Cloud based IoT and Big Data platforms

Documentation

Balázs Tóth - MWZX0D

Tartalomjegyzék

1. Introduction

The project consists of two parts:

1. Data collection and processing
2. Data visualization

The application can collect data from three websites, which can be exported in json format.

1. Google Play - play.google.com
2. MarkMyProfessor - markmyprofessor.com
3. Gépigény - gepigeny.hu

These collected datas (hungarian language based only) are comments which can be useful for natural language processing tasks.

2. Requirements

2.1. Installing Python 3

1. Visit the official Python website at: `https://www.python.org/downloads/`.
2. Select the latest Python 3 version and click on the download link.
3. Run the downloaded file and follow the instructions to install Python on your system.
4. Verify the installation by running Python version in the command line or terminal:

   ```
   python3 --version
   ```

   Your system should respond with the installed Python 3 version.

2.2. Installing Tkinter

Tkinter is usually part of Python 3, so a separate installation is typically not required. If it is not installed on your system, you can install it with the following commands:

1. **For Debian/Ubuntu systems:**

```
sudo apt-get update
sudo apt-get install python3-tk
```

2. **For Red Hat/Fedora systems:**

```
sudo dnf install python3-tkinter
```

3. **For Windows systems:** Tkinter is usually automatically installed with Python.

2.3. Installing Packages

CustomTkinter

```
    pip install customtkinter
```

- **Ghostscript:** For saving diagrams.

For Windows systems:

1. Download the Ghostscript installer from the official Ghostscript website.
2. Run the installer and follow the installation instructions.
3. Add the directory where the `gs.exe` is located to the system PATH environment variable. This is usually `C:\Program Files\gs\gsX.XX\bin` (where `X.XX` is the version number).

For macOS systems:

Ghostscript can be installed using Homebrew, a package manager for macOS:

```
brew install ghostscript
```

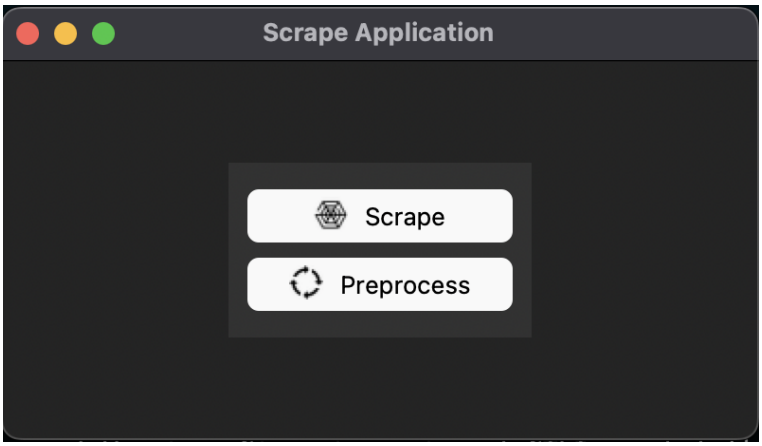
For Linux systems:

Most Linux distributions can install Ghostscript through their package manager. For example, on Ubuntu or Debian-based systems:

