Web Scraping with Python: Requests, BeautifulSoup, and Pandas

1. Install Required Libraries

Before starting, make sure you have the necessary libraries installed. You can install them using pip:

pip install requests beautifulsoup4

2. Import Libraries

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

3. Make a Request

Use the `requests.get()` method to fetch the HTML content of a webpage.

```
url = 'https://www.example.com'
response = requests.get(url)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    webpage_content = response.text
else:
    print(f"Error: Unable to fetch the webpage. Status code: {response.status_code}")
```

2. Handling 403 Response Code

```
headers = {
   'User-Agent': 'Mozilla/5.0 (Windows NT 6.3; Win 64; x64) Apple WeKit /537.36(KHTML, like Gecko) Chrome/80.0.3987.162 Safari/537.36'
}

# Checking if the response code is 403
if response_code == 403:
   response = requests.get('url', headers=headers).text
```

4. Parse HTML with BeautifulSoup

Use `BeautifulSoup` to parse the HTML content and navigate through the elements.

```
soup = BeautifulSoup(webpage_content, 'html.parser')
```

5. Locate Elements

Identify the HTML elements containing the data you want to extract. This may involve inspecting the webpage source code.

```
# Example: Extracting all links
links = soup.find_all('a')
for link in links:
   print(link.get('href'))
```

6. Extract Data

Create variables to store the extracted data. Use try-except blocks to handle exceptions gracefully.

```
# Example: Extracting titles from a webpage
titles = []
for article in soup.find_all('article'):
   try:
      title = article.find('h2').text.strip()
      titles.append(title)
```

```
    except AttributeError as e:
        print(f"Error: {e}")
```

7. Store Data in DataFrame

Use `pandas` to organize the extracted data into a DataFrame.

```
data = pd.DataFrame({
    'Title': titles,
    # Add other extracted data columns as needed
})
```

8. Display or Save Data

Display the data or save it to a file for further analysis.

```
# Display the DataFrame
print(data)

# Save to CSV
data.to_csv('output.csv', index=False)
```

Conclusion

Web scraping is a powerful tool for extracting data from websites. However, it's important to be respectful of websites' terms of service and policies, and to avoid sending too many requests in a short period, which could lead to IP bans. Always check a website's `robots.txt` file for guidelines on web scraping.