly-15.1.0-py_0
```

1.1.3 Folder Structure

```
github_scraper/
|
├── scrapy.cfg
└── github_scraper/
    ├── __init__.py
    ├── items.py
    ├── middlewares.py
    ├── pipelines.py
    ├── settings.py
    └── spiders/
        └── __init__.py
```



github_repos.xml	4/13/2025 9:01 PM	Microsoft Edge HT...	1 KB
scrapy.cfg	4/13/2025 12:29 PM	Configuration Sou...	1 KB
github_scraper	4/13/2025 7:09 PM	File folder	



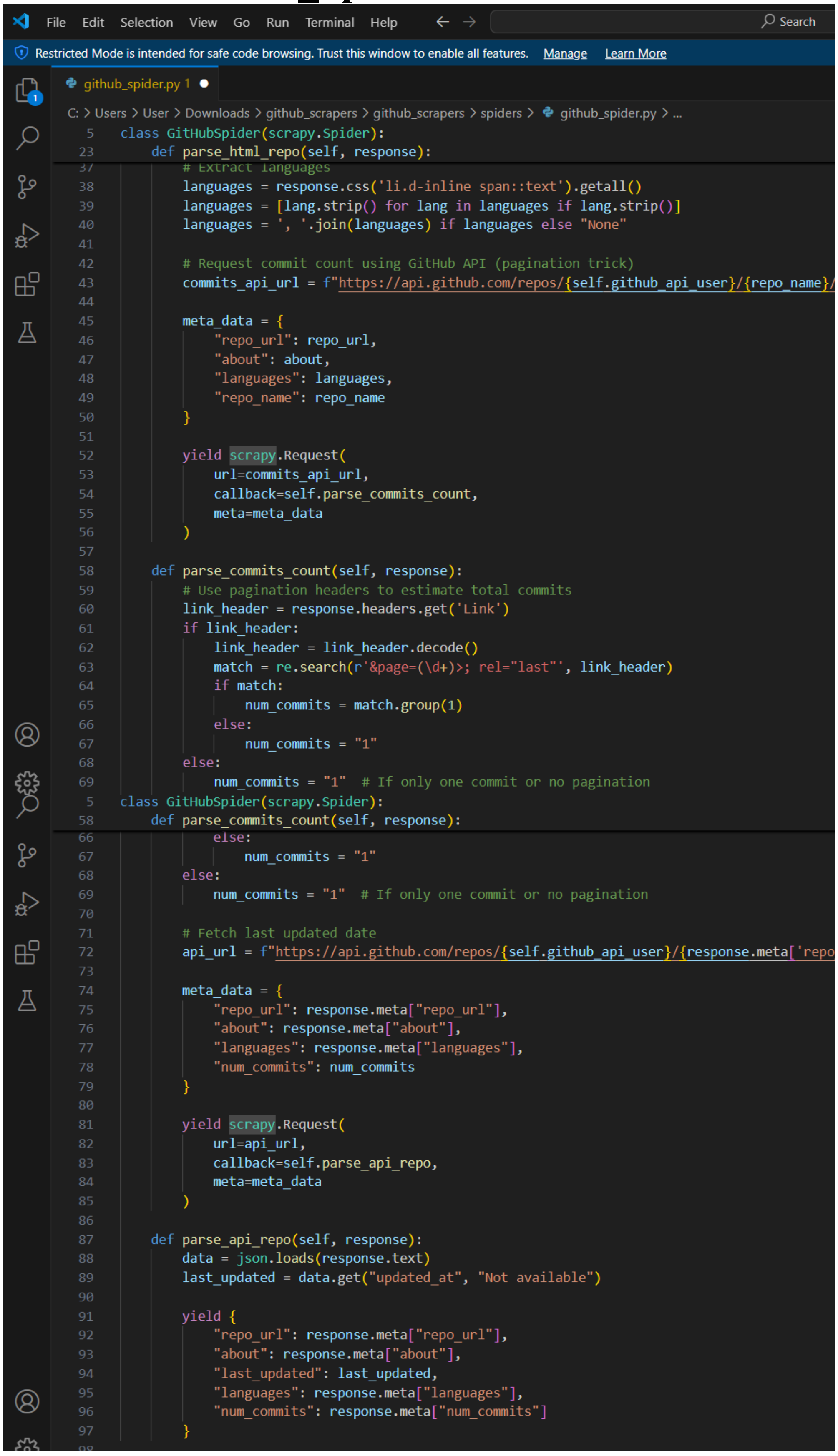
Yesterday

settings.py	4/13/2025 10:38 PM	Python.File	4 KB
items.py	4/13/2025 9:08 PM	Python.File	1 KB
