

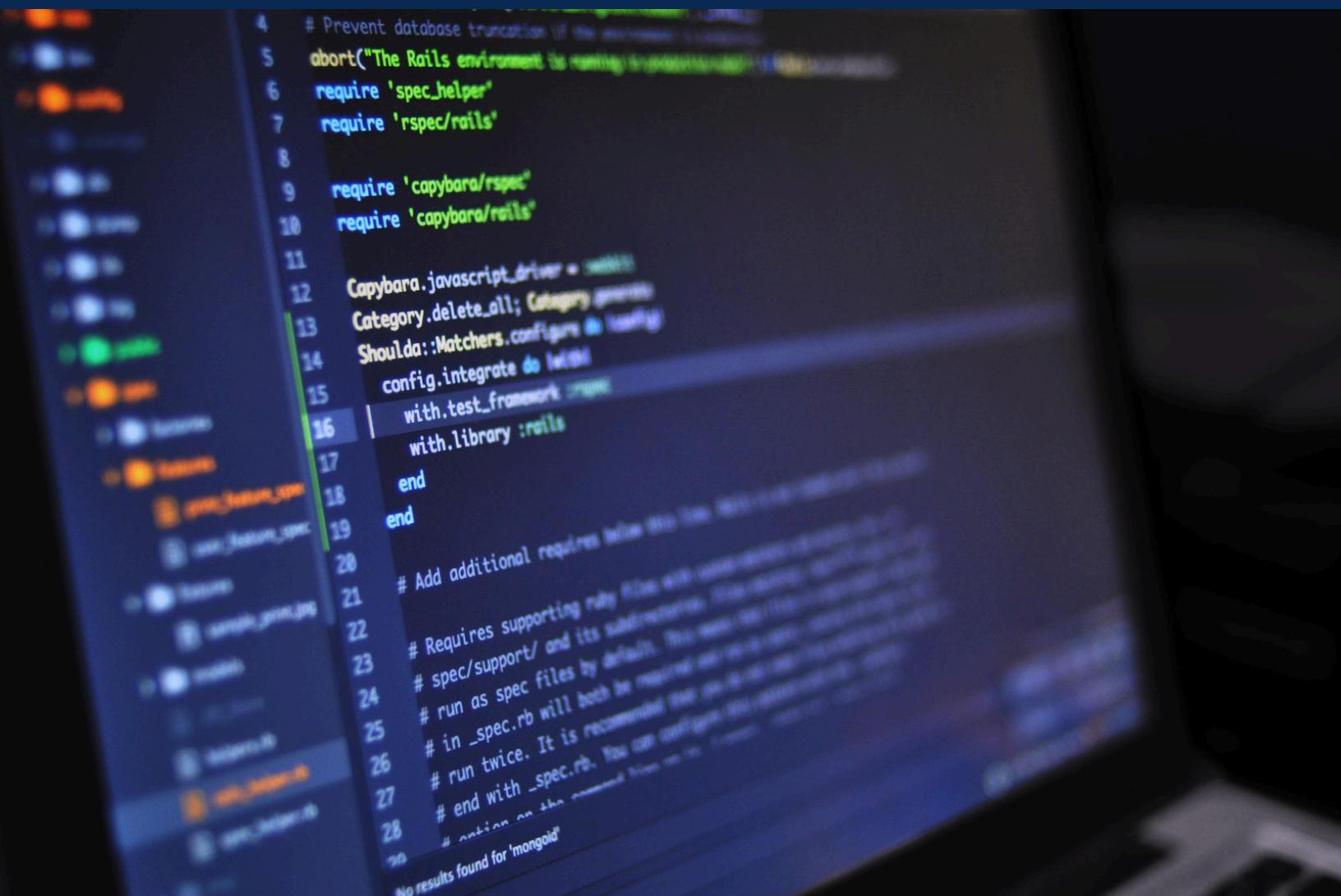
# WEB SCRAPING

## HTML & JS

ALLA HARUTYUNYAN  
PROYECTO PÍLDORA 24 FEBRERO DE 2025

README.md en mi GITHUB

# PRESENTATION

A blurred screenshot of a computer monitor showing a code editor with Ruby code for web scraping. The code includes require statements for 'spec\_helper', 'rspec/rails', 'capybara/rspec', and 'capybara/rails'. It also includes configuration for Capybara's JavaScript driver and database truncation. A sidebar on the left shows a file tree with files like 'spec\_helper.rb' and 'spec/rails.rb'.

Today we are going to talk about **web scraping** with **HTML** and **JS**.

It is a set of practices used to **automatically extract data** from websites.



+





## ¿CUÁNDO SE SUELE UTILIZAR ESTA TÉCNICA?

Sepuede scrapear una página web por varios motivos:

- Estudio de mercado
- Generación de leads para marketing
- Seguimiento de precios

NO TODOS LOS SITIOS WEB  
PERMITEN EL SCRAPING Y HAY  
VARIAS FORMAS DE AVERIGUARLO.

