1. Write Python code for extracting Title and href attributes from a HTML document?

   ```
   import requests
   from bs4 import BeautifulSoup
   url = 'https://example.com'
   r=rerquests.get(url)
   soup = BeautifulSoup(response.content, 'html.parser')
   title = soup.title.string if soup.title else 'No title found' print(f"Title of the webpage: {title}")
   links = soup.find_all('a')
    for link in links:
    href = link.get('href')
   if href:
    print(f"Link: {href}")
   else:
    print(f"Failed to retrieve the webpage.)
   ```

2. Write a Python function that uses CSS selectors to extract all links from a given webpage.

   ```
   import requests
    from bs4 import BeautifulSoup
   def extract_links(url):
    response = requests.get(url)
   soup = BeautifulSoup(response.content, 'html.parser')
   links = soup.select('a[href]')
   return [link.get('href') for link in links]
   url = 'https://example.com'
   links = extract_links(url)0
   for link in links:
    print(link)
   ```

3. Describe the process of making an HTTP GET request in Python using the requests library and print the response status code and its content.

   Install the requests library: First, you need to make sure that you have the requests library installed. If you don't have it installed, you can use pip to install it:

   bash

   Copy code

   ```
   pip install requests
   ```

   Make the HTTP GET request: You can make a GET request to a URL using the requests.get(url) function, where url is the website you want to request data from.

```
Import requests
url='www.google.com'
response=requests.get(url)
print(r.status_code)
print(r.content)
```

4. How would you handle session cookies while scraping a website that requires login?

1. **Create a session**:

   Use requests.Session() to create a session object. This object will automatically manage cookies across requests, so you don't have to manually handle the cookies.

2. **Login to the website**:

   Send a POST request with your login credentials (such as username and password) to the website's login form.

   The server will set a session cookie upon successful login, which will be stored and sent with subsequent requests made using the same session object.

3. **Scrape the website**:

   After logging in, you can use the session object to make subsequent requests to protected pages, and the session cookie will be automatically included in the request headers.

4. **Handle the cookies**:

   The requests library automatically handles cookies for you within a session. However, you can access the cookies manually if needed using session.cookies.

Example Code:

Here's how you can use the requests library to log into a website that requires login and handle session cookies:

Step-by-Step Example:

```
import requests



# Step 1: Create a session object to manage cookies

session = requests.Session()
```

```
login_url = 'https://example.com/login'

login_data = {

    'username': 'your_username',  # Replace with your actual username

    'password': 'your_password'   # Replace with your actual password

}

login_response = session.post(login_url, data=login_data)

if login_response.status_code == 200:

    print("Login successful!")

else:

    print(f"Login failed with status code {login_response.status_code}")

    exit()

protected_url = 'https://example.com/protected-page'  # Replace with a protected URL

protected_response = session.get(protected_url)

print("Protected page content:")

print(protected_response.text)

print("\nSession Cookies:")

print(session.cookies.get_dict())
```

5. How does Scrapy handle requests and responses?

Scrapy handles requests and responses through an asynchronous, event-driven system designed for efficient web scraping. When a spider starts, it generates requests that are managed by Scrapy's scheduler, which prioritizes them. These requests pass through downloader middleware, allowing modifications such as setting user-agents or proxies. Using the Twisted library, Scrapy sends multiple requests concurrently, allowing high-speed scraping. Responses are then processed by the middleware and returned to the spider's parse method, where data is extracted and new requests are generated. Extracted data is then sent through item pipelines for cleaning, validation, and storage. This streamlined process enables Scrapy to handle large volumes of data effectively.

6. What is Scrapy, and how does it differ from other web scraping libraries like BeautifulSoup?

Scrapy is an open-source web scraping and web crawling framework written in Python. It is specifically designed for large-scale web scraping tasks, making it more robust and efficient for complex and high-volume scraping needs. Unlike other scraping libraries like BeautifulSoup, Scrapy is a full-fledged framework that includes tools for managing the entire scraping process—from extracting data to handling requests and managing large volumes of data, making it more versatile and powerful.

7. How would you handle complex scraping projects with multiple spiders and data pipelines in Scrapy?

   To handle complex Scrapy projects:
   Use Multiple Spiders: Create individual spiders for each source, using CrawlSpider for link-heavy sites.
   Spider-Specific Settings: Customize settings per spider for rate limits and delays.
   Data Pipelines: Define pipelines for data cleaning, validation, and storage.
   Custom Middleware: Manage retries, proxies, and user-agent rotation.
   Run Concurrently: Use scripts to run multiple spiders in parallel for efficiency.

