



TUNIS BUSINESS SCHOOL
UNIVERSITY OF TUNIS

Web Scraping Project:

Currency Web Scraping

Prepared by:

Mohamed Malek Kaouach

Wael Zouaoui

Alya Barcous

Fatma Jallouli

Title:

Web Scraping Project Report: Currency Comparison Calculation and Exchange Rates Stability Analysis from X-Rates Website

Introduction:

Web scraping is a technique used to extract data from websites programmatically. In this project, the focus was on scraping currency exchange rates and performing currency comparison calculations from the X-Rates website. Additionally, the project aimed to analyze the stability of exchange rates over a specific period.

Objectives:

The main objectives of the project were as follows: Scrape currency exchange rates from the X-Rates website. Perform currency comparison calculations for various currency pairs. Analyze the stability of exchange rates over a specified time frame

Methodology:

The project was implemented using the Python programming language, along with the following libraries and tools: BeautifulSoup: A Python library for web scraping. Requests: A Python library for making HTTP requests. Pandas: A data manipulation and analysis library. The following steps were followed to accomplish the objectives:

Step 1: Data Collection Used the Requests library to send an HTTP request to the X-Rates website. Retrieved the HTML content of the webpage. Utilized BeautifulSoup to parse the HTML and extract relevant data, such as currency codes, exchange rates, and dates.

Step 2: Currency Comparison Calculation Extracted the necessary data from the scraped HTML, including the exchange rates for different currency pairs. Performed currency comparison calculations by applying appropriate formulas to the exchange rates data. Created a data structure or DataFrame using the Pandas library to store the calculated currency comparison results.

Step 3: Exchange Rate Stability Analysis Determined a specific time frame for the stability analysis, such as a week, month, or year. Calculated the average exchange rate

for each currency pair over the specified time frame. Assessed the stability of exchange rates by comparing the standard deviation of the exchange rates with a predefined threshold.

Results and Findings: The web scraping project yielded the following results and findings:

- *Collected exchange rate data* for various currency pairs from the X-Rates website. Performed currency comparison calculations, allowing for easy comparison between different currencies.
- *Analyzed the stability of exchange rates* over a specified time frame.
- *Identified* currency pairs with stable exchange rates based on the calculated standard deviation.

Challenges and Limitations: During the course of the project, the following challenges and limitations were encountered:

- *Website Structure:* Changes in the structure or layout of the X-Rates website may require updates to the web scraping script.
- *Rate Limiting:* The X-Rates website may have rate limiting measures in place, which could restrict the frequency of data retrieval.

Data Accuracy: The accuracy of the scrap exchange rate data depends on the reliability and timeliness of the X-Rates website.

Conclusion: This web scraping project successfully extracted currency exchange rates from the X-Rates website and performed currency comparison calculations. The stability analysis provided insights into the stability of exchange rates over a specific time frame.

→ The project can be further extended to include additional features such as visualization of exchange rate trends or integration with other financial data sources. Web scraping offers a powerful approach to gather data from websites for analysis and decision-making purposes.

However, it is essential to consider legal and ethical aspects of web scraping and comply with website terms of service and applicable laws during any web scraping activities.

1/Clear explanation of the main concepts:

What is web scraping?

Web scraping is the process of collecting structured web data in an automated manner. It's also widely known as extracting data from websites or web data scraping.

The data can be extracted from various sources on the web such as web pages, online databases, social media platforms, and other online sources.

Do we use web scraping in our daily life?

In general, we use web scraping in a manual way.

We often copy and paste information from a website which is the same performance as any web scraper, only we manually went through the data scraping process.

Instead of extracting data all by ourselves which can take too much time, web scraping with its intelligent automation, retrieves hundreds and millions of data points from the internet.

Why do web scraping?

Web scraping can be used for different purposes. For example market research, lead generation, content aggregation, price monitoring, price intelligence, news monitoring, and competitive analysis.

In general, web scraping is used by people and businesses who want to make use of publicly available web data to generate valuable insights and make better decisions based on their findings.

Is web scraping legal or not?

However, web scraping can sometimes be illegal or unethical if it violates website terms of service, copyrights, or privacy laws. Therefore, it is important to follow some guidelines and obtain permission before scraping data from websites.

How can we do Web scraping?

Web scraping can be done in a variety of ways.

These include using online services like APIs (application programming interfaces that will allow two applications to communicate with each other), or writing your own customized web scraping code from scratch.

Many large websites, such as Google, Twitter, and Facebook have APIs that allow you to access structured data.

This is the best option, but some websites do not allow users to access large amounts of data in a structured format.

In that case, it's best to scrape the website for data using **Web Scraping**.

What are the components of web scraping?

Web scraping requires two components: a **crawler** and a **scraper**.

The crawler: it is generally called a “spider”, an artificial intelligence algorithm that searches the web for specific data by following links across the internet and searching for content. In general, we first “crawl” the web or one specific website to discover URLs which then we pass to scraping.

