

Data, Data Storage, Data Collection

Lecture 4: Programmatic Data Collection

Romain Pascual

MICS, CentraleSupélec, Université Paris-Saclay

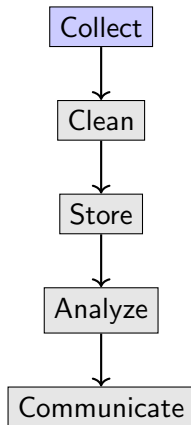
Recap

Recap: Generating and Measuring Data

- ① Data can be observational, experimental, or simulated, each granting a different relationship to the phenomenon
- ② Whichever mode, we only ever approximate the phenomenon.
- ③ Measurement error decomposes into bias and variance (MSE)
- ④ Sampling too slowly in time hides or disguises fast variation (aliasing)
- ⑤ Data collection error has two sides: representation (Session 2) and measurement (this session)

Introduction

Within The Lifecycle



Session Objectives

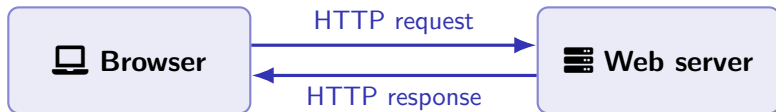
At the end of this session, you should be able to:

- explain the basic structure of an HTTP request and response;
- use an HTTP API to retrieve data;
- explain the basic idea behind web scraping;
- identify some practical and ethical limitations of automated collection;
- document collected data so that it can be found, accessed, interpreted and reused.

HTTP

HTTP: HyperText Transfer Protocol

When we collect data from a website, whatever mechanism we use relies on the same underlying exchange:



The browser sends a request to a URL and receives data in response. The protocol used for this exchange is **HTTP**.

A (Python) program can also be an HTTP client.

The HTTP Request

An HTTP request tells a server **what we want**.

Method	what kind of operation? GET, POST, PUT, DELETE
URL	where should the request go?
Parameters	what exactly are we asking for?
Headers	additional information about the request
Body	additional data, when needed

For example:

<u>GET</u>	<u>https://example.org/weather</u>	<u>?city=Paris</u>
Method	URL	Parameters

The HTTP Response

The server sends an HTTP response back to the client.

Status code	did the request succeed?
Headers	information about the response
Body	the content returned by the server

200	OK
401	Authentication required
403	Forbidden
404	Not found
429	Too many requests
500	Server error

Important

A successful HTTP request does not necessarily mean that the data is correct.

APIs

What Is an API?

An **API (Application Programming Interface)** is an interface that allows software components to interact through a defined set of operations. It specifies:

- What can be requested,
- How to request it,
- What is returned.



HTTP APIs

Expose **endpoints**: GET <https://example.org/weather>
Parameters refinement: ?city=Paris

Parameters in Python

requests let us specify query parameters separately from the URL.

```
1 import requests
2
3 response = requests.get(
4     "https://example.org/weather",
5     params={
6         "city": "Paris",
7     }
8 )
```

The library constructs the corresponding URL:

`https://example.org/weather?city=Paris`

Why use params?

Let the library handle URL encoding and parameter formatting.

(Semi-)Structured Responses

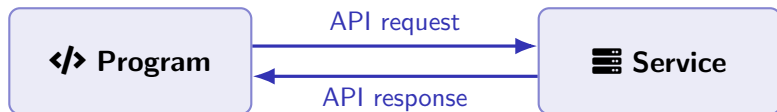
APIs often return data in a machine-readable format such as JSON, CSV or XML.

```
1 {  
2     "city": "Paris",  
3     "temperature": 21.4,  
4     "unit": "C"  
5 }
```

In Python:

```
1 data = response.json()  
2 temperature = data["temperature"]
```

API Constraints



 An API defines an interface, but the service still controls access:

- authentication or an API key;
- an account or specific permissions;
- rate limits or quotas;
- pagination for large results;
- restrictions on historical data or request size.

API: A Typical Collection Workflow

- 1 Identify the API and its documentation
- 2 Identify the endpoint(s) we need
- 3 Determine parameters and authentication requirements
- 4 Send requests
- 5 Check the responses
- 6 Parse the returned data
- 7 Store the collected data

Web Pages

HTML, CSS and JavaScript

The server typically sends a response formatted in **HTML**:

```
1 <h1>Paris</h1>
2 <p>
3     The temperature is
4     <span class="temperature">21.4</span>
5     degrees.
6 </p>
```

HTML (HyperText Markup Language) uses annotations to describe the structure and content of a document.

