Week 1 Lab: Web Scraping Workshop

CS 203: Software Tools and Techniques for AI

Duration: 3 hours

Prof. Nipun Batra & Teaching Assistants

Lab Overview

Today's Goals

By the end of this lab, you will:

- [YES] Master Chrome DevTools for finding data sources
- [YES] Build scrapers with Requests + BeautifulSoup
- [YES] Automate browsers with Playwright
- [YES] Create a real data collection project
- [YES] Understand ethical scraping practices

Structure

- **Part 1:** DevTools Exploration (45 min)
- **Part 2:** Static Scraping with BeautifulSoup (90 min)

Setup Check (10 minutes)

Verify Your Environment

```
# Check Python version (need 3.8+)
python --version

# Install required packages
pip install requests beautifulsoup4 playwright pandas matplotlib

# Install Playwright browsers
playwright install chromium

# Verify installations
python -c "import requests, bs4, playwright; print('All packages installed!')"
```

Troubleshooting

- **Permission errors**: Use `pip install --user`

Part 1: Chrome DevTools Mastery

Exercise 1.1: Find the Hidden API (15 min)

Website: https://www.goodreads.com or https://www.imdb.com

Your Task:

1. Open the website in Chrome
2. Open DevTools (Cmd+Option+I / Ctrl+Shift+I)
3. Go to Network tab
4. Filter by XHR/Fetch
5. Search for a book/movie
6. Find the API request that returns search results
7. Copy the request as cURL

Exercise 1.1: Solution Example

IMDB Search Example

```
# What you might find in Network tab
https://v3.sg.media-imdb.com/suggestion/x/avengers.json

# The response will be JSON like:
{
  "d": [
    {
      "i": {"imageUrl": "..."},
      "l": "Avengers: Endgame",
      "q": "feature",
      "rank": 43,
      "s": "Robert Downey Jr., Chris Evans",
      "y": 2019
    },
     ...
  ]
}
```

Exercise 1.2: cURL to Python (15 min)

Task: Convert your cURL command to Python

Step 1: Copy as cURL

Right-click request → Copy → Copy as cURL

Step 2: Use curlconverter.com (or do manually)

- Visit https://curlconverter.com
- Paste your cURL command
- Select "Python Requests"
- Get Python code!

Step 3: Test it

```python
import requests
```

Exercise 1.3: GitHub API Exploration (15 min)

Task: Use DevTools to find GitHub's API endpoints

1. Go to https://github.com/microsoft/vscode
2. Open DevTools → Network
3. Click on "Issues" tab
4. Find the API call for issues
5. Examine the response structure

Questions to Answer:

- What's the API endpoint?
- What parameters does it use?
- How is pagination handled?
- What authentication is needed?

Part 1 Checkpoint

What You've Learned

[YES] How to find API endpoints using DevTools
[YES] How to copy requests as cURL
[YES] How to convert cURL to Python
[YES] How to identify request parameters

Quick Quiz (5 min)

1. What's the difference between XHR and Fetch?
2. Where do you find authentication tokens in DevTools?
3. Why is copying as cURL useful?

Share your findings: Each team shares one interesting API they found!

Part 2: Static Scraping with BeautifulSoup

Exercise 2.1: Quotes Scraper (20 min)

Website: http://quotes.toscrape.com

Task: Build a complete quotes scraper

```
import requests
from bs4 import BeautifulSoup

# TODO: Write code to scrape:
# 1. All quotes from the first page
# 2. Author of each quote
# 3. Tags for each quote
# 4. Save to a CSV file

# Starter code:
url = 'http://quotes.toscrape.com/'
response = requests.get(url)
```

Exercise 2.1: Solution

```python
import requests
from bs4 import BeautifulSoup
import csv

url = 'http://quotes.toscrape.com/'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

quotes_data = []

# Find all quote containers
quotes = soup.find_all('div', class_='quote')

for quote in quotes:
    text = quote.find('span', class_='text').text
    author = quote.find('small', class_='author').text
    tags = [tag.text for tag in quote.find_all('a', class_='tag')]

    quotes_data.append({
        'quote': text,
        'author': author,
        'tags': ', '.join(tags)
    })

# Save to CSV
with open('quotes.csv', 'w', newline='', encoding='utf-8') as f:
    writer = csv.DictWriter(f, fieldnames=['quote', 'author', 'tags'])
    writer.writeheader()
    writer.writerows(quotes_data)

print(f"Saved {len(quotes_data)} quotes to quotes.csv")
```

Exercise 2.2: Pagination (25 min)

