University of 20 Aout 1955, Skikda
Department of Computer Science

Course material
Computer Science Bachelor's level

Web Application Development

Prepared by:
Dr. Laroum Toufik

Academic Year 2025-2026

ABSTRACT

This course material on web application development (intended for undergraduate Computer Science students) has the main objective of providing a clear and structured introduction to the fundamental web technologies, including the World Wide Web, HTML, CSS, JavaScript, PHP, XML, as well as web services. It is by no means a complete and exhaustive course. Each chapter presented here could be developed into a full standalone course.

This course material will enable students to acquire the necessary skills (Front-end and Back-end) to develop and implement a functional and well-structured Web application.

At the end of this course, the student will be able to:

- Understand the fundamental principles of Web application development.
- Master the steps of creating a Web project, from design to deployment.
- Use appropriate tools, languages, and frameworks to produce a dynamic website or application.
- Apply best practices in terms of structure, usability, security, and maintenance.

Recommended prerequisites:

To fully benefit from this module, students are advised to have already acquired:

- Fundamental knowledge in algorithms: knowing how to break down a problem into logical steps and design a simple algorithm.
- Basics of programming: knowing at least one programming language and understanding the concepts of variables, conditionals, loops, functions, etc.
- Basic knowledge of the Internet and networks: knowing what an IP address is, understanding the role of protocols (HTTP, TCP/IP), and knowing the basics of client-server functioning.
- Knowledge of database design, creation, and querying.

The course is composed of 7 chapters:

1. Chapter 01: Introduction to the World Wide Web
2. Chapter 02: HTML Language
3. Chapter 03: CSS Language
4. Chapter 04: JavaScript Language
5. Chapter 05: PHP Language
6. Chapter 06: XML Language
7. Chapter 07: Web Services

Table of Contents

List of Figures

List of Tables

CHAPTER

1

INTRODUCTION TO THE WORLD WIDE WEB

1 Introduction

The Web today plays a central role in our daily lives, whether it is searching for information, communicating, or entertaining ourselves. Before creating a website or a web application, it is essential to understand its fundamental principles. In this chapter, we will discover what the World Wide Web is, the client–server model on which it is based, as well as the role of the HTTP protocol in the exchanges between browser and server. We will also learn to distinguish between static sites, with fixed content, and dynamic sites, able to adapt to users' needs.

2 Definition and history of the Web

The *World Wide Web* (often abbreviated as "Web") is a public hypertext information system running on the Internet, allowing the consultation of pages linked together by hypertext links, via a web browser.

It was invented in 1989 by Tim Berners-Lee, a researcher at CERN, with the aim of

facilitating the sharing and consultation of scientific information. The first web page was published online in 1991. Over the years, the Web has evolved from *Web 1.0*, consisting mainly of static pages, to *Web 2.0*, more interactive and collaborative, and then to emerging concepts such as *Web 3.0* and the *Semantic Web*.

Today, the Web has become an essential tool in the fields of communication, commerce, education, and entertainment.

3 Internet and the Web

It is common to confuse **Internet** and the **World Wide Web**, although they are two distinct concepts.

- **Internet** is a global network of interconnected computers, using a set of communication protocols (TCP/IP) to exchange data.
- **The Web** is a service that runs on the Internet, allowing access to hypertext documents (web pages) via the HTTP protocol, generally with the help of a browser.

In summary, *Internet* is the infrastructure, while the *Web* is one of the applications that operate thanks to this infrastructure.

4 Client-Server Architecture

The **client-server architecture** is a model of distributed application organization in which tasks are shared between resource or service providers, called *servers*, and service requesters, called *clients*.

In this model:

- The **client** is often software installed on the user's machine (e.g., web browser) that sends requests.
- The **server** is a machine or a program that responds to client requests by providing the requested data or services.

The Web is a representative example of the client-server architecture. In this model, resources are hosted on servers that provide a constant service to clients, represented here by web browsers. The role of the server is to wait for requests from clients and to provide in return the requested data, whether static or dynamically generated as a result

of processing. The client, for its part, sends an HTTP request to the server and, once the response is received and the data transferred, presents it to the end user in a usable form. [7].

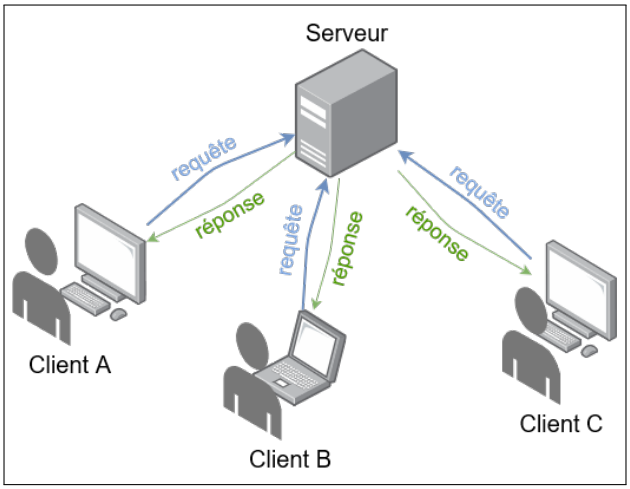

Figure 1.1: Client server architecture

5 The HTTP Protocol

The *HyperText Transfer Protocol* (HTTP) is the standard communication protocol used for data exchange on the Web. It defines how a client (often a browser) and a web server interact, by organizing exchanges in the form of requests and responses.

5.1 HTTP Request

An HTTP request is a message sent by the client to the server to request a resource. It generally consists of:

- a **request line** (e.g., `GET /index.html HTTP/1.1`), indicating the method, the URI, and the protocol version;
- **headers** providing additional information (content type, language, cookies, etc.);
- optionally a **message body** containing data to be transmitted (mainly in `POST` or `PUT` requests).

The server interprets this request to determine the resource or action requested.

5.2 HTTP Response

The HTTP response is the message sent by the server to the client following a request. It generally includes:

- a **status line** containing the response code (e.g., `200 OK`, `404 Not Found`) and the protocol version;
- **response headers** providing metadata (MIME type, content size, encoding, etc.);
- a **message body** that contains the requested resource (HTML, image, JSON, etc.).

The combination of the request and the response forms the basis of HTTP communication.

6 Web Page

A **web page** is an electronic document accessible via the World Wide Web, generally written in HTML, and which may contain text, images, videos, or interactive scripts. Each page is identified by a unique address, called a *Uniform Resource Locator* (URL), and is interpreted by a web browser that displays it to the user.

7 Website

A **website** is an organized set of web pages linked together by hyperlinks and hosted under the same domain name. It is generally designed to provide information, offer services, or allow interactions with users. The homepage, or *homepage*, often serves as the main entry point to the site [6].

There are mainly two types of websites:

7.1 Static Websites

also called showcase sites, their content is fixed and identical for all visitors. The pages are often created directly in HTML and CSS, and any modification requires manual intervention.

Examples include: a simple personal site (portfolio), an event presentation site, static documentation published in pure HTML (e.g., old online manuals)... etc.

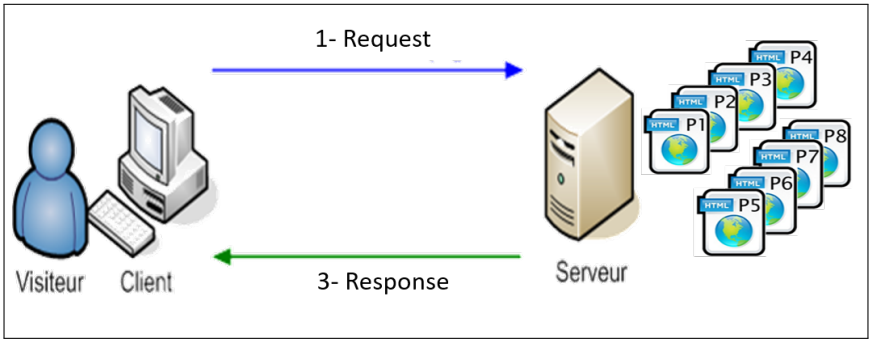

Figure 1.2: Principle of static websites

7.1.1 Advantages

- Fast and easy to create (does not require specific knowledge of dynamic scripting languages, databases, etc.).
- Reduced development cost: simple creation with only HTML and CSS.
- Low hosting cost.

7.1.2 Disadvantages

- No customization offered to visitors (the same content for everyone).
- No interaction is possible with the user.
- When the site is large, it is difficult to maintain, keep consistency, and update it manually.

7.2 Dynamic Websites

also called web applications, their content is generated automatically (dynamically) by a server depending on the user's actions, combining the use of programming languages (such as PHP, Java, Python, or JavaScript) and a database installed on the server. Examples include: social networks, webmails, discussion forums, search engines, e-commerce sites, online games... etc.

7.2.1 Advantages

- Interactive features: forms, search, account management, etc.
- Personalized content: display adapted according to the user (profile, preferences, history).

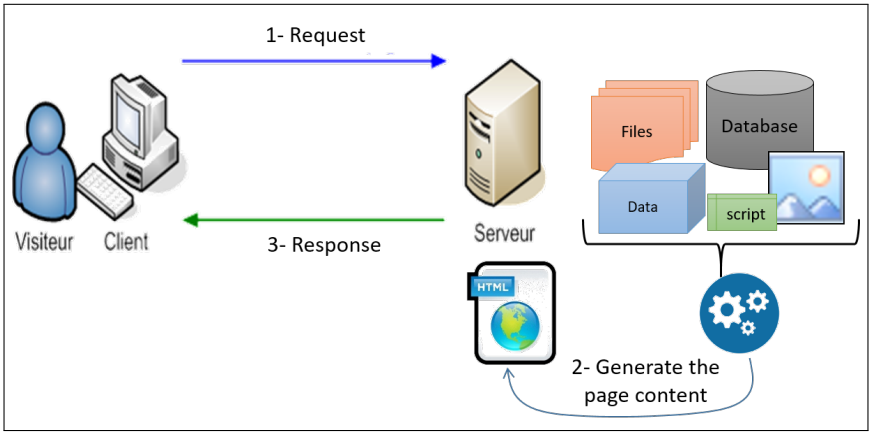

Figure 1.3: Principle of dynamic websites

- Easy and automatic updates: the content can be modified without touching the code.

7.2.2 Disadvantages

- Higher cost and complexity of development and hosting.
- Requires skills in server-side programming and databases.
- More demanding maintenance and security: increased risk of vulnerabilities (SQL injections, XSS, etc.) if the code is not secured.

8 Web Application Development

Web application development is the set of techniques, tools, and practices used to design, create, and maintain websites and applications accessible via a web browser.

It is generally divided into two main parts: Frontend and Backend.

8.1 Frontend (or client-side development)

This is the visible and interactive part with which the user directly interacts. It includes layout, design, ergonomics, and animations, implemented with technologies such as `HTML`, `CSS`, and `JavaScript`, as well as frameworks like `React`, `Vue.js`, or `Angular`.

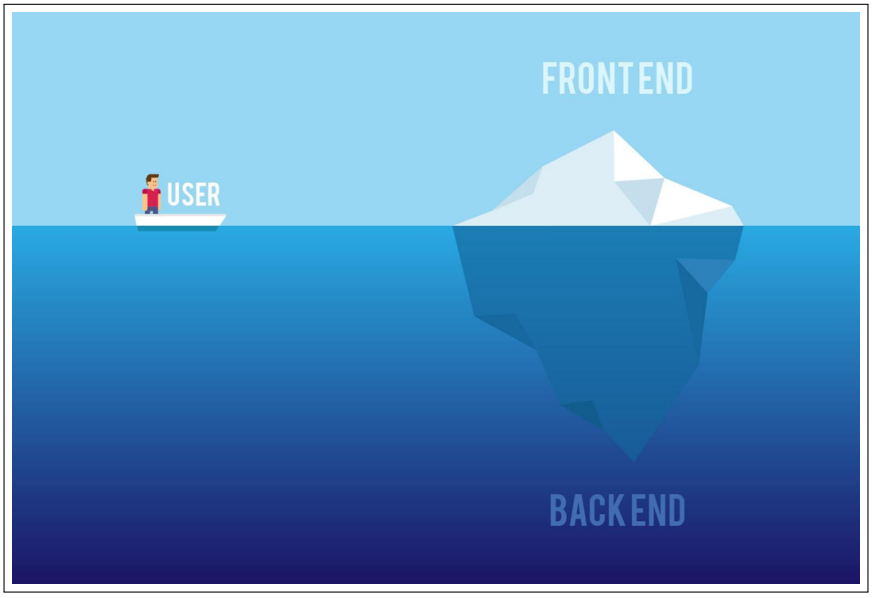

Figure 1.4: Front-end and Back-end

8.2 Backend (or server-side development)

This is the invisible part that handles business logic, data processing, and communication with databases. It uses server-side languages and frameworks such as `PHP`, `Java (Spring)`, `Python (Django, Flask)`, `Node.js`, etc., as well as database management systems like `MySQL`, `PostgreSQL`, or `MongoDB`.

9 Conclusion

In this first chapter, we have discovered the essential elements to understand how the Web works. We have seen what the World Wide Web is, the principle on which the client–server model is based, as well as the HTTP protocol that manages exchanges between a browser and a server. We have also learned to distinguish between a static site or page, whose content remains fixed, and a dynamic site or page, capable of adapting to the user's needs or actions. The next chapter is dedicated to the HTML language, the basic language to start web development.

10 Exercise: Creating a first web page

Without directly writing HTML tags, use the software **Dreamweaver** in *Design* mode to build a web page that represents a **simple CV** (see the figure below).

Figure 1.5: First web page

CHAPTER

2

HTML LANGUAGE

1 Introduction

In this chapter, we will discover the HTML language, which forms the basis of every web page. We will see how to structure content, use the main tags, and efficiently organize a page. The chapter will cover text formatting, inserting tables, creating forms, adding links and images, etc. It will lay the necessary foundations for creating static web pages and prepare for the addition of styles and interactions in the following chapters.

2 Definition

HTML (*HyperText Markup Language*) is the standard language used to **structure and organize the content of a web page** [10]. It describes the elements of the page (headings, paragraphs, images, links, tables, forms...) using **tags**, interpreted by the browser for display.

Note: HTML is *not* a programming language, but a **markup language** used only to structure content.

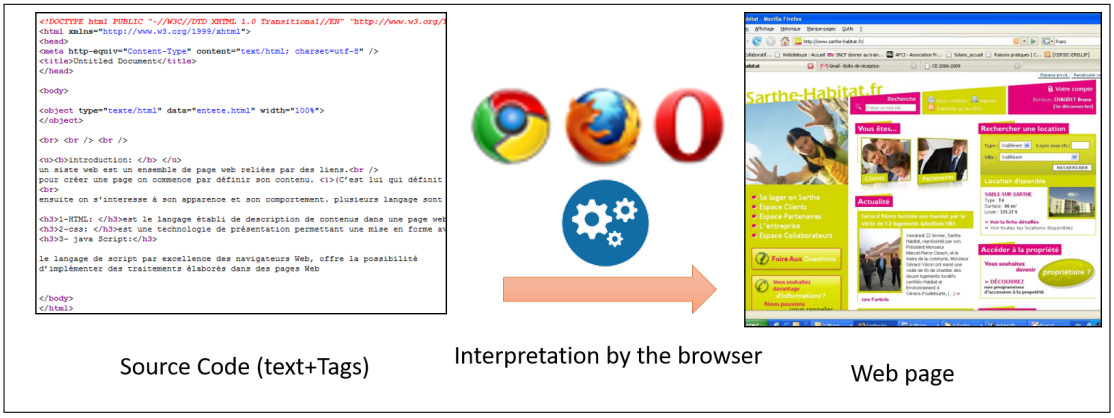

Figure 2.1: principle of HTML

3 History

The history of HTML is closely linked to that of the World Wide Web [4].

- **1991** – Tim Berners-Lee (CERN) designed the first prototype of HTML, with only 18 tags.
- **1993** – First official publication of the HTML specification (HTML 1.0) [10].
- **1995** – HTML 2.0 becomes the official standard, defined by the IETF.
- **1997–1999** – HTML 3.2 and then 4.01 are published by the W3C.
- **2000–2008** – Emergence of XHTML (a strict version based on XML).
- **2014** – Official release of HTML5, with semantic tags, audio/video, ‘<canvas>‘ and SVG.
- **Today** – HTML is now a Living Standard, managed by the WHATWG, which means it evolves continuously rather than being released as successive versions.