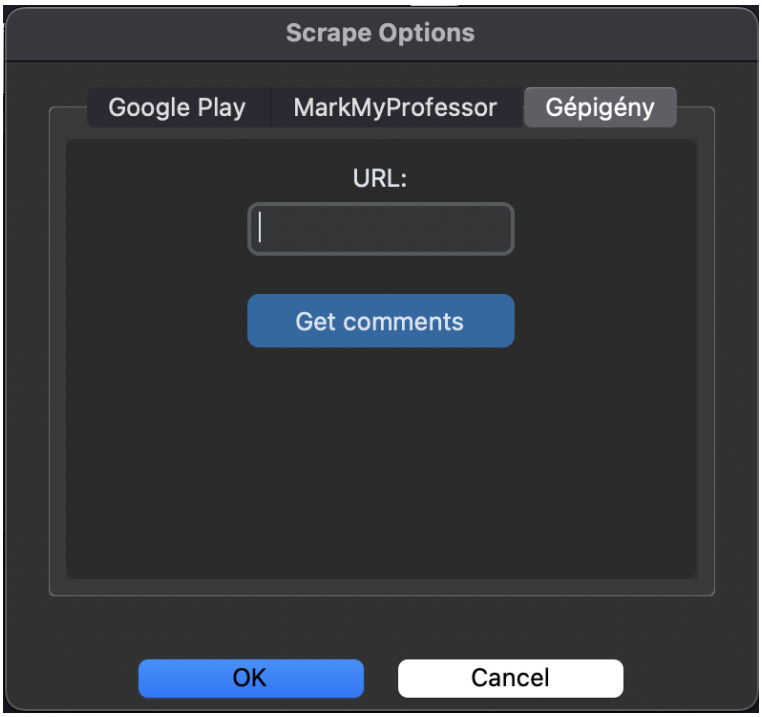
```
sudo apt-get update
sudo apt-get install ghostscript
```

3. Data collection and processing

Main screen of the application, especially **Scrape** option.



3.1. Gépigény scraping



Initializing the Scraper

The `GepigenyScraper` class is designed to facilitate scraping comments from the Gepigeny website. Its constructor (`__init__` method) takes three parameters: `base_url`, `save_callback`, and `status_label`.

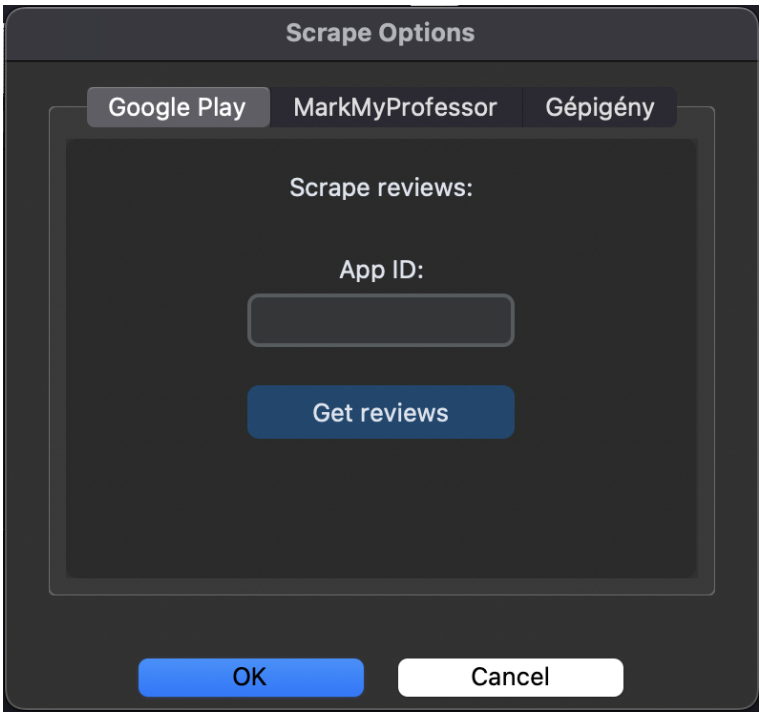
- `base_url`: This parameter represents the base URL of the Gepigeny website from which comments will be scraped.
- `save_callback`: This parameter is a function that will be called to save the scraped data.
- `status_label`: This parameter is a reference to a status label widget that will display information about the scraping process.

Scraping Gepigeny Comments

The `scrape_gepigeny_comments` method of the `GepigenyScraper` class is responsible for actually scraping comments from the Gepigeny website.

- It starts by retrieving the base URL from the `base_url` attribute of the instance.
- Then, it sends a request to the base URL to retrieve the content of the page.
- It uses BeautifulSoup to parse the HTML content and extract the total number of pages of comments available.
- After that, it iterates over each page of comments, extracting the comments and appending them to a list.
- It saves the comments to a JSON file after each page.
- It also updates the status label with information about the scraping progress.
- Finally, it prints a message indicating the completion of the scraping process along with the total number of comments scraped.

