# Client-Side Web Technologies – HTML

## Databases

Pierre Senellart

PSL UNIVERSITÉ PARIS


13 May 2026

# Outline

# Where we are

- We have a relational database (SQLite, MySQL, PostgreSQL...)
- We can query it from Python with SQL or pandas
- But the user is not going to type SQL or Python at a terminal

# Where we are

- We have a relational database (SQLite, MySQL, PostgreSQL...)
- We can query it from Python with SQL or pandas
- But the user is not going to type SQL or Python at a terminal

We need an interface: a way for a non-technical user to

- see the data (rows, tables, lists)
- enter data or ask questions (forms, buttons)

# Why the Web?

Of all the ways to build an interface (desktop GUI, mobile app, terminal...) the Web is by far the cheapest:

- no installation for the user: a browser is enough
- runs on any operating system, any device
- interface is just text files you can write in any editor
- can be served by a very small server-side program
- universal ecosystem of free tools

# Why the Web?

Of all the ways to build an interface (desktop GUI, mobile app, terminal...) the Web is by far the cheapest:

- no installation for the user: a browser is enough
- runs on any operating system, any device
- interface is just text files you can write in any editor
- can be served by a very small server-side program
- universal ecosystem of free tools

Caveat: we are not building a modern, polished webapp here. The goal is to put a usable face on top of code and data.

# The three Web languages

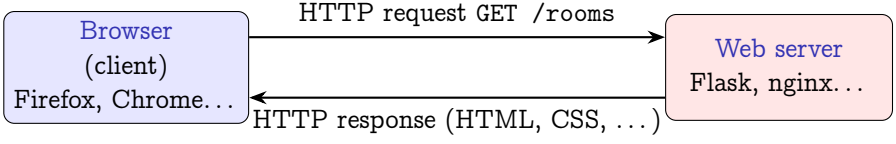
| | |
|---|---|
| HTML | structure and content of a page<br>*(this lecture)* |
| CSS | style of a page<br>*(next lecture)* |
| JavaScript | behavior on the client side<br>*(later)* |

# The three Web languages

| | |
|---|---|
| HTML | structure and content of a page<br>*(this lecture)* |
| CSS | style of a page<br>*(next lecture)* |
| JavaScript | behavior on the client side<br>*(later)* |

These are interpreted by a Web browser (Firefox, Chrome, Safari, Edge...) which fetches them from a Web server via the HTTP protocol.

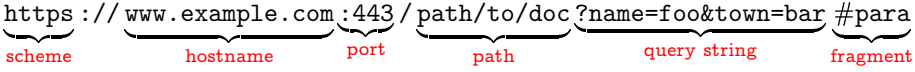
# Client and server




- The server runs our Python program, talks to the database, and returns HTML pages
- The client parses the HTML and displays it
- More on the server side in a later lecture

# URL: Uniform Resource Locator

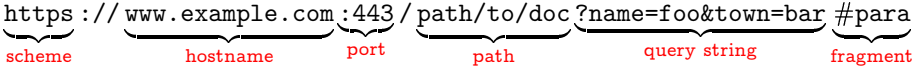
Every resource on the Web is identified by a URL:




| | |
|---|---|
| scheme | how to access the resource: usually `http` or `https` |
| hostname | domain name of the server (resolved via DNS; `www.` is just a conventional subdomain) |
| port | TCP port; default 80 for `http`, 443 for `https` (then omitted) |
| path | logical path of the document on the server |
| query string | optional parameters for dynamic pages |
| fragment | optional sub-part of the document (the `id` of an element) |

# URL: Uniform Resource Locator

Every resource on the Web is identified by a URL:




| | |
|---|---|
| scheme | how to access the resource: usually `http` or `https` |
| hostname | domain name of the server (resolved via DNS; `www.` is just a conventional subdomain) |
| port | TCP port; default 80 for `http`, 443 for `https` (then omitted) |
| path | logical path of the document on the server |
| query string | optional parameters for dynamic pages |
| fragment | optional sub-part of the document (the `id` of an element) |

Relative URLs are resolved against the current page's URL: `/foo` → root of the server, `bar` → same directory.