 A modern web page typically combines three technologies:

HTML

structure
and content

CSS

presentation
and layout

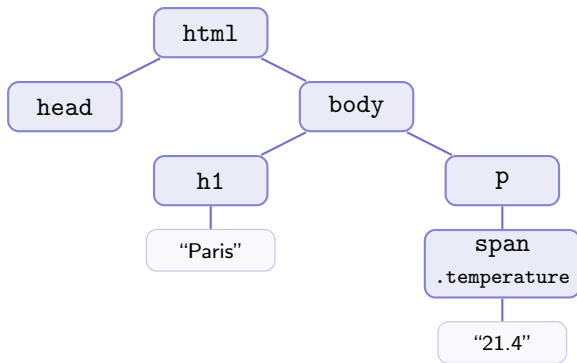
JavaScript

behaviour
and interaction

CSS classes and IDs are also useful for **locating** elements when scraping.

The DOM: A Tree Representation

The HTML **Document Object Model (DOM)** allows treating an HTML document as a tree structure: the document is the root while elements, attributes, text, and comments are nodes.



We can navigate the tree to find the elements containing the information we want!

Web Scraping

Web Scrapping

Web scraping is the automated extraction of information from a web page



```
1 <p>
2   The temperature is
3   <span class="temperature">
4     21.4
5   </span>
6   degrees .
7 </p>
```

`.temperature`
is a **CSS selector**
that identifies the
element with class
temperature.

⚠ The scraper works with the **HTML/DOM representation**, not with the pixels displayed on the screen.

Parsing HTML in Python

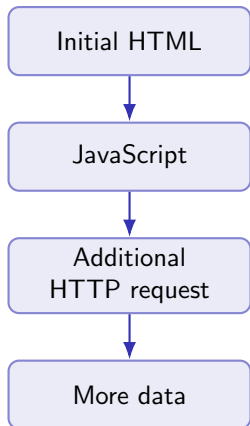
BeautifulSoup can parse HTML and navigate the DOM.

Python

```
1 from bs4 import BeautifulSoup
2
3 soup = BeautifulSoup(
4     response.text,
5     "html.parser"
6 )
7
8 elements = soup.select(".temperature")
9
10 for element in elements:
11     print(element.get_text())
```

Here `.temperature` is a CSS selector.

Dynamic Pages



The browser may execute JavaScript that makes additional HTTP requests.

The data we see in the browser may come from a **later request**, not the initial HTML.

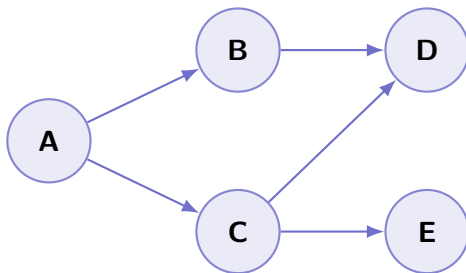
What to do

Inspect the browser's network requests: there might be an **API** that we can call directly.

 Browser automation and JavaScript execution are beyond the scope of this lecture.

Following Links

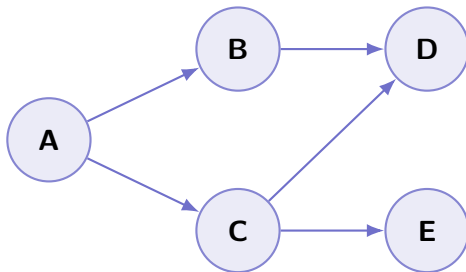
Suppose we want to collect information from many pages.



We can therefore view a website as a **graph** where pages are nodes and hyperlinks are edges.

Following Links

Suppose we want to collect information from many pages.



We can therefore view a website as a **graph** where pages are nodes and hyperlinks are edges.

A **crawler** builds and **traverses** this graph, repeatedly downloading pages, extracting content to find new links and following them.

Automated Collection: What Can Go Wrong?

Automated collection makes it easy to collect large amounts of data. But three questions matter:

① Can we access it?

Authentication, access restrictions, rate limits, blocking

② Should we collect it?

Terms of service, crawling policies (`robots.txt`), licensing, personal data

③ Will our collection remain reliable?

Changing HTML, URLs or APIs, dynamic content

Technically possible does not mean appropriate, permitted, or reliable.

API or Web Scraping?

	API	Web scraping
Designed for programs	Yes	Usually no
Structured response	Usually	Not necessarily
Documentation	Often	Rarely
Stable interface	Often	Not guaranteed
Easy to target specific data	Often	Depends on page
Access restrictions	Yes	Yes

General rule

Prefer an official API when an appropriate one is available.