3.2. Google Play scraping



Initializing the Scraper

The `GooglePlayScraper` class is designed for scraping reviews from the Google Play Store. Its constructor (`__init__` method) takes two parameters: `app_id_entry` and `save_callback`.

- `app_id_entry`: This parameter represents the Entry widget for app ID input.
- `save_callback`: This parameter is a function that will be called to save the scraped data.

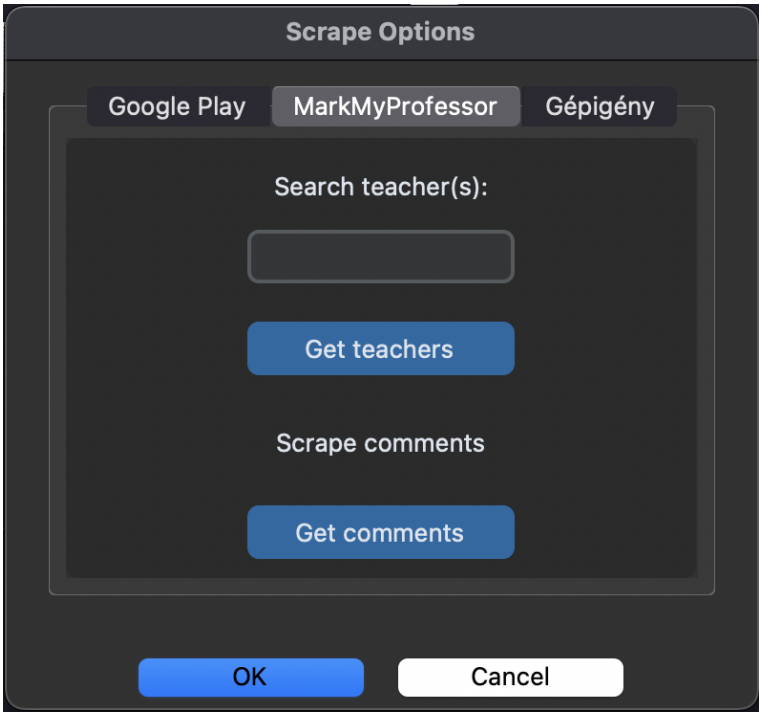
Scraping Google Play Reviews

The `scrape_reviews` method of the `GooglePlayScraper` class is responsible for scraping reviews for the specified app ID.

- It retrieves the app ID from the `app_id_entry` widget.
- Then, it calls the `reviews_all` function from the `google_play_scraper` library to scrape all reviews for the specified app ID.

- It specifies the language and country parameters as 'hu' (Hungarian) for the Hungarian version of Google Play.
- It saves the scraped reviews using the `save_callback` function.
- If an error occurs during scraping, it displays an error message using a messagebox.

3.3. MarkMyProfessor scraping



Initializing the Scraper

The `MarkMyProfessorScraper` class is designed for scraping comments and ratings for teachers from the MarkMyProfessor website. Its constructor (`__init__` method) takes two parameters: `name_entry` and `save_callback`.

- `name_entry`: This parameter represents the Entry widget for teacher name input.
- `save_callback`: This parameter is a function that will be called to save the scraped data.

Searching for Teachers

The `search_teachers` method of the `MarkMyProfessorScraper` class is responsible for searching for teachers based on the provided name query.

- It retrieves the name query from the `name_entry` widget.
- Constructs a URL for the search query to the MarkMyProfessor backend API.
- Sends a GET request to the constructed URL and retrieves the response.
- Parses the JSON response to extract information about the teachers matching the query.
- Calls the `save_callback` function to save the retrieved teacher data.