# ESTRUCTURA DE UNA PÁGINA WEB CON HTML

## HTML

```
<body>
  <header>
    <h1 class="titulo">Esto es un h1 </h1>
    <ul class="lista-no-ordenada">
      <li><a href="#seccion1">Elemento 1</a></li>
      <li><a href="#seccion2">Elemento 2</a></li>
    </ul>
  </header>

  <div class="conetenedor">
    <h4>Esto es un h4 dentro de un contenedor</h4>
  </div>

  <section id="seccion1">
    <h2 class="subtitulo">Esto es un h2</h2>
    <p class="parrafo">Este es un párrafo con</p>
    
  </section>

</body>
```

## DOM

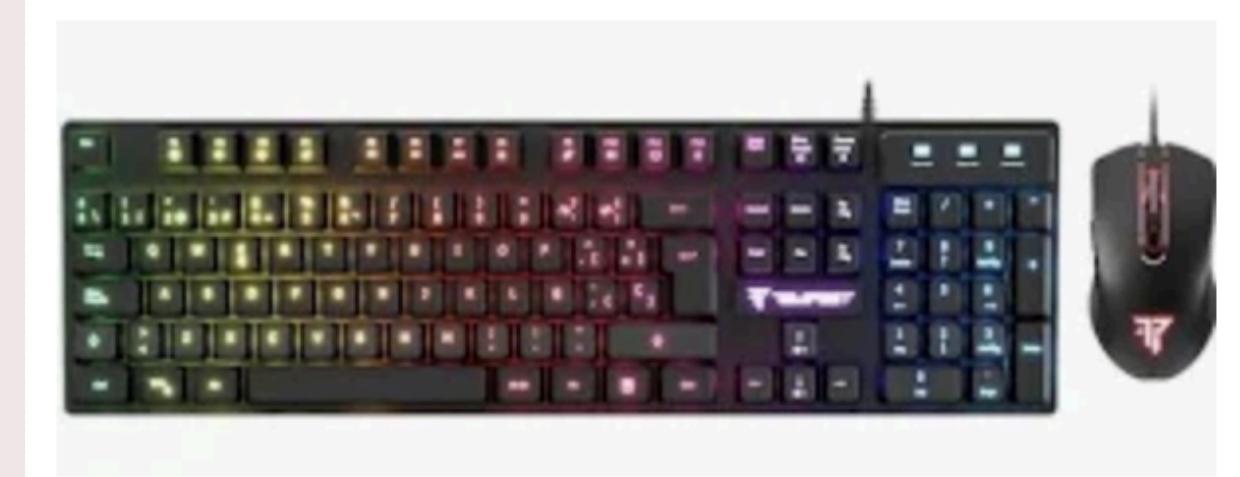
**Esto es un h1**

- [Elemento 1](#)
- [Elemento 2](#)

**Esto es un h4 dentro de un contenedor**

**Esto es un h2**

Este es un párrafo con



# CONSOLA + JS = SCRAPING

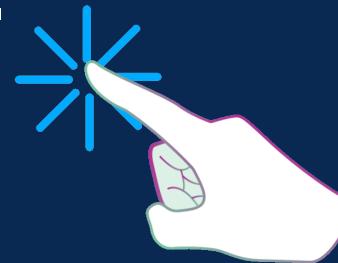
La consola web es parte de las herramientas de desarrollo que nos proporciona el navegador, y es clave para el scraping, ya que junto con JavaScript que es un lenguaje que el navegador puede interpretar podemos hacer peticiones y scripts en linea desde el propio navegador.

Los scripts que podemos usar en la consola para hacer scraping son los siguientes:

- `document.getElementById("id_del_elemento") # elemento por su id`
- `document.getElementsByClassName("clase_del_elemento") # todos con una clase específica`
- `document.querySelectorAll(".clase_del_elemento") # todos con una clase específica`
- `document.querySelector(".clase_del_elemento") # el primer elemento con una clase específica`
- `document.getElementsByTagName("h1") # por etiqueta, h1, h2, p, etc`
- `document.querySelectorAll("[atributo='valor']") # por atributo=valor`
- `document.querySelector("h1").textContent # el texto en un h1`
- `document.querySelector("h1").innerHTML # el código html en un h1`

HAGAMOS UN POCO  
DE SCRAPING

[books.toscrape.com](http://books.toscrape.com)



# LIBRERÍAS JS PARA SCRAPING

## PUPPEETEER VS CHEERIO

Característica	Puppeteer	Cheerio
Motor de renderizado	Uso un navegador real (Chrome/Chromium)	Solo procesa HTML como texto
Soporte para JS	Sí (carga dinámicamente JS, CSS, AJAX)	No (solo HTML estático)
Velocidad	Lento (abre un navegador)	Súper rápido
Uso de recursos	Alto consumo de CPU/RAM	Ligero
Interacción con la página	Puede hacer clic, llenar formularios, tomar capturas	No puede interactuar
Facilidad de scraping	Funciona con cualquier página (JS, AJAX, etc.)	No puede scrapear contenido cargado con JS

MUCHAS GRACIAS

ALLA HARUTYUNYAN

```
render() {
  return (
    <React.Fragment>
      <div className="py-5">
        <div className="container">
          <Title name="our" title="product">
          <div className="row">
            <ProductConsumer>
              {(value) => {
                |   |   |   console.log(value)
                |   |   |
              }}
            </ProductConsumer>
          </div>
        </div>
      </div>
    <React.Fragment>
```