

|  |
| --- |
| **Web scraper**  **Harry Fisher**  CMP320: Advanced ethical hacking  2023/24 |

*Note that Information contained in this document is for educational purposes.*

Abstract

This project aimed to automate the process of extracting data from web pages and generating SQL insert statements for a mySQL database. Manual data extraction from web sources is very labor-intensive and error-prone, posing challenges to efficient information management. The objective was to develop a Python-based solution using web scraping techniques to streamline the extraction of data from web pages and insert it into a database. By automating these tasks, the project would help to save some effort in making insert statements.

The project uses Python libraries such as BeautifulSoup and Pandas to implement automated data extraction and SQL insert statement generation. HTTP requests were made to target web pages to retrieve their HTML content, which was then parsed using BeautifulSoup to extract relevant HTML <TABLE> tags. Extracted data was formatted and cleansed then processed and formatted using Pandas DataFrame, ensuring compatibility with the database schema. SQL insert statements were generated based on the extracted data. The entire process was all done within a Python program, providing a seamless and efficient solution for web scraping and database population tasks.

The implementation of the automated web scraping and SQL insert statement generation solution proved to be effective in streamlining data extraction and making insert statments. By eliminating the need for everything to be done by hand, the project significantly reduced the time and effort required for these tasks. However, challenges such as handling dynamic web content and ensuring data accuracy in diverse web page structures were encountered during the development process. Future iterations of the project could explore advanced techniques for handling such challenges, thereby improving the robustness and reliability of the automated solution. Overall, the project demonstrated the feasibility and benefits of automating web scraping and database filling, on top an issue with automatically inserting into the database was encountered so there is two versions of code talked about, version 7.4 and version 8 (or dev build). Version 7.4 works and auto-generates the insert statements, version 8 was taking the next step in getting them to auto-insert into the database but under given time constraints couldn’t be fully made and fixed

Contents

[1 Introduction 1](#_Toc165352859)

[1.1 Background 1](#_Toc165352860)

[1.2 Aim 1](#_Toc165352861)

[2 Procedure 3](#_Toc165352862)

[2.1 overview 3](#_Toc165352863)

[2.2 libraries 3](#_Toc165352864)

[2.3 limitations 4](#_Toc165352865)

[2.4 dev build (non functional) 5](#_Toc165352866)

[3 Discussion 7](#_Toc165352867)

[3.1 General Discussion 7](#_Toc165352868)

[3.2 Legal and Ethical Considerations 7](#_Toc165352869)

[3.3 testing 8](#_Toc165352870)

[3.4 problems 9](#_Toc165352871)

[3.5 Future Work 9](#_Toc165352872)

[4 References 11](#_Toc165352873)

[Appendices 12](#_Toc165352874)

[Appendix A working code 12](#_Toc165352875)

[Appendix B dev build 13](#_Toc165352876)

[Appendix C code output 18](#_Toc165352877)

# Introduction

## Background

Web scraping technology functions by sending HTTP requests to web pages, retrieving their HTML content, and parsing the content to extract desired data elements. Tools like BeautifulSoup and Scrapy provide frameworks for navigating through web page structures and extracting data based on specified criteria.

Despite its potential benefits, web scraping also presents challenges and ethical considerations. One of the primary issues is the legality and ethicality of scraping data from websites without permission. Many websites have terms of service or robots.txt files that explicitly prohibit web scraping activities. Additionally, the dynamic nature of web content, including JavaScript-rendered pages and CAPTCHA protections, complicates the scraping process. These challenges hinder the scalability and reliability of web scraping initiatives, leading to concerns about data quality and compliance with regulations. According to a survey conducted by Distil Networks, 46% of web traffic consists of bots, with 20% of these bots being malicious in nature.

(https://cdn2.hubspot.net/hubfs/258389/White\_Papers/WP\_2016\_Bad\_Bot\_Landscape\_Report-1-1-1.pdf, 2016)

This prevalence of bot activity underscores the need for organizations to implement measures to protect their websites from unwanted scraping activities. Moreover, incidents of legal disputes and enforcement actions related to web scraping highlight the contentious nature of this technique. As organizations increasingly rely on web scraping for competitive intelligence and data acquisition, addressing the legal, ethical, and technical challenges.

