亞洲大學
ASIA UNIVERSITY

# Midterm Project Report

# Advanced Computer Programming

**Student Name : Joel Nelson Wijaya**

**Student ID : 113021135**

**Teacher : DINH-TRUNG VU**

**2025-04**

# Chapter 1 Introduction

## 1.1 Github

1) **Personal Github Account**: https://github.com/113021135
2) **Group Project Repository**: https://github.com/113021134/acp-1132

## 1.2 Overview

In this project, a Python program was created to extract metadata from a GitHub user's public repositories using the GitHub REST API. The project was executed in a Google Colab notebook environment, which made development and file handling more streamlined. The primary goal was to extract specific metadata fields from each repository—namely: the URL, description (About), date of last update, programming languages used, and an estimate of the number of commits. The extracted data was structured into an XML file named repos.xml. This file serves as a summary, record, or portfolio of a GitHub user's public development activity.

Advanced Python features and libraries were utilized, including:

- **requests library**: used to send HTTP GET requests to GitHub's API endpoints for repositories, languages, and commits.
- **xml.etree.ElementTree**: a built-in XML processing library in Python, used to build and export the structured XML file.
- **JSON parsing**: used to convert the JSON API responses into Python dictionaries for easier manipulation.
- **Google Colab files module**: used to download the resulting XML file directly to the local machine after generation.

My program has successfully extracted and structured information about the URL, About, Last_Updated, Languages, and Number_of_Commits of repositories from the GitHub user page https://github.com/113021135. This demonstrates how data from public developer profiles can be processed and structured for presentation or further analysis.

# Chapter 2 Implementation

## 1.1 Class 1

This part of the program acts as the main component responsible for fetching data from GitHub and creating a structured XML output. While no explicit class syntax (i.e., class GitHubRepoExtractor:) was used, the program's logic is functionally organized in a modular and class-like manner.

### 1.1.1 Fields

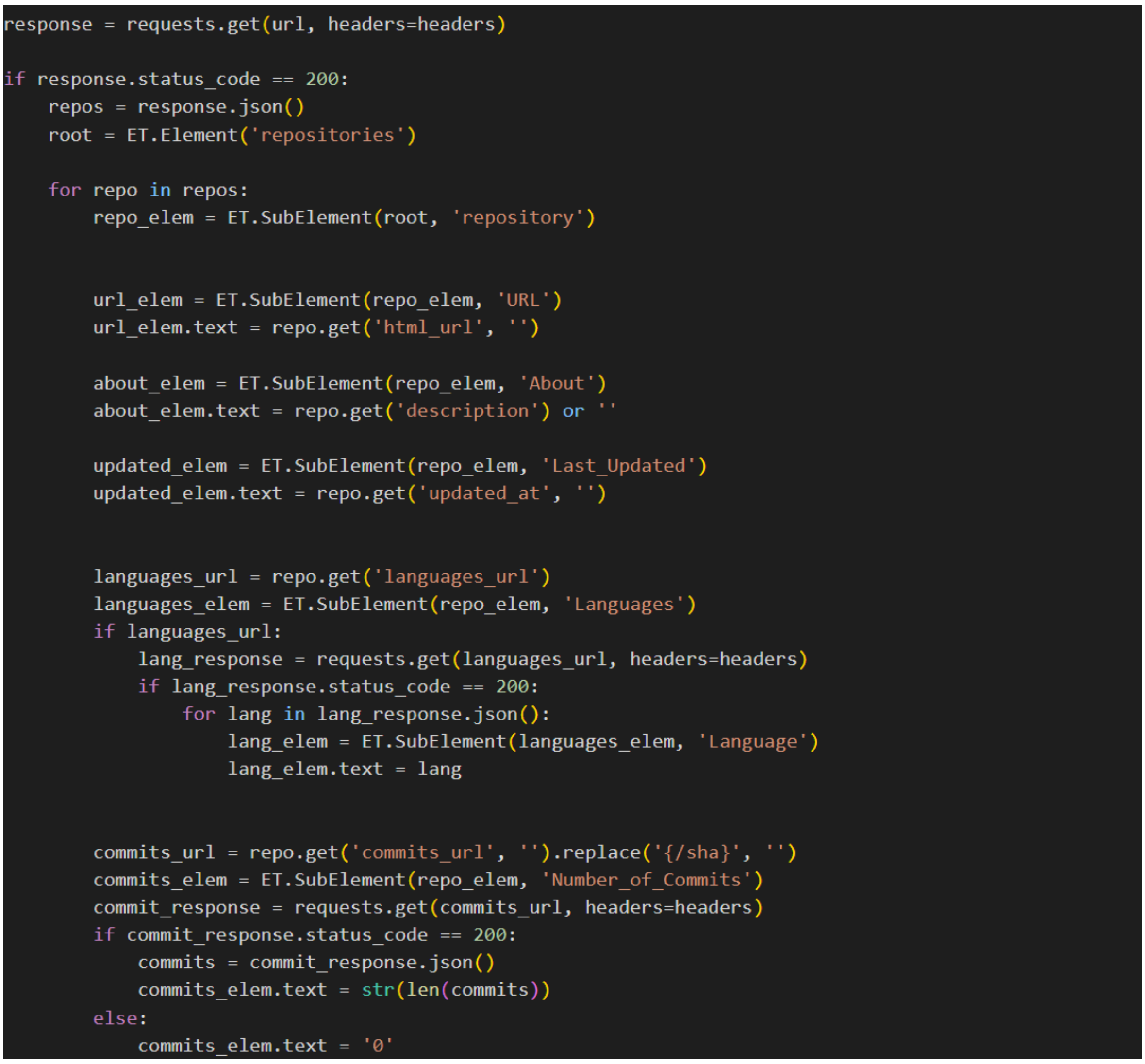
Username: The GitHub username whose repositories will be analyzed.