Scraping MarkMyProfessor Comments

The `scrape_markmyprofessor_comments` method of the `MarkMyProfessorScraper` class is responsible for scraping comments and ratings for teachers from the MarkMyProfessor website.

- Prompts the user to select a JSON file containing teacher data.
- Parses the selected JSON file and extracts data about the teachers.
- Iterates over each teacher in the data.
- For each teacher, retrieves the slug (unique identifier) and the last page of comments.
- Iterates over each page of comments for the teacher, scraping the comments and saving them to JSON files.

Helper Functions

The `get_markmyprofessor_teacher_get_last_page` method is a helper function that retrieves the last page of comments for a given teacher slug from the MarkMyProfessor backend API.

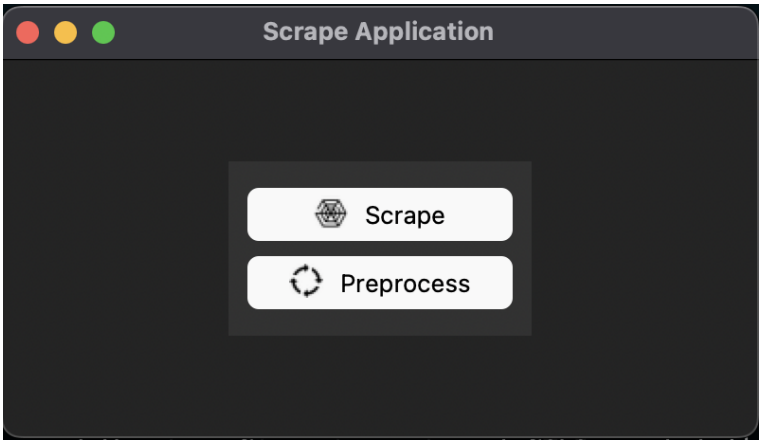
- Constructs a URL for the teacher's ratings page to the MarkMyProfessor backend API.
- Sends a GET request to the constructed URL and retrieves the response.
- Parses the JSON response to extract the last page number of comments.
- Returns the last page number.

The `scrape_markmyprofessor_comment` method is a helper function that scrapes comments for a given teacher slug and page number from the MarkMyProfessor backend API.

- Constructs a URL for the teacher's ratings page to the MarkMyProfessor backend API.
- Sends a GET request to the constructed URL and retrieves the response.
- Saves the retrieved comments to a JSON file.
- Returns the JSON response.

3.4. Preprocessing data

Select **Preprocess** option.



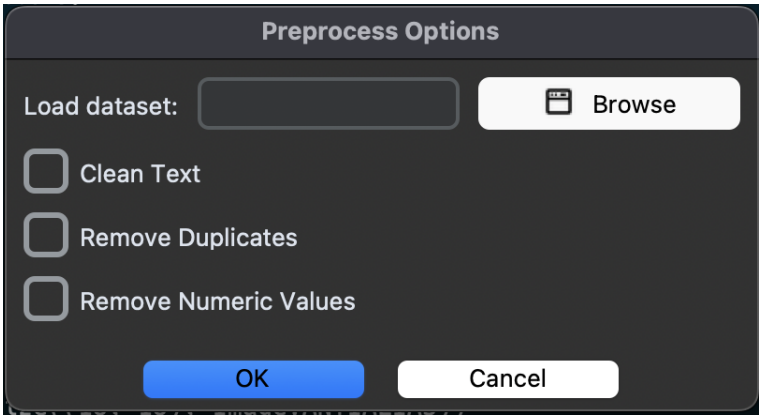
Data processing

1. Load data (csv file format)
2. Options: Clean text, Remove duplicates, Remove numeric values

Important to mention that the clean text option cleans the data with these methods:

- It converts text to lowercase to ensure consistency.
- It removes URLs using regular expressions to eliminate any web links present in the text.
- It removes mentions by matching and replacing patterns starting with '@'.
- It removes hashtags by matching and replacing patterns starting with '#'.