# Outline

# HTML: HyperText Markup Language

- Open standard maintained by the W3C and WHATWG
- Plain text files with tags
- Describes the structure and content of a document, not its appearance
- Designed for accessibility (screen readers, assistive technology rely on the structure)
- HTML5 is the current "living standard"
- Reference:
  - `https://html.spec.whatwg.org/multipage/`
  - `https://developer.mozilla.org/en-US/docs/Web/HTML`

# Tags

Syntax:

HTML

```
<tag attribute1="v1" attribute2="v2">content</tag>
```

or, for elements with no content:

HTML

```
<tag attribute1="v1">
```

| | |
|---|---|
| tag | keyword identifying an HTML element |
| content | text and/or other tags (nested) |
| attributes | list of `name="value"` pairs separated by spaces |

Comments are written `<!-- like this -->` and ignored by the browser.

# Tags

Syntax:

HTML

```
<tag attribute1="v1" attribute2="v2">content</tag>
```

or, for elements with no content:

HTML

```
<tag attribute1="v1">
```

| | |
|---|---|
| tag | keyword identifying an HTML element |
| content | text and/or other tags (nested) |
| attributes | list of `name="value"` pairs separated by spaces |

Comments are written `<!-- like this -->` and ignored by the browser. Rule of thumb: lowercase names, double-quoted values, always close your tags, nest them in the right order.

# Character entities

Some characters cannot appear literally in HTML text:

- `<` for < (would otherwise start a tag)
- `>` for >
- `&` for & (would otherwise start an entity)
- `&quot;` for " (inside attribute values)

Example:

HTML

```
<code>SELECT * WHERE id &lt; 10 AND name = &quot;X&quot;</code>
```

# Character entities

Some characters cannot appear literally in HTML text:

- `<` for < (would otherwise start a tag)
- `>` for >
- `&` for & (would otherwise start an entity)
- `&quot;` for " (inside attribute values)

Example:

HTML

```
<code>SELECT * WHERE id &lt; 10 AND name = &quot;X&quot;</code>
```

Aside: hand-concatenating HTML from user input is a classic source of XSS vulnerabilities. When we generate pages from data (next lectures), the templating library (Jinja) escapes these for us automatically.

# Structure of an HTML document

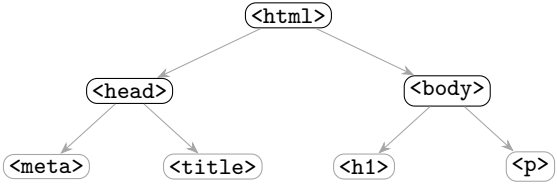
HTML

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>My Hotel</title>
  </head>
  <body>
    <h1>Welcome</h1>
    <p>Browse our rooms and bookings.</p>
  </body>
</html>
```

- `<!DOCTYPE html>`: this is HTML5
- `lang="en"`: language of the page (used by screen readers for pronunciation, by browsers for hyphenation, by search engines)
- `<head>`: metadata (title, charset, links to CSS…)
- `<body>`: the visible content

# The DOM: an HTML document is a tree

The browser parses your HTML into a tree, the Document Object Model (DOM):




This tree is what:

- CSS selectors target (next lecture)
- JavaScript navigates and modifies (later)
- Python templates build, node by node (later)

# Outline

# Headings and paragraphs

Six levels of headings, by decreasing importance:

HTML

```
<h1>Page title</h1>
<h2>Main section</h2>
<h3>Subsection</h3>
... up to <h6>
```

Paragraphs:

HTML

```
<p>This is one paragraph.</p>
<p>This is another paragraph.</p>
```

# Headings and paragraphs

Six levels of headings, by decreasing importance:

HTML

```
<h1>Page title</h1>
<h2>Main section</h2>
<h3>Subsection</h3>
... up to <h6>
```

Paragraphs:

HTML

```
<p>This is one paragraph.</p>
<p>This is another paragraph.</p>
```

Whitespace, line breaks and indentation in the source are collapsed by the browser. To force a line break inside a paragraph use <br>.

# Lists

Unordered list (bullet points):

HTML

```
<ul>
  <li>Single room</li>
  <li>Double room</li>
  <li>Suite</li>
</ul>
```

Ordered list (numbered):

HTML