Task: Extend your scraper to handle multiple pages

Challenge:

1. Scrape quotes from pages 1-10
2. Add a page number column
3. Include delay between requests
4. Handle the case when there are no more pages

Hints:

```python
import time

base_url = 'http://quotes.toscrape.com'
page = 1

while True:
```

Exercise 2.2: Solution

```python
import requests
from bs4 import BeautifulSoup
import csv
import time

base_url = 'http://quotes.toscrape.com'
all_quotes = []
page = 1

while page <= 10:  # Limit to 10 pages
    url = f'{base_url}/page/{page}/'
    print(f"Scraping page {page}...")

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

    quotes = soup.find_all('div', class_='quote')

    if not quotes:  # No more quotes
        print("No more pages!")
        break

    for quote in quotes:
        text = quote.find('span', class_='text').text
        author = quote.find('small', class_='author').text
        tags = [tag.text for tag in quote.find_all('a', class_='tag')]

        all_quotes.append({
            'page': page,
            'quote': text,
            'author': author,
            'tags': ', '.join(tags)
        })

    page += 1
    time.sleep(1)  # Polite delay

# Save to CSV
with open('all_quotes.csv', 'w', newline='', encoding='utf-8') as f:
    writer = csv.DictWriter(f, fieldnames=['page', 'quote', 'author', 'tags'])
    writer.writeheader()
    writer.writerows(all_quotes)

print(f"Total quotes scraped: {len(all_quotes)}")
```

Exercise 2.3: News Aggregator (45 min)

Task: Build a news headline aggregator

Website Options (choose one):

- https://news.ycombinator.com (Hacker News)
- https://www.reddit.com/r/technology (Reddit)
- https://www.bbc.com/news (BBC News)

Requirements:

1. Scrape headlines and links
2. Extract publish date/time if available
3. Get vote count or popularity metric
4. Save to CSV
5. Create a simple visualization with pandas

Exercise 2.3: Hacker News Solution

```python
import requests
from bs4 import BeautifulSoup
import pandas as pd
import matplotlib.pyplot as plt

url = 'https://news.ycombinator.com/'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

stories = []

# Find all story rows
story_links = soup.find_all('span', class_='titleline')
subtexts = soup.find_all('td', class_='subtext')

for link_span, subtext in zip(story_links, subtexts):
    link = link_span.find('a')
    title = link.text
    url = link['href']

    # Get points
    score = subtext.find('span', class_='score')
    points = int(score.text.split()[0]) if score else 0

    # Get author and time
    author = subtext.find('a', class_='hnuser')
    author_name = author.text if author else 'Unknown'

    stories.append({
        'title': title,
        'url': url,
        'points': points,
        'author': author_name
    })

# Create DataFrame
df = pd.DataFrame(stories)

# Save to CSV
df.to_csv('hackernews.csv', index=False)

# Visualize top stories
df.nlargest(10, 'points').plot(
    x='title', y='points', kind='barh',
    figsize=(10, 6), title='Top 10 HN Stories'
)
plt.tight_layout()
plt.savefig('top_stories.png')

print(f"Scraped {len(stories)} stories")
print(f"\nTop 5 stories:")
print(df.nlargest(5, 'points')[['title', 'points']])
```

Data Cleaning & Analysis (10 min)

Task: Clean your scraped data with pandas

```
import pandas as pd

df = pd.read_csv('hackernews.csv')

# Clean data
df['title'] = df['title'].str.strip()
df['points'] = df['points'].fillna(0).astype(int)

# Analysis
print("Statistics:")
print(df['points'].describe())

print("\nMost popular domains:")
df['domain'] = df['url'].str.extract(r'https?://([^/]+)')
print(df['domain'].value_counts().head())

print("\nMost prolific authors:")
print(df['author'].value_counts().head())

# Word frequency in titles
from collections import Counter
all_words = ' '.join(df['title']).lower().split()
common_words = Counter(all_words).most_common(20)
```

Part 2 Checkpoint

What You've Built

[YES] Quotes scraper with pagination
[YES] News aggregator with data analysis
[YES] CSV export and pandas integration
[YES] Basic visualization

Discussion (10 min)

Questions:

1. What was the hardest part of scraping?
2. Did you encounter any errors? How did you fix them?
3. What patterns did you notice in HTML structure?

Challenge: Who scraped the most data? Share your stats!

