Web Scraping using Python

Design Laboratory (CS69202)
Jan 15, 2025

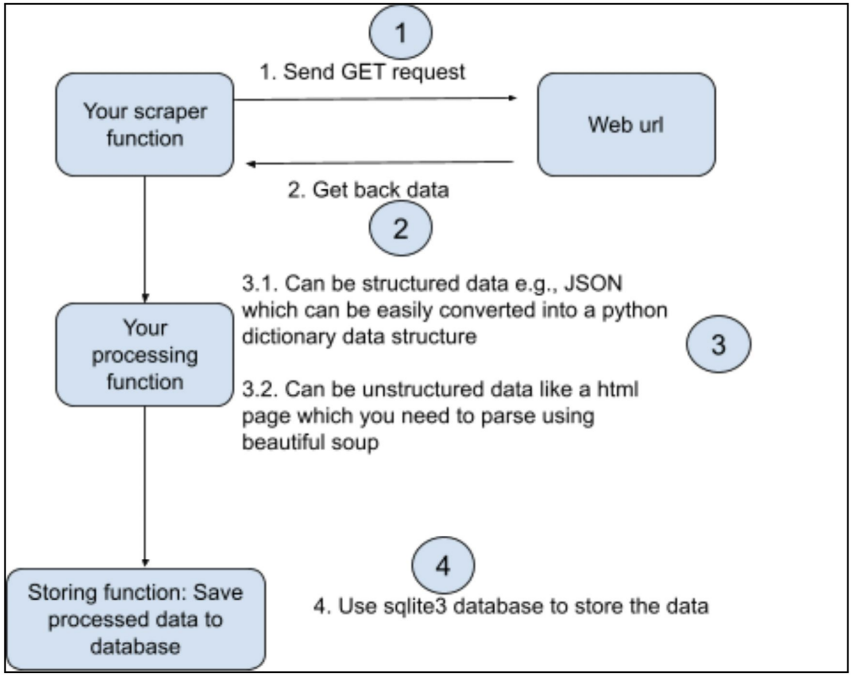
What is Web Scrapping

Web scrapers are programs that can collect data from websites, parse it and store the data (e.g., for analysis later).

A web request can be GET or POST. Using GET requests you can get data from the world wide web (WWW).

Pipeline for Scraping

1. Send the request (GET/POST)
2. Get the response
3. Based on response type; process the data
4. Store the data



Tools required for this assignment

1. Requests: “allows you to send HTTP/1.1 requests extremely easily”
2. BeautifulSoup (BS4): “scrape information from web pages. Sits atop an HTML parser, assists iterating, searching, and modifying the parse tree.”
3. Selenium: “Primarily it is for automating web applications for testing purposes, but is certainly not limited to just that.”
4. Sqlite3: “provides a lightweight disk-based database that doesn’t require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language.”

Requests: snippet to retrieve data from a url

```
import requests
url = '...'
response = requests.get(url)
if response.status_code == 200:
  output = response.text
else:
  print('Exception: url not correct or No response')
```

Caveat: *Best to set browser headers*

```
headers = {

    'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36'

}

...

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

BeautifulSoup

```
html_doc = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
"""
```

```
from bs4 import BeautifulSoup

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

Here are some simple ways to navigate that data structure:

```
soup.title
# <title>The Dormouse's story</title>

soup.title.name
# u'title'

soup.title.string
# u'The Dormouse's story'

soup.title.parent.name
# u'head'

soup.p
# <p class="title"><b>The Dormouse's story</b></p>

soup.p['class']
# u'title'

soup.a
# <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>

soup.find_all('a')
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]

soup.find(id="link3")
# <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
```

BeautifulSoup: Searching the tree

Two most popular methods: find() and find_all()

You can search for an element using different kinds of filters:

- `String: soup.find_all('b')`
- `Regex : soup.find_all(re.compile("^b"))`
- `List  : soup.find_all(["a", "b"])`

find_all(name, attrs, recursive, string, limit, **kwargs)

```
soup.find_all("title")
# [<title>The Dormouse's story</title>]

soup.find_all("p", "title")
# [<p class="title"><b>The Dormouse's story</b></p>]

soup.find_all("a")
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]

soup.find_all(id="link2")
# [<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]

import re
soup.find(string=re.compile("sisters"))
# u'Once upon a time there were three little sisters; and their names were\n'
```

find(name, attrs, recursive, string, **kwargs)

The *find_all()* method scans the entire document looking for results, but sometimes you only want to find one result. — use **find()**

```
soup.find_all('title', limit=1)
# [<title>The Dormouse's story</title>]

soup.find('title')
# <title>The Dormouse's story</title>
```

Other methods: find_all_next, find_next

These methods use iterates over whatever tags and strings that come after it in the document

```
first_link = soup.a
first_link
# <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>

first_link.find_all_next(string=True)
# [u'Elsie', u',\n', u'Lacie', u' and\n', u'Tillie',
#  u';\nand they lived at the bottom of a well.', u'\n\n', u'...', u'\n']

first_link.find_next("p")
# <p class="story">...</p>
```

Selenium: Writing your first script

- Start the session

```
driver = webdriver.Chrome()
```

- Take action on browser

```
driver.get("https://www.selenium.dev/selenium/web/web-form.html")
```

- Request browser information

```
title = driver.title
```

- Find an element

```
text_box = driver.find_element(by=By.NAME, value="my-text")
submit_button = driver.find_element(by=By.CSS_SELECTOR, value="button")
```

Selenium: Writing your first script

- Take action on element

```
text_box.send_keys("Selenium")
submit_button.click()
```

- Request element information

```
text = message.text
```

- End the session

```
driver.quit()
```

Sqlite3: Writing your first script

- Create a new database and open a database connection

```
import sqlite3
con = sqlite3.connect("tutorial.db")
```

- Create a database cursor: to execute SQL statements and fetch results from SQL queries

```
cur = con.cursor()
```

- Create a database table movie with columns for title, release year, and review score

```
cur.execute("CREATE TABLE movie(title, year, score)")
```

- Add data supplied as SQL literals by executing an INSERT statement

```
cur.execute("""
     INSERT INTO movie VALUES
     ('Monty Python and the Holy Grail', 1975, 8.2)
""")
```

Sqlite3: Writing your first script

- Fetching results from a table by executing SELECT

```
res = cur.execute('SELECT score FROM movie")
res.fetchall()
```

- Saving the changes

```
con.commit()
```

- Closing the db connection

```
con.close()
```