A scraper is a tool designed to extract data from a website. These tools vary widely in design and complexity, depending on the project and the type of website we are dealing with.

Other components needed for web scraping:

Web Scraping Engine: This is the core component of the website scraper that retrieves and extracts data from the targeted website. It can use various web scraping techniques such as Python and the library BeautifulSoup.

Data Storage: Once the data is extracted, it must be stored in a database for later analysis.

User Interface: A user interface is usually provided to allow users to interact with the scraper, set up configurations, and view the extracted data.

The functional flow of a website scraper involves several steps:

Configuration: The user specifies which website to scrape, which data to extract, and any other settings required.

Data Retrieval: The scraper retrieves the web page from the website and extracts the relevant data.

Data Storage: The extracted data is stored in a database.

Analysis: The stored data is analyzed and used to generate reports or alerts based on predefined rules.

Overall, the website scraper serves to monitor and analyze the behavior of websites for potential security threats. By automatically collecting and analyzing data from websites, it can help identify vulnerabilities and risks, and provide early warning of security breaches.

Functional flow of web scraping:

1. Identify the target website:

Identify the website where you want to scrape data from it and ensure that the website permits web scraping so you comply with ethical guidelines.

2. Determine the data to be scrapped:

After identifying the target website, determine the specific data to scrape. This could be product information, contact details, pricing, or any other type of data.

3. Choose a web scraping tool:

There are so many different web scraping tools available, both free and paid, to extract data from websites. The choice of the tool is based on the needs and expertise.

4. Develop a scraping script:

Develop a scraping script that instructs the tool on how to extract data from the website. This involves identifying the HTML elements that contain the data you want to scrape and using XPath or CSS selectors to extract the data.

5. Execute the scraping script:

After developing the scraping script, you can execute it to extract the data from the website. The script will access the website, locate the HTML elements containing the data, and extract the data into a structured format such as CSV, JSON, or a database.

7. Store and analyze the data:

The extracted data needs to be stored in a structured format for further analysis and processing.

We can use data analysis tools to analyze the data and gain insights into customer behavior, and market trends.

8. Monitor and maintain the scraping process:

Web scraping is an ongoing process, and we need to monitor and maintain the scraping process to ensure that it continues to work effectively.

We should regularly check for changes to the website structure or data format that may affect the scraping process and adjust the scraping script accordingly.

2/ Overview of the main existing solutions:

Custom scripts:

Characteristics: Custom scripts are highly customizable and can be tailored to meet the specific needs of a web scraping project. They are typically written in programming languages such as Python, JavaScript, or Ruby, and they rely on web scraping libraries such as BeautifulSoup or Scrapy.

Advantages: Custom scripts offer a high degree of flexibility and can scrape a wide variety of websites. They are also highly efficient, allowing users to scrape large amounts of data quickly.

Limitations: Writing custom scripts requires programming knowledge, which can be a barrier for non-technical users. Additionally, websites may implement anti-scraping measures that can make it difficult to scrape their data.

Browser extensions:

Characteristics: Browser extensions such as Web Scraper and Data Miner provide a user-friendly interface for scraping websites. They allow users to select data fields and use pre-built templates to extract data.

Advantages: Browser extensions are easy to use and do not require any programming knowledge. They are also highly customizable, allowing users to select which data to scrape and how to organize it.

Limitations: Browser extensions are limited in their ability to scrape complex websites. They may also be detected by anti-scraping measures and blocked by websites.

Cloud-based scraping services:

Characteristics: Cloud-based scraping services such as Scrapinghub and Octoparse allow users to scrape websites without needing to write any code. These services typically offer a user-friendly interface and can handle large-scale scraping projects.

Advantages: Cloud-based scraping services are easy to use and do not require any programming knowledge. They are also highly scalable and can handle large amounts of data.

Limitations: Cloud-based scraping services may be limited in their ability to scrape certain websites. They may also be expensive for large-scale projects.

API-based scraping:

Characteristics: API-based scraping involves using APIs provided by websites to extract data programmatically. APIs typically provide a more reliable and efficient way to extract data than web scraping.

Advantages: API-based scraping is reliable and efficient, and it does not require any web scraping knowledge. APIs also provide a more consistent way to extract data across multiple websites.

Limitations: APIs are not available for all websites, and they may require payment for access. Additionally, APIs may be limited in the amount of data they can provide.

Data marketplaces:

Characteristics: Data marketplaces such as Import.io and Diffbot provide access to pre-scraped datasets for a fee. These datasets can be downloaded in a variety of formats and used for a wide range of applications.

Advantages: Data marketplaces are fast and easy to use, and they provide access to high-quality datasets without requiring any scraping knowledge. They are also highly scalable and can handle large amounts of data.

Limitations: Data marketplaces are limited in the types of data they provide, and they may not be suitable for all applications. They can also be expensive for large-scale projects.

3/High Level Design of the proposed solution:

Introduction:

The proposed solution aims to develop a web scraping application to extract data from multiple URLs. The application will utilize the Python programming language along with the BeautifulSoup and requests libraries for web scraping. This high-level design document outlines the main components and the flow of data/messages in the proposed solution.