Part 3: Playwright for Dynamic Content

Exercise 3.1: First Playwright Script (15 min)

Task: Scrape Google search results

```python
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    # Launch browser (headless=False to see what's happening)
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()

    # Search on Google
    page.goto('https://www.google.com')

    # TODO:
    # 1. Find search box and type "web scraping python"
    # 2. Click search button
    # 3. Wait for results
```

Exercise 3.1: Solution

```python
from playwright.sync_api import sync_playwright
import time

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()

    # Go to Google
    page.goto('https://www.google.com')

    # Search
    page.fill('textarea[name="q"]', 'web scraping python')
    page.press('textarea[name="q"]', 'Enter')

    # Wait for results
    page.wait_for_selector('h3')

    # Extract results
    results = page.query_selector_all('h3')

    print("Top 10 results:")
    for i, result in enumerate(results[:10], 1):
        title = result.inner_text()
        # Get parent link
        link_elem = result.query_selector('xpath=ancestor::a')
        url = link_elem.get_attribute('href') if link_elem else 'No URL'

        print(f"{i}. {title}")
        print(f"   {url}\n")

    time.sleep(2)  # Pause to see results
    browser.close()
```

Exercise 3.2: Infinite Scroll (15 min)

Task: Scrape content from an infinite scroll page

Website: https://www.reddit.com or similar

```python
from playwright.sync_api import sync_playwright
import time

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()

    page.goto('https://www.reddit.com/r/programming/')

    # TODO:
    # 1. Scroll down 5 times to load more posts
    # 2. Wait between scrolls for content to load
    # 3. Extract all post titles
    # 4. Save to CSV

    # Hint: use page.evaluate() to scroll
```

Exercise 3.2: Solution

```python
from playwright.sync_api import sync_playwright
import time
import csv

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()

    page.goto('https://www.reddit.com/r/programming/')

    # Wait for initial content
    page.wait_for_selector('h3')

    # Scroll 5 times
    for i in range(5):
        print(f"Scroll {i+1}/5...")

        # Get current height
        previous_height = page.evaluate('document.body.scrollHeight')

        # Scroll to bottom
        page.evaluate('window.scrollTo(0, document.body.scrollHeight)')

        # Wait for new content
        time.sleep(2)

        # Wait for height to change (new content loaded)
        new_height = page.evaluate('document.body.scrollHeight')
        if new_height == previous_height:
            print("No more content to load")
            break

    # Extract all posts
    posts = page.query_selector_all('h3')

    results = []
    for post in posts:
        title = post.inner_text()
        results.append({'title': title})

    # Save to CSV
    with open('reddit_posts.csv', 'w', newline='', encoding='utf-8') as f:
        writer = csv.DictWriter(f, fieldnames=['title'])
        writer.writeheader()
        writer.writerows(results)

    print(f"Scraped {len(results)} posts")
    browser.close()
```

Exercise 3.3: Login & Authentication (15 min)

Task: Automate login to a website

Practice Site: http://quotes.toscrape.com/login

```python
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()

    # Go to login page
    page.goto('http://quotes.toscrape.com/login')

    # TODO:
    # 1. Fill in username (any username works)
    # 2. Fill in password (any password works)
    # 3. Click login button
    # 4. Verify you're logged in
    # 5. Navigate to a protected page
```

Exercise 3.3: Solution

```python
from playwright.sync_api import sync_playwright
import time

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()

    # Login
    page.goto('http://quotes.toscrape.com/login')

    page.fill('input[name="username"]', 'admin')
    page.fill('input[name="password"]', 'admin')
    page.click('input[type="submit"]')

    # Wait for redirect
    page.wait_for_url('**/quotes.toscrape.com/')

    # Verify login
    if page.query_selector('a[href="/logout"]'):
        print("[OK] Successfully logged in!")
    else:
        print("✗ Login failed")

    # Now scrape protected content
    response = page.goto('http://quotes.toscrape.com/')

    # Extract quotes (same as before, but now we're authenticated)
    quotes = page.query_selector_all('div.quote')

    for quote in quotes[:5]:
        text = quote.query_selector('span.text').inner_text()
        author = quote.query_selector('small.author').inner_text()
        print(f"{text}\n  — {author}\n")

    time.sleep(2)
    browser.close()
```

Part 3 Checkpoint

Playwright Skills Acquired

[YES] Browser automation basics
[YES] Form filling and clicking
[YES] Infinite scroll handling
[YES] Login automation
[YES] Dynamic content extraction

When to Use Playwright

Use Playwright when:

- Content loads via JavaScript
- Need to interact with page
- Infinite scroll or lazy loading

Part 4: Mini Project (30 minutes)

Your Own Scraping Project!