Headers: A dictionary for optional authentication headers (e.g., a personal access token), used to avoid API rate limiting.

Url: GitHub API endpoint used to retrieve the list of repositories for the given username.

### 1.1.2 Methods/Functions

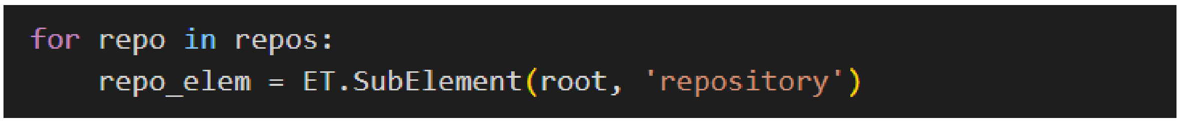
The program uses several key methods and functions to collect, process, and export repository metadata from GitHub.



```python
response = requests.get(url, headers=headers)

if response.status_code == 200:
    repos = response.json()
    root = ET.Element('repositories')

    for repo in repos:
        repo_elem = ET.SubElement(root, 'repository')


        url_elem = ET.SubElement(repo_elem, 'URL')
        url_elem.text = repo.get('html_url', '')

        about_elem = ET.SubElement(repo_elem, 'About')
        about_elem.text = repo.get('description') or ''

        updated_elem = ET.SubElement(repo_elem, 'Last_Updated')
        updated_elem.text = repo.get('updated_at', '')


        languages_url = repo.get('languages_url')
        languages_elem = ET.SubElement(repo_elem, 'Languages')
        if languages_url:
            lang_response = requests.get(languages_url, headers=headers)
            if lang_response.status_code == 200:
                for lang in lang_response.json():
                    lang_elem = ET.SubElement(languages_elem, 'Language')
                    lang_elem.text = lang


        commits_url = repo.get('commits_url', '').replace('{/sha}', '')
        commits_elem = ET.SubElement(repo_elem, 'Number_of_Commits')
        commit_response = requests.get(commits_url, headers=headers)
        if commit_response.status_code == 200:
            commits = commit_response.json()
            commits_elem.text = str(len(commits))
        else:
            commits_elem.text = '0'
```

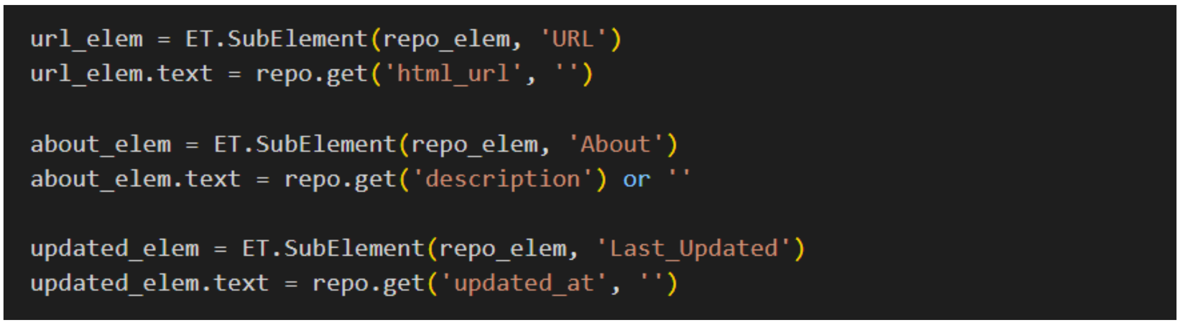
The requests.get() method is central to the program, allowing it to perform HTTP GET requests to access the GitHub REST API. This includes retrieving the user’s public repositories, fetching the programming languages used in each repository, and obtaining commit data to estimate repository activity.