## Aim

1. **Web Scraping Program Development:**
   * Create a robust and versatile Python program using libraries like BeautifulSoup and Pandas to scrape data from websites.
2. **SQL Insert Statement Generation:**
   * Design a mechanism within the program to automatically generate SQL insert statements based on the extracted data.
   * Ensure that the generated insert statements are formatted correctly and compatible with MySQL databases.
3. **Legal and Ethical Investigation:**
   * Conduct research into the legal landscape surrounding web scraping, including relevant laws and regulations.
   * Explore the ethical considerations associated with web scraping, such as respect for website terms of service and data privacy concerns.
   * Evaluate the potential risks and consequences of web scraping activities in violation of website policies and legal requirements.

# Procedure

## overview

In the development process of the web scraping script, version 7.4, several key steps were taken to achieve the desired functionality. Initially, the script was designed to extract specific data from a table on a provided web page and prepare it for future integration into a database. To accomplish this, the script utilized the requests library to make HTTP requests to the target web page and the BeautifulSoup library to parse the HTML content and locate the desired table. This approach facilitated the extraction of structured data from the HTML document, enabling further processing.

A screenshot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

Moreover, the script employed the pandas library to convert the extracted HTML table data into a DataFrame, which provided a structured format for easier manipulation and analysis. Subsequently, the data underwent formatting and cleaning operations within the sortDataFrame method to ensure consistency and remove unnecessary characters. This step involved splitting the titles and removing exes information, such as brackets and quotation marks, so it would be easier to make the insert statement

A computer code with text

Description automatically generated

Furthermore, a temporary test method, printDataFrame, was implemented to verify the extracted data and visualize it in the form of SQL insert statements. This allowed for quick validation of the data extraction process and ensured that the extracted information was accurately captured and formatted for database insertion. However, despite the successful extraction and formatting of data, the script lacked functionality for database integration, as it only printed the insert statements instead of executing them against a database.

A computer screen shot of a computer code

Description automatically generated

In summary, the development of version 7.4 of the web scraping script involved several steps, including data extraction, formatting, and validation. While the script successfully extracted and formatted the desired data from the target web page, it lacked functionality for database integration, which would be a key focus for future iterations of the script.

In the development of version 8 of the web scraping script, notable enhancements and considerations were incorporated to advance its functionality and usability. The script retained its primary objective of extracting data from a provided web page, specifically targeting tables for subsequent processing. However, an important addition in this version was the aspiration to automate database integration, a feature absent in the previous iteration. The script now aimed to generate SQL insert statements for future database insertion, signifying a significant advancement towards achieving a fully automated data pipeline.

To facilitate this database integration, the script utilized the mysql.connector library to establish a connection to a MySQL database. This connection allowed for the execution of SQL commands, enabling the insertion of extracted data into the database tables. Additionally, the script implemented methods

A computer code with text

Description automatically generated with medium confidence

such as makeDataBase and addData to manage database interactions, including schema creation and data insertion processes. However, despite these advancements, the database integration functionality remained a work-in-progress, with the script currently limited to printing SQL insert statements instead of executing them against a live database. It also had a function to make a sql database built in from a pre made script it would read from.

A close-up of text

Description automatically generated

A screenshot of a computer

Description automatically generated

## libraries

Pandas is a Python library for data manipulation and analysis. It provides easy-to-use data structures and tools for reading, cleaning, transforming, and analyzing structured data. With Pandas, you can handle tasks like importing data from different sources, cleaning messy datasets, performing data aggregation, and grouping.

(Pandas, 2024)

Requests is a Python library for making HTTP requests. It simplifies the process of sending HTTP requests to web servers and handling their responses. With Requests, you can easily perform tasks like fetching web pages, interacting with web APIs, and downloading files from the internet. It provides a simple and intuitive interface for making various types of requests (GET, POST, PUT, DELETE, etc.) and allows you to customize requests with headers, parameters, and authentication credentials. Requests is widely used in web scraping, web development, and data fetching applications for its ease of use and flexibility.

(PyPi, 2024)

Beautiful Soup is a Python library for parsing HTML and XML documents. It provides tools for navigating, searching, and modifying parsed HTML/XML content, making it easier to extract specific data from web pages. Beautiful Soup handles poorly formatted HTML/XML gracefully, allowing you to extract information even from messy web pages. It is commonly used in web scraping projects to extract structured data from web pages by traversing the HTML/XML tree and locating elements based on tags, attributes, and CSS selectors. Overall, Beautiful Soup simplifies the process of extracting data from web pages and is widely used in web scraping and data extraction tasks.

(Beautiful Soup Documentation — Beautiful Soup 4.4.0, 2024)

## limitations

the script heavily relies on the HTML structure of the target website, particularly when locating specific elements such as tables. If the HTML structure of the website undergoes significant changes, the script may fail to locate the desired data elements accurately, resulting in parsing errors or missing data.