- It removes extra whitespace by replacing consecutive whitespace characters with a single space.
- Finally, it removes emojis from the text using the demoji library to ensure that only text data remains.



4. Data visualization

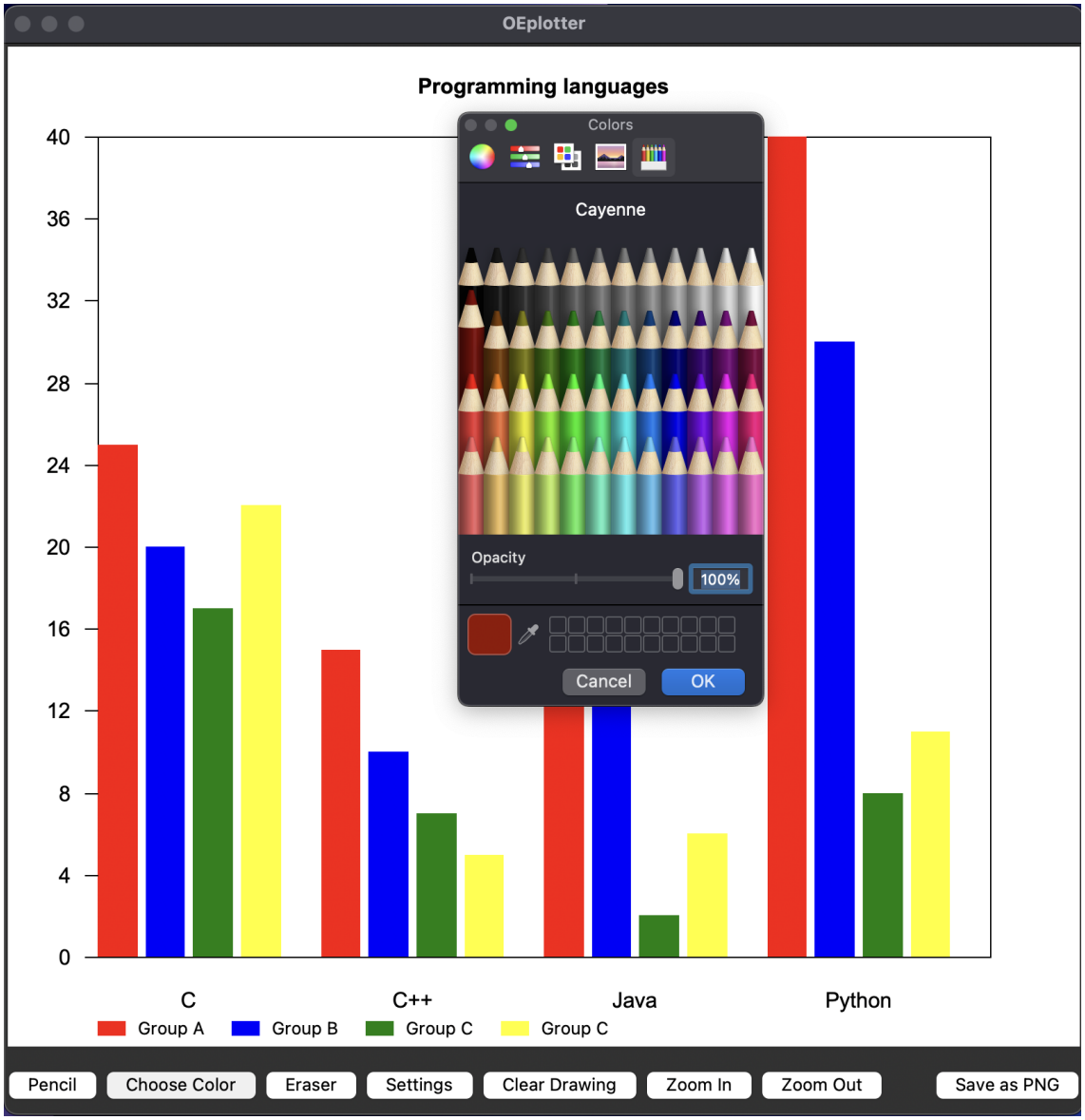
Responsive data visualization 'package' where barchart and linechart can be used where it is possible to use different groups.

