**Name: ANJALI K M**

**Roll No. : 231057028**

**1. What is Web Scraping, explain with an example?**

Web Scraping is important technique used for extracting unstructured data from the websites and transforming that data into structured. Web Scraping is also identified as web data extraction, web data scraping, web harvesting or screen scraping.

Basic architecture of web scraping:

Structured data

Web scraping mechanism

World wide websites

example of web scraping using Python and the BeautifulSoup library to extract the titles of articles from a hypothetical blog website:

import requests

from bs4 import BeautifulSoup

# Send an HTTP request to the website

url = "https://www.exampleblog.com"

response = requests.get(url)

html\_content = response.content

# Parse the HTML using BeautifulSoup

soup = BeautifulSoup(html\_content, 'html.parser')

# Find article titles using appropriate HTML tags and classes

article\_titles = soup.find\_all('h2', class\_='article-title')

# Extract and print the article titles

for title in article\_titles:

print(title.text)

In this example, we assume that the blog website's articles are structured with <h2> tags and a class named 'article-title' for the article titles. The script sends an HTTP request to the website, retrieves the HTML content, uses BeautifulSoup to parse it, and then extracts and prints the article titles.

**2. What are the challenges in performing web scraping?**

Some common challenges we might encounter when performing web scraping:

1. Website Changes: Websites often update their layout and structure, causing your scraping code to break because the expected elements have moved or changed.

2. Data Format Variation: Websites might present data in different formats on different pages, making it tricky to consistently extract the information you need.

3. CAPTCHA: Some websites require users to solve CAPTCHA challenges to prove they're human, which is difficult to automate and can block your scraping efforts.

4. IP Blocking: Frequent or aggressive scraping can lead to your IP address being blocked by the website's server to prevent overloading.

5. Rate Limiting: Websites may restrict the number of requests you can make in a given time frame, slowing down your scraping process.

6. Empty or Missing Data: Websites might not always have the data you expect, leading to missing or incomplete information in your scraped dataset.

7. Anti-Scraping Measures: Websites can use techniques like user-agent detection to identify and block scraping bots.

8. Legal and Ethical Concerns: Scraping without proper permissions or for unethical purposes can lead to legal issues and ethical dilemmas.

9. Data Cleanup: Scraped data can have inconsistent formatting, requiring you to clean and transform it into a usable format.

10. Ongoing Maintenance: Websites evolve, necessitating regular updates to your scraping code to ensure it continues to work.

**3. Describe the basic architecture for performing web scraping?**

Web scraping involves extracting information from websites. The basic architecture for performing web scraping typically involves the following steps:

Sending a Request:

The process begins by sending a request to the target website's server. This request can be made using libraries like requests in Python. The server responds by sending back the requested web page's HTML content.

Receiving the HTML Content:

Once the server responds, you receive the HTML content of the web page. This content includes all the text, images, links, and structure that make up the page.

Parsing the HTML:

The next step is to parse the HTML content to extract the specific information you're interested in. This is typically done using HTML parsing libraries like BeautifulSoup or lxml in Python. These libraries allow you to navigate the HTML structure and extract elements like headings, paragraphs, tables, links, etc.

Extracting Data:

Using the parsed HTML, you can identify the specific elements you want to extract. For example, you might want to extract product prices, article titles, or contact information. You access these elements using selectors like CSS selectors or XPath expressions, depending on the library you're using.

Storing Data:

After extracting the desired data, you need to store it. This could involve writing the data to a file (such as CSV, JSON, or a database), using it for analysis, or displaying it on your own website or application.

Handling Pagination and Dynamic Content:

Many websites have multiple pages or load content dynamically through JavaScript. To scrape these, you might need to deal with pagination (navigating through different pages) and handling JavaScript-rendered content. Tools like Selenium can help automate the interaction with dynamic content.

Crawling and Rate Limiting:

When scraping multiple pages, it's essential to be respectful of the website's server resources. Excessive requests in a short time can lead to your IP address being blocked or the website's server becoming slow. Implementing rate limiting (controlled number of requests per unit of time) can help prevent this issue.

Handling Errors and Exceptions:

Websites can change their structure or content, leading to unexpected issues during scraping. Implementing error handling and exceptions ensures that your scraping script can gracefully handle unexpected situations and continue functioning without breaking.

**4. Describe various techniques of web scraping.**

Various techniques of web scraping are:

A. Classical copy and paste: The human’s manual examination and copy and-paste method is the best and the workable webscraping technology. But this is an tending to implement or cause errors, and tiresome technique when user need to analyse and store lots of datasets.

B. Hypertext Transfer Protocol (HTTP) Programming: By using this technique user can be extract information from static and dynamic web pages. Data can be retrieved by posting HTTP requests to the remote web server using socket programming .

C. Hyper Text Markup Language (HTML) Parsing: Semi-structured data query languages, like XQuery and the Hyper Text Query Language (HTQL), can be used to parse HTML pages and to retrieve and transform page content.

D. Document Object Model (DOM)Parsing: By embedding a full-fledged web browser, such as the Internet Explorer or the Mozilla browser control, programs can retrieve the dynamic content generated by client-side scripts. These browser controls also parse web pages into a DOM tree, based on which programs can retrieve parts of the pages. E. Web Scraping Software: Now a days many software tools are available, that can be used to customize web-scraping solutions. This software may attempt to automatically recognize the data structure of a page or provide a recording interface that removes the necessity to manually write web-scraping code, or some scripting functions that can be used to extract and transform content, and database interfaces that can store the scraped data in local databases.

F. Computer vision web-page analysers: There are efforts using machine learning and computer vision that attempt to identify and extract information from web pages by interpreting pages visually as a human being might .