Moreover, the script assumes that the desired data is presented in the form of a table with a specific class name ("tracklist"). This is very specialized for the site used but not for many others

Another limitation of the script is its lack of robust error handling mechanisms. In the event of exceptions or edge cases, such as failed HTTP requests or unexpected HTML content from the target website, the script will crash. Implementing comprehensive error handling would further the script's reliability in handling unexpected scenarios.

continuing, while the script generates SQL insert statements for the database, it does not include functionality for directly interacting with a MySQL database. Users must manually execute the generated insert statements. The version 8 was set to fix this but was never fully implemented

Lastly, the script does not incorporate mechanisms to address the legal and ethical considerations surrounding web scraping activities. Scraping data from websites without permission may violate website terms of service or infringe upon data privacy rights, potentially leading to legal consequences and an expulsion from the university. Users must exercise caution and adhere to applicable laws and regulations when using the script for web scraping purposes.

## dev build (non functional)

version 8 was made to address various issues and make the program overall more usable and funtional

In addition to the existing libraries, the script now imports the mysql.connector library, to allow it to connect to the mySQL database to make the table and import the data

The script establishes a connection to a MySQL database named "music" hosted on a local server, using the provided credentials. This connection is essential for accessing and manipulating data within the database, allowing the inserts

A computer code with text

Description automatically generated with medium confidence

A new method named makeDataBase is introduced in the webScraper class. This method reads an SQL file containing database creation queries and executes them to create the necessary database.

A close-up of text

Description automatically generated

The addData method is updated to use parameterized SQL queries for inserting data into the MySQL database. This modification improves security and efficiency by preventing SQL injection attacks and optimizing query execution.

Another new method named checkSongs is added to the webScraper class. This method executes a SELECT query to retrieve data from the "songs" table in the MySQL database and prints the results, letting you test to see if it works

A screenshot of a computer program

Description automatically generated

After initializing the webScraper class instance w, the methods scrapeTable, makeDataFrame, sortDataFrame, makeDataBase, printDataFrame, and checkSongs are invoked sequentially to execute various tasks related to web scraping, database creation, data insertion, and verification. This structured approach ensures the orderly execution of operations and for debugging and troubleshooting.

Overall, these updates expand the script's functionality to include database interaction, automate database creation, and provide data validation capabilities. But it doesn’t work properly and would require further work

# Discussion

## General Discussion

The updated script would have demonstrates significant progress in automating the process of web scraping and database population. By integrating functionalities to interact with a MySQL database, automate database creation, and validate data insertion, the script enhances efficiency and reliability in handling extracted data from websites.

The script's functionalities align with established practices in web scraping and database management, reflecting a convergence of methodologies and techniques employed in related projects. The use of libraries such as BeautifulSoup, Pandas, and mysql.connector mirrors common approaches in web scraping and database interaction tasks. Additionally, the emphasis on data validation and adherence to database best practices underscores a commitment to maintaining data integrity and reliability, consistent with industry standards and best practices.

The version 8 sscript's development aligns closely with its stated aims of scraping a website for data and generating insert statements for a MySQL database. Through sequential execution of web scraping, database creation, data insertion, and verification tasks, the script effectively fulfills its primary objectives. But unfortunately, the working version 7.4 does not

## Legal and Ethical Considerations

web scraping activities are subject to various legal and ethical considerations that must be carefully navigated to ensure compliance with applicable laws and regulations. Firstly, adherence to website terms of service is crucial. Violating these terms, such as by scraping data without permission, may constitute a breach of contract and could result in legal action. Therefore, it's necessary to review and adhere to the terms of service of each website scraped to avoid potential legal consequences.

Data privacy regulations, particularly the General Data Protection Regulation (GDPR), impose strict requirements on the collection and processing of personal data in the UK. So Scraping any personal data without consent may violate GDPR regulations and other relevant data protection laws. Scraper developers must exercise caution and ensure compliance with applicable data privacy laws to protect individuals' privacy rights and avoid legal liabilities and being removed from the university.

Intellectual property rights are also a significant consideration in web scraping activities in the UK. Scraping copyrighted content or intellectual property without permission may infringe upon the rights of content creators or website owners. Scraper developers should respect intellectual property rights and seek permission from copyright holders before scraping and using their content to avoid potential legal consequences for copyright infringement.

