CADT

Cambodia Academy of Digital Technology

Web Search and Web Scraping

Lecturer

Mr. Natt KORAT

Lecturer-Researcher,

Institute of Digital Research & Innovation (IDRI),

Cambodia CADT Academy of Digital Technology (CADT)

Recap: where did the docs come from?

- A fixed, hand-built 30-document collection
- Real search engines do not get their documents handed to them
- Someone has to COLLECT them first

Web search, crawling, and scraping

Concept	Main question
Crawling	Which URLs should we discover or visit next?
Scraping	Which fields do we extract from fetched pages?
Indexing	How do we organize documents for fast retrieval?
Web search	Which documents answer a user query?

Today’s pipeline

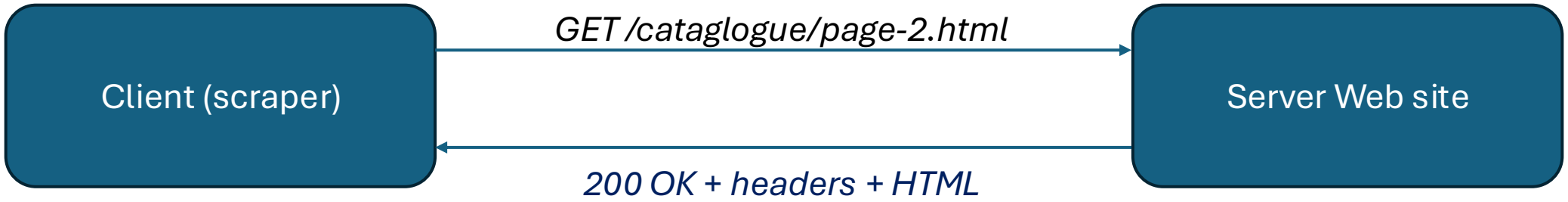
- URL -> HTTP request -> HTML response -> parse -> structured record -> (eventually) index
- Each arrow is one part of Lab 5

URL anatomy

- https://books.toscrape.com/catalogue/page-2.html?sort=title#top
- Scheme: https://
- Host: books.toscrape.com
- Path: /catalogue/page-2.html
- Query: ?sort=title
- Fragement: #top

HTTP: a request and a response

- Client sends a REQUEST (method, URL, headers)
- Server sends back a RESPONSE (status code, headers, body/conent)
- **GET**: fetch a page. (POST, PUT, etc. exist but are rare in scraping.)