Components:

URL List: The solution expects a list of URLs as input. These URLs represent the web pages from which data will be scraped.

Web Scraper:

The web scraper component is responsible for making HTTP requests to the URLs, parsing the HTML content of the web pages using BeautifulSoup, and extracting the desired data. It consists of the following sub-components:

Requests:

This sub-component handles the HTTP requests to the URLs using the requests library.

HTML Parser: The BeautifulSoup library is used to parse the HTML content of the web pages and extract the required data.

Data Extraction: The web scraper component extracts data such as product reference numbers, prices, names, availability status, and features from the web pages.

Data Storage: The extracted data is stored in separate lists for each data attribute (product reference, price, name, availability, and features). These lists will hold the scraped data from each URL.

Output:

The final output of the solution is the collected data in the form of lists. The data can be further processed or stored in a database, exported to a file, or used for further analysis.

Flow of Data/Messages:

The list of URLs is provided as input to the web scraper component.

The web scraper component iterates over each URL in the list.

For each URL, the web scraper sends an HTTP request using the requests library.

If the request is successful (status code = 200), the HTML content of the web page is parsed using BeautifulSoup.

The web scraper component extracts the desired data (product reference, price, name, availability, and features) from the parsed HTML.

The extracted data is stored in separate lists (list_ref, list_price, list_name, list_avail, list_feat).

After scraping all the URLs, the collected data is available in the lists.

The lists can be used for further processing, storage, or analysis as required.

Conclusion:

The high-level design of the proposed solution outlines the key components and the flow of data/messages. By utilizing the web scraper component and the data storage mechanism, the solution enables the extraction of data from multiple URLs. This design provides a foundation for implementing the web scraping application and serves as a guide for future development and enhancements.

4/ Tools and Development phases:

Tools :

1. Programming Language: Python is a popular choice for web scraping due to its simplicity and a wide range of libraries available.

2. Web Scraping Libraries: Python offers several powerful libraries for web scraping. The most commonly used ones are:

- **BeautifulSoup:** A library for parsing HTML and XML documents, making it easy to extract data from web pages.
- **Scrapy:** A more comprehensive web scraping framework that provides a powerful set of tools for crawling and scraping websites.
- **Selenium:** A library for automating web browsers, useful for scraping dynamic websites that require JavaScript execution.
- **Requests:** A library for making HTTP requests, used for retrieving the HTML content of web pages.

Development Phases:

1. Understanding the Website: Analyze the target website's structure, identify the data you want to scrape, and inspect the HTML source code of the web pages.

2. Planning and Design: Define the scope of your web scraping project, determine the specific data to extract, and plan the logic and flow of your scraping script.

3. Setting up the Environment: Install Python and the required libraries (e.g., BeautifulSoup, Scrapy, Selenium) using a package manager like pip.

4. Writing Code:

- Use the chosen library (e.g., BeautifulSoup, Scrapy, Selenium) to make HTTP requests and retrieve the HTML content of the web pages.
- Parse the HTML content and extract the desired data using the library's features and functions.

- Write code to handle pagination, form submissions, or any other website-specific interactions required for scraping.
- Implement error handling and logging mechanisms to handle exceptions and track the scraping process.

5. Data Processing: Clean and process the extracted data as needed, such as removing unwanted characters, formatting, or transforming the data.

6. Storing or Exporting Data: Save the scraped data to a file (e.g., CSV, JSON, or a database) or export it to other systems for further analysis or use.

7. Respecting Website Policies: Ensure your web scraping activities comply with the target website's terms of service, robots.txt file, and legal considerations. Implement delays between requests to avoid overloading the server.

8. Testing and Refinement: Test your web scraping script with different scenarios, including edge cases, to ensure its reliability and accuracy. Refine and optimize the code as needed.

9. Maintenance and Updates: Regularly review and update your web scraping code to accommodate any changes in the website's structure or policies. Monitor the scraped data quality and make necessary adjustments.

Main Project goals:

Overall Process.py

```
import os
import tkinter as tk

# Get the absolute path to the directory where the main Python file is located
dir_path = os.path.abspath(os.path.join(os.path.expanduser("~"), "Desktop", "Security_Project"))

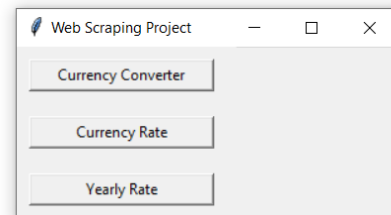
# Define the function to execute the first Python file
def execute_file1():
    os.system(f'python "{os.path.join(dir_path, "currency_converter.py")}"')

# Define the function to execute the second Python file
def execute_file2():
    os.system(f'python "{os.path.join(dir_path, "rate.py")}"')

# Define the function to execute the third Python file
def execute_file3():
    os.system(f'python "{os.path.join(dir_path, "yearly_rate.py")}"')

# Create the window and the buttons
window = tk.Tk()
window.title("Web Scraping Project")
button1 = tk.Button(window, text="Currency Converter", command=execute_file1, height=1, width=20)
button1.grid(row=0, column=0, pady=10, padx=10)
button2 = tk.Button(window, text="Currency Rate", command=execute_file2, height=1, width=20)
button2.grid(row=1, column=0, pady=10, padx=10)
button3 = tk.Button(window, text="Yearly Rate", command=execute_file3, height=1, width=20)
button3.grid(row=2, column=0, pady=10, padx=10)

# Start the main loop of the window
window.mainloop()
```



→ This code calls all 3 main functions of the program: currency converter, currency rate table generator, yearly rate.

