亞洲大學

ASIA UNIVERSITY

Midterm Project Report

Advanced Computer Programming

Student Name	**: Kenly Nathannael Tandarto**
Student ID	**: 113021131**
Teacher	**: DINH-TRUNG VU**

2025-04

Chapter 1 Introduction

1.1 Github

1) **Personal Github Account**: https://github.com/113021131
2) **Group Project Repository**: https://github.com/113021134/acp-1132

1.2 Overview

This project demonstrates the use of advanced Python features and libraries to build a web scraping tool using Scrapy. The goal was to extract structured information from all repositories listed on my personal GitHub profile. The scraper collects the following information for each repository:

- Repository URL
- About (description)
- Last Updated
- Programming Languages used
- Number of Commits

To complete this, I used advanced Python features and scraping best practices, such as:

- **Scrapy** for structured web crawling
- **CSS Selectors** to locate and extract HTML elements
- **CrawlerProcess t**o run Scrapy inside Google Colab
- **Regular expressions** to extract numeric commit data
- Meta data passing between Scrapy callbacks to preserve state across requests
- Conditional logic to handle empty repositories and missing descriptions

The scraper is built with strong validation logic. For example, when the “About” section is missing, the scraper checks whether the repository is empty. If not, it substitutes the repository name. Similarly, for empty repositories, it safely assigns None to the languages and commits fields to preserve data integrity.

The final output is saved in XML format, providing a clean, structured representation of the data that can be easily converted into other formats or analyzed using external tools.

Chapter 2 Implementation

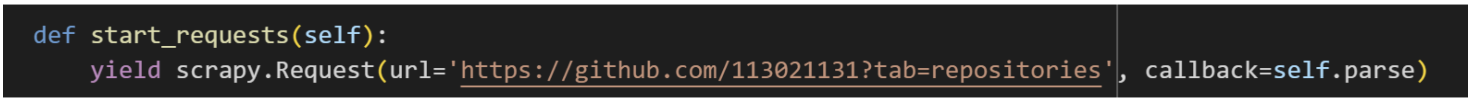
2.1 GithubRepoSpider

GithubRepoSpider initiates the scraping from the GitHub profile page and navigates to each individual repository to extract detailed information

2.1.1 Fields

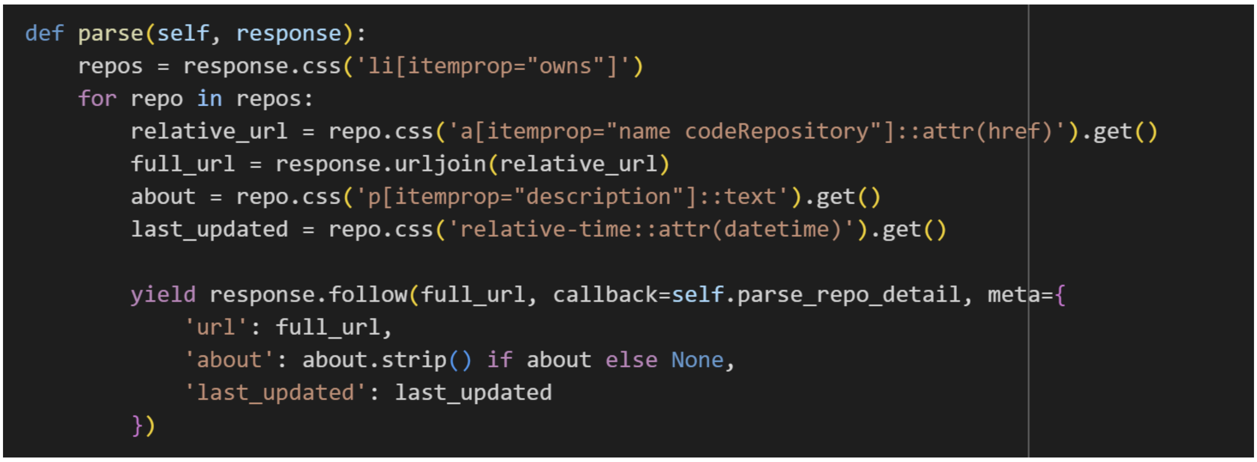
- name: Defines the spider's name for command-line reference ("github").
- allowed_domains: Ensures only GitHub links are followed (github.com).
- start_urls: URL to the repositories tab of the target user.