4 Structure of an HTML page

An HTML page has a hierarchical structure composed of main tags. It always begins with the declaration <!DOCTYPE html>, which tells the browser that it is an HTML5 document. The <html> tag encloses all the content of the page. Inside it, the <head> section contains the metadata (page title, encoding, links to style sheets, etc.), while the <body> section contains the content visible to the user (headings, paragraphs, images, links, etc.):

```
<!DOCTYPE html>

<html>
<head>
<title>Page Title</title>
</head>
<body>
<h1>My first heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```

An HTML element is defined by a start tag, some content, and then an end tag: <nomBalise> Content here... </nomBalise>

An HTML element corresponds to everything between the opening tag and the closing tag: <h1>My first heading</h1> My first paragraph.

The role of a web browser (Chrome, Edge, Firefox, Safari) is to read HTML documents and display them correctly.

A browser does not display HTML tags, but uses them to determine how to present the document on the screen.

Figure 2.2: display of the HTML page in the browser

5 Development Environments (IDE)

To develop web pages, it is recommended to use an Integrated Development Environment (IDE). An IDE combines in a single tool a code editor, writing assistance features (syntax highlighting, auto-completion), etc. Examples of popular IDEs for web development are Visual Studio Code, Sublime Text, Dreamweaver, Atom, Notepad++, etc.

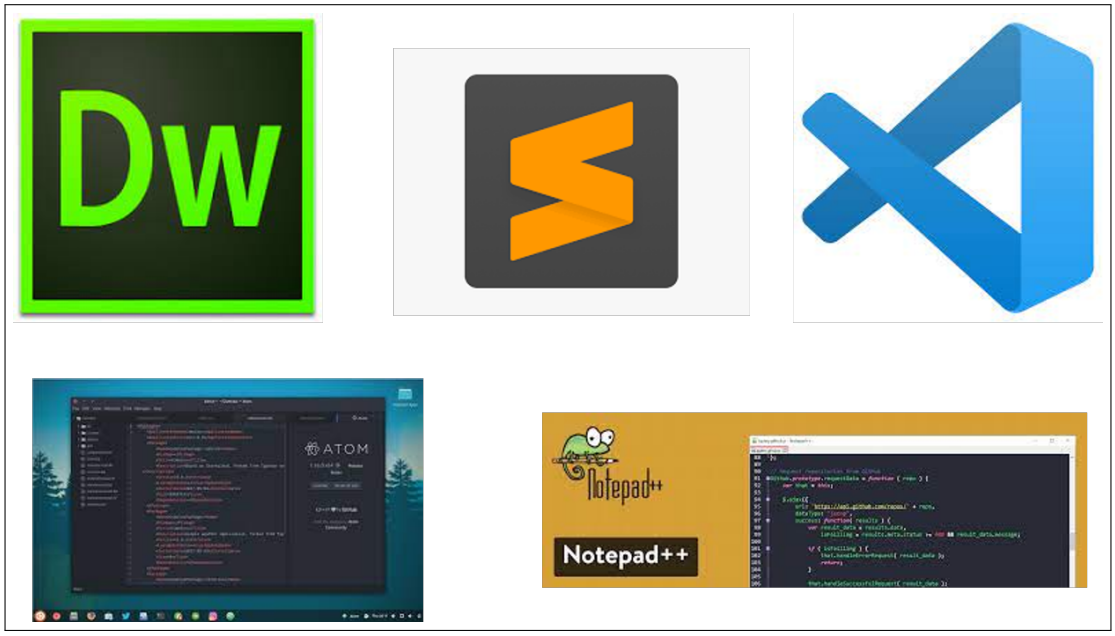

Figure 2.3: Development Environments (IDE)

6 Text formatting with the tag

Before the advent of style sheets (CSS), the tag was widely used to modify the appearance of text directly in the HTML code. It allowed changing the size, color, and font of the text.
General syntax: Text to format

- size : Defines the text size (value between 1 and 7)
- color : Defines the text color (name or hexadecimal code)
- face : Defines the font to use (Arial, Times, ...)

Example:
Hello World

```
<!DOCTYPE html>
<html>
<body>

<font size="15" color="red" face="Arial">Hello World</font>

</body>
</html>
```

Hello World

Figure 2.4: font tag

Key point

The `<font>` tag belongs to older versions of HTML. It may still work in some browsers, but it is recommended to use CSS styles for text formatting. It is used here only for educational purposes, to explain the principle of an HTML formatting tag.

7 Other formatting tags

HTML also provides a set of tags to locally modify the appearance of text. Below, we present some other commonly used formatting tags:

Tag	Description	Example
`b`	Makes text bold (old tag, replaceable by `strong`)	`<b>Bold text</b>`
`strong`	Makes text bold while indicating semantic importance	`<strong>Important</strong>`
`i`	Displays text in italics (old tag)	`<i>Italic text</i>`
`em`	Displays text in italics with emphatic meaning	`<em>Emphasis</em>`
`u`	Underlines text	`<u>Underlined text</u>`
`mark`	Highlights text	`<mark>Highlighted text</mark>`
`code`	Displays a piece of code in a monospaced font	`<code>printf()</code>`
`br`	Line break (no closing tag)	`Line1 Line2`
`sup`	Displays text as superscript	`H<sup>2</sup>O`
`sub`	Displays text as subscript	`x<sub>i</sub>`
`center`	Horizontally centers the content (deprecated tag)	`<center>Centered text</center>`

Table 2.1: Some HTML tags for text formatting

Key point

These tags were widely used in older versions of HTML. It is recommended to use CSS to define text formatting rather than directly applying these tags in HTML code.

8 Some useful tags

8.1 Lists of elements

HTML lists allow web developers to group a set of related items in the form of lists. There are 2 types of lists:

- An unordered list starts with the <ul> tag. Each list item starts with the <li> tag. By default, list items are marked with bullets (small black circles).
- An ordered list starts with the <ol> tag. Each list item starts with the <li> tag. By default, list items are numbered.

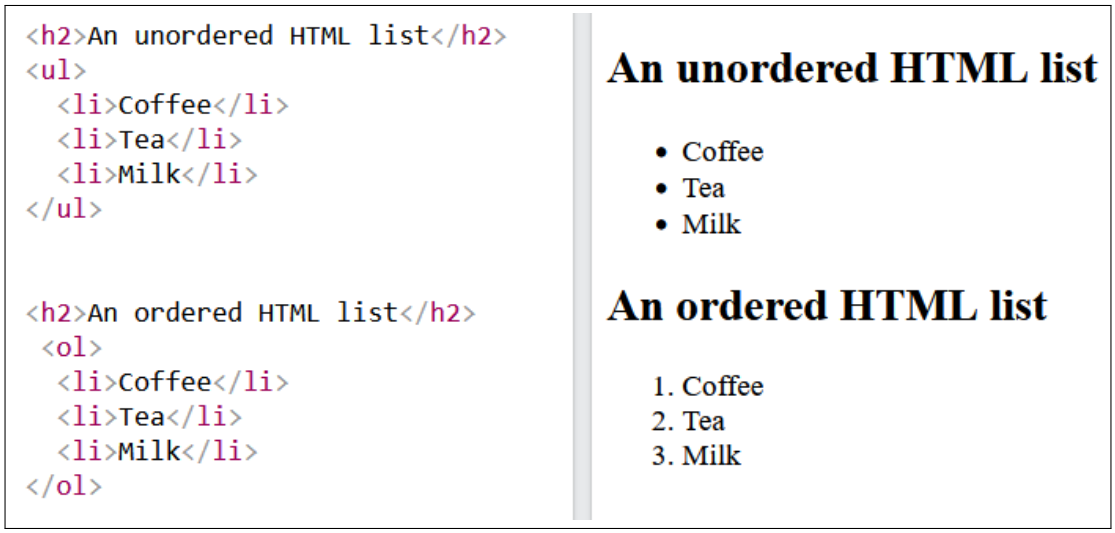

Figure 2.5: Lists in HTML

8.2 Heading tags <h1> to <h6>

The <h1> to <h6> tags are used to define headings in HTML. The <h1> tag defines the most important heading, while the <h6> tag defines the least important heading.

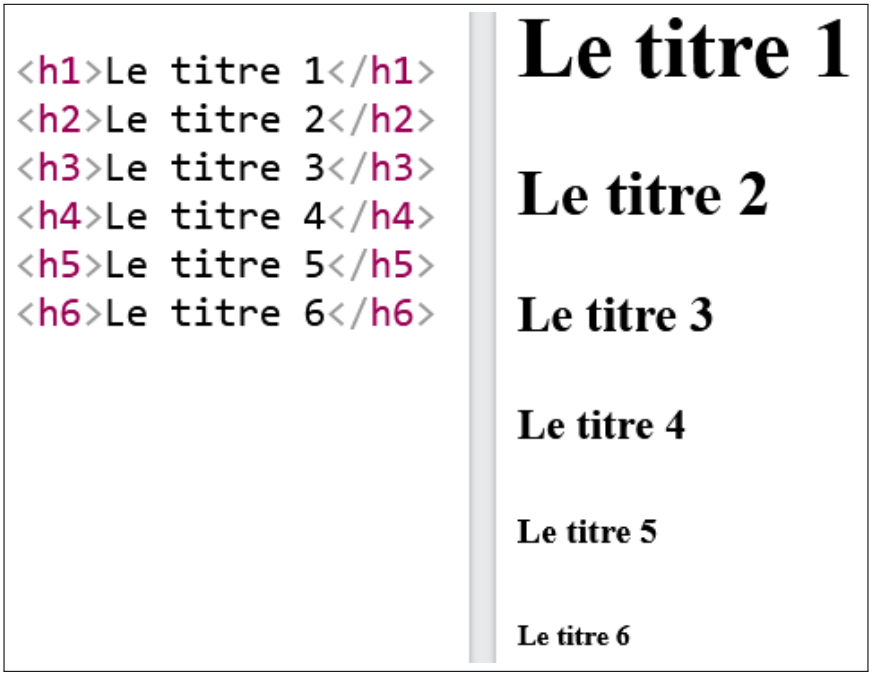

Figure 2.6: Headings in HTML

8.3 Paragraphs

The HTML element <p> defines a paragraph.

A paragraph always starts on a new line (it is a **Block**-level tag), and browsers automatically add some white space (a margin) before and after the paragraph.

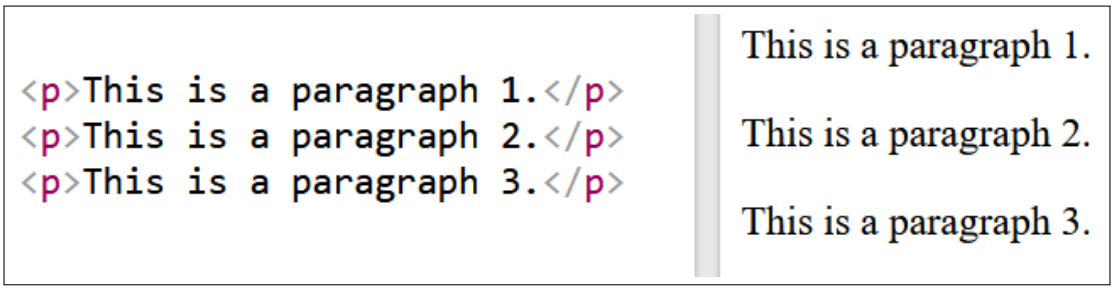

Figure 2.7: Paragraphs in HTML

A paragraph can contain text, but also other HTML elements such as links, images, lists, or even formatting tags.

8.4 Images

The HTML <img> tag is used to insert an image into a web page. The <img> tag is empty: it only contains attributes and has no closing tag.

The <img> tag has the following attributes:

- src: Specifies the path to the image
- alt: Specifies an alternative text for the image
- width and height: Allow you to define the dimensions of the image (currently, this is rather done with CSS).

```
<img src="img_Algeria.jpg"
alt="Flowers in Algeria"
width="200" height="150">
```

Figure 2.8: Images in HTML

8.5 Hyperlinks

Links allow users to navigate from one page to another by clicking. When you hover the mouse over a link, the cursor turns into a small hand.
The HTML <a> tag is used to define a hyperlink. It has the following syntax: link text. The most important attribute of the <a> element is the href attribute, which specifies the link destination.

```
To visit the department's website
<a href="https://www.depinfoskikda.com/">
  click here
</a>
```

To visit the department's website click here

Figure 2.9: Links in HTML

The link text is the part that will be visible to the reader. By clicking on this text, the reader will be directed to the specified URL.

Key point

- Links can be used to **download a document**, for example:

```
<a href="document.pdf">Download the document</a>
```

- They can also lead to a **specific location in the page** using anchors, for example:

```
<a href="#exercises">Go to the exercises section</a>
```

9 Tables

HTML tables allow structuring data into rows and columns, for example to display the list of students registered in the database.
Example: A table containing the list of students with the columns: code, name, and average.

- An HTML table is composed of cells arranged within rows and columns.
- Each row of a table begins with a <tr> tag and ends with a </tr> tag.
- Each cell of a table is defined by a <td> tag and a </td> tag.
- <th> means table header. By default, the text contained in the <th> tags is bold and centered, but this can be modified with CSS.

When creating a table in HTML, it is sometimes necessary to merge multiple cells to improve the presentation of the data.

9.1 Rowspan (merging rows)

Allows a cell to span several rows. For example, cell 3 in the following table:

9.2 Colspan (merging columns)

Allows a cell to span several columns. For example, cell 4 in the following table:

```
<h2>Liste des étudiants</h2>
<table>
        <tr>
            <th>Code</th>
            <th>Nom</th>
            <th>Moyenne</th>
        </tr>

        <tr>
            <td>ET001</td>
            <td>Amina Bensalah</td>
            <td>15.2</td>
        </tr>
        <tr>
            <td>ET002</td>
            <td>Karim Meziane</td>
            <td>13.7</td>
        </tr>
        <tr>
            <td>ET003</td>
            <td>Sofia Hamdi</td>
            <td>16.5</td>
        </tr>
        <tr>
            <td>...</td>
            <td>...</td>
            <td>...</td>
        </tr>
</table>
```

Liste des étudiants

Code	Nom	Moyenne
ET001	Amina Bensalah	15.2
ET002	Karim Meziane	13.7
ET003	Sofia Hamdi	16.5
...	...	...

Figure 2.10: Table in HTML

```
<table>
  <tr>
    <td>cell1</td>
    <td>cell2</td>
    <td rowspan=2>cell3</td>
  </tr>

  <tr>
    <td>cell4</td> <td>cell5</td>
  </tr>

  <tr>
    <td>cell6</td> <td>cell7</td> <td>cell8</td>
  </tr>

</table>
```

cell 1	cell 2	cell 3
cell 4	cell 5	
cell 6	cell 7	cell 8

Figure 2.11: Merging rows

Note

- It is possible to combine the rowspan and colspan attributes to create more complex tables.
- It is clear that a table cell may contain text, images, a complete table, or even an entire HTML page.

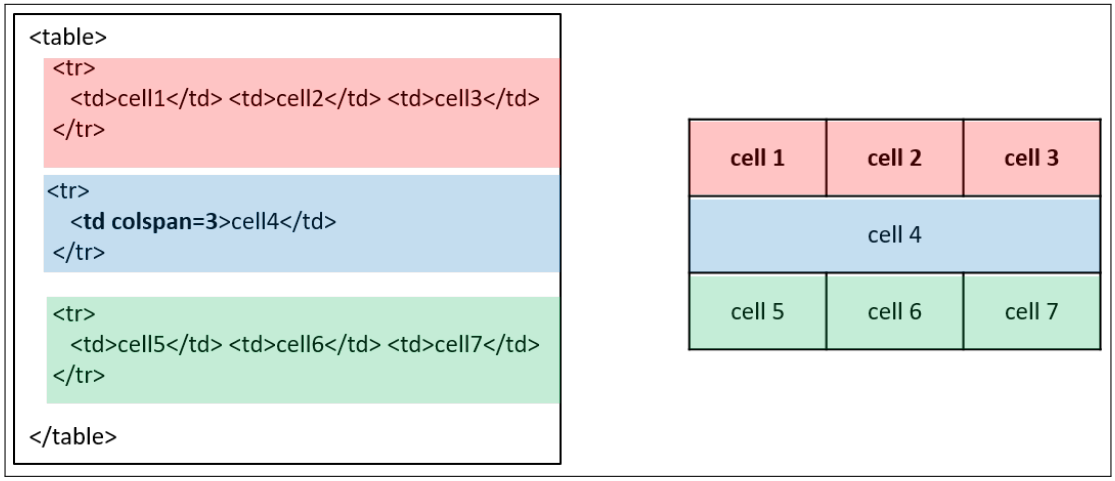

Figure 2.12: Merging columns

10 Forms

Forms are a way to enter data in order to process it later on the server side (most often), or using scripts in dynamic pages. They ensure interaction and communication with the user through different graphical components.
To insert a form we use the tag <form>....</form> which has the attributes method and action:

```
<form   method="..."   action="..." >
...... <--! The content of the form -->
</form>
```

10.1 Method Attribute

The `method` attribute specifies the HTTP method to be used when sending the form data.

Form data can be sent:

- as variables integrated into the URL (with `method="get"`) ;
- or as an HTTP **POST** request (with `method="post"`).

Note: The default HTTP method used when sending a form is `GET`.

Notes regarding the `GET` method

- The form data is added to the URL as *name/value* pairs.
- This method should never be used to transmit sensitive information, as the data is visible in the URL.

- The length of the URL is limited (about 2048 characters).

Notes regarding the `POST` method

- The form data is sent in the body of the HTTP request (it does not appear in the URL).
- The `POST` method does not have any particular size limitation and allows the sending of a large amount of information.

10.2 Action Attribute

The `action` attribute defines the action to be performed when the form is submitted. In general, the form data is sent to a file on the server when the user clicks the submit button.
In the example below, the form data is sent to a file called *inscription.php*. This file contains a server-side script responsible for processing the form data.

```
<form   method="post"   action="inscription.php" >
...... <--! The content of the form -->
</form>
```

Note: if the action attribute is omitted, the action is applied to the current page.

10.3 Form Elements

All interactive elements (components) of the form are mainly grouped within the tags:

1. Input: contains most of the components.
2. Select: choice list.
3. textArea: multi-line text area.

10.3.1 The input tag

One of the most commonly used form elements is the <input> element. The <input> element can be displayed in different ways depending on the value of its type attribute. This tag has the following syntax:

```
<input type="....." id="...." name="....">
```

To remember

- The **id** attribute allows to uniquely identify an element in the page (therefore it must not be used more than once). It is used, for example, to target an element with CSS or to create an anchor to that element.
- The **name** attribute, on the other hand, is mainly used in forms to give a name to fields and allow the server to retrieve the values sent.

Here are the different types of <input> elements you can use in HTML:

Input Type Text: defines a single-line text input field.