When the responses are received, response.json() is used to convert the returned JSON into Python dictionaries or lists for easier data manipulation.



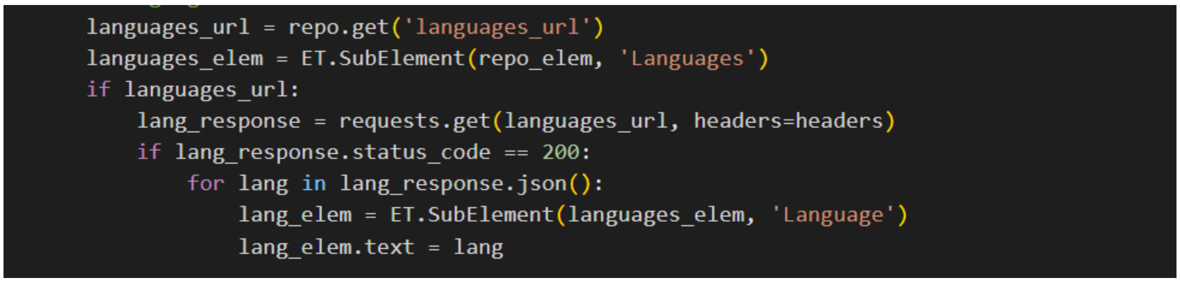
```python
for repo in repos:
    repo_elem = ET.SubElement(root, 'repository')
```

The main processing flow is handled through a structured loop that iterates over each repository. this structured loop handles each repo one by one and collects info.In this loop, the program:



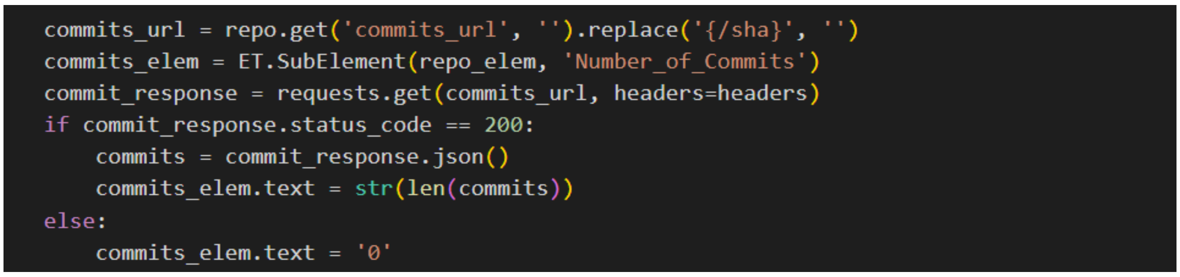
```python
url_elem = ET.SubElement(repo_elem, 'URL')
url_elem.text = repo.get('html_url', '')

about_elem = ET.SubElement(repo_elem, 'About')
about_elem.text = repo.get('description') or ''

updated_elem = ET.SubElement(repo_elem, 'Last_Updated')
updated_elem.text = repo.get('updated_at', '')
```

- Extracts basic information is like grabbing simple text from the reposition for example like: the repository URL, description (About), and last updated date



```python
    languages_url = repo.get('languages_url')
    languages_elem = ET.SubElement(repo_elem, 'Languages')
    if languages_url:
        lang_response = requests.get(languages_url, headers=headers)
        if lang_response.status_code == 200:
            for lang in lang_response.json():
                lang_elem = ET.SubElement(languages_elem, 'Language')
                lang_elem.text = lang
```

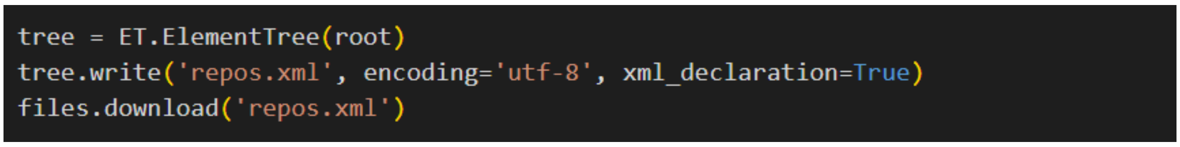
- Sends a nested request to the languages_url to fetch or get the programming languages used in the repository.



```python
commits_url = repo.get('commits_url', '').replace('{/sha}', '')
commits_elem = ET.SubElement(repo_elem, 'Number_of_Commits')
commit_response = requests.get(commits_url, headers=headers)
if commit_response.status_code == 200:
    commits = commit_response.json()
    commits_elem.text = str(len(commits))
else:
    commits_elem.text = '0'
```

- Send another request to the commits_url and count items (with the {sha} placeholder removed) to retrieve the list of commits and approximate the number of commits by counting the items in the response.