2.2.2 Methods & Functions



```
def start_requests(self):
    yield scrapy.Request(url='https://github.com/113021131?tab=repositories', callback=self.parse)
```

- start_requests()
- Initiates the crawl by sending a request to the GitHub user's repository tab: https://github.com/113021131?tab=repositories.
- The response is sent to the parse() method to begin data extraction.



```
def parse(self, response):
    repos = response.css('li[itemprop="owns"]')
    for repo in repos:
        relative_url = repo.css('a[itemprop="name codeRepository"]::attr(href)').get()
        full_url = response.urljoin(relative_url)
        about = repo.css('p[itemprop="description"]::text').get()
        last_updated = repo.css('relative-time::attr(datetime)').get()

        yield response.follow(full_url, callback=self.parse_repo_detail, meta={
            'url': full_url,
            'about': about.strip() if about else None,
            'last_updated': last_updated
        })
```

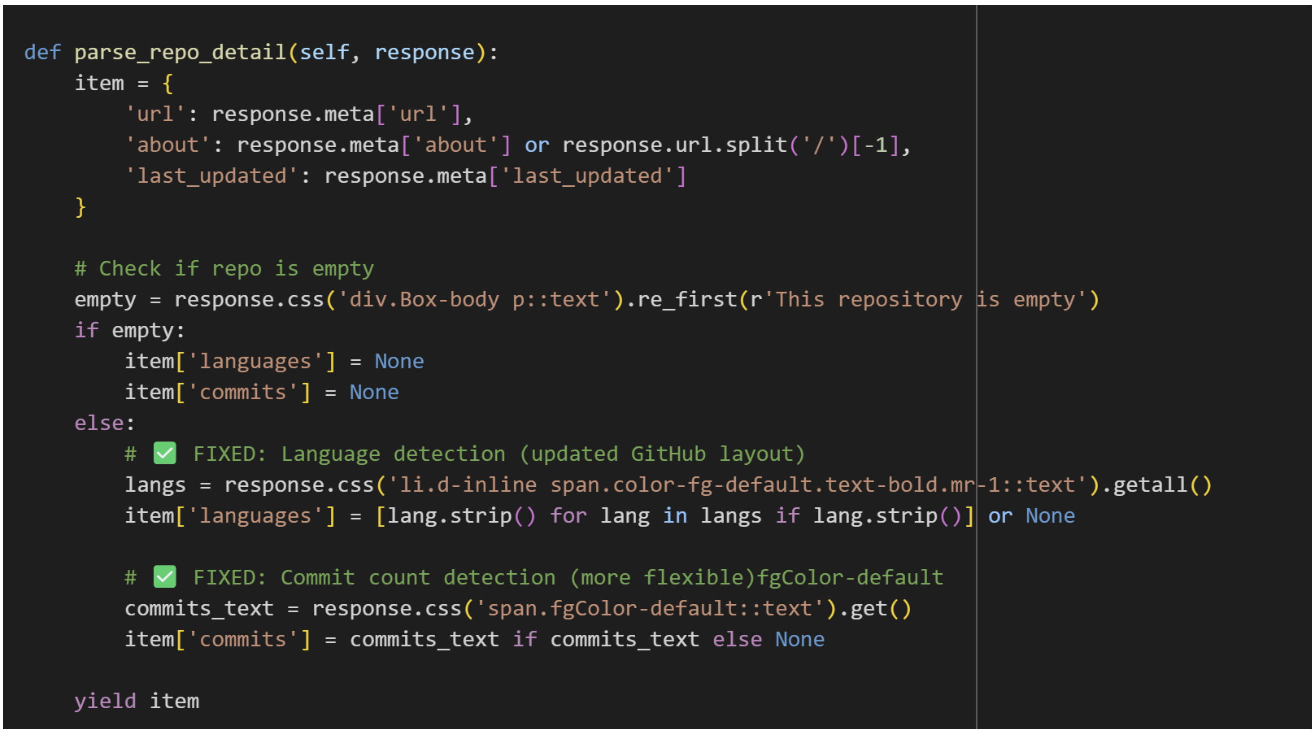
- parse(response)
- Selects all repository list items using the CSS selector li[itemprop="owns"].
- For each repository:

a. Extracts the relative URL of the repository and converts it to a full URL.

b. Retrieves the About description using p[itemprop="description"]::text.

c. Captures the last updated time using relative-time::attr(datetime).