It is used to facilitate access to all 3 operations provided in the project in just a click of the button.

An instance of the Tk class is created to initialize the main window of the application.

The window title is “Web Scraping Project” that contains 3 buttons that calls a given python file.

We also use os library in order to call and execute python files.

The project has **3 main functionalities**:

A currency converter: convert a currency to other selected currency (we have 54 available currencies). It shows also the rate of exchange and the result.

A currency rate table generator: Its main functionality is to scrap the content of table of a selected exchange currency rate compared to all other available currencies. The output will be a form of two tables: you can choose the Top10 exchange currency rate or show all available exchange currency rate.

Yearly Rate: This tool scrap the list of the yearly exchange rate of a given currency (the currency we want to convert) to the currency that we want to convert to in a given year (the user can choose the years between 2013 and our actual year.)

The provided code is a simple currency converter application built using *Python and Tkinter*, a standard GUI toolkit.

The application allows users to select a base currency, a target currency, and an amount to convert. It then retrieves the exchange rate from the X-Rates website and calculates the converted amount.

Currency_converter.py

```
import requests
from bs4 import BeautifulSoup
from tkinter import *

window = Tk()
window.title("Currency Converter")

convert = "USD"
to_convert = "EUR"
amount = 1

def list_of_currency_names():
    currency_list_url = f"https://www.x-rates.com/calculator/?from={convert}&to={to_convert}&amount={1}"
    currency_list = requests.get(currency_list_url).text
    currency_list_page = BeautifulSoup(currency_list, "html.parser")

    currencies = currency_list_page.find_all(class_="currencyList currencycalculator")
    short_name = []
    currency_name = ""

    for currency in currencies:
        currency_name += currency.text
        currency_symbols = currency.find_all('a')
        for symbol in currency_symbols:
            symbol_text = symbol.get("href")
            short_name.append(symbol_text.split("=")[1])
    currency_name_list = currency_name.split("\n")

    full_list = []
    for i, x in zip(short_name, currency_name_list):
        full_list.append(f"{i} - {x}")

    return full_list

short_name = list_of_currency_names()
short_name.sort()
```

```

def generate(amount):
    result = am.get(1.0,"end")
    url = f"https://www.x-rates.com/calculator/?from={var1.get()[ :3]}&to={var2.get()[ :3]}&amount={int(result)}"
    results = requests.get(url).text
    page = BeautifulSoup(results, "html.parser")
    prices = page.find_all(class_="ccOutputRs1t")
    for price in prices:
        price_text= price.text
        print(price.text)
    l3.config(text=price_text)
    price_text = price_text.split(" ")[0].replace(",","")
    price_text = str(round(float(price_text)/int(result),4))
    l4.config(text=price_text)
    print(round(float(price_text)/int(result),4))

convert = Label(text="Convert",height=3,width=20)
convert.grid(row=0,column=0)
var1 = StringVar()
OptionMenu(window,var1,*short_name).grid(row=0,column=1)
to_convert = Label(text="Convert to",height=3,width=20).grid(row=1,column=0)
var2 = StringVar()
OptionMenu(window,var2,*short_name).grid(row=1,column=1)
am = Text(window,height=1,width=10)
am.grid(row=2,column=1)
Label(text="Amount").grid(row=2,column=0, pady=15)
b3 = Button(text="Generate", command=lambda: generate(2), height=1, width=20)
b3.grid(row=3, column=0, columnspan=2, padx=10, pady=20)
l3 = Label(text="",width= 20,height=3)
l3.grid(row=5,column=0)
Label(text="Unit Price").grid(row=4,column=1)
Label(text="Convert Result").grid(row=4,column=0)
l4 = Label(text="",width=20,height=3)
l4.grid(row=5,column=1)
|
window.mainloop()

```

Here's a brief description of the code:

Importing Required Libraries: The requests library is imported for making HTTP requests to fetch web page data. The BeautifulSoup class from the bs4 module is imported for parsing HTML content.

Creating the GUI: An instance of the Tk class is created to initialize the main window of the application. The window title is set to "Currency Converter".

Function Definitions: The list_of_currency_names() function scrapes the X-Rates website to obtain a list of currency names and their respective short codes.