```
<form action="action_page.php">
  First name:  <input type="text" id="fname" name="fname"><br>
  Last name:  <input type="text" id="lname" name="lname">
</form>
```

First name: Mohamed
Last name: Ben ali

Figure 2.13: input type text

Input Type password: defines a password field. The characters entered in this field are hidden (displayed as asterisks or circles).

```
  Password:  <input type="password" id="psd" name="psd">
```

Password: ••••••••••••

Figure 2.14: input type password

Note:

- The `placeholder` attribute (in text fields) corresponds to a help text displayed inside a form field when it is empty. This text automatically disappears as soon as the user starts typing.
- The `required` attribute means that this field must be filled in when submitting the form, otherwise validation will be refused and the form will not be submitted.

Input Type checkbox: defines a checkbox:

- Checkboxes allow the user to select zero or multiple options from a limited set of choices.
- Use the attribute `checked='checked'` to make the box checked by default.
- The `value` attribute contains the text that will be sent to the server if the user checks the box. It is mandatory.

```
Select your programming languages:

<form action="action_page.php">
  <input type = "checkbox" name="language1"  value="java" /> java <br>
  <input type = "checkbox" name="language2"  value="c++" /> c++ <br>
  <input type = "checkbox" name="language3"  value="python" /> python <br>
```

Select your programming languages:
☑ java
☐ c++
☑ python

Figure 2.15: input type checkbox

Input Type radio: defines a radio button:

- Radio buttons allow the user to select only one option from a limited set of choices.
- Use the attribute `checked="checked"` to have the button selected by default.
- Radio buttons must have the same `name` attribute. This groups them together and forces the user to choose only one option from this group.
- The `value` attribute is mandatory since it represents the value that will be sent to the server when the user selects this radio button.

```
Select your Favorite programming language:

<form action="action_page.php">
  <input type = "radio" name="favorite_language"  value="java" /> java <br>
  <input type = "radio" name="favorite_language"  value="c++" /> c++ <br>
  <input type = "radio" name="favorite_language"  value="python" /> python <br>
```

Select your Favorite programming languages:
◉ java
○ c++
○ python

Figure 2.16: input type radio

Input Type submit: defines a button used to send all the form data to a form handler:

- The form handler is usually a server-side page containing a script responsible for processing the submitted data.
- The form handler is specified in the `action` attribute of the <form> tag.
- The text displayed on the button is defined by the `value` attribute.

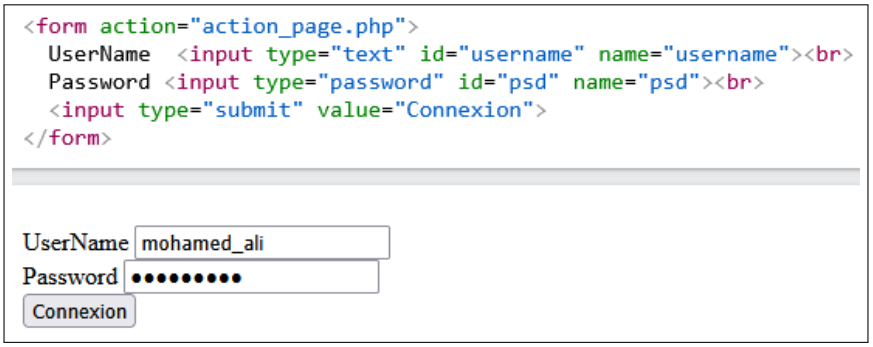

Figure 2.17: input type submit

Input Type reset: defines a reset button that will restore all form values to their default values.

Input Type button: allows displaying a simple button on a web page.

- `<input type="button" onclick="alert('Hello !')" value="Click Me!">`
- Unlike the submit type, it does not automatically trigger form submission.
- The form handler is specified in the `action` attribute of the <form> tag.
- It is mainly used to execute a custom action, usually via JavaScript.

Input Type file: defines a file selection field along with a Browse button to upload a file.

- A window opens to allow the user to select the file.
- You must add the attribute `enctype="multipart/form-data"` to your <form> tag.
- The `accept` attribute specifies the allowed file extension to be uploaded.

Input Type hidden: defines a hidden input field (not visible to the user).

- Example: `<input type="hidden" id="userId" name="userId" value="35">`

```
<form action="action_page.php">
  Select a file:
  <input type="file" id="myfile" name="myfile"><br>
  <input type="submit" value="Submit">
</form>
```

Select a file: Browse... No file selected.

Submit

Figure 2.18: input type file

- A hidden field allows web developers to include data that cannot be seen or modified by users when submitting a form.
- For example, a hidden field can store the identifier of the database record to be updated when the form is submitted.

Input Type file: defines a file selection field along with a *Browse* button allowing the upload of a file.

- A window appears to select the file.
- You must add the attribute `enctype="multipart/form-data"` to your `<form>` tag.
- The `accept` attribute indicates the allowed file extensions.

```
<form action="action_page.php">
  Select a file:
  <input type="file" id="myfile" name="myfile"><br>
  <input type="submit" value="Submit">
</form>
```

Select a file: Browse... No file selected.

Submit

Figure 2.19: File type input field

Other types of the `<input>` tag: HTML5 introduced several new types of form fields to improve user experience and make data entry easier. These types allow certain validations to be performed automatically, without using JavaScript.

Type	Description
`email`	Field intended for entering an email address. The browser automatically checks that the format of the address is correct.
`number`	Numeric field. The user can enter only digits and use small arrows to increase or decrease the value.
`tel`	Allows entering a phone number (numeric keyboard displayed on mobile).
`url`	Field reserved for entering a web address (URL syntax verification).
`range`	Displays a horizontal slider allowing selection of a value within a range.
`date`	Displays a date picker (calendar).
`time`	Allows entering a time using a selector.
`color`	Displays a palette allowing the user to choose a color.

Table 2.2: Some new input field types introduced by HTML5.

10.3.2 The Select Tag

The SELECT tag allows you to create a drop-down list of elements from which the user can make a choice.

- Each element is defined by an <option> tag inside the select tag.
- To define a preselected option, add the selected attribute to the <option> element.
- Each option must have a value attribute that defines the value of this option.

Figure 2.20: the select tag.

10.3.3 The TextArea Tag

The <textarea> element defines a multi-line input field (a text area).

```
Describe yourself : <br>
<textarea name='description' rows='10' cols='50'>500 characters max </textarea>
```

Describe yourself :

500 characters max

Figure 2.21: the textarea tag.

- The name attribute is used to name it, and the rows and cols attributes indicate the height and width respectively.
- To set a default value, insert it between the opening and closing tags.

10.3.4 Other tags:

The Video tag:

- The HTML <video> element is used to display a video on a web page.
- The controls attribute adds video controls, such as play, pause, and volume.
- The <source> element allows specifying alternative video files from which the browser can choose. The browser will use the first format it recognizes.

The Audio tag:

- The HTML <audio> element is used to play an audio file on a web page.
- The controls attribute adds audio controls, such as play, pause, and volume.
- The text placed between the <audio> and </audio> tags will only appear in browsers that do not support the <audio> element.

```
<video width="300" height="200" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  Your browser does not support the video tag.
</video>
```

Figure 2.22: the video tag.

```
<audio controls>
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
Your browser does not support the audio element.
</audio>
```

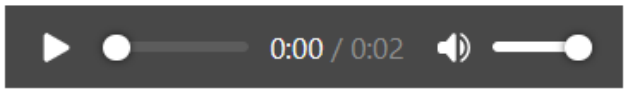

Figure 2.23: the audio tag.

11 HTML5 Semantic tags:

HTML5 introduces several semantic tags that allow for better structuring of a web page's content and make its code clearer and more understandable, both for developers and for search engines or assistive technologies.

- <**header**>: represents the header of a page or a section. It can contain the title, a logo, a navigation menu, or any other introductory element.
- <**footer**>: represents the footer of a page or a section. It generally contains contact

information, credits, or links to legal notices.

- <**section**>: defines a thematic section of a document, grouping content around the same subject.
- <**article**>: represents an independent and self-contained piece of content, such as a blog post, a news item, or a comment.
- <**nav**>: contains a navigation block, such as a menu or a list of links to different parts of the site.
- <**aside**>: defines sidebar content, complementary to the main content, for example a sidebar or information boxes.
- <**main**>: represents the main content of a document, unique and central compared to the rest of the content.

These HTML5 semantic tags make it possible to **give meaning to the content**, while maintaining presentation flexibility via CSS. They allow for:

- Improved **code readability** for developers.
- Better **accessibility** for users of assistive technologies.
- Optimized **SEO referencing** thanks to a clear and hierarchical structure.

12 Inline and Block type tags

In HTML, tags can be classified into two main categories: block and inline.

12.1 Inline tags

The content of inline elements is displayed directly in the text flow, without causing a line break. They are generally used to highlight part of a sentence, such as a bold word, an italicized passage, or a hyperlink [2]. Unlike block-level elements, they do not create new distinct areas in the page and are not designed for precise positioning (although CSS allows manipulating them). Their size is therefore generally determined by their content and context.
Example: `<span>`, `<a>`, `<strong>`, `<em>`, `<b>`, `<i>`, `<u>`, `<small>`, `<sup>`, `<sub>`, `<img>`, `<label>`, `<input>`, `<button>`, `<select>`, `<textarea>`.

12.2 Block tags

Block-level elements are distinguished by their visual rendering which takes up the entire available width and forms a block in the page. This is the case, for example, of a paragraph.

This organization allows them to:

- define precise dimensions (width, height);
- contain other sized elements;
- manage internal (*padding*) and external (*margin*) spacing.

Their essential characteristic is the ability to be **positioned independently of the normal document flow**, which makes them essential for layout.

Example: `<div>`, `<p>`, `<h1>` to `<h6>`, `<header>`, `<footer>`, `<section>`, `<article>`, `<nav>`, `<ul>`, `<ol>`, `<li>`, `<table>`, `<form>`.

Note

The `<div>` and `<span>` tags are "neutral" containers, meaning they have no semantic meaning of their own.

- `<div>` is a **block** element: used to structure large blocks.
- `<span>` is an **inline** element: used to format or manipulate fragments of text.

Finally, thanks to the CSS `display` property, it is possible to transform a block-level element into an inline element (and vice versa), which offers great flexibility in the design of a page (This point will be addressed in the chapter related to CSS).

13 Conclusion

This chapter focused on learning the different tags used to structure an HTML page and to insert various elements (text, images, links, tables, forms, etc.). However, HTML does not allow precise control over the visual appearance of a website. That is why, in the next chapter, we will cover the CSS language, which makes it possible to define the formatting and styling of web pages (colors, fonts, layout, etc.).

14 Exercises

14.1 Creating and formatting an HTML page

Create a web page using only **HTML tags** (content and formatting) in order to obtain a result similar to the figure below. You must use *structure* HTML tags (headings, paragraphs, lists...) as well as *formatting* tags (bold, italic, underline, color, alignment, etc.).

Figure 2.24: HTML formatting

14.2 Tables in HTML

Using **tables** as well as the other **HTML tags** covered in the course, create the web page shown in the figure below. This page should present a **simple CV** structured using tables to organize the different sections (personal information, education, work experience, skills, etc.).

Figure 2.25: Formatting with Tables

14.3 HTML Forms

Create a web page that contains the **form** shown in the figure below. To properly organize and align the different elements of the form (input fields, buttons, etc.), use an **HTML table**.

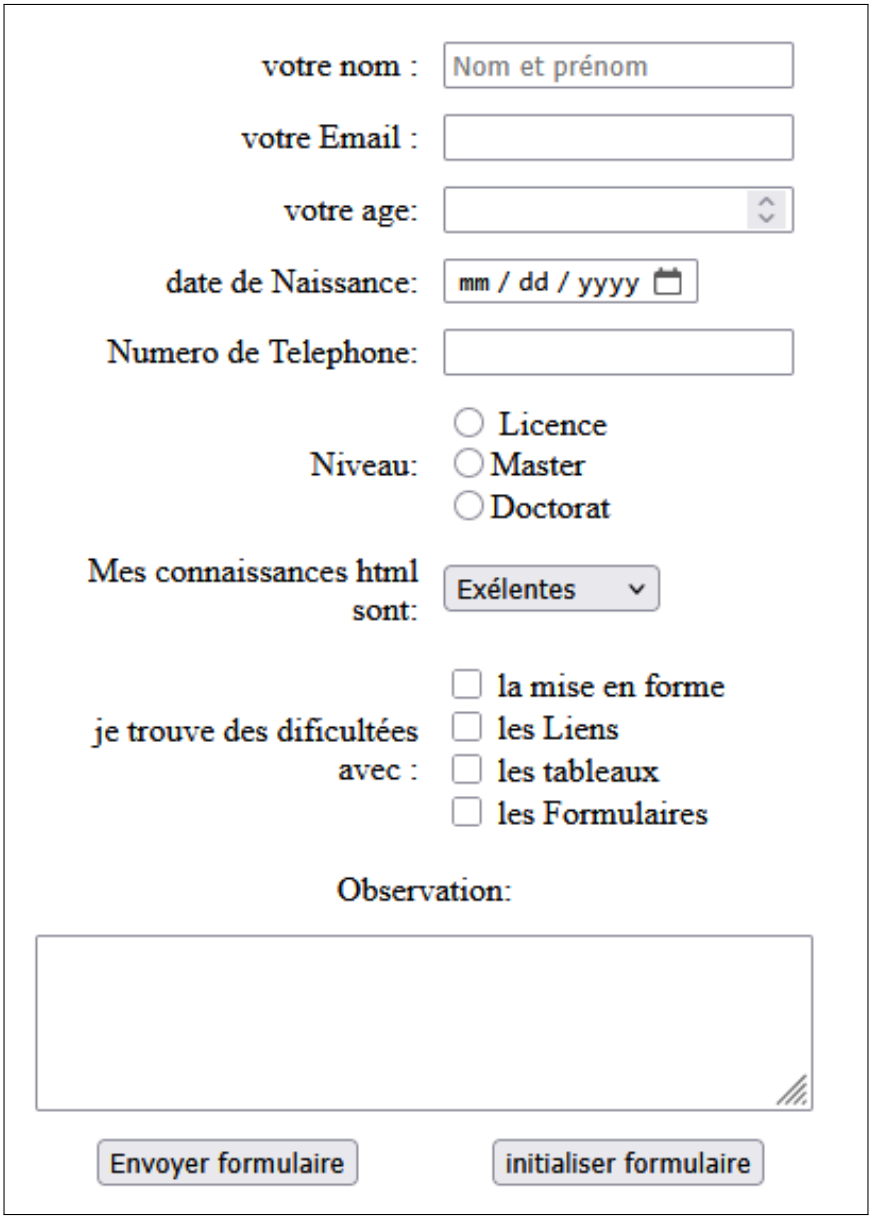

Figure 2.26: HTML forms

CHAPTER

3

CSS LANGUAGE (CASCADING STYLE SHEETS)

1 Introduction

In web development, it is essential to distinguish between the content and the presentation of a page. This distinction makes the code more organized, maintainable, and scalable.

- **The content** corresponds to the structure and elements of the web page: text, images, videos, titles, paragraphs, lists, forms... It is mainly defined with the HTML language, which handles the semantics and organization of the elements.
- **The presentation** corresponds to the visual appearance of the page: colors, fonts, margins, spacing, alignments, visual effects, etc. The presentation is managed by the CSS language (Cascading Style Sheets).

The separation between content and presentation has several advantages:

1. **Easier maintenance:** modifying the style of a page does not require touching the HTML content.

2. **Code reusability:** the same CSS file can be applied to multiple web pages with the same style, reducing code duplication.
3. **Faster display:** the style is loaded only once.
4. **Flexibility in design:** different designs (screens, tablets, printing, etc.) can be experimented with without modifying the site's structure.
5. **Possibility of collaborative development:** several team members can work simultaneously, some on the HTML content (the structure) and others on the CSS (the design).

In summary, **HTML handles the content**, and **CSS handles the presentation**. This separation is a **fundamental best practice** for building professional, clear, and easily scalable websites.

2 Definition

CSS is a language used in web development as a complement to the HTML language, and its function is to create style sheets responsible for the formatting of web documents. It manages aesthetics (colors, typography) and various functionalities such as visual structure, interactive behavior, adaptability, and dynamic effects of a web page [2].

Thanks to the separation of content and presentation, a website is composed of several **XHTML** pages, which define the site's content, and a specific file with the extension `.css`, which defines the appearance and formatting of all the pages.

Thus, modifying the appearance of the site means modifying **only one CSS file**, without having to touch each XHTML page. This approach makes website maintenance much simpler and faster, while ensuring visual consistency across all pages.

To link an HTML page to an external CSS file, we use the `<link>` tag. This tag is placed in the `<head>` section of the HTML page:

```
<link rel="stylesheet" href="style.css">
```

- `rel="stylesheet"`: indicates that the linked file is a style sheet.
- `href="style.css"`: specifies the path to the CSS file.

Note

In CSS, there are three main ways to apply style to a web page:

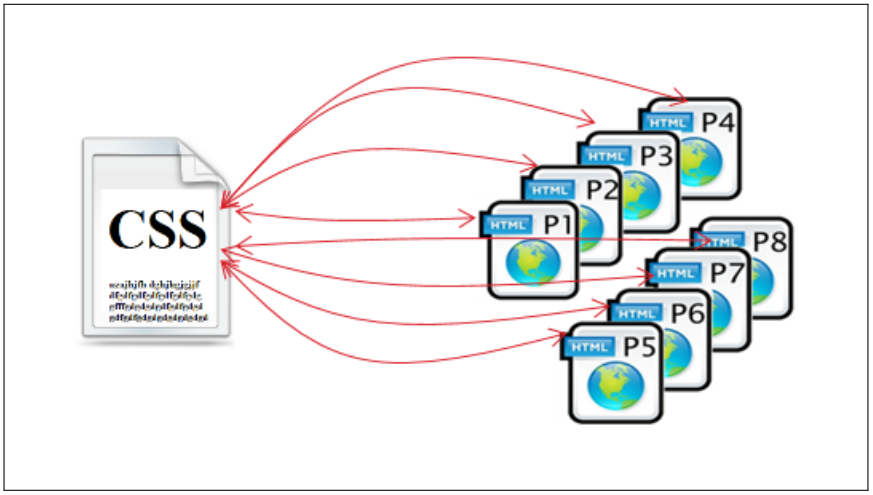

Figure 3.1: the link tag.

- Create a separate file with the .css extension (e.g., style.css).
- Place the CSS directly inside the HTML page, within a <style> tag.
- Apply the style directly to an element using the style attribute.

3 CSS Syntax

A CSS rule consists of a selector and a declaration block. A style sheet can of course contain as many rules as desired.

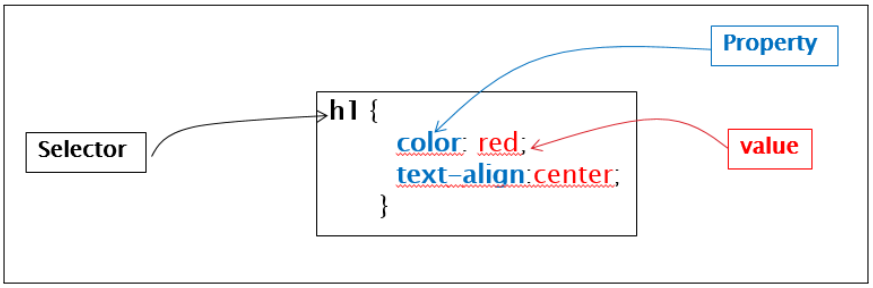

Figure 3.2: a CSS rule

A CSS rule consists of a selector and a declaration block:

- **The selector** indicates the HTML element you want to style.
- **The declaration block** contains one or more declarations separated by semicolons.
- **Each declaration** includes a CSS property name and a value, separated by a colon.

- **Multiple CSS declarations** are separated by semicolons, and declaration blocks are enclosed in curly braces {}.

Example: Center and display paragraphs in red

- **p** is a CSS selector (it designates the HTML element you want to style: <p>).
- **color** is a property, and **red** is the value of this property.
- **text-align** is a property, and **center** is the value of this property.

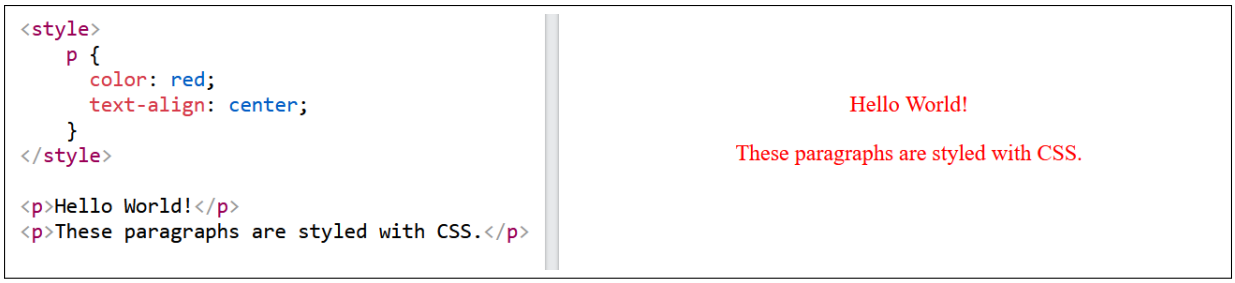

Figure 3.3: example of a CSS rule

4 Types of selectors

The CSS language accepts different forms of selectors. The most important ones are:

4.1 Tag selector

The type selector (or tag selector) in CSS is a simple selector that allows you to target all HTML elements corresponding to the name of this tag (or a set of tags). In other words, it selects all instances of a certain type of tag in an HTML document.

Example all the <h1> elements of the page will be displayed in uppercase with red text color.

4.2 Class selector (.class)

A class is a name freely chosen to represent the concerned elements (which are not necessarily of the same tag). A class selector takes its name by prefixing it with a dot "." [2]

Example We will display in red the hyperlinks of the navigation bar (which belong to the class nav-link).

```
<style>
    h1 {
    color: red;
    text-transform: uppercase;
}
</style>

<h1> this is H1 title </h1>
<p>Every H1 tag will be affected by the style.</p>

<h2> this is H2 title </h1>
<p>only H1 tag will be affected by the style.</p>
```

THIS IS H1 TITLE

Every H1 tag will be affected by the style.

this is H2 title

only H1 tag will be affected by the style.

Figure 3.4: tag selector

```
<style>
  .nav-link {
             color: red;
            }
</style>

<a  href="home.html" class="nav-link"> home </a> <br>
<a  href="login.html" class="nav-link"> login </a> <br>
<a  href="contact.html" class="nav-link"> contact </a> <br>

<h3>Introduction</h3>
To learn web development, the website
<a href="3wschool.com">3wschool</a> is a very good reference.
```

home
login
contact

Introduction

To learn web development, the website 3wschool is a very good reference.

Figure 3.5: class selector

This rule does not apply to <a> tags that do not belong to the nav-link class.

4.3 ID selector (#id)

The **id** selector uses the **id** attribute of an HTML element to select a specific element.

The **id** of an element is unique in a page, so the id selector is used to target a single unique element.

To select an element with a specific id, we write the hash character (#) followed by the id of the element.

Example Only the hyperlink with the attribute id="home" is displayed in red. The rule does not apply to other <a> tags.

4.4 Descendant selector

This selector allows you to target an element contained inside another element.
For example, display <a> tags in red when they are inside <p> tags.

```
<style>
  #home {
         color: red;
        }
</style>

<a  href="home.html" id="home"> home </a> <br>
<a  href="login.html"> login </a> <br>
<a  href="contact.html"> contact </a> <br>

<h3>Introduction</h3>
To learn web development, the website
<a href="3wschool.com">3wschool</a> is a very good reference.
```

home
login
contact

Introduction

To learn web development, the website 3wschool is a very good reference.

Figure 3.6: ID selector

```
<style>
  p a {
         color: red;
        }
</style>

<a  href="home.html"> home </a> <br>
<a  href="login.html"> login </a> <br>
<a  href="contact.html"> contact </a> <br>

<h3>Introduction</h3>
<p>
    To learn web development, the website
    <a href="3wschool.com">3wschool</a> is a very good
    reference.
</p>
```

home
login
contact

Introduction

To learn web development, the website 3wschool is a very good reference.

Figure 3.7: descendant selector

4.5 Pseudo-classes and pseudo-elements

Pseudo-classes and pseudo-elements allow applying styles based on states or relationships that cannot be directly expressed in HTML. Indeed, CSS generates specific elements linked to certain interactions (such as hovering over a link) or to particular structures in the document hierarchy (such as the first paragraph of a block). These methods offer the possibility to style content that is not even present in the HTML code itself.

Example A common example is the use of the :hover pseudo-class, which takes effect when the mouse pointer hovers over the concerned element.

```
<style>
a:hover {
        color: white;
        background-color: red;
}
</style>
```

In this example, when hovering over a hyperlink with the mouse, it is displayed in white with a red background.

4.6 CSS grouping selector

The CSS grouping selector allows you to apply the same style rules to several HTML elements at the same time. Thus, instead of repeating the same code for each element, they are grouped in a single declaration.

For example, in the following code, the <h1>, <h2>, and <p> tags share exactly the same style definitions:

```
h1, h2, p {
        text-align: center;
        color: red;
}
```

5 CSS Properties

5.1 Text-related properties

In CSS, several properties allow controlling the appearance and formatting of text. These properties influence color, size, font, and alignment, thus offering great flexibility in presenting the content of a web page.

Main text-related properties are:

5.1.1 Color

Defines the text color. A color can be specified by its name, hexadecimal code, RGB, or other formats. Example: `color: red;`

5.1.2 font-family

Specifies the font to be used for the text. It is recommended to provide a list of fallback fonts. Example: `font-family: Arial, sans-serif;`

5.1.3 font-size

Controls the size of the text. It can be expressed in pixels (px), em (em), percentage (%), or other units. Example: `font-size: 16px;`

5.1.4 font-weight

Defines the thickness of the text (bold, normal, light). Common values are normal, bold, or a numeric value from 100 to 900. Example: `font-weight: bold;`

5.1.5 text-align

Horizontally aligns the text to the left, right, center, or justified. Example: `text-align: center;`

5.1.6 text-decoration

Allows adding decorations such as underline, line-through, or highlight. Example: `text-decoration: underline;`

5.1.7 line-height

Controls line spacing, i.e., the vertical space between lines of text. Proper handling improves readability. Example: `line-height: 1.5;`

5.1.8 text-transform

Changes the case of the text: uppercase, lowercase, or capitalization of words. Example: `text-transform: uppercase;`

5.1.9 letter-spacing and word-spacing

Respectively adjust the spacing between characters and between words. Example: `letter-spacing: 2px;` or `word-spacing: 5px;`

5.2 Color and background-related properties

CSS properties related to colors and backgrounds allow defining the visual appearance of elements by controlling text color, background colors, and background images. They largely contribute to the graphic identity and ergonomics of a website.

5.2.1 opacity

Defines the transparency of an element and its content. The value ranges from 0 (completely transparent) to 1 (opaque). Example: `opacity: 0.8;`

5.2.2 background-color

Specifies the background color of an element. As with text color, the color can be defined in different formats. Example: `background-color: lightblue;`

5.2.3 background-image

Allows adding an image as a background. This image can be a local file or an online image (URL). Example: `background-image: url('image.jpg');`

There are other properties such as: background-repeat, background-position, background-size, and background-attachment.

5.3 Box-related properties (Box Model)

The CSS Box Model is a fundamental concept that describes the structure of an HTML element as a rectangular box composed of different parts: the content, the padding (inner spacing), the border, and the margin (outer spacing). Box model properties allow controlling these different areas to manage the layout and spacing of elements on the page.

5.3.1 border

Defines the width, style (solid, dotted, dashed, none, etc.), and color of the border. Example: `border: 2px solid black;`

The `border-style` property allows defining the type of border of an HTML element. Not all browsers necessarily recognize all ten possible border styles. Here are the main ones:

- `dashed` : dashed border.
- `dotted` : dotted border.
- `double` : two solid lines of equal thickness, separated by a space of the same thickness.

- `groove` : 3D engraved effect in the page (opposite of `ridge`).
- `hidden` : no border, but affects the adjoining border.
- `inset` : inset effect, the element looks embedded into the page (opposite of `outset`).
- `none` : no border (equivalent to `border-width: 0;`).
- `outset` : outset effect, the element looks extruded from the page (opposite of `inset`).
- `ridge` : 3D raised effect coming out of the page (opposite of `groove`).
- `solid` : solid line.

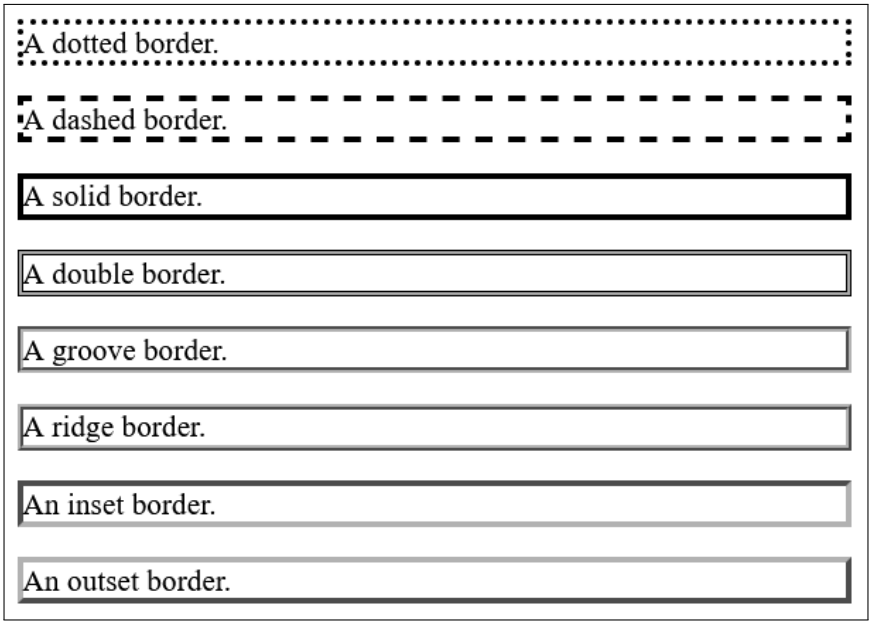

Figure 3.8: types of borders.

5.3.2 width and height

Defines the width and height of the element's content. Example: `width: 200px; height: 100px;`

5.3.3 padding

The `padding` property defines the inner margin of an element, i.e., the space between its content and its border. It can be defined separately for each side:

- `padding-top` : defines the top inner margin of an element.

- `padding-right` : defines the right inner margin of an element.
- `padding-bottom` : defines the bottom inner margin of an element.
- `padding-left` : defines the left inner margin of an element.

Example: `padding: 10px;` or `padding-top: 5px;`

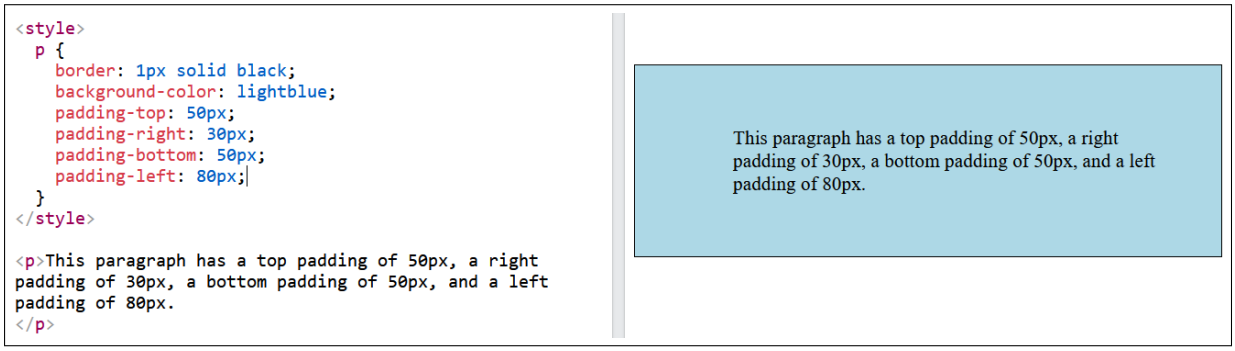

Figure 3.9: padding property.

5.3.4 margin

The `margin` property defines the outer space between the border and adjacent elements. It can also be modified globally or per side. Example: `margin: 20px;` or `margin-left: 15px;`

- `margin-top` : defines the top outer margin of an element.
- `margin-right` : defines the right outer margin of an element.
- `margin-bottom` : defines the bottom outer margin of an element.
- `margin-left` : defines the left outer margin of an element.

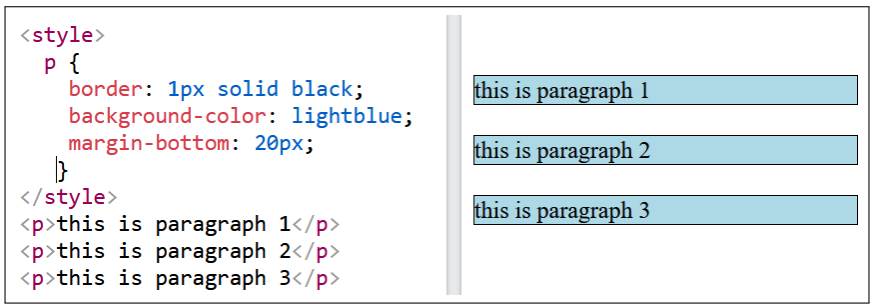

Figure 3.10: margin property.

Notes:

- `margin: auto;` : automatically centers an element horizontally (i.e., `margin-right = margin-le` within its container.
- `margin: 5px;` (without specification): sets all four margins globally in a single declaration.

6 Positioning properties

Positioning and display properties play a central role in the layout of elements on a web page. They determine where and how elements are placed, as well as how they interact with other components of the page.

6.1 display

The `display` property is one of the most important in CSS. It defines how an HTML element is displayed on the page, i.e., its behavior in the document flow.

6.1.1 Main values

- `block`: the element takes up the entire available width and always starts on a new line. (Examples: `<div>`, `<p>`, `<h1>`).
- `inline`: the element is displayed within the text flow without a line break and only takes up the space of its content. (Examples: `<span>`, `<a>`).
- `inline-block`: similar to `inline` but allows defining dimensions (`width`, `height`).
- `none`: the element is hidden and takes up no space on the page.

6.1.2 Modern values for layout

- `flex`: transforms the element into a flexible container to create dynamic alignments.
- `grid`: transforms the element into a grid container for building complex layouts.

Note: Thanks to the `display` property in CSS, it is possible to change the default behavior of elements. For example, an inline tag (such as `<a>`) can be displayed as a block with `display: block`, and conversely, a block tag (such as `div`) can be displayed inline with `display: inline` or `display: inline-block`. This makes it possible, for instance, to place several `div` blocks or several paragraphs on the same line, side by side.

6.2 visibility

The `visibility` property in CSS allows you to make an element visible or invisible. Unlike `display: none`, which completely removes the element from the page flow, an element with `visibility: hidden` becomes invisible but still retains its reserved space.

- `visibility: visible`: the element is displayed (default value).
- `visibility: hidden`: the element is invisible but still occupies space.

To remember

- `display: none` the element is completely removed from the flow (as if it didn't exist).
- `visibility: hidden` the element is hidden but keeps its place, used when you want to temporarily hide content while preserving the layout.

6.3 z-index

The `z-index` property in CSS allows managing the stacking order of elements when they overlap. An element with a higher `z-index` is displayed above those with a lower value. **Note:** The `z-index` property only works on positioned elements (`position: absolute`, `position: relative`, `position: fixed`, or `position: sticky`) as well as on flex items (i.e., direct children of an element with `display: flex`).