middlewares.py	4/13/2025 9:08 PM	Python.File	4 KB
pipelines.py	4/13/2025 9:08 PM	Python.File	1 KB
__pycache__	4/13/2025 11:15 PM	File folder	
spiders	4/13/2025 11:15 PM	File folder	

A long time ago

__init__.py	11/20/2024 4:03 PM	Python.File	0 KB

1.2 Class: Github_Spider


File
Edit
Selection
View
Go
Run
Terminal
Help
Search
github_spider.py 1
C: > Users > User > Downloads > github_scrapers > github_scrapers > spiders > github_spider.py > ...

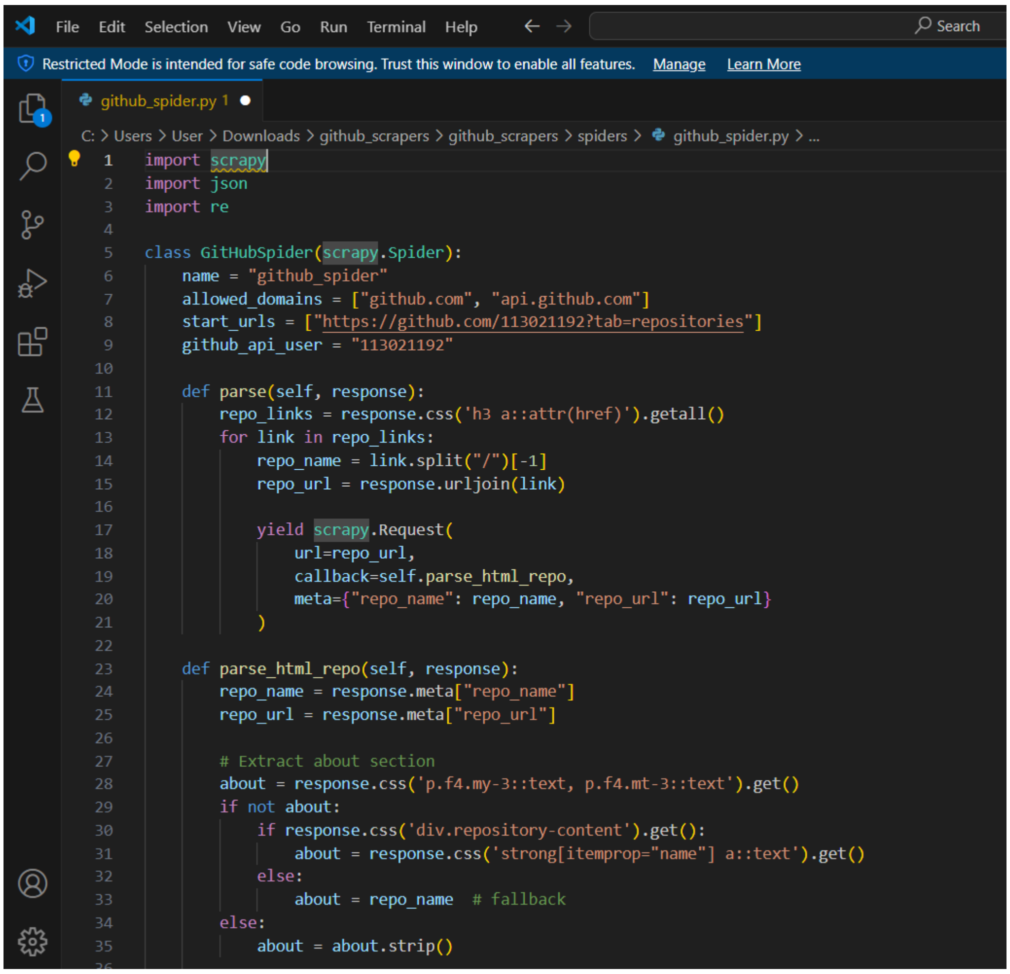
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