The generate() function is called when the user clicks the "Generate" button. It retrieves the selected currencies and the amount to convert, sends a request to X-Rates, and extracts the converted amount from the HTML response.

The converted amount is displayed in a label widget (l3) and the unit price in another label widget (l4).

GUI Layout: Various labels, dropdown menus, and an entry field are created using Tkinter widgets and placed on the window using the grid() method.

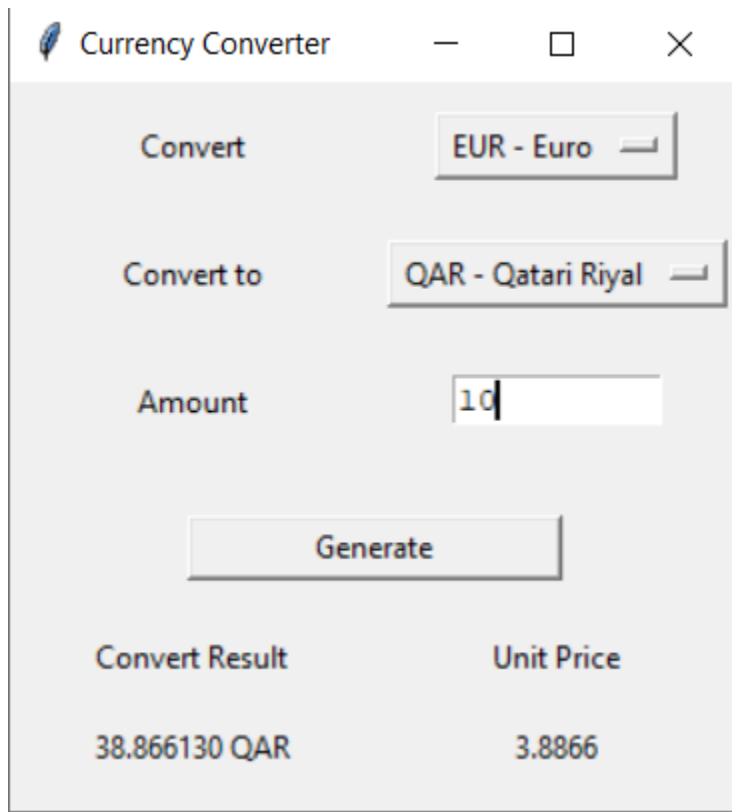
The "Generate" button is created and assigned the generate() function as its command.

Main Event Loop: The window.mainloop() function is called to start the main event loop of the application, which listens for user interactions and updates the GUI accordingly.

Overall, this code provides a basic interface for a currency converter, using web scraping techniques to fetch the exchange rates from the X-Rates website and displaying the converted results to the user.

The script imports necessary libraries: requests for making HTTP requests, BeautifulSoup from the bs4 module for parsing HTML content, Tk from the tkinter module for creating the GUI.

Example:



Convert	EUR - Euro
Convert to	QAR - Qatari Riyal
Amount	10
Generate	
Convert Result	Unit Price
38.866130 QAR	3.8866

Rate.py

This code takes as an input the currency that you want to convert and gives you two choices of table generation:

Generating the Top10 actual currency exchange rate of the chosen currency

Or Generating all actual currency exchange rate of the chosen currency

The code doesn't change in terms of finding the list of the currency names. We just change the class name where to find the currency names.

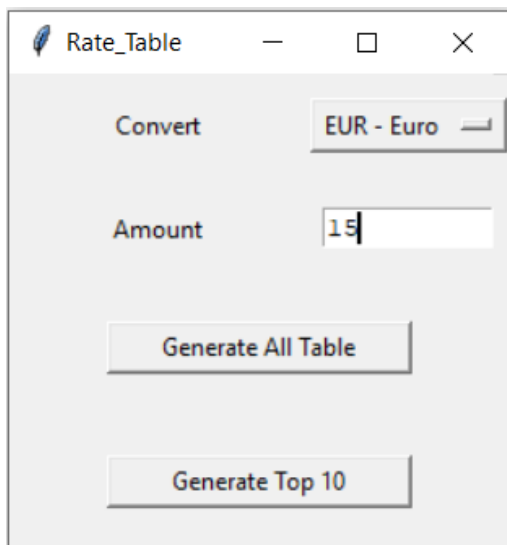
- The function **generate1(amount)**: it shows the total rate table.

It searches for the total rate table in the website and gets the cells' content and put it in an excel file as an output with headers that we insert (`headers = ["Currency", "Rate", "Change"]`)

The function **generate2(amount)**: it shows the top 10 rate table.

It searches for the top10 rate table in the website by specifying its class and gets the cells' content and put it in an other excel file as an output with headers that we insert (`headers = ["Currency", "Rate", "Change"]`)

→We use pandas for data manipulation such the process of creating excel tables in `generate1` and `generate2` functions.