To organize the data, the program uses Python's xml.etree.ElementTree to construct a well-formed XML structure.For each repository, the program creates a <repository> section in the XML file. Inside it, it adds details like <URL> for the link, <About> for the description, <Last_Updated> for the last update time, <Languages> for the programming languages used, and <Number_of_Commits> for how many commits it has.Inside <Languages>, each programming language is listed as a <Language> item. This helps organize all the important info about each repo in a clear way.

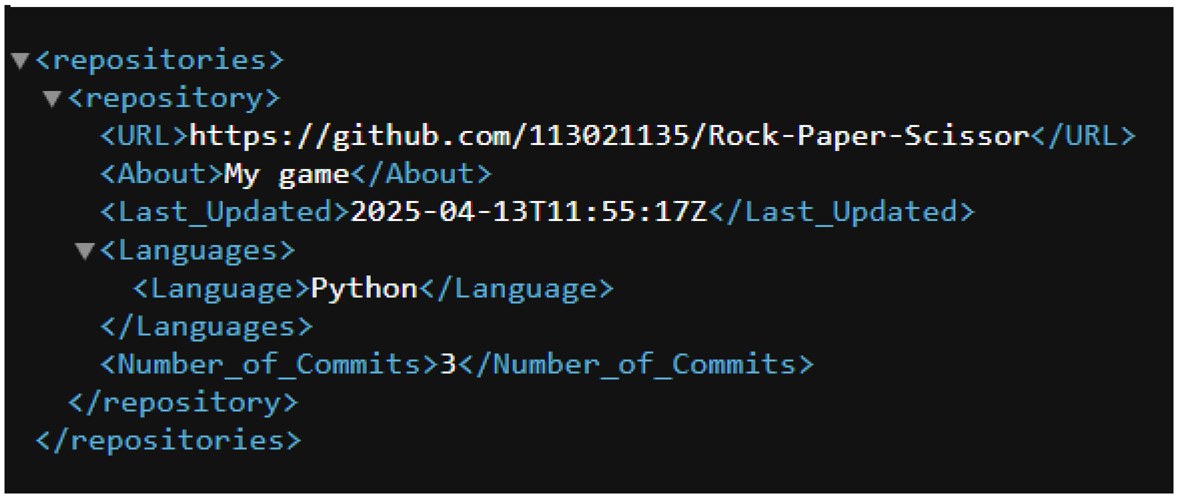


```
tree = ET.ElementTree(root)
tree.write('repos.xml', encoding='utf-8', xml_declaration=True)
files.download('repos.xml')
```

Finally, the XML tree is written to a file named repos.xml. This means the program saves the entire XML structure into a file so it can be stored, viewed, or downloaded later.and then files.download() function from Google Colab is used to enable the user to download the file to their local machine for storage or presentation.

# Chapter 3 Results

## 3.1 Result 1



```
▼<repositories>
  ▼<repository>
     <URL>https://github.com/113021135/Rock-Paper-Scissor</URL>
     <About>My game</About>
     <Last_Updated>2025-04-13T11:55:17Z</Last_Updated>
   ▼<Languages>
       <Language>Python</Language>
     </Languages>
     <Number_of_Commits>3</Number_of_Commits>
   </repository>
 </repositories>
```

A single XML file titled repos.xml was successfully generated as a result of this project. This XML file contains structured metadata for all public repositories under the GitHub account 113021135. The data includes:

- **URL**: The direct link to the repository on GitHub.
- **About**: The textual description provided by the repository owner.
- **Last_Updated**: The timestamp indicating the most recent update to the repository.
- **Languages**: A list of programming languages used in the repository, retrieved via GitHub's languages_url endpoint.
- **Number_of_Commits**: An estimate of the number of commits made to the repository. This number is based on counting the items in the API response from the commits_url, which typically contains a paginated list of recent commits.
- The XML file serves as a comprehensive and structured summary of the GitHub account's activity. It can be used for documentation, profiling, analysis, or academic purposes. The file format allows further processing in other tools, such as Excel, data parsers, or visualization frameworks.

# Chapter 4 Conclusions

In conclusion, this project shows that Python is a useful tool for getting and organizing data from the web. It helps collect GitHub reposition information automatically and saves it in a clear format. Even though the program is simple, it works well and can be improved in the future. This kind of project is a good example of how Python can be used in real-world tasks.