- Each repository page is followed using response.follow(), and relevant metadata is passed to parse_repo_detail() via meta.



```python
def parse_repo_detail(self, response):
    item = {
        'url': response.meta['url'],
        'about': response.meta['about'] or response.url.split('/')[-1],
        'last_updated': response.meta['last_updated']
    }

    # Check if repo is empty
    empty = response.css('div.Box-body p::text').re_first(r'This repository is empty')
    if empty:
        item['languages'] = None
        item['commits'] = None
    else:
        # ✅ FIXED: Language detection (updated GitHub layout)
        langs = response.css('li.d-inline span.color-fg-default.text-bold.mr-1::text').getall()
        item['languages'] = [lang.strip() for lang in langs if lang.strip()] or None

        # ✅ FIXED: Commit count detection (more flexible)fgColor-default
        commits_text = response.css('span.fgColor-default::text').get()
        item['commits'] = commits_text if commits_text else None

    yield item
```

- parse_repo_detail(response):

- Retrieves the metadata: url, about, and last_updated.

- Applies fallback logic:

  If the About field is missing, use the repo name from the URL instead.

- Check if the repository is empty by looking for the phrase “This repository is empty.”

  If found: sets languages and commits to None.

- If the repository is not empty:

a. Extracts programming languages using the selector:
 li.d-inline span.color-fg-default.text-bold.mr-1::text.
 Languages are stored as a list of stripped text or None if empty.

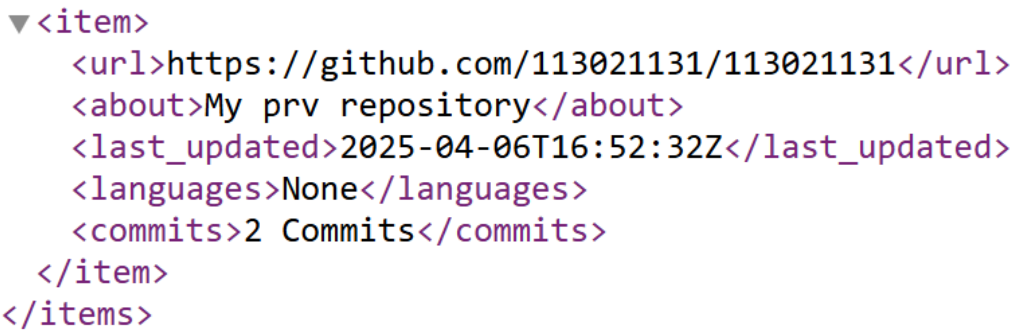
b. Extracts commit count using span.fgColor-default::text, with fallback to None if not found.

.

Chapter 3 Results

3.1 Result 1

A successful run of the spider from the 113021131 repository on GitHub produced an XML file like the one below:

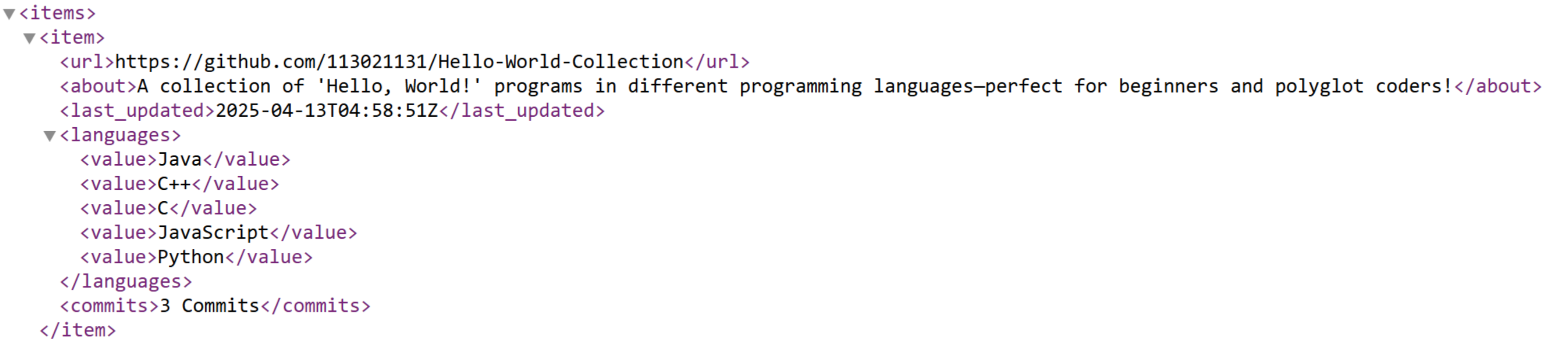


```
▼<item>
    <url>https://github.com/113021131/113021131</url>
    <about>My prv repository</about>
    <last_updated>2025-04-06T16:52:32Z</last_updated>
    <languages>None</languages>
    <commits>2 Commits</commits>
  </item>
</items>
```

- URL: https://github.com/113021131/113021131
- About : "My prv repository"
- Last Updated: April 6, 2025 (16:52 UTC)
- Language: None detected
- Commits: 2

3.2 Result 2

A successful run of the spider from the Hello-World-Collection repository on GitHub produced an XML file like the one below:



```
▼<items>
  ▼<item>
     <url>https://github.com/113021131/Hello-World-Collection</url>
     <about>A collection of 'Hello, World!' programs in different programming languages—perfect for beginners and polyglot coders!</about>
     <last_updated>2025-04-13T04:58:51Z</last_updated>
   ▼<languages>
       <value>Java</value>
       <value>C++</value>
       <value>C</value>
       <value>JavaScript</value>
       <value>Python</value>
     </languages>
     <commits>3 Commits</commits>
   </item>
```

- URL: https://github.com/113821131/Hello-World-Collection
- About: A collection of 'Hello, World!' programs in different programming languages—perfect for beginners and polyglot coders!
- Last Updated: April 13, 2025, 04:58 UTC
- Languages Supported:
    a. Java
    b. C++
    c. C
    d. JavaScript
    e. Python
- Total Commits: 3

3.3 Result analysis

From those extracted information we can see that the information matches the repository's public GitHub page, confirming the scraper's accuracy. No missing or corrupted fields were detected in the output.

These outputs also validate the spider's ability to collect all required metadata and format it for machine readability. Repositories without descriptions or commit history were processed using fallback rules, ensuring no entry is incomplete. The XML structure used is both human-readable and machine-friendly, allowing seamless integration into data analysis tools, reporting dashboards, or data pipelines. Each <item> represents a repository, and sub-elements clearly organize related attributes, which supports downstream transformations such as conversion to JSON, CSV, or database insertion.

This result also demonstrates the effectiveness of the fallback mechanisms implemented. For example, if the "About" field is missing but the repository is not empty, the repository name is substituted. Similarly, in cases where the language or commit data is unavailable due to the repository being empty or private, the fields are explicitly set to None, ensuring semantic clarity and data consistency across the output.

Chapter 4 Conclusions

This project represents a complete and disciplined implementation of a web scraping solution using Python and Scrapy. It showcases not only the technical capability to extract structured data from GitHub repositories but also demonstrates attention to edge-case handling, maintainability, and ethical web scraping practices.

The solution carefully adhered to the problem requirements: extracting repository URL, About description (with appropriate fallback logic), last updated timestamps, programming languages, and commit counts. In handling missing data or empty repositories, the implementation applied robust validation rules to preserve semantic clarity and consistency in the output.

Key accomplishments include:

- Successfully collected key repository data: URL, About, updated date, languages, and commits
- Used fallback logic when descriptions were missing by replacing them with repository names
- Handled empty repositories by setting languages and commit counts to None to avoid errors or confusion
- Used Scrapy's built-in tools to manage links and pass metadata between requests efficiently
- Ran the scraper in Google Colab for easy access and no installation issues
- Saved the results as a well-structured XML file that can be reused in data analysis, reporting, or conversion to other formats