```
<style>
  .box{width: 150px;height: 150px;position: absolute;}

  .red{background: red;top: 50px;left: 50px;z-index: 1;}

  .blue{background: blue;top: 100px;left: 100px;z-index: 2;}

  .green{background: green; top: 150px;left: 150px;z-index: 3;}
</style>

  <div class="box red">bloc1 z-index=1</div>
  <div class="box blue">bloc2 z-index=2</div>
  <div class="box green">bloc3 z-index=3</div>
```

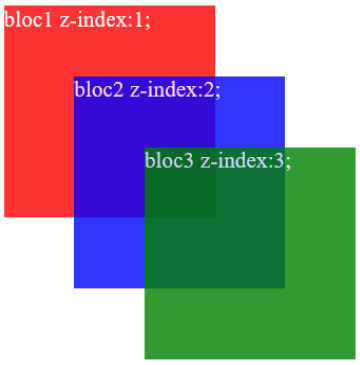

Figure 3.11: z-index property.

6.4 position

The `position` property in CSS defines how an element is positioned on a web page. It plays an essential role in layout and in controlling the placement of elements.

- `static`: default value. Elements are placed according to the normal flow of the page.
- `relative`: the element is positioned relative to its normal position. It can be moved using `top`, `left`, `right`, `bottom`.
- `absolute`: the element is removed from the normal flow and positioned relative to its nearest ancestor with a non-`static` position.
- `fixed`: the element is fixed relative to the browser window, even when scrolling.
- `sticky`: the element behaves like `relative` until it reaches a specified position, then it becomes `fixed`.

These positioning techniques will be detailed in the next section.

7 CSS Positioning Techniques

Positioning in CSS allows precise control over the placement of elements on a web page. The `position` property is used to define the positioning mode, which makes it possible to go beyond the normal document flow. Elements are then placed in their final location using the `top`, `bottom`, `left`, and `right` properties.

7.1 Static Positioning

All HTML elements are positioned using `static`, meaning by default they are always placed according to the normal flow of the page and are therefore not affected by the `top`, `bottom`, `left`, and `right` properties.

The specific behavior and placement of elements in the normal flow are as follows:

- Block-level elements stack vertically: each new block is automatically placed below the previous one and, by default, occupies the full width of its container.
- Inline elements are placed next to each other on the same line. When there is no more space in the container, a line break occurs automatically.

Thus, by default, each element depends on its immediate siblings, and two successive <p> paragraphs appear one under the other.

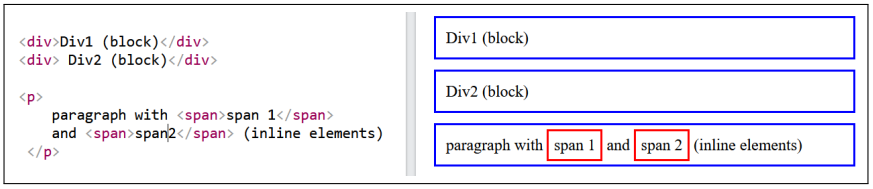

Figure 3.12: static positioning.

7.2 Relative Positioning

An element with `position: relative;` remains in the normal flow of the document, which means it still occupies its initial space. However, it can be moved relative to its normal position using the `top`, `right`, `bottom`, and `left` properties. Despite this visual shift, the other elements on the page behave as if the element had not moved, thus keeping their original position.

Figure 3.13: relative positioning.

In this figure, the blue block is, by default, displayed below the red block, but with relative positioning, we moved it 20 pixels to the right and 20 pixels upward.

7.3 Absolute Positioning

An element with `position: absolute;` is positioned relative to its nearest positioned ancestor (that is, an ancestor with a position other than `static`).

If no positioned ancestor exists, then the element is positioned relative to the initial containing block, usually the browser's viewport.

Concretely, an element with `position: absolute` no longer takes up space in the normal flow, which means other elements behave as if this element does not exist. Its position is defined by the `top`, `right`, `bottom`, and `left` properties, which specify the distance between the element and the edges of its positioned container.

Note: Absolutely positioned elements are removed from the normal document flow and can overlap other elements.

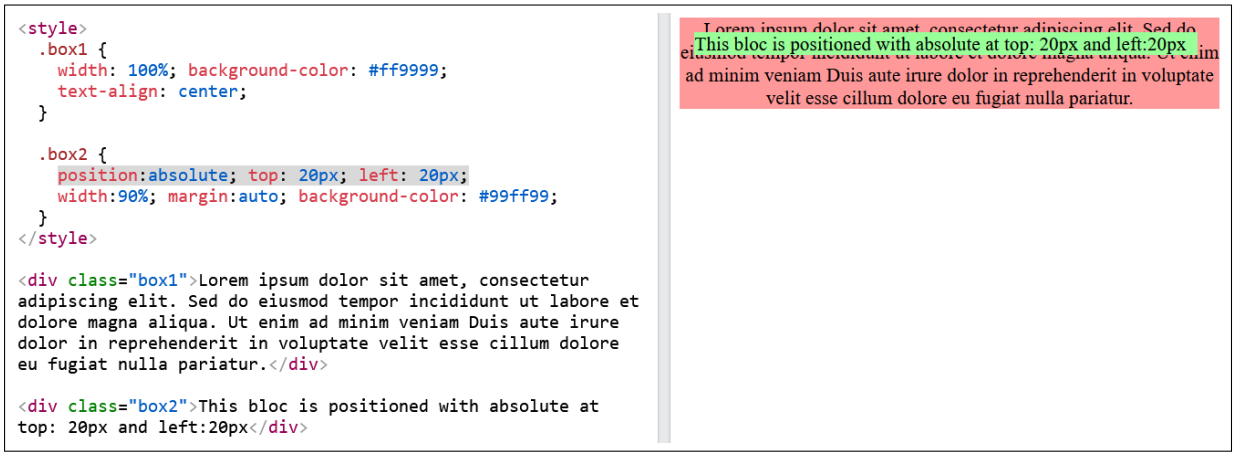

Figure 3.14: absolute positioning.

7.4 Fixed Positioning

Fixed positioning (`position: fixed`) in CSS is a positioning technique that works similarly to absolute positioning, with one essential difference: a fixed element is positioned relative to the visible browser window, not relative to a positioned ancestor. Its position is set using the `top`, `right`, `bottom`, and `left` properties.

This means that the element always remains visible in the same place, even when the page is scrolled. For example, a menu or a button fixed at the top or bottom of the screen will remain visible at all times, regardless of page scrolling.

7.5 Floating Positioning

The `float` property in CSS allows an element to "float" to the left or right of its container. Unlike absolute positioning, a floating element is partially removed from the normal flow, and other inline elements (text, images, etc.) will wrap around it.

- `float: left;` : the element is aligned to the left of its container, with text and other elements wrapping on the right.

- `float: right;` : the element is aligned to the right, with text and other elements wrapping on the left.

```
<style>
    img {
      float: right;
      margin-left:15px;
      width: 200px;
      height: 200px;
    }
</style>

<p align="justify">In this example, the image will float to the
right in the paragraph, and the text in the paragraph will wrap
around the image.
<img src="pineapple.jpg" width="300" height="300">
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus
imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae
scelerisque enim ligula venenatis dolor. Maecenas nisl est,
ultrices nec congue eget, auctor vitae massenenatis. Integer
fringilla congue eros non fermentum. Sed dapibus pulvinar nibh
tempor porta. Cras ac leo purus. Mauris quis diam velit.
</p>
```

In this example, the image will float to the right in the paragraph, and the text in the paragraph will wrap around the image. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massenenatis. Integer fringilla congue eros non fermentum. Sed dapibus pulvinar nibh tempor porta. Cras ac leo purus. Mauris quis diam velit.

Figure 3.15: floating positioning.

Normally, block-level elements (such as `div`, `p`, or `section`) are displayed one below the other in the normal flow of the page. However, with the `float: left;` or `float: right;` property, it is possible to force these blocks to align horizontally, as long as the available space in the container allows it. This technique was widely used for many years to design multi-column layouts before the emergence of more modern and powerful solutions like Flexbox and Grid.

```
<style>
  .bloc {
    width: 150px; height: 50px;
    margin: 10px;
    background-color: lightblue;
    border: 2px solid blue;
    float: left;
  }
</style>

<div class="bloc">Bloc 1</div>
<div class="bloc">Bloc 2</div>
<div class="bloc">Bloc 3</div>
<div class="bloc">Bloc 4</div>
```

Bloc 1 Bloc 2 Bloc 3 Bloc 4

Figure 3.16: Horizontal alignment of blocks using floating positioning.

7.6 Sticky Positioning

Sticky positioning in CSS is a mix between relative positioning and fixed positioning.

An element with `position: sticky` is positioned relatively as long as the page has not been scrolled to a certain point (defined by `top`, `left`, `right`, or `bottom`). Once that

scroll point is reached, the element becomes "sticky" and stays fixed at a defined position (for example, at the top of the window), remaining visible even while scrolling. This mechanism makes it possible, for example, to create menus that remain visible at the top of the page while scrolling, or headers that stay attached while the content continues to scroll.

7.7 Flex positioning

Flexbox (Flexible Box Layout) is a layout model introduced in CSS3. It allows distributing space and aligning elements in a container in a much simpler and more flexible way than float or inline-block.
Flexbox applies to a parent container that becomes a **flex container** thanks to the CSS property `display: flex;`. The direct child elements of this container then become **flex items**.

Flex items are arranged according to:

- a **main axis** defined by the `flex-direction` property (by default horizontal, from left to right),
- a **cross axis**, perpendicular to the main axis defined by the `align-items` property.

What Flexbox allows you to do:

- Align elements in a row or a column, regardless of their size.
- Distribute the available space between the elements (uniform spacing or grouping on one side).
- Design a *responsive* layout where elements adapt to the screen size.
- Reverse the display order of the elements without modifying the HTML code.
- Easily center elements, horizontally or vertically.

Important properties of the `flex container`:

1. `flex-direction`: defines the direction of the main axis
    - `row` (default): horizontal layout,
    - `column`: vertical layout,

- `row-reverse` or `column-reverse`.

2. `flex-wrap`: manages wrapping of elements when they overflow
   - `nowrap` (default),
   - `wrap`,
   - `wrap-reverse`.
3. `justify-content`: controls the distribution of elements along the main axis
   - values: `flex-start`, `flex-end`, `center`, `space-between`, `space-around`, `space-evenly`.
4. `align-items`: sets the alignment of elements on the cross axis
   - values: `stretch` (default), `flex-start`, `flex-end`, `center`, `baseline`.
5. `align-content`: controls the spacing of lines when multiple lines are present.

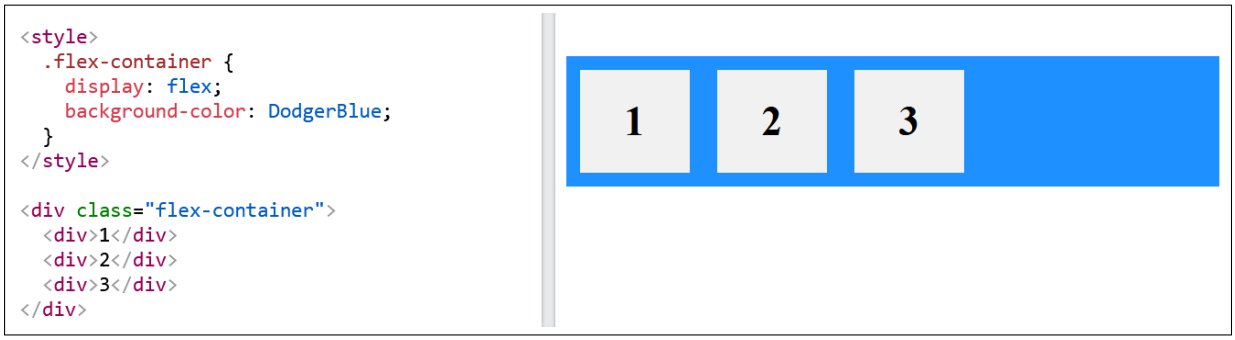

Figure 3.17: flex positioning.

7.8 Grid Positioning

The Grid Layout is a powerful system designed to create two-dimensional layouts (rows and columns). Unlike Flexbox, which is mainly oriented on one axis (row or column), Grid allows managing both rows and columns simultaneously.

What Grid allows you to do:

- Create page structures with well-defined areas (header, menu, content, footer).
- Organize image galleries, portfolios, data tables..
- Do *responsive design* by adapting the grid to screen sizes.
- Overlap elements using the same grid cell.

Basic principles:

1. suppose we have the following HTML structure:

```
<div class="container">
<header class="header">HEADER</header>
<nav class="menu">MENU</nav>
<main class="content">CONTENT</main>
<footer class="footer">FOOTER</footer>
</div>
```

A container becomes a *grid* using the property `display: grid`.

```
.container {
        display: grid;
}
```

The direct children of the container become *grid items*.

2. A grid made of rows and columns is defined using the properties `grid-template-rows` and `grid-template-columns`.

```
.container {
 display: grid;
 grid-template-columns: 1fr 3fr;   /* 2 columns */
 grid-template-rows: auto auto auto;  /* 3 rows */
}
```

Here, we ask for 2 columns: the first column takes 1 fraction (1fr) and the second column takes 3 fractions (3fr). We have 4 fractions: 25% for the first column and 75% for the second.
We can define named areas to organize the grid. for example:

```
.container {
 display: grid;
 grid-template-columns: 1fr 3fr;   /* 2 columns */
 grid-template-rows: auto auto auto;  /* 3 rows */
```

```
 grid-template-areas:
 "header header"
 "menu content"
 "footer footer";
}
```

this means: the first row contains a header that takes the full width (two columns). The second row is divided: menu on the left and content on the right, and the last row contains the footer which takes the full width (two columns).

3. The *grid items* can be placed automatically according to the flow order, or explicitly with the properties `grid-row` and `grid-column` or using the names of the areas. For example, each child element is placed in the area defined by grid-template-areas:

```
.header  { grid-area: header; }
.menu    { grid-area: menu; }
.content { grid-area: content; }
.footer  { grid-area: footer; }
```

Or we can use the properties `grid-row` and `grid-column`

```
.container {
 display: grid;
 grid-template-columns: 1fr 3fr;   /* 2 columns */
}

/* Placement of elements with grid-row and grid-column */
.header {
 grid-column: 1 / 3;  /* occupies 2 columns (from the 1st to the 3rd) */
 grid-row: 1;         /* 1st row */
}

.menu {
 grid-column: 1;      /* 1st column */
 grid-row: 2;         /* 2nd row */
}

.content {
```

```
 grid-column: 2;       /* 2nd column */
 grid-row: 2;          /* 2nd row */
}