```python
class GitHubSpider(scrapy.Spider):
    def parse_html_repo(self, response):
        # Extract languages
        languages = response.css('li.d-inline span::text').getall()
        languages = [lang.strip() for lang in languages if lang.strip()]
        languages = ', '.join(languages) if languages else "None"

        # Request commit count using GitHub API (pagination trick)
        commits_api_url = f"https://api.github.com/repos/{self.github_api_user}/{repo_name}/

        meta_data = {
            "repo_url": repo_url,
            "about": about,
            "languages": languages,
            "repo_name": repo_name
        }

        yield scrapy.Request(
            url=commits_api_url,
            callback=self.parse_commits_count,
            meta=meta_data
        )

    def parse_commits_count(self, response):
        # Use pagination headers to estimate total commits
        link_header = response.headers.get('Link')
        if link_header:
            link_header = link_header.decode()
            match = re.search(r'&page=(\d+)>; rel="last"', link_header)
            if match:
                num_commits = match.group(1)
            else:
                num_commits = "1"
        else:
            num_commits = "1"  # If only one commit or no pagination
```

```python
class GitHubSpider(scrapy.Spider):
    def parse_commits_count(self, response):
```

```python
            else:
                num_commits = "1"
        else:
            num_commits = "1"  # If only one commit or no pagination
```

```python

        # Fetch last updated date
        api_url = f"https://api.github.com/repos/{self.github_api_user}/{response.meta['repo

        meta_data = {
            "repo_url": response.meta["repo_url"],
            "about": response.meta["about"],
            "languages": response.meta["languages"],
            "num_commits": num_commits
        }

        yield scrapy.Request(
            url=api_url,
            callback=self.parse_api_repo,
            meta=meta_data
        )

    def parse_api_repo(self, response):
        data = json.loads(response.text)
        last_updated = data.get("updated_at", "Not available")

        yield {
            "repo_url": response.meta["repo_url"],
            "about": response.meta["about"],
            "last_updated": last_updated,
            "languages": response.meta["languages"],
            "num_commits": response.meta["num_commits"]
        }
```


File
Edit
Selection
View
Go
Run
Terminal
Help
Search
Manage
Learn More
github_spider.py 1
C: > Users > User > Downloads > github_scrapers > github_scrapers > spiders > github_spider.py > ...

Restricted Mode is intended for safe code browsing. Trust this window to enable all features.

```python
import scrapy
import json
import re

class GitHubSpider(scrapy.Spider):
    name = "github_spider"
    allowed_domains = ["github.com", "api.github.com"]
    start_urls = ["https://github.com/113021192?tab=repositories"]
    github_api_user = "113021192"

    def parse(self, response):
        repo_links = response.css('h3 a::attr(href)').getall()
        for link in repo_links:
            repo_name = link.split("/")[-1]
            repo_url = response.urljoin(link)

            yield scrapy.Request(
                url=repo_url,
                callback=self.parse_html_repo,
                meta={"repo_name": repo_name, "repo_url": repo_url}
            )

    def parse_html_repo(self, response):
        repo_name = response.meta["repo_name"]
        repo_url = response.meta["repo_url"]