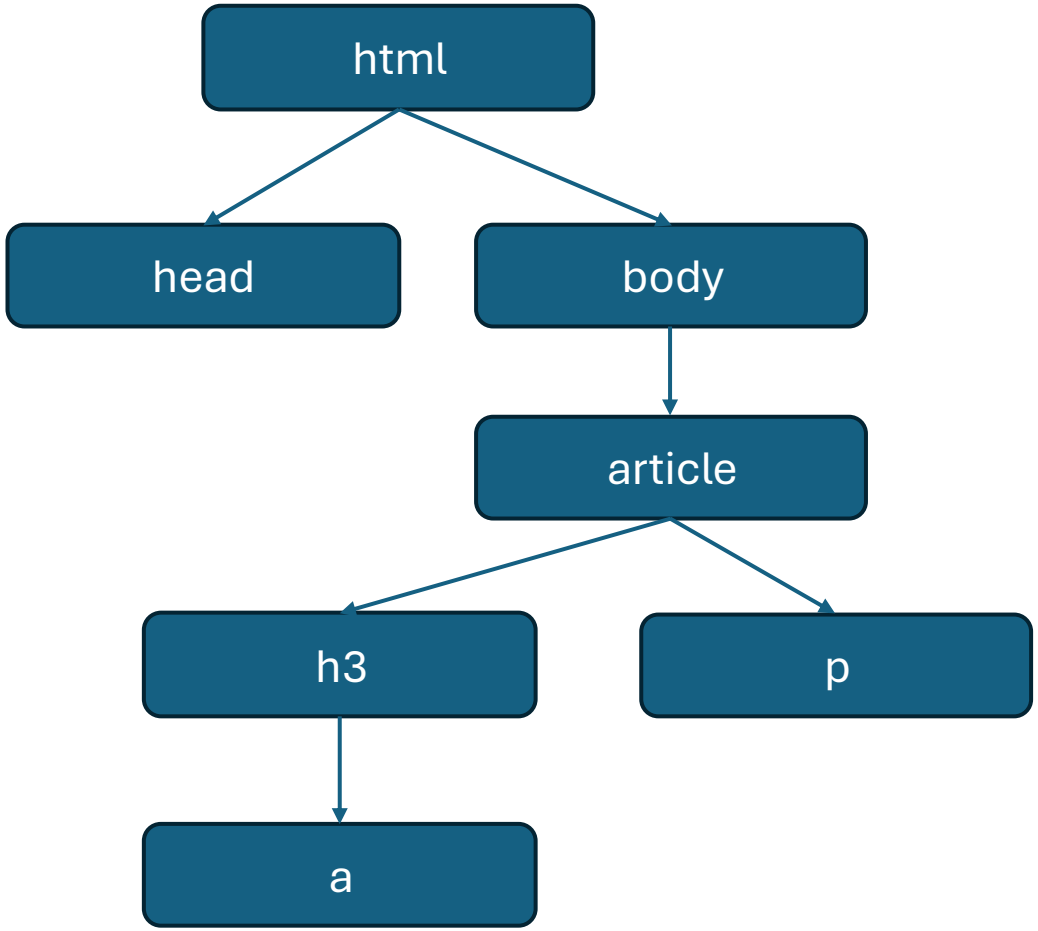


HTTP status codes to recognize

Status	Meaning / crawler response
200	OK –> inspect response body
401 / 403	Authentication required / forbidden -> do not bypass
404	Not found -> log and skip or fix URL
429	Too many request -> back off, check retry-after
5xx	Server problem -> stop or retry cautiously

HTML and the DOM

```
<html>
  <head>...</head>
  <body>
    <article class="product">
      <h3>
        <a title="My book" href="detail.hmt">my book</a>
      </h3>
      <p class="price">$12.50</p>
    </article>
  </body>
</html>
```



Finding content in markup

- Tags, classes and ids are the “hook” a scraper grabs onto
- **CSS selectors**: the same syntax used to STYLE a page can be used to SELECT from it
- **Browser**: view page source / dev tools elements panel: you can find these hooks on the real site

Live code: download one page

```python
import requests

url = "https://books.toscrape.com"
r = requests.get(
    url=url,
    headers={
        "User-Agent": "CADT-IR-W5/1.0"
    },
    timeout=20
)

r.raise_for_status

print(r.status_code, r.url)
print(r.text[:100])
```

A simple reqeust from Python code

```python
from bs4 import BeautifulSoup
from urllib.parse import urljoin

soup = BeautifulSoup(r.content, "html.parser")
for card in soup.select("article.product_pod"):
    link = card.select_one("h3 a")
    title = link.get("title")
    href = link.get("href")
    price = card.select_one("p.price_color").get_text(strip=True)

    print(title, price, href)
```

Parsing content using BeautifulSoup

A limitation: JavaScript-render pages

Server-rendered HTML

- HTML response cntains the data
- Request + BeautifulSoup can read it
- Our Books to Scrape works here

Client-rendered content

- Initila HTML may be an emtpy shell
- JavaScript loads fields later
- Use an API or browser tool when permitted

Tool like Selenium/Playwright exist for this, but are beyornd today's scope (explore yourselves)

robots.txt: instructions for crawlers

- https://example.com/robots.txt
- Published at the wbsite root
- Contains rules for matching user agents and URL paths
- Not a login mechanism or a content license
- Respect rules before fetching project pages

Read a robots.txt file

```
User-agent: *
Disallow: /private/
Allow: /public/

# Optional non-standard extension on some sites:
Crawl-delay: 2
```

What does this say?

- All bots match *
- Do not crawl /private/
- /public/ is allowed
- Crawl-delay support varies

Real example: https://en.wikipedia.org/robots.txt

Check robots programmatically

```python
from urllib import robotparser

url = "https://en.wikipedia.org/robots.txt"

rp = robotparser.RobotFileParser(url)
rp.read()

allowed = rp.can_fetch(
    "*",
    "https://en.wikipedia.org/api/rest_v1/"
)

delay = rp.crawl_delay("*")
print(allowed, delay)
```

Politeness is more than robots.txt

- Limit the crawl scope: 2 listing pages, not 50
- Wait between requests; keep concurrency low
- Identify the crawler with a descrptive User-Agent
- Honor 429 responses and stop rather than evade restrictions
- User an official API or cached data where possible

What makes the real Web difficult?

- Millions of pages change continuously
- Duplicate and near-duplicate content
- Noisy HTML, missing fields, incosistent formats
- JavaScript-rendered content and complex URLs
- Terms, privacy, rate limits, and access rules

Permission, copyright, and privacy

Before collection:

- Review terms, licenses, robots
- Prefere an offical API when available
- Verify redistribution conditions

Before publication:

- Do not publish content withous rights
- Minimize personal information
- Store data securely with retention rules

Recap

- HTTP request/response, HTML and the DOM are the raw material of scraping
- requests + BeautifulSoup: fetch, then parse (Scrapy is a real proejct)
- robots.txt states what is allowed; politeness and terms of use go further
- Static scraping cannot see JavaScript-rendered content