# Course notes:
# Web scraping and API fundamentals in Python

# Web Scraping

## Definition

Web scraping, also known as web data extraction, is formally known as the process of obtaining and structuring data from the web using intelligent automation.

It can be used to potentially retrieve hundreds, millions, or even billions of data points from the internet's seemingly endless frontier.

## Ethics of scraping:

- Do not engage in piracy or other unauthorized commercial use regarding the data you extract.
- Read the ToS and robots.txt of the site you are about to scrape.
- Do not spam the website with multiple requests in a short amount of time, as that may hurt them and/or may be classified as a DDOS attack.

# HTTP requests

On the web, servers and clients usually communicate through HTTP requests.

HTTP stands for 'HyperText Transfer Protocol' and it specifies how requests and responses are to be formatted and transmitted. These requests are how most of your web surfing is happening. When opening a page, the browser sends a request to the server of that page, and the server responds with the relevant resources (HTML, images, etc.).

The two most popular request types are **GET** and **POST.**

## GET

- Obtain data from server
- Can be bookmarked
- Parameters are added directly into the URL
- Not used to send sensitive info (such as passwords)

## POST

- Usually used when a state needs to be altered (such as adding items to you shopping cart) or when sending passwords
- Parameters are added in a separate body, thus it is more secure
- Cannot be bookmarked

**Request headers** are pieces of information about the request itself - information, such as the encoding and language of the expected response, the length and type of data provided, who is making the request, cookies and so on. These pieces of information, referred to as headers, are intended to make communications on the web easier and more reliable, as the server has a better idea of how to respond.

Two of the most common header fields are the *User-Agent* (identification string for the software making the request) and *cookies* (special type of header that has a variety of uses).

# Response

The response contains 2 main pieces of information – the **status code** and the **body of the response**.

The status code indicates whether the request was successful and/or any errors. It is represented by a 3-digit number.

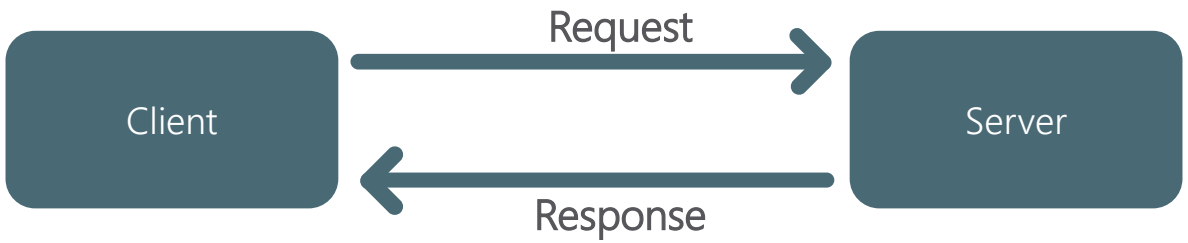
Codes in the ranges indicate:

- **2xx** – Success
- **3xx** – Redirects
- **4xx** – Client errors
- **5xx** – Server errors

The two most frequently encountered status codes are:

- **200 OK** – The request has succeeded
- **404 Not Found** – The server can not find the requested resource

The **body of the response** contains the requested information. Usually, it is either an HTML or a JSON file.



# JSON

JSON stands for 'JavaScript Object Notation' as it was derived from the JavaScript programming language. It is a standard for data exchange on the web.

The JSON format relies on 3 key concepts: it should be easy for humans to read and write; easy for programs to process and generate, regardless of the programming language; and, finally: written in plain text.

It achieves that by using conventions familiar to almost all programmers, by building upon 2 structures: dictionaries and lists.

## Dictionaries

A dictionary is a data structure that contains key-value pairs, surrounded by curly brackets

```
{
   "key 1": "value 1",
   "key 2": "value 2"
    …
}
```

## Lists

A list, on the other hand, is a collection of items. It is contained inside square brackets.

```
[ "item 1", "item 2" … ]
```

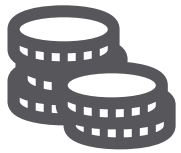
# API Overview

An API is an **Application Programming Interface**.

An API specifies how software components should interact. You may think of it as a contract between a client and a server. If the client makes a request in a specific format, the server will always respond in a documented format or initiate a defined action. Web-based APIs usually provide information.

Some APIs are free, most are either paid, or require registration. In the latter case, you are usually given a KEY and ID that must be incorporated into every request to that API.

**Examples of APIs:**



- Up-to-date currency exchange rates

- Real-time weather data and forecast

- Job listings

- Nutrition analysis

# HTML Overview

**HTML** is the underlying source code of all webpages, along with CSS and JavaScript.
It consists of nested **elements/tags**.


An **element** has the following syntax: `<tag_name> content </tag_name>`

These *elements may have additional information specified in tag attributes*.

Popular **tag attributes** are:

- 'class'
- 'id'

Example:

- `<div class="descriptive-class-name">….</div>`

**Popular tags** are:

- link - `<a href="url_of_link">Text of the link</>`
- paragraph - `<p>…</p>`
- tag to group other tags - `<div class="hahaha">…</div>`
- tag to mark part of the content - `<span id="hohoho">…</span>`

# Beautiful Soup

Beautiful Soup is a Python library for extracting data from an HTML document.
It achieves that by analyzing the HTML with a parser.

The most useful methods of the package are:

- `.find("p", class="footer-text")` – extracts the first paragraph tag with class = "footer-text"
- `.find_all("a")` – extracts all links from the page
- `tag.attrs` – returns a dictionary of all attributes of this tag
- `tag.text` – returns all the text in this tag
- `tag.contents` – returns a list of all children elements of this tag

# Common roadblocks when scraping

Identification:

Some websites may require us to identify ourselves – the solution is to **set the "User-Agent" request header to one of the common browsers.**

Cookies:

Other websites may require us to set cookies – solution: **use the session class of the request's library**.

Login:

Occasionally, the data we want to scrape may be locked behind a login. In that case, we need to **simulate a login attempt by inspecting the POST request** that is sent and its parameters.

Excessive requests:

Another roadblock may occur when we send too much requests in a short amount of time to a server. Therefore, we may get blocked. Out of ethical standpoint as well, it is a good idea to limit our rate of requests. We can simply do that with the sleep function of the time package -> **time.sleep(2).**

# Requests-html – dealing with JavaScript

The requests-html package was intended as a replacement of the requests + Beautiful Soup combo. However, its strongest point is that it has **full JavaScript support**, meaning it can execute JavaScript. This allows us to scrape dynamically generated content.

To run the JavaScript, we can use the render method:

```
from requests_html import HTMLSession
session = HTMLSession()
r = session.get("example.com")
r.html.render()
```