Sample data

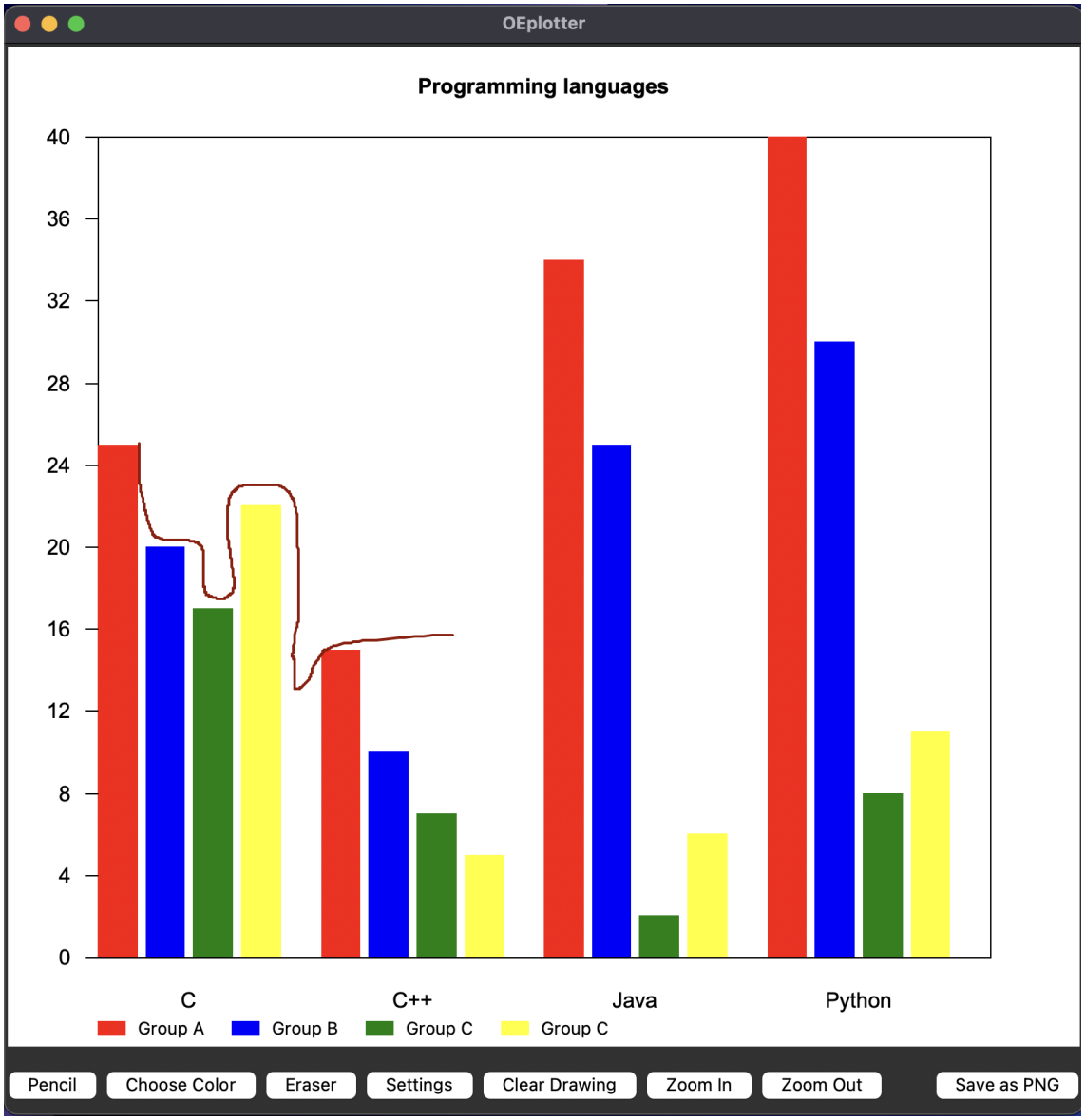
```
groups = [
    {'label': 'Group A', 'values': {'C': 25, 'C++': 15, 'Java': 34, 'Python': 40}},
    {'label': 'Group B', 'values': {'C': 20, 'C++': 10, 'Java': 25, 'Python': 30}},
    {'label': 'Group C', 'values': {'C': 17, 'C++': 7, 'Java': 2, 'Python': 8}},
    {'label': 'Group C', 'values': {'C': 22, 'C++': 5, 'Java': 6, 'Python': 11}}
]
```