## testing

Testing was exclusively conducted on Wikipedia due to ethical and legal considerations and a fear of accidentally breaking the law, with a commitment to upholding responsible web scraping practices. Wikipedia is a publicly accessible platform with permissive licensing terms that allow for the extraction of data for personal or non-commercial use. By limiting testing to Wikipedia, potential legal risks associated with scraping copyrighted or restricted content from other websites were mitigated. This approach ensured compliance with applicable laws, regulations, and ethical guidelines governing web scraping activities. Additionally, focusing testing efforts on Wikipedia provided a controlled environment for evaluating the script's functionality, performance, and scalability without infringing upon the rights of website owners or violating terms of service.

## problems

Additionally, the makeDataFrame method utilizes the pd.read\_html function from the pandas library to convert HTML table data into a DataFrame. However, the success of this conversion relies heavily on the structure and formatting of the HTML table, which may vary across different web pages. Inconsistent HTML structures or unexpected data formats will lead to errors.

Furthermore, the code includes methods for sorting the extracted data (sortDataFrame) and interacting with a MySQL database (makeDataBase, addData, checkSongs). However, these database interaction methods rely on the assumption that the DataFrame contains valid data, which may not always be the case if the data returns incomplete or inaccurate results.

## Future Work

First and foremost would be finishing version 8 and getting it fully working

One potential area for future work involves adding error handling mechanisms in the script. It would help manage exceptions and edge cases encountered during web scraping, database interaction, and in the data validation process. Implementing error handling would improve the script's resilience and reliability.

integrating mechanisms to address legal and ethical considerations surrounding web scraping activities. So it would automatically check the TOS and if it was not allowed to be scrapped it would skip it This includes ensuring compliance with website terms of service, data privacy regulations, and intellectual property rights.

Developing a user-friendly graphical interface is another avenue for future work. A graphical interface can streamline user interactions and enhance usability by providing intuitive controls for configuring ad inputting urls

Making it more universally applicable to be able to scrape across a smaller verity of sites.

# References

*Beautiful Soup Documentation — Beautiful Soup 4.4.0*. (2024, 02 02). Retrieved from https://beautiful-soup-4.readthedocs.io/en/latest/

*https://cdn2.hubspot.net/hubfs/258389/White\_Papers/WP\_2016\_Bad\_Bot\_Landscape\_Report-1-1-1.pdf*. (2016). Retrieved 03 02, 2024, from https://cdn2.hubspot.net/hubfs/258389/White\_Papers/WP\_2016\_Bad\_Bot\_Landscape\_Report-1-1-1.pdf

Pandas. (2024, 03 02). Retrieved from pandas.DataFrame — pandas 2.2.2 documentation - PyData |: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

PyPi. (2024, 03 02). *requests*. Retrieved from https://pypi.org/project/requests/

# Appendices

Note that Appendices should be referenced in the main body of the text.

## Appendix A working code

#Harry Fisher

#Web scrapper V 7.4

#WIP

"""

This is a program to go through the link provided find a table sort through the data to take out what i

need it will then... in the future auto put it into a database for now it just prints the insert statments

"""

#right now i am using pandas because i can get it to work, later i would like to be able to do it with the xml file

#but its a bit complecated so ill work it out later

import pandas as pd

#the requests library is the standard for making HTTP requests in Python

import requests

#Beautiful Soup is a Python package for parsing HTML and XML documents in other works its used for web scrapeing

from bs4 import BeautifulSoup

#i set up a class called webScraper purely to pass the local variable objects through the methods without having to make global vairiables

#im assuming this is bad practice or incredibly inefficient but it works...

class webScraper:

#this is the actual part of the program that gets the table

def scrapeTable(self):

#this is the actual part of the program that gets the xml file

response = requests.get("https://en.wikipedia.org/wiki/Greatest\_Hits\_(Queen\_album)")

soup = BeautifulSoup(response.text, 'html.parser')

#this is the part that looks through and gets the table

self.findTable = soup.findAll('table', {'class':"tracklist"})

def makeDataFrame(self):

self.df=pd.read\_html(str(self.findTable))

self.df=pd.DataFrame(self.df[0])

#this is here to sort throught the data and format it in the way i want

def sortDataFrame(self):

#this line of code removes everything after and including the ( this is done because on the page after the title there is a bunch of junk in brackets

self.df['Titles'] = self.df['Title'].str.split('(').str[0]

#this line of code removes the quotation marks from around the title of the song

self.df['Titles'] = [counter.replace('"','') for counter in self.df['Titles']]

#this line of code removes any extra space from the start and end of the title

self.df['Titles'] = [counter.strip() for counter in self.df['Titles']]

#this is a temp test method it runs through the lists and prints them in the form of an insert statment

def printDataFrame(self):

for counter in range(len(self.df['Titles'])):

print("INSERT INTO songs(title, duration, artist, genre\_code")

print(F"VALUES ('{self.df['Titles'][counter]}', '{self.df['Length'][counter]}', '{self.df['Writer(s)'][counter]}', 1);")