The screenshot shows a window titled "Rate_Table" with a standard Windows title bar (minimize, maximize, close buttons). The interface is a simple GUI for a currency converter. It features a "Convert" label next to a dropdown menu currently showing "EUR - Euro". Below this is an "Amount" label next to a text input field containing the number "15". At the bottom of the window, there are two buttons: "Generate All Table" and "Generate Top 10".

```

import requests
from bs4 import BeautifulSoup
from tkinter import *
import requests
import pandas as pd

window = Tk()
window.title("Rate_Table")

convert = "USD"
to_convert = "EUR"
amount = 1

def list_of_currency_names():
    currency_list_url = f"https://www.x-rates.com/table/?from={convert}&amount={1}"
    currency_list = requests.get(currency_list_url).text
    currency_list_page = BeautifulSoup(currency_list, "html.parser")

    currencies = currency_list_page.find_all(class_="currencyList ratestable")
    short_name = []
    currency_name = ""

    for currency in currencies:
        currency_name += currency.text
        currency_symbols = currency.find_all('a')
        for symbol in currency_symbols:
            symbol_text = symbol.get("href")
            short_name.append(symbol_text.split("=")[1])
    currency_name_list = currency_name.split("\n")

    full_list = []
    for i, x in zip(short_name, currency_name_list):
        full_list.append(f"{i} - {x}")

    return full_list

short_name = list_of_currency_names()
short_name.sort()

def generate1(amount):
    result = am.get(1.0, "end")
    url = f"https://www.x-rates.com/table/?from={var1.get()[0:3]}&amount={int(result)}"
    response = requests.get(url)
    html_content = response.content
    # Parse the HTML content using BeautifulSoup
    soup = BeautifulSoup(html_content, "html.parser")

    # Find the table on the page and extract the data
    table = soup.find("table", class_="tablesorter ratesTable")
    table_rows = table.find_all("tr")

    data = []
    for row in table_rows:
        cells = row.find_all("td")
        row_data = []
        for cell in cells:
            row_data.append(cell.text.strip())
        if row_data:
            data.append(row_data)

    # Convert the data to a pandas DataFrame
    headers = ["Currency", "Rate", "Change"]
    df = pd.DataFrame(data, columns=headers)

    # Print the DataFrame
    print(df)

    # Save the DataFrame to an Excel file
    df.to_excel("Total_rates_table.xlsx", index=False)

    # Print a success message
    print("Rates table saved to rates_table.xlsx!")

```

```

def generate2(amount):
    result = am.get(1.0,"end")
    url = f"https://www.x-rates.com/table/?from={var1.get()[ :3]}&amount={int(result)}"
    response = requests.get(url)
    html_content = response.content
    # Parse the HTML content using BeautifulSoup
    soup = BeautifulSoup(html_content, "html.parser")

    # Find the table on the page and extract the data
    table = soup.find("table", class_="ratesTable")
    table_rows = table.find_all("tr")

    data = []
    for row in table_rows:
        cells = row.find_all("td")
        row_data = []
        for cell in cells:
            row_data.append(cell.text.strip())
        if row_data:
            data.append(row_data)

    # Convert the data to a pandas DataFrame
    headers = ["Currency", "Rate", "Change"]
    df = pd.DataFrame(data, columns=headers)

    # Print the DataFrame
    print(df)

    # Save the DataFrame to an Excel file
    df.to_excel("Top10_rates_table.xlsx", index=False)

    # Print a success message
    print("Rates table saved to rates_table.xlsx!")

convert = Label(text="Convert",height=3,width=20)
convert.grid(row=0,column=0)

var1 = StringVar()
OptionMenu(window,var1,*short_name).grid(row=0,column=1)

am = Text(window,height=1,width=10)
am.grid(row=2,column=1)

Label(text="Amount").grid(row=2,column=0, pady=15)

b3 = Button(text="Generate All Table", command=lambda: generate1(2), height=1, width=20)
b3.grid(row=3, column=0, columnspan=2, padx=10, pady=20)

b4 = Button(text="Generate Top 10", command=lambda: generate2(2), height=1, width=20)
b4.grid(row=4, column=0, columnspan=2, padx=10, pady=20)

window.mainloop()

```

The script creates a window for the GUI and sets its title as "Rate_Table".

The script defines some variables:

- **convert:** The base currency to convert from (set as "USD").

- ***to_convert***: The currency to convert to (set as "EUR").
- ***amount***: The amount of currency to convert (set as 1).

The script defines a function called `list_of_currency_names()` that retrieves a list of currency names from a website. It uses the `convert` variable to construct a URL, sends a GET request to the URL, parses the HTML content using BeautifulSoup, and extracts the currency names from the page. The function returns a list of currency names.

The script calls the `list_of_currency_names()` function to get a list of currency names and assigns it to the variable `short_name`. It then sorts the list.

The script defines two functions: *generate1(amount)* and *generate2(amount)*.

→ These functions are called when the corresponding buttons in the GUI are clicked.

The functions retrieve currency rate data from a website, parse the HTML content using BeautifulSoup, extract the rate data from a table on the page, convert the data to a pandas DataFrame, print the DataFrame, save the DataFrame to an Excel file, and print a success message.