8. Write a Python script that scrapes the title, headings (h1, h2, etc.), and paragraphs (p) from a given webpage URL and outputs the following:
   The title of the page
   All the headings in the page (h1, h2, h3, etc.)
   All the paragraphs
   Example:
   - URL: https://www.example.com
   - Output:
     - Title: "Example Domain"
     - Headings: ['Heading 1', 'Heading 2']
     - Paragraphs: ['This is a paragraph.']

```
import requests
from bs4 import BeautifulSoup
def scrape_webpage(url):
   response = requests.get(url)
   if response.status_code != 200:
       print(f"Failed to retrieve the page. Status code: {response.status_code}")
       return
   soup = BeautifulSoup(response.text, 'html.parser')
   title = soup.title.string if soup.title else 'No title found'
   headings = []
   for level in range(1, 7):
       heading_tag = f'h{level}'
       for heading in soup.find_all(heading_tag):
           headings.append(heading.get_text(strip=True))
   paragraphs = [p.get_text(strip=True) for p in soup.find_all('p')]
   print(f"Title: \"{title}\"")
   print("Headings:", headings if headings else "No headings found.")
   print("Paragraphs:", paragraphs if paragraphs else "No paragraphs found.")
url = input("Enter the URL of the webpage to scrape: ").strip()
scrape_webpage(url)
```

9. Write a Python script to scrape a website with multiple pages (e.g., an e-commerce site or a blog with paginated articles). Your task is to:

   Extract the titles of articles/products.

   Iterate through multiple pages to get all the data. (Assume the pagination is implemented using "Next" buttons or page numbers in the URL).

   Example:

   - URL: https://example.com/products?page=1
   - Expected Output:
     - Titles of products from pages 1, 2, and 3.

```
import requests

from bs4 import BeautifulSoup

def scrape_products(base_url, total_pages=3):

    all_product_titles = []

    for page_num in range(1, total_pages + 1):

        url = f"{base_url}?page={page_num}"
        print(f"Scraping page {page_num}: {url}")

        response = requests.get(url)

        if response.status_code != 200:

            print(f"Failed to retrieve page {page_num}, status code: {response.status_code}")

            continue

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

        product_titles = [title.get_text(strip=True) for title in soup.find_all('h2', class_='product-title')]

        if not product_titles:

            print(f"No products found on page {page_num}.")

        else:

            print(f"Found {len(product_titles)} products on page {page_num}.")

        all_product_titles.extend(product_titles)

    return all_product_titles

def main():

    base_url = "https://example.com/products"

    product_titles = scrape_products(base_url, total_pages=3)

    print("\nScraped Product Titles:")

    for idx, title in enumerate(product_titles, 1):

        print(f"{idx}. {title}")

if __name__ == "__main__":

    main()
```

10. Write a Python script that scrapes product prices for the same product from multiple e-commerce websites (e.g., Amazon, Flipkart, TataCliQ).

    Your task is to:

    Search for the same product across multiple websites (e.g., "laptop").

Extract product name, price, and rating (if available).
Output the results in a structured format like a CSV or JSON.
Example:

- Search term: "Laptop"
- Expected Output:

```
[
 {
  "product_name": "Laptop 1",
  "price": "$500",
  "rating": "4.5/5",
  "website": "amazon.com"
 },
 {
  "product_name": "Laptop 2",
  "price": "$450",
  "rating": "4.0/5",
  "website": "tatacliq.com"
 }
]
```

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

def scrape_amazon(search_term):

    url = f"https://www.amazon.com/s?k={search_term.replace(' ', '+')}"
    headers = {
        "
    }

    response = requests.get(url, headers=headers)
    soup = BeautifulSoup(response.content, "html.parser")

    products = []
    for item in soup.find_all("div", {"class": "s-result-item"}):
        try:
            product_name = item.h2.text.strip()
            price = item.find("span", {"class": "a-price-whole"}).text.strip()
            rating = item.find("span", {"class": "a-icon-alt"}).text.strip()
            products.append({
                "product_name": product_name,
                "price": price,
                "rating": rating,
                "website": "amazon.com"
            })
        except AttributeError:
            continue


def scrape_flipkart(search_term):
    url = f"https://www.flipkart.com/search?q={search_term.replace(' ', '%20')}"
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
    }

    response = requests.get(url, headers=headers)
    soup = BeautifulSoup(response.content, "html.parser")

    products = []
    for item in soup.find_all("a", {"class": "IRpwTa"}):
        try:
            product_name = item.text.strip()
            price = item.find("div", {"class": "_30jeq3"}).text.strip()
            rating = item.find("div", {"class": "_3LWZlK"}).text.strip() if item.find("div", {"class": "_3LWZlK"}) else "N/A"
            products.append({
                "product_name": product_name,
                "price": price,
                "rating": rating,
                "website": "flipkart.com"
            })
        except AttributeError:
            continue
    return products

def scrape_tatacliq(search_term):
    url = f"https://www.tatacliq.com/search/?searchCategory=all&text={search_term.replace(' ', '%20')}"
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
    }

    response = requests.get(url, headers=headers)
    soup = BeautifulSoup(response.content, "html.parser")
```