#assigning w to the webScraper class then calling the methods

w = webScraper()

w.scrapeTable()

w.makeDataFrame()

w.sortDataFrame()

w.printDataFrame()

## Appendix B dev build

**#Harry Fisher**

**#Web scrapper V 8**

**#WIP**

**"""**

**This is a program to go through the link provided find a table sort through the data to take out what i**

**need it will then... in the future auto put it into a database for now it just prints the insert statments**

**"""**

**#right now i am using pandas because i can get it to work, later i would like to be able to do it with the xml file**

**#but its a bit complecated so ill work it out later**

**import pandas as pd**

**#the requests library is the standard for making HTTP requests in Python**

**import requests**

**#Beautiful Soup is a Python package for parsing HTML and XML documents in other works its used for web scrapeing**

**from bs4 import BeautifulSoup**

**import mysql.connector**

**mydb = mysql.connector.connect(**

**host="localhost",**

**user="harry",**

**password="1234",**

**database="music"**

**)**

**mycursor = mydb.cursor()**

**#i set up a class called webScraper purely to pass the local variable objects through the methods without having to make global vairiables**

**#im assuming this is bad practice or incredibly inefficient but it works...**

**class webScraper:**

**#this is the actual part of the program that gets the table**

**def scrapeTable(self):**

**#this is the actual part of the program that gets the xml file**

**response = requests.get("https://en.wikipedia.org/wiki/Greatest\_Hits\_(Queen\_album)")**

**soup = BeautifulSoup(response.text, 'html.parser')**

**#this is the part that looks through and gets the table**

**self.findTable = soup.findAll('table', {'class':"tracklist"})**

**def makeDataFrame(self):**

**self.df=pd.read\_html(str(self.findTable))**

**self.df=pd.DataFrame(self.df[0])**

**#this is here to sort throught the data and format it in the way i want**

**def sortDataFrame(self):**

**#this line of code removes everything after and including the ( this is done because on the page after the title there is a bunch of junk in brackets**

**self.df['Titles'] = self.df['Title'].str.split('(').str[0]**

**#this line of code removes the quotation marks from around the title of the song**

**self.df['Titles'] = [counter.replace('"','') for counter in self.df['Titles']]**

**#this line of code removes any extra space from the start and end of the title**

**self.df['Titles'] = [counter.strip() for counter in self.df['Titles']]**

**def makeDataBase(self):**

**with open('C:\\Users\\Harry Fisher\\Downloads\\MUSIC.sql') as MakeDataBase:**

**mycursor.execute(MakeDataBase.read(), multi=True)**

**def addData(self):**

**sql = "INSERT INTO songs(title, duration, artist, genre\_code) VALUES (%s, %s, %s, %s)"**

**for counter in range(len(self.df['Titles'])):**

**if counter < len(self.df['Titles']) -1:**

**mycursor.executemany(sql, (F" ('{self.df['Titles'][counter]}', '{self.df['Length'][counter]}', '{self.df['Writer(s)'][counter]}', '1'),"))**

**else:**

**mycursor.executemany(sql, (F" ('{self.df['Titles'][counter]}', '{self.df['Length'][counter]}', '{self.df['Writer(s)'][counter]}', '1')"))**

**def checkSongs(self):**

**mycursor.execute("SELECT \* FROM songs")**

**myresult = mycursor.fetchall()**

**for x in myresult:**

**print(x)**

**#this is a temp test method it runs through the lists and prints them in the form of an insert statment**

**def printDataFrame(self):**

**for counter in range(len(self.df['Titles'])):**

**if counter < len(self.df['Titles']) -1:**

**print("INSERT INTO songs(title, duration, artist, genre\_code)")**

**print(F"VALUES ('{self.df['Titles'][counter]}', '{self.df['Length'][counter]}', '{self.df['Writer(s)'][counter]}', 1),")**

**else:**

**print("INSERT INTO songs(title, duration, artist, genre\_code)")**

**print(F"VALUES ('{self.df['Titles'][counter]}', '{self.df['Length'][counter]}', '{self.df['Writer(s)'][counter]}', 1)")**

**#assigning w to the webScraper class then calling the methods**

**w = webScraper()**

**w.scrapeTable()**

**w.makeDataFrame()**

**w.sortDataFrame()**

**w.makeDataBase()**

**#w.addData()**

**w.printDataFrame()**

**w.checkSongs()**

## Appendix C code output

A screenshot of a computer

Description automatically generated