The script creates GUI elements using Tkinter:

Labels, buttons, and an option menu for selecting currencies and entering an amount to convert.

The "Generate All Table" button calls the `generate1(amount)` function when clicked.

The "Generate Top 10" button calls the `generate2(amount)` function when clicked.

The script enters the main event loop of the GUI, which allows the user to interact with the interface.

Overall, the code creates a GUI that fetches currency rate data from a website, allows the user to select currencies and enter an amount to convert, and generates tables of currency rates in a DataFrame format that can be printed and saved to an Excel file.

Output example:

Rate_Table

Convert

EUR - Euro

Amount

10

Generate All Table

Generate Top 10



Total_rates_table



Top10_rates_table

TOP10 Rate Table:

	A	B	C	D
1	Currency	Rate	Change	
2	US Dollar	10.678677	0.936446	
3	British Pou	8.633060	1.158338	
4	Indian Rup	882.78455	0.011328	
5	Australian	16.478125	0.606865	
6	Canadian I	14.566521	0.686506	
7	Singapore	14.467989	0.691181	
8	Swiss Fran	9.724817	1.028297	
9	Malaysian	49.375235	0.202531	
10	Japanese Y	1491.7531	0.006704	
11	Chinese Yu	75.882422	0.131783	
12				

	A	B	C	D
1	Currency	Rate	Change	
2	Argentine Peso	2550.598105	0.003921	
3	Australian Dollar	16.478125	0.606865	
4	Bahraini Dinar	4.015183	2.490547	
5	Botswana Pula	147.416900	0.067835	
6	Brazilian Real	53.781458	0.185938	
7	Bruneian Dollar	14.467989	0.691181	
8	Bulgarian Lev	19.558300	0.511292	
9	Canadian Dollar	14.566521	0.686506	
10	Chilean Peso	8594.132659	0.001164	
11	Chinese Yuan Ren	75.882422	0.131783	
12	Colombian Peso	46959.969933	0.000213	
13	Czech Koruna	237.042438	0.042187	
14	Danish Krone	74.484050	0.134257	
15	Hong Kong Dollar	83.639522	0.119561	
16	Hungarian Forint	3704.963635	0.002699	
17	Icelandic Krona	1493.075713	0.006698	
18	Indian Rupee	882.784557	0.011328	
19	Indonesian Rupia	160276.601223	0.000062	
20	Iranian Rial	452055.806710	0.000022	
21	Israeli Shekel	39.691333	0.251944	
22	Japanese Yen	1491.753143	0.006704	
23	Kazakhstani Tenge	4782.700658	0.002091	
24	South Korean Won	14167.281518	0.000706	
25	Kuwaiti Dinar	3.284756	3.044366	
26	Libyan Dinar	51.485278	0.194230	
27	Malaysian Ringgit	49.375235	0.202531	
28	Mauritian Rupee	486.590492	0.020551	
29	Mexican Peso	188.653263	0.053007	
30	Nepalese Rupee	1413.117379	0.007077	
31	New Zealand Doll	17.797763	0.561868	
32	Norwegian Krone	120.146694	0.083232	
33	Omani Rial	4.111293	2.432325	
34	Pakistani Rupee	3047.263197	0.003282	
35	Philippine Peso	600.035210	0.016666	
36	Polish Zloty	45.344350	0.220535	
37	Qatari Riyal	38.870386	0.257265	
38	Romanian New Le	49.690724	0.201245	
39	Russian Ruble	861.088557	0.011613	
40	Saudi Arabian Riy	40.045040	0.249719	
41	Singapore Dollar	14.467989	0.691181	
42	South African Ranc	211.317156	0.047322	
43	Sri Lankan Rupee	3147.247230	0.003177	
44	Swedish Krona	116.630270	0.085741	

Total Rate Table:

Yearly rate : The provided code is a modified version of the previous currency converter application, with the addition of a feature that allows the user to retrieve monthly average exchange rates between two currencies for a specified year from the X-Rates website.

```
import requests
from bs4 import BeautifulSoup
from tkinter import *
import tkinter as tk
from tkinter import messagebox
import pandas as pd

window = Tk()
window.title("Monthly Average")

convert = "USD"
to_convert = "EUR"
amount = 1

def list_of_currency_names():
    currency_list_url = f"https://www.x-rates.com/average/?from={convert}&to={to_convert}&amount={1}&year=2023"
    currency_list = requests.get(currency_list_url).text
    currency_list_page = BeautifulSoup(currency_list, "html.parser")

    currencies = currency_list_page.find_all(class_="currencyList monthlyaverage")
    short_name = []
    currency_name = ""

    for currency in currencies:
        currency_name += currency.text
        currency_symbols = currency.find_all('a')
        for symbol in currency_symbols:
            symbol_text = symbol.get("href")
            short_name.append(symbol_text.split("=")[1])
    currency_name_list = currency_name.split("\n")

    full_list = []
    for i, x in zip(short_name, currency_name_list):
        full_list.append(f"{i} - {x}")

    return full_list

short_name = list_of_currency_names()
short_name.sort()
```

```
def generate(amount,year):
    result = am.get(1.0,"end")

    url = f"https://www.x-rates.com/average/?from={var1.get()[ :3]}&to={var2.get()[ :3]}&amount={int(result)}&year={int(year)}"
    response = requests.get(url)
    html_content = response.content

    # Parse the HTML content using BeautifulSoup
    soup = BeautifulSoup(html_content, "html.parser")

    # Find the unordered list on the page and extract the data
    ul = soup.find('ul', {'class': 'OutputLinksAvg'})
    lis = ul.find_all('li')

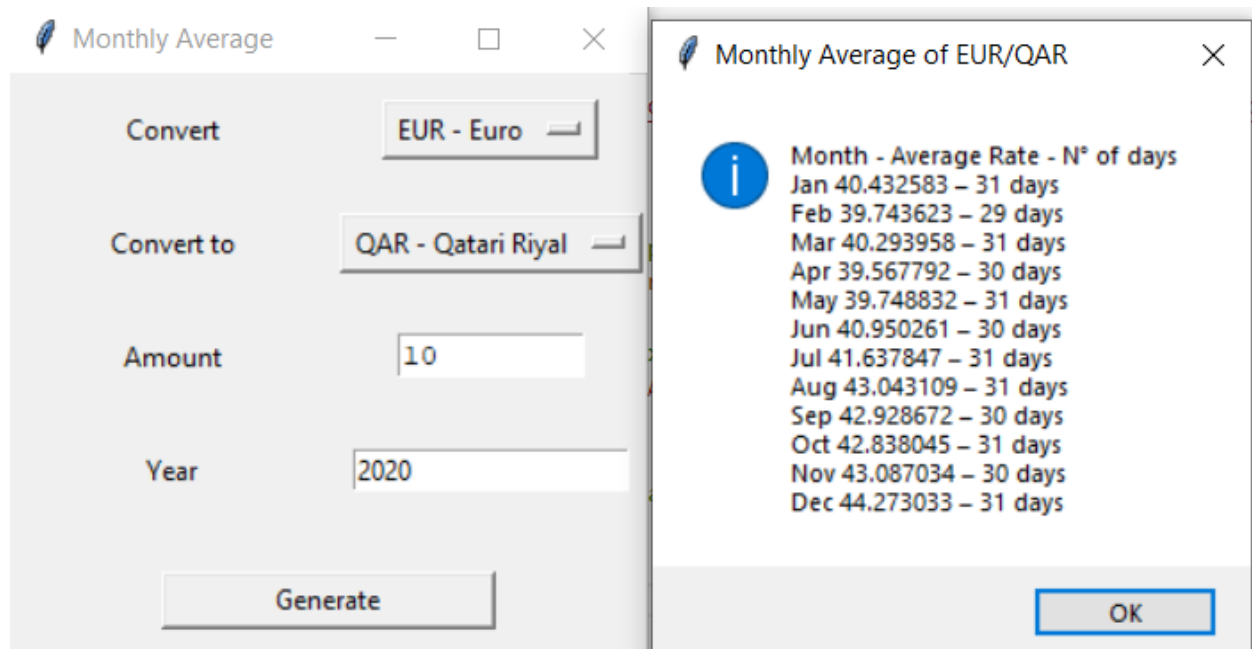
    # Create a string variable to store the data
    output_text = "Month - Average Rate - N° of days \n"
    for li in lis:
        output_text += li.text.strip() + "\n"

    # Show the data in a message box
    messagebox.showinfo(f"Monthly Average of {var1.get()[ :3]}/{var2.get()[ :3]}", output_text)

convert = Label(text="Convert",height=3,width=20)
convert.grid(row=0,column=0)
var1 = StringVar()
OptionMenu(window,var1,*short_name).grid(row=0,column=1)
to_convert = Label(text="Convert to",height=3,width=20).grid(row=1,column=0)
var2 = StringVar()
OptionMenu(window,var2,*short_name).grid(row=1,column=1)
am = Text(window,height=1,width=10)
am.grid(row=2,column=1)
Label(text="Amount").grid(row=2,column=0, pady=15)
Label(text="Year").grid(row=3, column=0, pady=15)
year_entry = Entry(window)
year_entry.grid(row=3, column=1)
b3 = Button(text="Generate", command=lambda: generate(2,year_entry.get()), height=1, width=20)
b3.grid(row=4, column=0, columnspan=2, padx=10, pady=20)

window.mainloop()
```

Example:



Monthly Average

Convert

EUR - Euro

Convert to

QAR - Qatari Riyal

Amount

10

Year

2023

Generate

Monthly Average of EUR/QAR

i

Month - Average Rate - N° of days
Jan 39.243020 – 31 days
Feb 38.988265 – 28 days
Mar 38.966596 – 31 days
Apr 39.957762 – 30 days
May 39.657088 – 31 days

OK