        # Extract about section
        about = response.css('p.f4.my-3::text, p.f4.mt-3::text').get()
        if not about:
            if response.css('div.repository-content').get():
                about = response.css('strong[itemprop="name"] a::text').get()
            else:
                about = repo_name  # fallback
        else:
            about = about.strip()
```

Description: This class is a Scrapy spider specifically designed to scrape information from GitHub repositories. It navigates through the user's repositories and extracts the required data.

Fields:

- name: The name of the spider, which is "github_spider".
- start_urls: The starting URL for the spider, which points to the user's GitHub repositories page.

Methods:

- parse(response): This method is responsible for extracting repository links from the main repositories page. It uses CSS selectors to find all repository links and initiate requests to parse each repository individually.
- parse_repo(response): This method handles the extraction of detailed information from each repository page. It extracts the "About" section, last

updated date, languages used, and number of commits. It also includes logic to handle cases where the "About" section is empty or the repository is empty.

***Functions*:**

- parse(response):
  - Extracting Repository Links: Uses CSS selectors to find all repository links on the main page.
  - Initiating Requests: For each repository link, it initiates a request to parse the repository details.
- parse_repo(response):
  - Extracting "About" Section: Attempts to extract the "About" section. If it's empty, it checks if the repository is empty. If not, it uses the repository name as the "About" section.
  - Extracting Last Updated Date: Uses CSS selectors to find the last updated date of the repository.
  - Extracting Languages: Extracts the programming languages used in the repository.
  - Extracting Number of Commits: Uses regular expressions to extract the number of commits. If the repository is empty, it sets the languages and number of commits to None.

1.3 Class XMLWriter

Description: This class is responsible for writing the scraped data into an XML file. It ensures that the data is structured correctly and encoded in UTF-8.
Fields:

- file_name: The name of the XML file to be created.

Methods:

- write_to_xml(data): This method takes the scraped data and writes it to the XML file.

Functions:

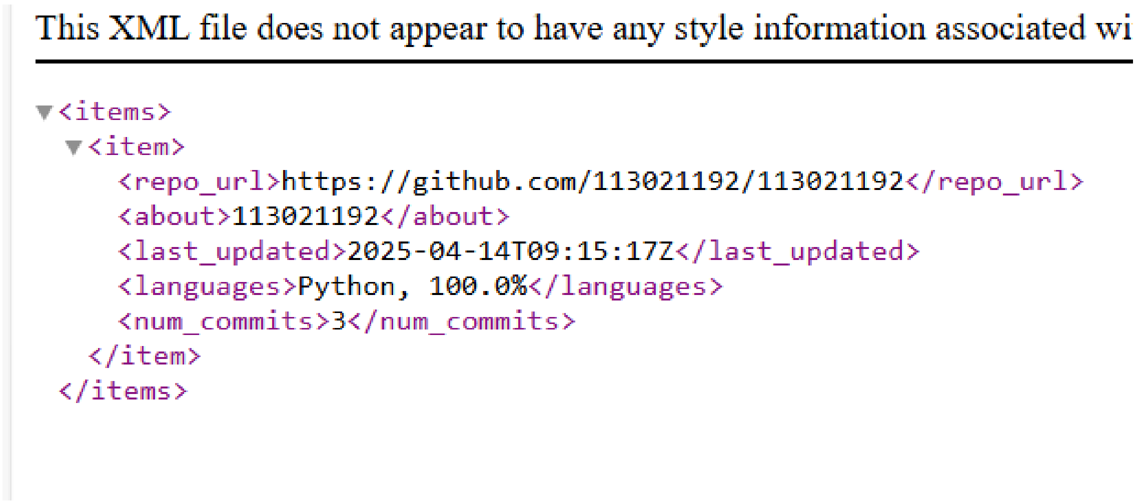
- write_to_xml(data):

Creating XML Structure: Constructs the XML structure with the necessary tags and attributes.

Writing Data: Writes the scraped data into the XML file, ensuring proper encoding and formatting.

Chapter 3 Results

The Scrapy spider was executed using the command: scrapy crawl github.
It generated an XML file in the 'output' directory named `github_repos.xml`. This file includes structured information about each repository as follows:



This XML file does not appear to have any style information associated wi

```
▼<items>
  ▼<item>
     <repo_url>https://github.com/113021192/113021192</repo_url>
     <about>113021192</about>
     <last_updated>2025-04-14T09:15:17Z</last_updated>
     <languages>Python, 100.0%</languages>
     <num_commits>3</num_commits>
   </item>
 </items>
```

This output can be used for analyzing GitHub activity and repository metadata.

.

Chapter 4 Conclusions

This project demonstrates the effective use of Scrapy for web scraping and data extraction. The spider is designed to handle various scenarios, such as empty "About" sections and empty repositories, ensuring robust and reliable data collection. The output XML file serves as a comprehensive dataset that can be used for further analysis or integration with other applications.

The use of advanced programming techniques and libraries, such as CSS selectors and regular expressions, highlights the flexibility and power of Python for web scraping tasks. This project not only achieves its goal of extracting detailed information from GitHub repositories but also provides a solid foundation for future web scraping projects.