4.1. Features

Color selection

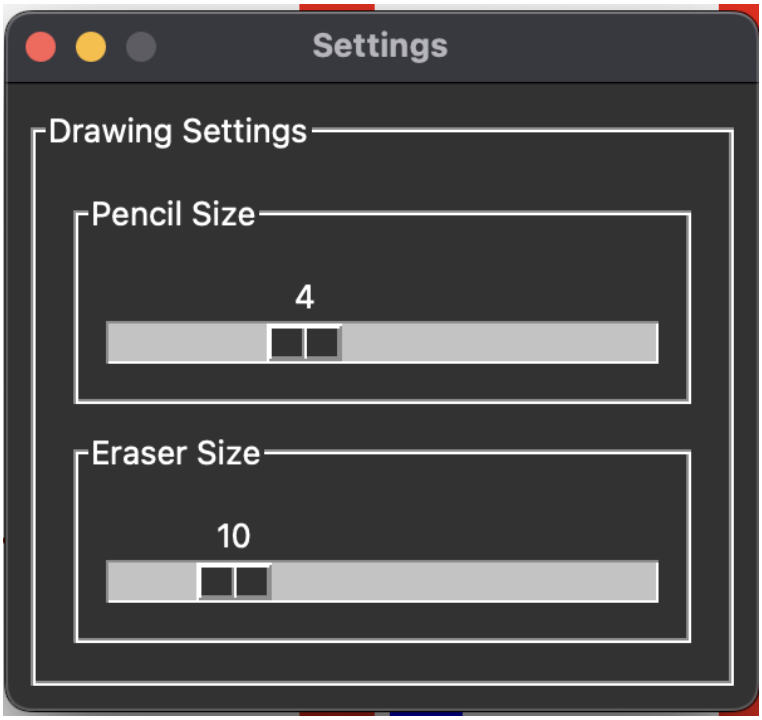


Drawing


OEplotter
Programming languages
40
36
32
28
24
20
16
12
8
4
0
C
C++
Java
Python
Group A
Group B
Group C
Group C
Pencil
Choose Color
Eraser
Settings
Clear Drawing
Zoom In
Zoom Out
Save as PNG

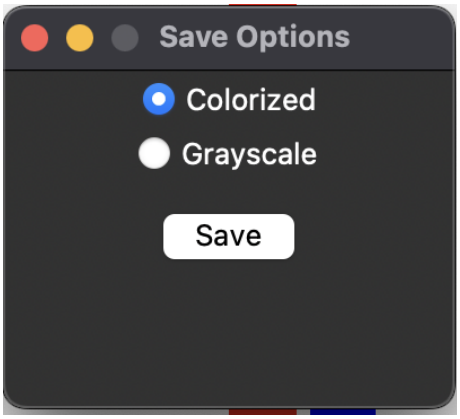
Settings

To increase or decrease the size of the pencil/eraser, choose settings option at the bottom of the application.