Task: Choose a website and build a complete scraper

Requirements

1. **Choose** a website you're interested in
2. **Decide** what data to collect
3. **Build** a scraper (Requests+BS4 or Playwright)
4. **Collect** at least 50 data points
5. **Clean** and analyze with pandas
6. **Visualize** one insight
7. **Present** to the class (2 min each)

Project Template

```python
"""
Project: [Your Project Name]
Goal: [What you're scraping]
Website: [URL]
"""

import requests
from bs4 import BeautifulSoup
import pandas as pd
import matplotlib.pyplot as plt

# Configuration
BASE_URL = 'https://...'
OUTPUT_FILE = 'data.csv'

# 1. Scraping function
def scrape_data():
    # Your scraping code here
    data = []
    # ...
    return data

# 2. Data cleaning
def clean_data(raw_data):
    df = pd.DataFrame(raw_data)
    # Clean data...
    return df

# 3. Analysis & Visualization
def analyze(df):
    print(df.describe())
    # Create plot...

# 4. Main execution
if __name__ == '__main__':
    raw_data = scrape_data()
    df = clean_data(raw_data)
    df.to_csv(OUTPUT_FILE, index=False)
    analyze(df)
```

Project Rubric

Grading Criteria (for reference)

Criteria	Points
Data collection works	30%
Code quality & comments	20%
Data cleaning	15%
Analysis/Visualization	15%
Follows ethical practices	10%
Presentation	10%

Ethical Checklist

- [YES] Checked robots.txt
- [YES] Added delays between requests
- [YES] Used appropriate User-Agent

Case Study 1: Price Tracking

Real-World Application

Scenario: Track laptop prices across e-commerce sites

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

def scrape_amazon_price(product_url):
    headers = {'User-Agent': 'Mozilla/5.0 ... '}
    response = requests.get(product_url, headers=headers)
    soup = BeautifulSoup(response.text, 'html.parser')

    # Find price element
    price = soup.find('span', class_='a-price-whole')
    if price:
        return float(price.text.replace(',', ''))
    return None

# Track multiple products
products = {
    'Laptop A': 'https://amazon.in/ ... ',
    'Laptop B': 'https://flipkart.com/ ... '
}

# Scrape daily and save to CSV
price_data = []
for name, url in products.items():
    price = scrape_amazon_price(url)
    price_data.append({
        'date': datetime.now()
```

Case Study 2: Research Paper Monitoring

Academic Use Case

Scenario: Track new papers on arXiv by topic

```python
import requests
from bs4 import BeautifulSoup
import re

def scrape_arxiv_recent(category='cs.AI', max_results=10):
    """Scrape recent papers from arXiv"""

    url = f'https://arxiv.org/list/{category}/recent'
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    papers = []
    paper_divs = soup.find_all('dt')
    desc_divs = soup.find_all('dd')

    for paper, desc in zip(paper_divs[:max_results], desc_divs[:max_results]):
        # Extract arXiv ID
        arxiv_id = paper.find('a', title='Abstract')
        if not arxiv_id:
            continue

        arxiv_id = arxiv_id.text.strip()

        # Extract title and authors
        title = desc.find('div', class_='list-title').text
        title = re.sub(r'^Title:\s*', '', title).strip()

        authors = desc.find('div', class_='list-authors').text
        authors = re.sub(r'^Authors:\s*', '', authors).strip()

        papers.append({
            'arxiv_id': arxiv_id,
            'title': title,
            'authors': authors,
            'url': f'https://arxiv.org/abs/{arxiv_id}'
        })
```

Case Study 3: Social Media Monitoring

Marketing Use Case

Scenario: Track brand mentions on Reddit

```python
from playwright.sync_api import sync_playwright
import time

def monitor_reddit_mentions(keyword, subreddit='all', limit=20):
    """Monitor Reddit for keyword mentions"""

    with sync_playwright() as p:
        browser = p.chromium.launch(headless=True)
        page = browser.new_page()

        # Search Reddit
        search_url = f'https://www.reddit.com/search/?q={keyword}&sort=new'
        page.goto(search_url)

        # Wait for results
        page.wait_for_selector('h3', timeout=10000)

        # Scroll to load more
        for _ in range(3):
            page.evaluate('window.scrollTo(0, document.body.scrollHeight)')
            time.sleep(2)

        # Extract posts
        posts = page.query_selector_all('div[data-testid="post-container"]')

        mentions = []
        for post in posts[:limit]:
            try:
                title_elem = post.query_selector('h3')
                if not title_elem:
                    continue

                title = title_elem.inner_text()

                # Get metadata
                author = post.query_selector('[data-testid="post_author_link"]')
                author = author.inner_text() if author else 'Unknown'