.footer {
 grid-column: 1 / 3;   /* occupies 2 columns */
 grid-row: 3;          /* 3rd row */
}
```

Important properties of the Grid container:

- `grid-template-columns`, `grid-template-rows` : define the structure of the grid.
- `gap` : space between rows and columns.
- `justify-items`, `align-items` : alignment of items within their cell.
- `justify-content`, `align-content` : global alignment of the grid inside the container.
- `grid-auto-rows`, `grid-auto-columns` : define the size of rows or columns added automatically.

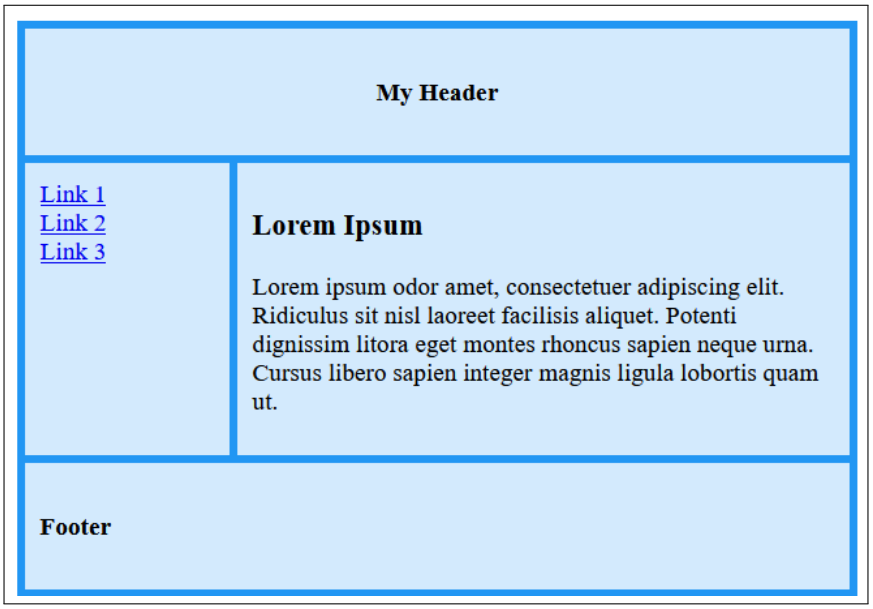

Figure 3.18: grid positioning.

8 Responsive Web Design

8.1 Definition

Web pages can be viewed from a wide variety of devices: desktops, tablets, and phones. A good web page must remain attractive and easy to use regardless of the device. Responsive design consists in adapting the display of a website according to the width (or height) of the viewport (the visible area of the browser).

Type d'appareil	Largeur approximative
Téléphone (portrait)	≤ 480px ou ≤ 600px
Téléphone (paysage)	600px – 768px
Tablette (portrait)	768px – 1024px
Ordinateur portable	1024px – 1280px
Écran de bureau	1280px et plus

Figure 3.19: different types of screens.

We talk about responsive web design when CSS and HTML are used to resize, hide, shrink, enlarge or move the content so that it displays correctly and remains functional on all types of screens. In CSS, this is mainly done through media queries.

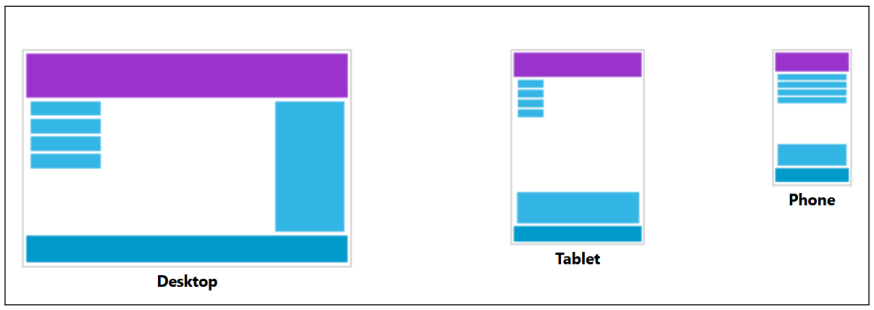

Figure 3.20: responsive web design.

8.2 Media Queries

Media query is a CSS feature introduced with CSS3. It uses the `@media` rule to apply a block of CSS styles only when a specific condition is met.

```
@media (max-width: 600px) {
 body {
  background-color: lightblue;
 }
}
```

In this example, the style defined inside the `@media` applies only if the screen width is less than or equal to 600 pixels. Thus, the background of the page becomes light blue only on small screens, such as those of mobile phones.

9 Example

Consider the following HTML page:

```
<div class="container">
<div class="box">A</div>
<div class="box">B</div>
<div class="box">C</div>
</div>
```

This container includes three blocks A, B, and C. With the CSS style below, the three blocks are displayed side by side thanks to Flexbox.

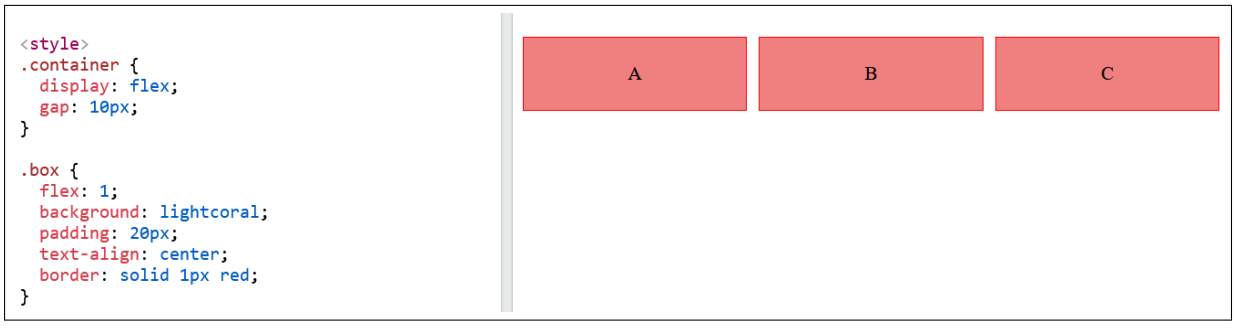

Figure 3.21: design on large screens.

But on small screens (smartphones), the width is reduced. Therefore, we want to display the blocks vertically, one below the other. To do this, we add a media query that adapts the display to this type of screen.

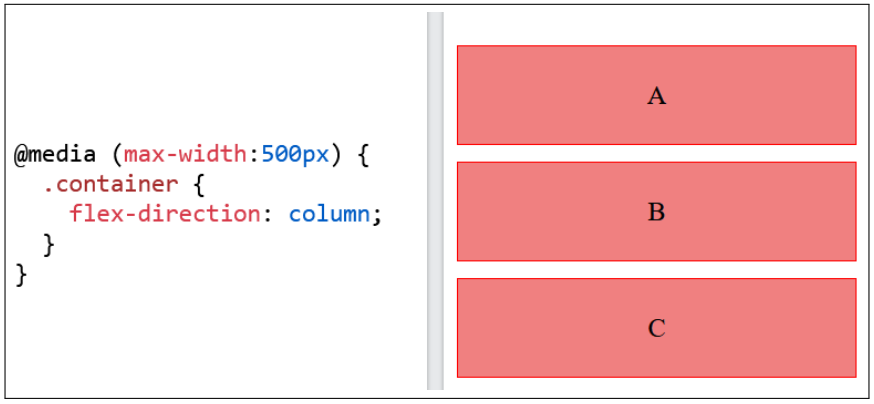

Figure 3.22: design on small screens.

10 Visual Effects and Advanced CSS Styles

CSS offers a wide variety of styles to customize the appearance of HTML elements, beyond simple colors or text sizes. Here are some commonly used style categories:

- **Borders and Rounded Corners**
  - `border-radius` : rounds the corners of an element for a softer look
- **Shadow Effects**
  - `text-shadow` : adds a drop shadow to text to make it stand out
  - `box-shadow` : creates a shadow around a block or box
- **Opacity and Transparency**
  - `opacity` : sets the overall transparency of an element, from 0 (transparent) to 1 (opaque)
- **Color Gradients**
  - `linear-gradient` : creates a linear gradient between multiple colors
  - `radial-gradient` : creates a circular gradient
  - These gradients can be applied as background using `background-image`
- **Transformations and Animations**
  - `transform` : allows rotating (`rotate`), scaling (`scale`), skewing (`skew`) or moving (`translate`) an element
  - `transition` : creates smooth transition effects when a CSS property changes (for example, color change on hover)

These properties allow creating modern, aesthetic, and dynamic interfaces by acting on shape, size, colors, and movements of elements.

11 Conclusion

In this chapter, we have explored the essentials of styling web pages with **CSS**. We have seen how to define selectors and style rules to control the appearance of HTML elements, as well as different positioning techniques: the normal flow, relative and absolute positioning, modern systems like **Flexbox** and **Grid**, and finally adapting to different devices using **media queries**.

12 Exercises

12.1 Styling an HTML Page with CSS

Use **HTML** and **CSS** to create the web page shown in **Figure 3.23**.

Figure 3.23: Styling with CSS

12.2 CSS Positioning

Use the CSS positioning techniques covered in the course to create the web page shown in **Figure 3.24**.

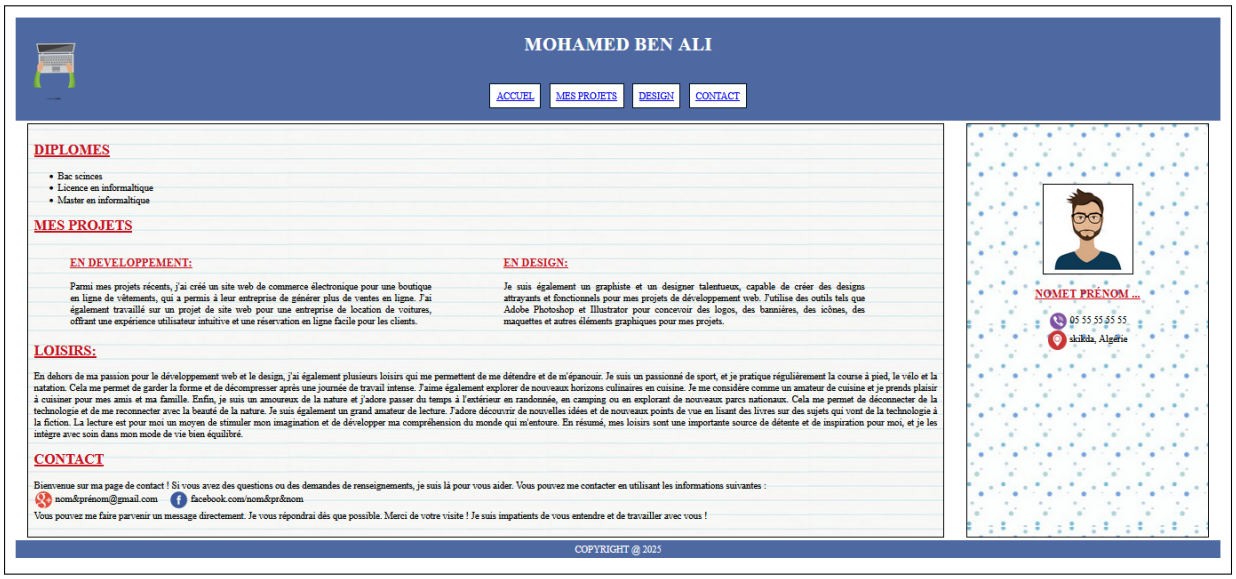

Figure 3.24: CSS Positioning

CHAPTER

4

JAVASCRIPT

1 Introduction

JavaScript is a dynamic, interpreted, and object-oriented programming language, mainly used to make web pages interactive. Designed in 1995 by Brendan Eich at Netscape, it was initially called *LiveScript*. Over time, JavaScript became a web standard, standardized by ECMAScript (ECMA-262). Today, it is used not only in browsers (client-side) but also on the server side through environments such as `Node.js` [5].

2 Possibilities offered by JavaScript

Thanks to JavaScript, it is possible to:

- **Make web pages interactive**: for example, handling button clicks, showing or hiding elements, or reacting to user actions in real time.
- **Modify the document content (DOM)**: add, delete, or dynamically update HTML elements without reloading the page.
- **Control style and appearance (CSS)**: change colors, sizes, positions, or animations of elements based on conditions.

- **Communicate with a server**: send and receive data in the background (AJAX, Fetch API), allowing part of the page to be updated without reloading it entirely.
- **Manage local storage**: store information in the user's browser using *cookies*, *localStorage*, or *sessionStorage*.
- **Create rich applications**: such as games, data visualizations, or complex client-side interfaces.
- **Enhance user experience**: by validating forms before submission, displaying error messages instantly, or adding dynamic effects.

3 Integrating JavaScript into a web page

JavaScript code can be inserted into an HTML page in two ways:

1. **Internal script:** A script can be placed anywhere in the page, and multiple scripts can be included in a single page:

   ```
   <script>
   alert("Hello, welcome to my site!");
   </script>
   ```

   This method is also used to declare functions (in the head tag, in the body tag, or in both).

2. **External script (separate .js file):** this method is useful when the same code is used on multiple web pages. In this case, the script file name is set in the **src** attribute of the `<script>` tag:

   ```
   <script src="myScript1.js"></script>
   <script src="myScript2.js"></script>
   ```

3. Inside an HTML tag: this is called an event handler, which allows scripts to interact with HTML elements:

```
<body>
<form>
name:
<input type="text" value="in uppercase" onFocus="this.value='' " />
</form>
</body>
```

4 Some Basic JavaScript Commands

4.1 alert()

To display a simple message to the user, we use the instruction `alert("message");`. This instruction displays the text between the quotes in a dialog box with an OK button. To continue on the page, the user must click this button.

Example:

```
<body>
<script>
alert("welcome to javaScript")
</script>
...  HTML code ...
</body>
</html>
```

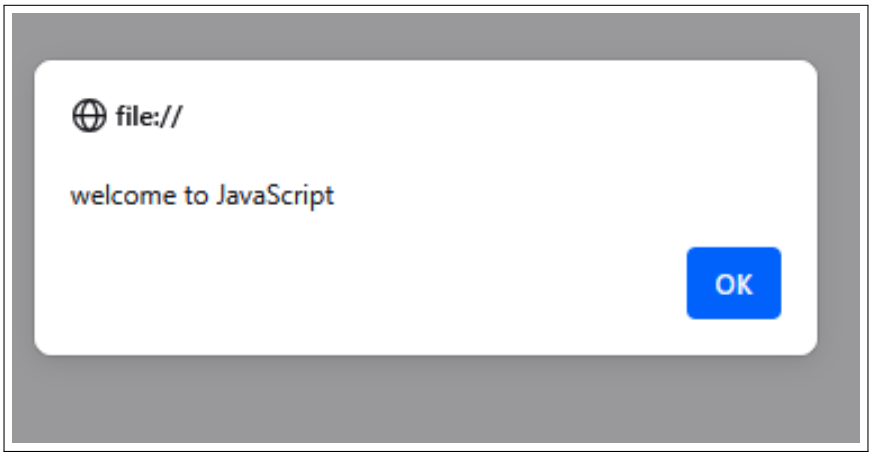

Figure 4.1: The alert function.

4.2 write()

The document.write method (currently obsolete) allows inserting HTML content directly into the web page at the time of loading. For example, one can write:

```
<body>
...  HTML code ...
<script>
document.write("<p> <i> welcome to javaScript </i> </p>")
</script>
...  HTML code ...
</body>
</html>
```

This injects the paragraph into the document flow.

4.3 prompt()

The prompt() function displays a modal dialog box that invites the user to enter text. It is frequently used to collect information directly through a simple interface.
Syntax: `let result = prompt(message, defaultValue);`

- message (optional): text displayed to prompt the user for input.
- defaultValue (optional): initial value displayed in the input field.

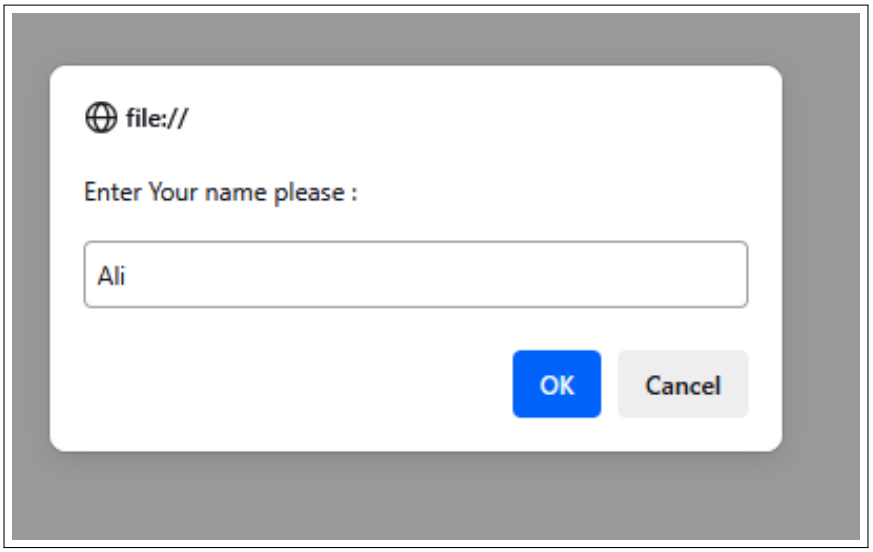

Figure 4.2: The prompt function.

When the box appears, the user can type a response and then click "OK" or cancel the input. The function then returns either the entered string if the user confirms, or null if the user cancels or closes the box.

Example

```
<!DOCTYPE html>
<html>
<body>
<script>
let name = prompt("Enter Your name please :");
if (name !== null) {
 document.write("Hello " + name + " !");
} else {
 document.write("Hello world !");
}
</script>
</body>
</html>
```

The prompt() function is a convenient tool for quick inputs, but it is often replaced in more complex applications by custom HTML forms or user interface libraries.

4.4 confirm()

The confirm() function displays a modal dialog box containing a message and two buttons: OK and Cancel. It is used to request user confirmation before performing an action.

Syntax: `let response = confirm(message);`

- message: an optional string to display in the dialog box.
- The modal box blocks interaction with the page until the user clicks OK or Cancel.
- The function returns a boolean: true if the user clicks OK, false if the user clicks Cancel or closes the window.

Example

```
<script>
if (confirm("Do you really want to delete this item?")) {
```

```
  // Action confirmed by the user
  alert("Item deleted.");
 } else {
  // Action canceled
  alert("Action canceled.");
 }
 </script>
```

Thus, confirm() is a simple and quick tool to validate a user decision before an important operation, especially to prevent accidental actions.

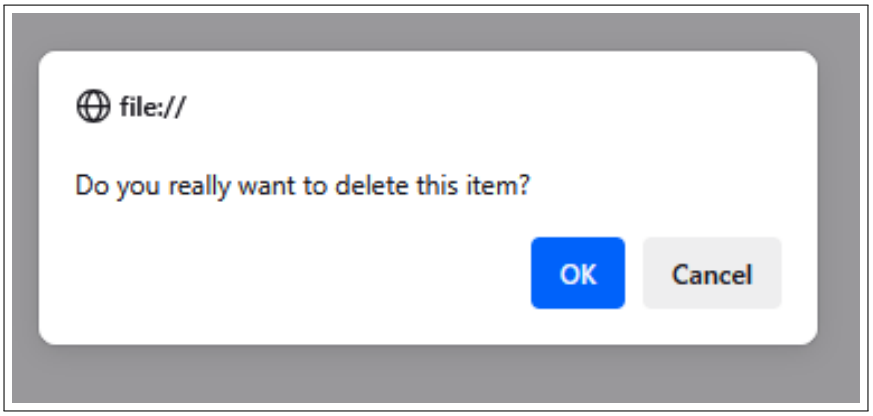

Figure 4.3: The confirm function.

Note

The functions `prompt()`, `confirm()`, and `alert()` are nowadays less used, because they block script execution and offer a limited user experience. They are gradually being replaced by custom forms or modern (modal) dialog boxes created with HTML, CSS, and JavaScript.

5 Variables and Types in JavaScript

5.1 Variable Declaration

In JavaScript, a variable is a memory space used to store a value. There are three main ways to declare a variable:

- `var`: old way to declare a variable (function scope, should be avoided in new code).
- `let`: allows declaring a block-scoped variable.

- `const`: allows declaring a constant whose value cannot be changed.