```
    products = []
    for item in soup.find_all("div", {"class": "ProductModule"}):
        try:
            product_name = item.find("a", {"class": "ProductName"}).text.strip()
            price = item.find("span", {"class": "ProductPrice"}).text.strip()
            rating = item.find("span", {"class": "ProductRating"}).text.strip() if
item.find("span", {"class": "ProductRating"}) else "N/A"
            products.append({
                "product_name": product_name,
                "price": price,
                "rating": rating,
                "website": "tatacliq.com"
            })
        except AttributeError:
            continue
    return products

def search_product(search_term):
    products = []
    products += scrape_amazon(search_term)
    products += scrape_flipkart(search_term)
    products += scrape_tatacliq(search_term)

    return products

search_term = "laptop"
product_data = search_product(search_term)

import json
with open("product_data.json", "w") as f:
    json.dump(product_data, f, indent=4)

df = pd.DataFrame(product_data)
df.to_csv("product_data.csv", index=False)

print("Data saved in product_data.json and product_data.csv.")
```

11. Write a Python script that scrapes weather data for a given location from a weather website (e.g., Weather.com). You need to extract:
    - Current temperature.
    - Weather conditions (sunny, rainy, etc.).
    - 7-day forecast (if available).

      Example Url: https://weather.com/weather/today/l/<Country_name>

```
import requests
```

```
from bs4 import BeautifulSoup
import json
import pandas as pd

def scrape_weather(location_url):
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
    }

    response = requests.get(location_url, headers=headers)
    soup = BeautifulSoup(response.content, "html.parser")

    weather_data = {}

    try:
        current_temp = soup.find("span", class_="CurrentConditions--tempValue--
3KcTQ").text
        weather_data["current_temperature"] = current_temp
    except AttributeError:
        weather_data["current_temperature"] = "N/A"


    try:
        condition = soup.find("div", class_="CurrentConditions--phraseValue--
2xXSr").text
        weather_data["current_condition"] = condition
    except AttributeError:
        weather_data["current_condition"] = "N/A"

    forecast = []
    try:
        days = soup.find_all("span", class_="DailyContent--daypartDate--3MM0J")
        temps = soup.find_all("span", class_="DailyContent--temp--_8DL5")
        conditions = soup.find_all("span", class_="DailyContent--narrative--3AcXd")

        for day, temp, cond in zip(days, temps, conditions):
            forecast.append({
                "day": day.text.strip(),
                "temperature": temp.text.strip(),
                "condition": cond.text.strip()
            })
        weather_data["7_day_forecast"] = forecast
    except AttributeError:
        weather_data["7_day_forecast"] = "N/A"

    return weather_data
location_url = "https://weather.com/weather/today/l/India"
weather = scrape_weather(location_url)

print(json.dumps(weather, indent=4))

with open("weather_data.json", "w") as f:
    json.dump(weather, f, indent=4)

if "7_day_forecast" in weather:
    df = pd.DataFrame(weather["7_day_forecast"])
    df.to_csv("7_day_forecast.csv", index=False)

print("Weather data saved in weather_data.json and 7_day_forecast.csv.")
```

12. Write a Python script to scrape job listings from a job board (e.g., Indeed, LinkedIn). You need to:

    - Extract job titles, company names, locations, and the job descriptions.
    - Filter the results by a specific keyword (e.g., "data scientist").
    - Store the results in a CSV or JSON file.

    Example:

    - Input keyword: "data scientist"
    - Output:

    job_title, company, location, job_description

"Data Scientist", "ABC Corp", "Hyderabad, India", "Looking for a data scientist with experience in machine learning..."

```
import requests

from bs4 import BeautifulSoup

import pandas as pd

import json




def scrape_jobs(keyword):
    query = keyword.replace(" ", "+")

    url = f"https://www.indeed.com/jobs?q={query}&l="

    headers = {

        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

    }


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

    soup = BeautifulSoup(response.content, "html.parser")


    job_listings = []


    for job_card in soup.find_all("div", class_="jobsearch-SerpJobCard"):

        try:

            title = job_card.find("h2", class_="title").text.strip()

             company = job_card.find("span", class_="company").text.strip()

            location = job_card.find("div", class_="location").text.strip()

            description = job_card.find("div", class_="summary").text.strip()

            job_listings.append({

                "job_title": title,

                "company": company,

                "location": location,

                "job_description": description

            })

        except AttributeError:
            continue  # Skip if any required field is missing

    return job_listings

def main(keyword):

    jobs = scrape_jobs(keyword)



    with open("job_listings.json", "w") as f:

        json.dump(jobs, f, indent=4)

    df = pd.DataFrame(jobs)

    df.to_csv("job_listings.csv", index=False)

    print("Job listings saved in job_listings.json and job_listings.csv.")

if __name__ == "__main__":

    keyword = "data scientist"

    main(keyword)
```