```
<ol>
  <li>Pick a room</li>
  <li>Pick dates</li>
  <li>Confirm booking</li>
</ol>
```

# Block vs inline elements

HTML elements come in two main flavors:

**block** starts on a new line, takes the full width available, may contain other blocks and inline elements:
`<p>`, `<h1>`...`<h6>`, `<ul>`, `<ol>`, `<li>`, `<table>`, `<header>`, `<main>`, `<section>`, `<div>`...

**inline** flows within a line of text, only as wide as its content, may contain only other inline elements:
`<a>`, `<strong>`, `<em>`, `<code>`, `<img>`, `<span>`...

# Block vs inline elements

HTML elements come in two main flavors:

**block** starts on a new line, takes the full width available, may contain other blocks and inline elements:
`<p>`, `<h1>`...`<h6>`, `<ul>`, `<ol>`, `<li>`, `<table>`, `<header>`, `<main>`, `<section>`, `<div>`...

**inline** flows within a line of text, only as wide as its content, may contain only other inline elements:
`<a>`, `<strong>`, `<em>`, `<code>`, `<img>`, `<span>`...

Rule: put inline inside block, never the other way around. (CSS can override the default with `display: block` / `inline`, but the default is what your page falls back to.)

# Inline formatting

Inside a paragraph:

HTML

```
<p>The room is <strong>fully booked</strong> on
that <em>weekend</em>. See <code>SELECT *</code> below.</p>
```

`<strong>` strong importance (rendered **bold**)

`<em>` emphasis (rendered *italic*)

`<code>` inline code (rendered `monospace`)

`<span>` generic inline container, no semantics

# Inline formatting

Inside a paragraph:

HTML

```
<p>The room is <strong>fully booked</strong> on
that <em>weekend</em>. See <code>SELECT *</code> below.</p>
```

- `<strong>` strong importance (rendered **bold**)
- `<em>` emphasis (rendered *italic*)
- `<code>` inline code (rendered `monospace`)
- `<span>` generic inline container, no semantics

Choose tags by meaning, not by appearance: styling is CSS's job.

# Links

Links are the essence of the Web.

HTML

```
<a href="https://www.psl.eu/">PSL University</a>

<a href="/rooms.html">All rooms</a>
<a href="/booking?room=504">Book room 504</a>
<a href="#contact">Contact section below</a>
   ...
<h2 id="contact">Contact</h2>
<p>Email us at info@hotel.example.</p>
```

- Absolute URL: `https://...`
- Relative URL: same server, possibly same page
- `#id` jumps to the element whose `id` attribute matches (here, the `<h2>`)
- Query string `?key=value`: passes data to the server

# Images

HTML

```
<img src="rooms/504.jpg"
     alt="Room 504, sea view">
```

- `src`: where the image comes from
- `alt`: textual description for screen readers, and shown if the image fails to load: always provide it
- Sizing and responsiveness are handled by CSS (next lecture)

# Outline

# Semantic structure

HTML5 introduced tags that describe what a region is, not just where:

HTML

```
<header>     ... site title, logo ...      </header>
<nav>        ... navigation menu ...        </nav>
<main>
  <article>  ... a self-contained piece ... </article>
  <section>  ... a thematic group ...       </section>
</main>
<aside>      ... side-related content ...   </aside>
<footer>     ... copyright, contact ...     </footer>
```

Before HTML5, the same layout was built with the generic block container `<div>` (the block counterpart of `<span>`); the new tags are visually equivalent but better for accessibility, search engines, and reading the source.

# Outline

# HTML tables

Tables are how we display tabular data: exactly what comes back from a SQL query.

HTML

```
<table>
  <tr><th>id</th><th>name</th><th>email</th></tr>
  <tr><td>1</td><td>John Smith</td><td>john.smith@gmail.com</td></tr>
  <tr><td>2</td><td>Alice Black</td><td>alice@black.name</td></tr>
</table>
```

`<table>` the table, `<tr>` a row, `<th>` a header cell, `<td>` a data cell.

# Structured tables

For larger tables, group rows semantically:

HTML

```
<table>
  <caption>Reservations</caption>
  <thead>
    <tr><th>id</th><th>guest</th><th>room</th>
        <th>arrival</th><th>nights</th></tr>
  </thead>
  <tbody>
    <tr><td>1</td><td>1</td><td>504</td>
        <td>2026-01-01</td><td>5</td></tr>
  </tbody>
</table>
```

`<caption>` table title, rendered above

`<thead>` header rows (column labels)

`<tbody>` body rows (the actual data)

`<tfoot>` footer rows (e.g. totals)

# From SQL result to HTML table

In a future lecture we will do this in Python:

1. run a query: `rows = db.execute("SELECT ... FROM reservation")`
2. loop over the rows in a template
3. for each row, emit an `<tr>` with one `<td>` per column
4. ship the resulting HTML page back to the browser

# From SQL result to HTML table

In a future lecture we will do this in Python:

1. run a query: `rows = db.execute("SELECT ... FROM reservation")`
2. loop over the rows in a template
3. for each row, emit an `<tr>` with one `<td>` per column
4. ship the resulting HTML page back to the browser

The same shape of HTML is generated for any query result. Tables are the natural rendering of relations.

# Outline

# Forms: getting data from the user

Forms are how an HTML page collects input from the user and sends it to the server, which can then run an SQL query or an INSERT.

- login pages, search bars, booking forms, comments...
- all built from a small set of HTML elements
- the actual processing happens on the server

# The <form> element

HTML