```
// Example:
var x = 10;        // old syntax
let y = 20;        // regular variable
const PI = 3.14;   // constant
```

5.2 Data Types in JavaScript

JavaScript is a loosely typed and dynamic language, which means that the type of a variable is determined at runtime and can change.

The main primitive types are:

- **Number**: integers and real numbers (e.g. 42, 3.14).
- **String**: character strings (e.g. `"Hello"`).
- **Boolean**: logical values `true` or `false`.
- **Null**: intentional null value.
- **Undefined**: a variable declared but not initialized.
- **Symbol**: unique identifiers (introduced with ES6).
- **BigInt**: for representing very large integers (introduced with ES11).

```
// Example:
let age = 25;            // Number
let name = "Ali";       // String
let isStudent = true;    // Boolean
let address = null;      // Null
let id;                  // Undefined
```

5.3 Objects

In addition to primitive types, JavaScript has the **Object** type, used to represent data collections and more complex entities (arrays, functions, literal objects, etc.).

```
// Example of an object:
let person = {
```

```
 name: "Ali",
 age: 20,
 isStudent: true
};
```

6 Control Structures

JavaScript provides several control structures to manage conditions, repetitions (*with 4 types of loops*), and interruptions in the program flow. These mechanisms, common to most programming languages, are easy to use and play a fundamental role in organizing the logic of a program [9].

6.1 Conditions

There are mainly two types of conditional structures in JavaScript. The first is based on the combined use of `if`, `else if`, and `else` statements. It allows testing a condition and executing different actions depending on whether it is true or false.

The general syntax is as follows:

```
if( condition ) {
 (...)
} else if( condition ) {
 (...)
} else {
 (...)
}
```

The second type of conditional structure is based on the use of `switch`, `case`, and `default` statements. It allows performing different actions depending on the value of a variable, thus offering an alternative to chaining `if...else if` statements.

```
switch( variable ) {
 case value:
 (...)
 break;
 default:
 (...)
}
```

6.2 The `for` Loop

The first iterative structure is based on the `for` keyword. It allows specifying simultaneously:

- the initialization performed at the beginning of the loop,
- the exit condition,
- as well as the operation executed after each iteration.

As long as the exit condition is true, the loop continues to execute.
Example

```
for( var i=0; i<10; i++ ) {
 alert("Counter value: "+i);
}
```

6.3 `For ... in` Loop

The `for...in` loop is a variant of the `for` loop. It combines the `for` keyword with the `in` keyword, the latter specifying the variable used during the iteration. This structure allows iterating over all properties of an object or the indices of an array.
Example:

```
var indexedArray = {
 "key1": "value1",
 "key2": "value2"
};
for( var key in indexedArray ) {
 alert("Value for key "+key+": "+indexedArray[key]);
}
```

6.4 The `while` Loop

The `while` loop allows defining a continuation condition. As long as this condition is true, the instructions within the loop are executed repeatedly.

The general syntax is as follows:

```
while( end condition ) {
 (...)
}
```

6.5 The `do ... while` Loop

The `do...while` loop is a variant of the `while` loop. Its particularity is that the block of instructions is executed at least once, even before the first condition check.

The general syntax is as follows:

```
do {
 (...)
} while( end condition );
```

Note

JavaScript provides the keywords `break` and `continue` to modify the normal flow of a loop:

- `break`: allows immediately exiting a loop and leaving its execution block.
- `continue`: allows skipping directly to the next iteration, ignoring the remaining instructions of the current iteration.

7 Arrays in JavaScript

7.1 Definition

An array in JavaScript is a special object used to store an ordered collection of values (called *elements*). Each element is identified by a numeric index, starting at 0.

- JavaScript arrays can contain elements of different types: numbers, strings, objects, functions, etc.
- The size of an array is not fixed and can change dynamically.
- Indices are always integers, unlike associative arrays (arrays with custom keys, which JavaScript does not directly support).

7.2 Creating Arrays

You can create an empty array or one initialized with elements using the literal notation with brackets []:

```
const emptyArray = [];
let fruits = ["apple", "banana", "orange"];
let numbers = [1, 2, 3, 4, 5];
let mixed = ["Ali", 25, true];
```

It is also possible to use the Array constructor:

```
const numbers = new Array(1, 2, 3, 4);
```

7.3 Accessing and Modifying Elements

Accessing an array element is done using its index:

```
console.log(fruits[0]); // Displays "apple"
fruits[2] = "orange";   // Modifies the 3rd element
```

7.4 Useful Properties and Methods

JavaScript provides many built-in methods to manipulate arrays:

- `length`: returns the number of elements in the array.
- `push()`: adds an element to the end of the array.
- `pop()`: removes the last element of the array.
- `shift()`: removes the first element of the array.
- `unshift()`: adds an element at the beginning of the array.
- `indexOf()`: returns the index of a given element.
- `forEach()`: iterates over the array and applies a function to each element.
- `map()`: creates a new array from the result of a function applied to each element.

7.5 Iterating Over an Array

There are several ways to iterate over the elements of an array in JavaScript.

- `for` **loop**: allows access to elements by their index.

```
let fruits = ["apple", "banana", "orange"];
for (let i = 0; i < fruits.length; i++) {
 console.log(fruits[i]);
}
```

- `for...of` **loop**: allows directly iterating over the values of the array.

```
for (let fruit of fruits) {
 console.log(fruit);
}
```

- `forEach` **method**: executes a given function on each element of the array.

```
fruits.forEach((fruit) => {
 console.log(fruit);
});
```

8 Functions in JavaScript

A function in JavaScript is a block of code that performs a specific task. It is defined once but can be called (executed) as many times as needed. Functions promote code reuse, readability, and organization.

Function Declaration

The general syntax to declare a function is as follows:

```
function functionName(param1, param2, ...) {
 // block of instructions
 return value;
}
```

- **functionName**: identifier chosen by the programmer.
- **parameters**: input values that the function can receive (optional).
- **return**: allows returning a result (optional).

Simple Example

```
function square(x) {
 return x * x;
}

let result = square(5);
console.log(result); // Displays 25
```

Here, the function `square` takes a parameter `x`, calculates its square, and returns the result. By calling `square(5)`, the value `25` is obtained.

Functions with Multiple Parameters

```
function addition(a, b) {
 return a + b;
}

console.log(addition(3, 7)); // Displays 10
```

A function can receive multiple parameters, separated by commas.

Functions without a Return Value

Not all functions need to return a value. They can simply execute instructions.

```
function greet(name) {
 console.log("Hello " + name);
}

greet("Ali"); // Displays "Hello Ali"
```

The `return` Keyword

The `return` keyword stops the execution of the function and returns a value to the calling code. Without `return`, the function returns `undefined` by default.

Note

JavaScript functions do not need to be declared only in the <head>. They can be written in any <script> tag on the page. However, they must be defined before being called.

9 Event Handling in JavaScript

JavaScript allows making a page interactive through **event handling**. An event is an action triggered by the user or the browser, for example: clicking a button, hovering over an element, typing in a field, or loading a page.
Example of events:

- `onclick`: triggered on a click.
- `onmouseover`: triggered when the mouse hovers over an element.
- `onkeydown`: triggered when a key is pressed.
- `onload`: triggered when the page loads.

9.1 Methods for Handling Events

1. HTML Attribute Directly in the Tag

```
<button onclick="alert('Button clicked!')">Click</button>
```

2. Assign a Function in JavaScript

```
let button = document.getElementById("btn");
button.onclick = function() {
 alert("Button clicked!");
};
```

3. Using addEventListener (Recommended Method)

```
let button = document.getElementById("btn");
button.addEventListener("click", () => {
 alert("Button clicked!");
});
```

Complete Example

```
<!DOCTYPE html>
<html>
<body>
<p id="message">Initial text</p>
<button id="btn">Change the text</button>

<script>
let button = document.getElementById("btn");
button.addEventListener("click", function() {
 alert('ok');
});
</script>
</body>
</html>
```

In this example:

- In JavaScript, the button is retrieved using `getElementById("btn")`.
- A `click` event is assigned to this button using the `addEventListener` method.
- When the user clicks, the function displays an alert message.

10 Main Events in JavaScript

Events in JavaScript are triggered either by the user (such as a mouse click or keyboard input) or by the browser (when a page loads, when the window is resized, etc.). They are classified into several categories according to their nature. Here is a list of the most commonly used events in JavaScript, grouped by category:

1. Mouse Events

- `onclick`: click on an element.
- `ondblclick`: double-click.
- `onmouseover`: mouse passes over an element.
- `onmouseout`: mouse leaves an element.
- `onmousemove`: mouse movement.
- `onmousedown`: mouse button pressed.
- `onmouseup`: mouse button released.

2. Keyboard Events

- `onkeydown`: when a key is pressed.
- `onkeyup`: when a key is released.
- `onkeypress`: when a key is pressed and released.

3. Form Events

- `onsubmit`: form submission.
- `onreset`: form reset.
- `onfocus`: when an element gains focus.
- `onblur`: when an element loses focus.
- `onchange`: when a value changes in a field.
- `oninput`: when a user types text in a field.

4. Window / Document Events

- `onload`: complete loading of the page or an image.
- `onunload`: page closing or refresh.
- `onresize`: window resizing.

- `onscroll`: scrolling within the page.
- `onerror`: when an error occurs while loading a script or an image.

11 DOM and Accessing HTML Elements

11.1 Definition

The **DOM** (Document Object Model) is a hierarchical representation of an HTML (or XML) document. In JavaScript, the DOM allows access to and dynamic manipulation of the content, structure, and style of a web page. Each HTML tag becomes a **node** in this tree: document, elements, attributes, and text.

The DOM allows JavaScript to read, modify, add, or remove elements in a web page dynamically. For example:

- **Access HTML elements**: by their `id`, `class`, `name`, or tag type.
- **Read and modify content**: text, attributes (such as `src`, `href`, `value`, `checked`), or CSS styles.
- **React to user actions**: click, hover, input in a field, option selection, etc.
- **Create or remove elements dynamically**: add a new row to a table, insert a paragraph, delete a button.

Example consider the following HTML code:

```
<div id="myZone">
<span><b>A text</b></span>
<div id="myOtherZone">
Another text
</div>
</div>
```

This HTML code fragment corresponds to the hierarchical structure illustrated in the following figure. The document object is the root. HTML tags are connected through parent or child relationships.

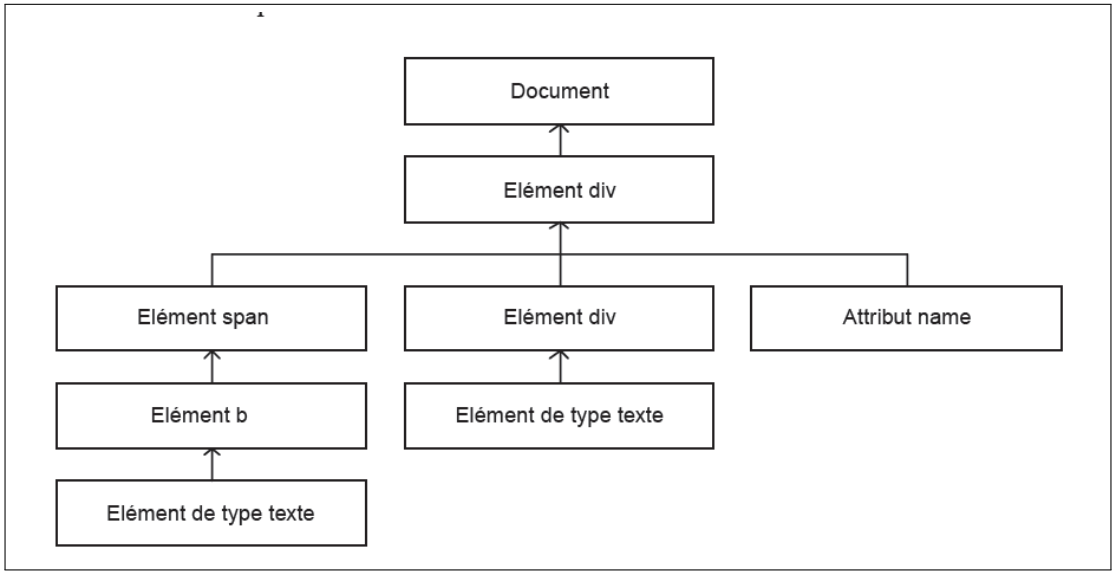

Figure 4.4: DOM tree structure of the HTML code.

11.2 Accessing HTML Elements

JavaScript provides several methods to access elements of a page. The most commonly used are:

- `document.getElementById(id)`: selects an element by its ID.
- `document.getElementsByTagName(name)`: selects all elements with the same tag name (returns a collection).
- `document.getElementsByClassName(name)`: selects all elements with the same class (returns a collection).
- `document.querySelector(selector)`: selects the first element matching the CSS selector.
- `document.querySelectorAll(selector)`: selects all elements matching the CSS selector (returns a NodeList).

11.3 Example of Access and Modification

The `innerHTML` property is one of the most commonly used for manipulating the DOM. It allows you to **read** or **modify** the HTML content of an element.

- **Reading:** retrieve the HTML content of an element.
- **Writing:** insert or replace the content of an element.