                # Get score
                score = post.query_selector('[id^="vote-arrows"]')
                score_text = score.inner_text() if score else '0'

                mentions.append({
                    'title': title,
                    'author': author,
                    'score': score_text
                })
```

Debugging Common Issues

Problem 1: Empty Results

```python
# Issue: soup.find_all() returns empty list
soup.find_all('div', class_='item')  # []

# Debug steps:
print(soup.prettify())  # See actual HTML
print(len(response.text))  # Check if page loaded

# Common causes:
# 1. Wrong selector
# 2. JavaScript-rendered content (use Playwright)
# 3. Need to login first
# 4. Blocked by anti-scraping (add headers)
```

Problem 2: 403 Forbidden

Debugging Common Issues (Continued)

Problem 3: Element Not Found

```python
# Issue: AttributeError: 'NoneType' object has no attribute 'text'

# Bad code:
title = soup.find('h1').text  # Crashes if h1 not found

# Good code:
title_elem = soup.find('h1')
if title_elem:
    title = title_elem.text
else:
    title = 'No title found'

# Or use get_text with default:
title = soup.find('h1').get_text(default='No title') if soup.find('h1') else 'No title'

# Better: Use try-except
try:
    title = soup.find('h1').text
```

Best Practices Checklist

Before You Scrape

- [ ] Check if API exists (easier than scraping!)
- [ ] Read `robots.txt` (site.com/robots.txt)
- [ ] Review Terms of Service
- [ ] Check data usage licenses
- [ ] Plan your rate limiting strategy

While Scraping

- [ ] Use descriptive User-Agent
- [ ] Add delays between requests (1-2s minimum)
- [ ] Handle errors gracefully
- [ ] Log your activities

Presentation Guidelines

What to Present (2 minutes)

1. **What** did you scrape? (10 sec)
    - Website and data type
2. **How** did you scrape it? (30 sec)
    - Requests or Playwright?
    - Main challenges overcome
3. **What** did you find? (60 sec)
    - Show your data (table or visualization)
    - One interesting insight
4. **Code snippet** (20 sec)

Sample Presentation Structure

~~~
# Project: IITGN Mess Menu Scraper

## What
Scraped weekly mess menu from IITGN website

## How
- Used Requests + BeautifulSoup
- Challenge: Menu in table format with merged cells
- Solution: Used pandas.read_html()

## Findings
- 73 unique dishes over 4 weeks
- Most common: "Dal Fry" (appears 12 times)
- Visualization: Word cloud of dishes

## Code Highlight
```python
# pandas.read_html() saved the day!
tables = pd.read_html(url)
menu_df = tables[0]  # First table is the menu
~~~

Resources for Continued Learning

Documentation

- **Requests:** https://docs.python-requests.org/
- **BeautifulSoup:** https://www.crummy.com/software/BeautifulSoup/
- **Playwright:** https://playwright.dev/python/
- **pandas:** https://pandas.pydata.org/docs/

Practice Sites

- http://quotes.toscrape.com - Practice scraping
- http://books.toscrape.com - E-commerce practice
- https://scrapethissite.com - Various challenges
- https://webscraper.io/test-sites - Test different patterns

Advanced Topics (Next Week)

Lab Submission

What to Submit

1. **Code** (Python files or Jupyter notebook)
    - Well-commented
    - Includes requirements.txt
2. **Data** (CSV file)
    - Cleaned and formatted
    - At least 50 data points
3. **Report** (README.md or PDF)
    - What you scraped
    - How you did it
    - One insight/visualization

Homework for Next Week

Assignments

1. **Extend your scraper**
    - Collect 200+ data points
    - Add error handling
    - Create 3 visualizations
2. **Read**
    - Requests documentation (focus on sessions)
    - BeautifulSoup CSS selectors guide
    - Web scraping ethics article (link on course site)
3. **Install for next week**

Great Work Today!

You've Learned:

[YES] Chrome DevTools for finding APIs
[YES] Web scraping with Requests + BeautifulSoup
[YES] Browser automation with Playwright
[YES] Data collection, cleaning, and analysis

Next Week:

Data Validation, Labeling, and Quality Control

Questions? Office hours tomorrow 3-5 PM

Quick Feedback (5 min)

Anonymous Poll

1. **Pace**: Too fast / Just right / Too slow
2. **Difficulty**: Too easy / Just right / Too hard
3. **Most useful**: DevTools / Requests / Playwright
4. **What to improve**: [Open feedback]

Scan QR code or visit:

[feedback link]

Thank you! See you next week!