Save as PNG

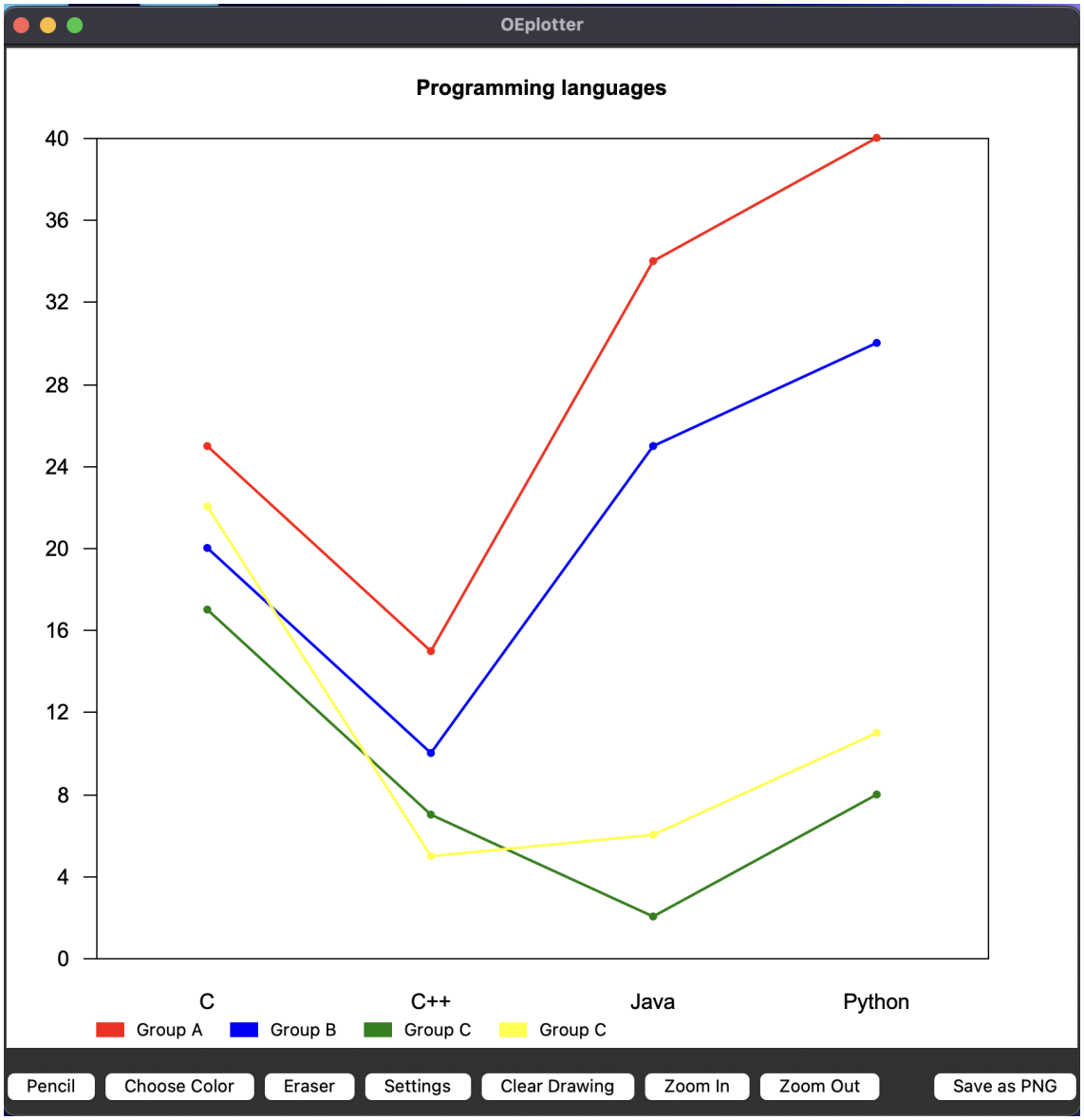
It is possible to save your plot by clicking on Save as PNG option where you can save in colorized and grayscaled mode.



Other features

- Clear drawing
- Zoom in and out

4.2. Linechart example



4.3. Barchart example