Provenance & Documentation

The FAIR Data Principles

Mark D. Wilkinson et al. “The FAIR Guiding Principles for Scientific Data Management and Stewardship”. In: **Scientific Data** 3.1 (Mar. 2016). ISSN: 2052-4463. DOI: 10.1038/sdata.2016.18

Motivation:

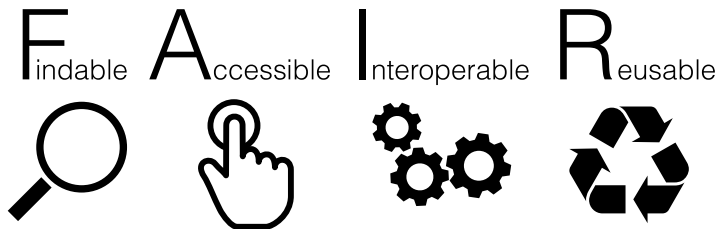
- Digital data abundant but hard to find, access, combine, or reuse
- Need for reproducible, interoperable, trustworthy data

FAIR principles guide good data management to make datasets usable by both computers and humans. It is about both the **data itself** and its **metadata**.

The FAIR Principles: Making Data Usable

Data should be:

- **Findable:** Easy to locate
- **Accessible:** Retrievable with standard tools
- **Interoperable:** Usable with other datasets
- **Reusable:** Well-documented for future use



How to Make Data FAIR

Findable, **A**ccessible, Interoperable, **R**eusable

Which is the principle associated with each advice?

- Use **standard formats** (CSV, JSON)
- Include a **clear license** (e.g., Creative Commons)
- Assign a **DOI** (e.g., via <https://zenodo.org/>)
- Allow downloading over **standard protocols** (e.g., HTTP, FTP)
- Document with **metadata standards** (e.g., Dublin Core)

How to Make Data FAIR

Findable, **A**ccessible, Interoperable, **R**eusable

Which is the principle associated with each advice?

- (I) Use **standard formats** (CSV, JSON)
 - Include a **clear license** (e.g., Creative Commons)
 - Assign a **DOI** (e.g., via <https://zenodo.org/>)
 - Allow downloading over **standard protocols** (e.g., HTTP, FTP)
 - Document with **metadata standards** (e.g., Dublin Core)

How to Make Data FAIR

Findable, **A**ccessible, Interoperable, **R**eusable

Which is the principle associated with each advice?

- (I) Use **standard formats** (CSV, JSON)
- (R) Include a **clear license** (e.g., Creative Commons)
 - Assign a **DOI** (e.g., via <https://zenodo.org/>)
 - Allow downloading over **standard protocols** (e.g., HTTP, FTP)
 - Document with **metadata standards** (e.g., Dublin Core)

How to Make Data FAIR

Findable, **A**ccessible, Interoperable, **R**eusable

Which is the principle associated with each advice?

- (I) Use **standard formats** (CSV, JSON)
- (R) Include a **clear license** (e.g., Creative Commons)
- (F) Assign a **DOI** (e.g., via <https://zenodo.org/>)
 - Allow downloading over **standard protocols** (e.g., HTTP, FTP)
 - Document with **metadata standards** (e.g., Dublin Core)

How to Make Data FAIR

Findable, **A**ccessible, Interoperable, **R**eusable

Which is the principle associated with each advice?

- (I) Use **standard formats** (CSV, JSON)
- (R) Include a **clear license** (e.g., Creative Commons)
- (F) Assign a **DOI** (e.g., via <https://zenodo.org/>)
- (A) Allow downloading over **standard protocols** (e.g., HTTP, FTP)
 - Document with **metadata standards** (e.g., Dublin Core)

How to Make Data FAIR

Findable, **A**ccessible, Interoperable, **R**eusable

Which is the principle associated with each advice?

- (I) Use **standard formats** (CSV, JSON)
- (R) Include a **clear license** (e.g., Creative Commons)
- (F) Assign a **DOI** (e.g., via <https://zenodo.org/>)
- (A) Allow downloading over **standard protocols** (e.g., HTTP, FTP)
- (R) Document with **metadata standards** (e.g., Dublin Core)

What to Document: The Essentials

Who?

collector

When?

date / time

Where?

source / endpoint

How?

code / method

What?

variables / meaning

Why?

purpose / selection

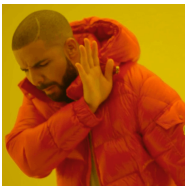
For programmatic collection:

- the endpoint/URL and version;
- the parameters used;
- the **raw** response, saved before parsing;
- any authentication or access constraint.

Conclusion

Takeaways

- ① **HTTP** is the common mechanism behind downloading, calling an API, and scraping a page
- ② **APIs** provide a structured interface for programs.
- ③ **Web scraping** extracts data from web representations such as HTML.
- ④ **Crawling** extends scraping across linked pages.
- ⑤ **Reproducibility** requires documenting how, when and what you collected, even for automated collection.



Manually Scraping Data From Medium



Using the Medium API

see <https://mediumapi.com/manual-web-scraping-vs-medium-api.html>

```
1 import requests
2 image_url = "https://mediumapi.com/imgs/blogs/scrape-
  medium-data-vs-medium-api.png"
3 img_data = requests.get(image_url).content
4 with open('drake.png', 'wb') as handler:
5     handler.write(img_data)
```