```
<form action="/search" method="get">
  <label for="q">Guest name:</label>
  <input type="text" id="q" name="q">
  <button type="submit">Search</button>
</form>
```

`action` URL to submit the form to

`method` `get` (data in the URL) or `post` (data in the request body)

`name` key under which the value will arrive at the server

`<label>` accessible description, linked via `for`/`id`

# Common input types

HTML

```
<input type="text"     name="name">
<input type="password" name="pwd">
<input type="email"    name="email">
<input type="number"   name="nights" min="1" max="30">
<input type="date"     name="arrival">
<input type="checkbox" name="breakfast" value="yes">
<input type="radio"    name="bed" value="single">
<input type="radio"    name="bed" value="double">
<input type="hidden"   name="guest_id" value="42">

<textarea name="comment" rows="4" cols="40"></textarea>
```

Browsers render appropriate widgets (date pickers, number spinners...) and do basic client-side validation.

# Drop-down lists

HTML

```
<label for="room">Room type:</label>
<select id="room" name="room">
  <option value="single">Single</option>
  <option value="double" selected>Double</option>
  <option value="suite">Suite</option>
</select>
```

Multiple selection: add the `multiple` attribute; the browser then sends every selected value under the same name.

# A complete booking form

HTML

```
<form action="/book" method="post">
  <label>Guest:   <input type="text"   name="name" required></label>
  <label>Arrival: <input type="date"   name="arrival" required></label>
  <label>Nights:  <input type="number" name="nights" min="1"></label>
  <label>Room:
    <select name="room">
      <option>504</option><option>107</option><option>302</option>
    </select>
  </label>
  <button type="submit">Book</button>
</form>
```

# GET vs POST

**GET** form data goes in the URL: `/search?q=alice&room=504`

- visible, bookmarkable, cacheable
- suitable for queries (search, filters)
- not for sensitive data, not for actions that change the database

**POST** form data goes in the request body

- not visible in the URL bar
- suitable for actions (booking, login, deletion)
- the standard for any `INSERT`/`UPDATE`/`DELETE`

# GET vs POST

**GET** form data goes in the URL: `/search?q=alice&room=504`

- visible, bookmarkable, cacheable
- suitable for queries (search, filters)
- not for sensitive data, not for actions that change the database

**POST** form data goes in the request body

- not visible in the URL bar
- suitable for actions (booking, login, deletion)
- the standard for any `INSERT`/`UPDATE`/`DELETE`

More on this in future lectures.

# Outline

# What HTML gives us

- A universal way to display text, tables, lists, images... in any browser
- A form mechanism to send user input back to a server
- A semantic structure that is accessible and machine-readable
- Plain text files we can generate from Python once we have a server

# What HTML does *not* do

- Looks (colors, fonts, layout): that is CSS (next lecture)
- Behavior in the browser (dynamic updates, animation): that is JavaScript (future lecture)
- Server-side logic (routing, queries, authentication): Python and Flask (future lectures)

# What HTML does *not* do

- Looks (colors, fonts, layout): that is CSS (next lecture)
- Behavior in the browser (dynamic updates, animation): that is JavaScript (future lecture)
- Server-side logic (routing, queries, authentication): Python and Flask (future lectures)

But on its own, HTML is already enough to put a usable face on a database.

# To explore on your own

- Officiel specification: `https://html.spec.whatwg.org/multipage/`
- MDN Web Docs, the Web reference:
  `https://developer.mozilla.org/en-US/docs/Web/HTML`
- Validate any page with the W3C validator: `https://validator.w3.org/`
- Try View Source (Ctrl-U) on any website you visit to see the HTML
- In the browser, the developer tools (F12) let you inspect and edit live HTML

# License