```
<!DOCTYPE html>
<html>
<head>
<title>DOM Example</title>
</head>
<body>
<h1 id="title">Hello</h1>
<p class="text">Paragraph 1</p>
<p class="text">Paragraph 2</p>

<script>
// Access by ID
document.getElementById("title").innerHTML = "Hello in JavaScript";

// Access by class
let paragraphs = document.getElementsByClassName("text");
paragraphs[0].style.color = "red";   // change the color of the 1st paragraph
paragraphs[1].innerHTML = "Modified text!";

// Access with querySelector
let h = document.querySelector("h1");
h.style.backgroundColor = "yellow";
</script>
</body>
</html>
```

Note

- `innerHTML` completely replaces the content of an element.
- You can insert both **plain text** and **HTML tags**.
- To insert only text without HTML interpretation, use `textContent` instead.
- With the DOM, it is possible to **modify content**, **change CSS styles**, or **add/remove elements** in real time, making web pages interactive and dynamic.

Conclusion

In this chapter, we explored the **JavaScript** language, an essential programming language for making web pages **dynamic** and **interactive**. We saw how to manipulate variables, arrays, functions, manage events, and interact with the **DOM** to modify the content of an HTML page in real time.

JavaScript thus acts as a **client-side** language, executed directly in the user's browser.

In the next chapter, we will move to the **server-side** with the **PHP** language. It allows dynamic web content generation, communication with a database, and the development of full web applications.

12 Exercises

12.1 Calculate an Operation in JavaScript

Using JavaScript, create a web page in **HTML** that allows performing an arithmetic operation between two numbers a and b.

- The user must be able to select the operation to execute using a **dropdown list**.
- The **result** of the operation must be displayed in a designated area.

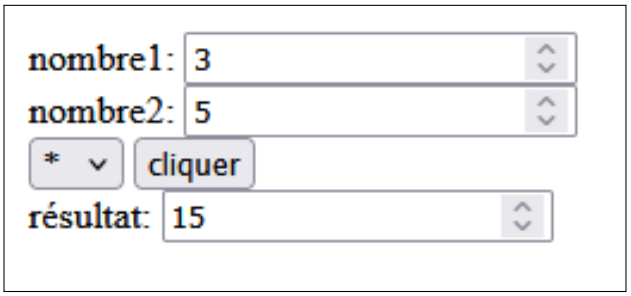

Figure 4.5: Calculate an operation with JavaScript

12.2 Form Validation with JavaScript

Create a web page in **HTML** containing a registration form for a new student. The form must include the following fields:

- **Username**: must contain more than 5 characters.

- **Last Name**, **First Name**, and **Password**: must not be empty.
- **Password** (entered twice): both values must be identical and contain at least 6 characters.
- **Age**: must be between 10 and 100.

Using a function in **JavaScript**, check the value entered in each field and display an **error message** if the corresponding rule is not respected.

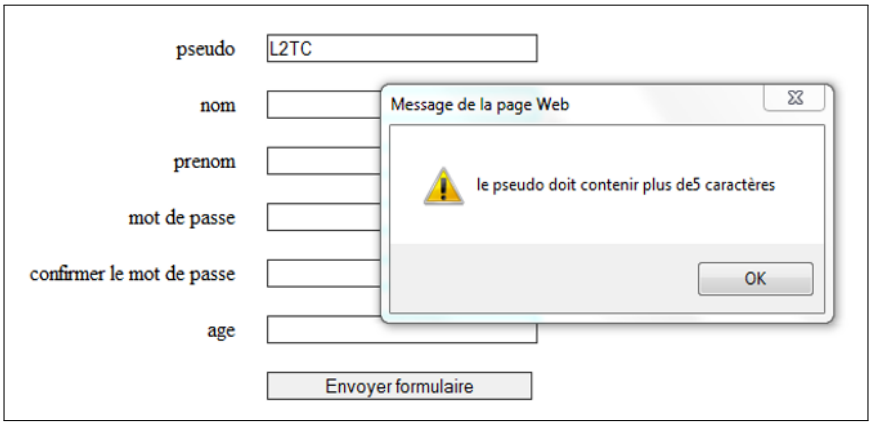

Figure 4.6: Validate a form with JavaScript

12.3 Creating an Interactive Multiple-Choice Test with JavaScript

Create a web page in **HTML** containing a small test in the form of a **multiple-choice questionnaire (MCQ)**. The user must select exactly **2 choices out of 3 options**.

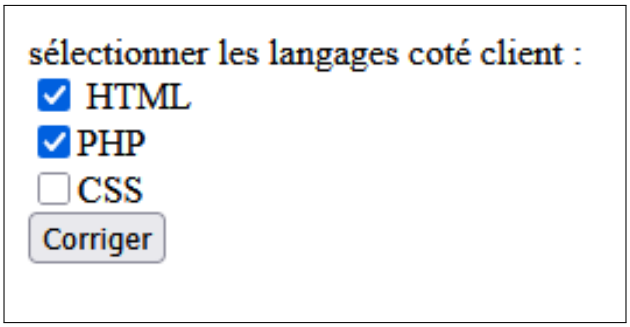

Figure 4.7: Interactive MCQ with JavaScript

- A function in **JavaScript** must check the user's answers.
- If the selected combination corresponds to the correct answer, display a success message.
- Otherwise, display an **error message**.

CHAPTER

5

PHP LANGUAGE

1 Introduction

PHP (Hypertext Preprocessor) is an open-source programming language primarily used for creating dynamic web pages. Unlike static HTML, which always displays the same content, PHP allows generating web pages that can vary depending on users, data, or other interactions.

PHP was created by Rasmus Lerdorf in 1994 for his personal needs. It allows developers to quickly design dynamic web pages capable of displaying the results of calculations or SQL queries performed on a database management system (DBMS).

Its syntax is inspired by C, Java, and Perl, while providing some unique features. PHP is suitable for both beginners who want to easily explore dynamic web techniques and professionals seeking a solution that is simple, powerful, and reliable. Today, PHP powers millions of websites, including popular CMSs such as WordPress, Drupal, and Joomla.

2 Operating Principle

A PHP script is interpreted by a server-side program, which distinguishes PHP from a language like JavaScript that runs directly in the user's browser. Generally, the PHP interpreter is integrated into the Apache web server as a module, which greatly simplifies its use. When a page with a .php extension is requested, the server loads the file into memory and identifies the PHP code portions to execute. The interpreter then processes these scripts, generating HTML code that replaces the PHP instructions in the document sent to the browser. Thus, the user never receives PHP code, only HTML ready to be displayed [8].

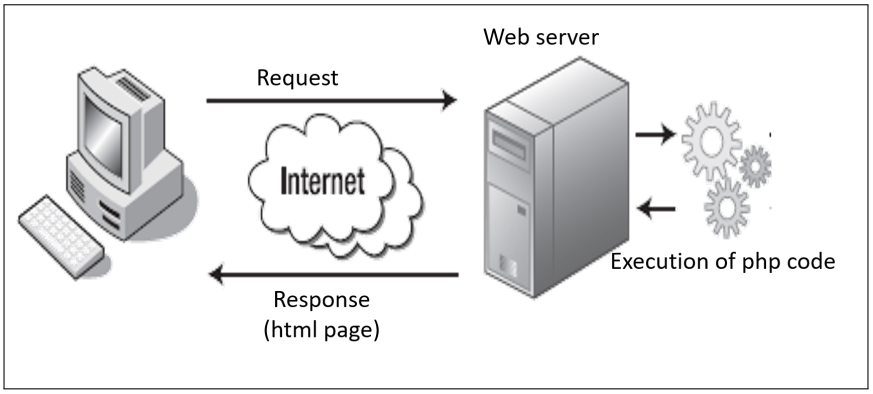

Figure 5.1: PHP operating principle.

3 Development Environment

To run a website developed in **PHP**, it is essential to set up a **minimal platform** composed of three complementary elements:

- **The web server** (such as Apache or IIS): it acts as an intermediary between the user and the application. It receives requests sent by the browser, interprets them, and delivers the appropriate response.
- **The PHP interpreter**: this is the engine that executes PHP scripts on the server side. It transforms PHP code into content understandable by the browser, most often **HTML**, sometimes enriched with CSS and JavaScript.
- **The Database Management System (DBMS)**: usually **MySQL**, it allows storing and managing data in a structured way (users, articles, orders, etc.) and makes it accessible to PHP code.

These three components can be installed separately and configured manually. However, to simplify the setup of a complete environment, there are integrated solutions called **distribution packs**. These packs combine the web server, PHP interpreter, and DBMS into a single installation. Among the most popular are:

1. **EasyPHP**: easy to use and particularly suitable for Windows environments.
2. **XAMPP**: cross-platform (Windows, Linux, macOS) and offering a richer configuration, commonly used for both development and learning purposes.

Figure 5.2: Platforms for running a PHP website.

Thanks to these packs, even a beginner can quickly have a functional platform to develop, test, and run PHP/MySQL applications. There are also other options, such as **WampServer** (widely used on Windows), **MAMP** (mainly for macOS), and **Laragon** (lightweight and modern for Windows). These alternatives provide similar functionalities and allow developers to choose the tool best suited to their environment and needs.

4 First PHP Page

To execute a PHP script, the file must be placed in the root folder of the local web server. With EasyPHP, this folder is usually called `www`, whereas with XAMPP, it is the `htdocs` folder located in the XAMPP installation directory (for example, `C:\xampp\htdocs` on Windows).

When the Apache web server is running, it treats these folders as the roots of the local site. To display your PHP script in a browser, you must access `http://localhost/file_name.php` or, if your files are organized in subfolders, `http://localhost/folder_name/file_name.php`. The server will then interpret the PHP code in the file and send only the generated HTML content to the browser.

5 PHP Syntax

A PHP script can be inserted anywhere within a document. It starts with `<?php` and ends with `?>`. In practice, a PHP file usually contains **HTML** tags along with **PHP** code intended to produce dynamic content. By convention, the extension assigned to PHP files is `.php`.

Figure 5.3: Example of a PHP page.

The `echo` statement in PHP is used to display content on a web page. It directly outputs the text placed between parentheses (or quotes) in the PHP code, and this text becomes visible in the HTML sent to the browser.
For example, the following code displays the message "Hello World!" underlined in the generated page:

```
<?php
echo " <u> Hello World! </u>";
?>
```

6 Variables

In PHP, a variable is a container used to store data of different types, such as numbers, text, or boolean values. Each variable always starts with a dollar sign $ followed by the variable name.
PHP is a weakly typed and dynamic language, which means the variable type is determined automatically according to the assigned value and can change during the script execution.

6.1 Variable Declaration

In PHP, there is no explicit variable declaration. The interpreter automatically creates a variable as soon as a new name, preceded by the symbol `$`, appears in the script. Thus, the following statement assigns the value `1` to the variable `$myVariable`, even if it was not previously defined.

Example

```
<?php
$name = "Mohamed";          // string with double quotes
$firstname = 'Ben ali';     // string with single quotes
$age = 22;                  // integer
$height = 1.75;             // float
$isStudent = true;          // boolean
?>
```

6.2 Displaying Variables

Displaying variables in PHP is mainly done using the `echo` statement. It allows the content of a variable to appear in the HTML code sent to the browser. The `print` statement can also be used, but `echo` is more common as it is slightly faster and accepts multiple parameters.

Example 1: Simple Display

```
<?php
$name = "Mohamed";
echo $name;   // Displays: Mohamed
?>
```

Example 2: Concatenation of Text and Variables

```
<?php
$firstname = "Mohamed";
$name = "Ben Ali";
echo "Hello " . $firstname . " " . $name;
// Displays: Hello Mohamed Ben Ali
?>
```

Example 3: Interpolation in Strings

When using double quotes (" "), PHP directly interprets the content of variables inside the string:

```
<?php
$age = 22;
echo "I am $age years old";
// Displays: I am 22 years old
?>
```

7 Control Structures in PHP

Control structures allow modifying the flow of a program based on conditions or repetitions. As in many languages, PHP provides several types of structures: conditions, loops, and jump statements.

7.1 Conditions

Conditions allow executing a block of instructions only if a logical expression is true.

```
<?php
$age = 20;

if ($age >= 18) {
 echo "You are an adult.";
} else {
 echo "You are a minor.";
}
?>
```

7.2 Loops

Loops are used to repeat a block of instructions multiple times. PHP provides several types of loops:

- **while** : executes as long as a condition is true;

- **do ... while** : executes at least once, then continues while the condition is true;
- **for** : used when the number of iterations is known;
- **foreach** : used to iterate over an array or a collection.

```
<!DOCTYPE html>
<html>
<body>
<h4> Table with php </h4>
<table border='1'>
  <?php
        for($i=1; $i<=5; $i++)
        echo "<tr><td> row $i </td></tr>" ;
  ?>
</table>
</body>
</html>
```

Table with php

row 1
row 2
row 3
row 4
row 5

Figure 5.4: Example of a for loop in PHP.

7.3 Jump Statements

PHP also provides statements that allow modifying the execution flow within loops:

- **break** : completely exits a loop;
- **continue** : skips the current iteration and moves to the next one.

```
<?php
for ($i = 1; $i <= 5; $i++) {
 if ($i == 3) {
  continue;  // Skip displaying 3
 }
 echo "Value: $i <br>";
}
?>
```

8 Organizing Code with include and require

In web development, it is common to separate certain parts of the code into distinct files to avoid repetition and to make maintenance easier. PHP allows including these files in a script using the `include` and `require` statements.

Example 1 You can put the header code (`header.php`) and the footer (`footer.php`) in separate files, then include them in `index.php`:

```
<!-- file header.php -->
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>My PHP Site</title>
</head>
<body>
<h1>Welcome to my site</h1>

<!-- file footer.php -->
<footer>
<p>© 2025 - My site</p>
</footer>
</body>
</html>

<!-- file index.php -->
<?php
include 'header.php';
echo "<p>Main content of the page.</p>";
include 'footer.php';
?>
```

Example 2 You can also centralize configuration settings in a `config.php` file (for example, for database connection) and include it in other scripts:

```
<!-- file config.php -->
<?php
$db_host = "localhost";
$db_user = "root";
$db_pass = "";
$db_name = "my_database";
?>

<!-- file connexion.php -->
```

```
<?php
require 'config.php';
echo "Connecting to the database: " . $db_name;
?>
```

Note

- `include` continues execution even if the file is not found (warning).
- `require` stops execution if the file is not found (fatal error).

9 Functions in PHP

A function is a reusable block of code that performs a specific task. Using functions helps organize code, avoid repetition, and makes programs easier to maintain.

9.1 Declaring a Function

In PHP, a function is declared using the `function` keyword:

```
<?php
function sayHello() {
 echo "Hello everyone!";
}

// Calling the function
sayHello();
?>
```

9.2 Functions with Parameters

A function can receive parameters to customize its execution:

```
<?php
function greet($name) {
 echo "Hello, " . $name . "!";
}
```

```
greet("Mohamed");
greet("Ali");
?>
```

9.3 Functions with Return Value

A function can also return a value using the `return` statement:

```
<?php
function add($a, $b) {
 return $a + $b;
}

$result = add(5, 7);
echo "The sum is: " . $result;
?>
```

9.4 Notes

- Function names are case-insensitive in PHP.
- There are many predefined functions in PHP (for strings, arrays, files, etc.).

10 PHP and Forms

HTML forms allow users to input data and send it to the server. In PHP, this data can be retrieved and processed using the superglobals `$_GET` or `$_POST`.
When the form is submitted using the POST method, data is retrieved with the superglobal `$_POST`, whereas if it is submitted using the GET method, data is retrieved with the superglobal `$_GET`.

10.1 Simple Example

```
<form action="welcome.php" method="POST">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
```

```html
<!DOCTYPE HTML>
<html>
<body>

<form action="welcome.php" method="post">
  Name: <input type="text" name="name"><br>
  E-mail: <input type="text" name="email"><br>
  <input type="submit" value ="send">
</form>

</body>
</html>
```

Name: Ali
E-mail: ali@gmail.com
send

Figure 5.5: PHP and Forms.

In the file `welcome.php`, the form data is retrieved as follows:

```
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>
```

In this example, the page will display:

```
Welcome ali
Your email address is: ali@gmail.com
```

11 Sending Data via Hyperlinks

When clicking on a hyperlink containing parameters in the URL, they are sent to the server using the `GET` method. In PHP, this data is accessible through the superglobal `$_GET`.

For example, the following link:

```
<a href="page.php?nom=ali&age=20">Link with parameters</a>
```

will send to the script `page.php` the data with the keys `nom` and `age`, which can be retrieved using the following PHP code:

```
<?php
echo "Name: " . $_GET['nom'] . "<br>";
echo "Age: " . $_GET['age'];
?>
```

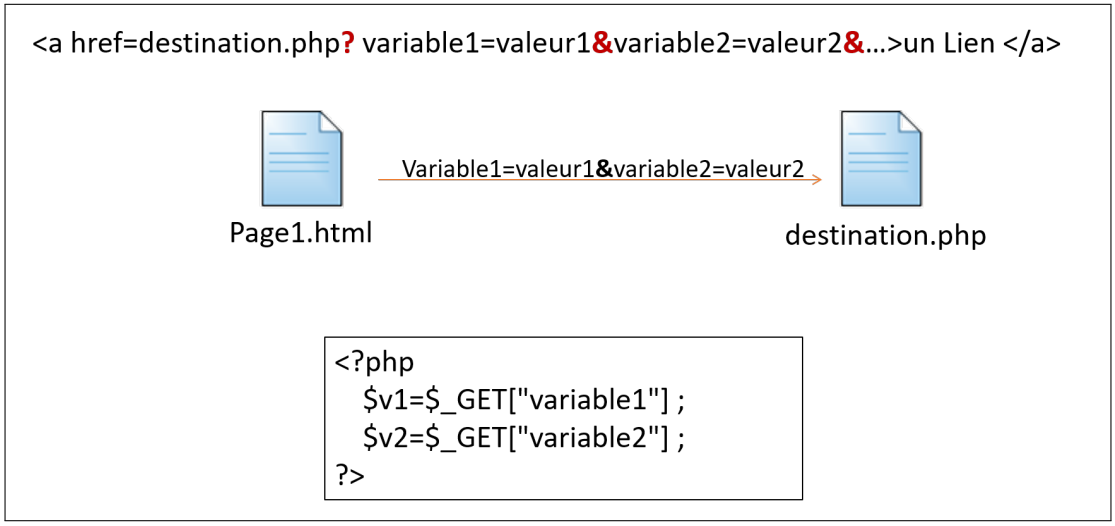

Figure 5.6: Sending data via hyperlinks.

Thus, PHP automatically handles parameters passed in the URL and makes them available via `$_GET`, allowing data to be transmitted between pages without using a form.

Note

When submitting a form to the **same page**, it is recommended to check if the variables exist before using them. In PHP, this is done using the `isset()` function:

```
<?php
if (isset($_POST['name'])) {
 echo "Hello " . $_POST['name'];
}
?>
```

This avoids errors when accessing undefined indices in `$_POST` or `$_GET`.

12 Database Manipulation with PHP

PHP works natively with a **MySQL** database. **MySQL** is a *Database Management System* (DBMS) that allows information to be structured into **databases**, **tables**, **fields**, and **records**, and manipulated using a specific query language: **SQL** (Structured Query Language).

The workflow can be summarized as follows:

1. The user requests, via their browser, access to a `.php` web page containing instructions to connect to a MySQL database.

2. The **web server** forwards this request to the PHP interpreter.
3. The **PHP interpreter** executes the script code. When it encounters database-related instructions, it sends the corresponding SQL query to the **MySQL server**.
4. The **MySQL server** executes the query (read, insert, update, or delete) and returns the resulting data set or a success/error message to PHP.
5. PHP continues processing, dynamically builds the final **HTML** content, and returns it to the **web server**.
6. Finally, the web server delivers the generated HTML page to the user's browser.

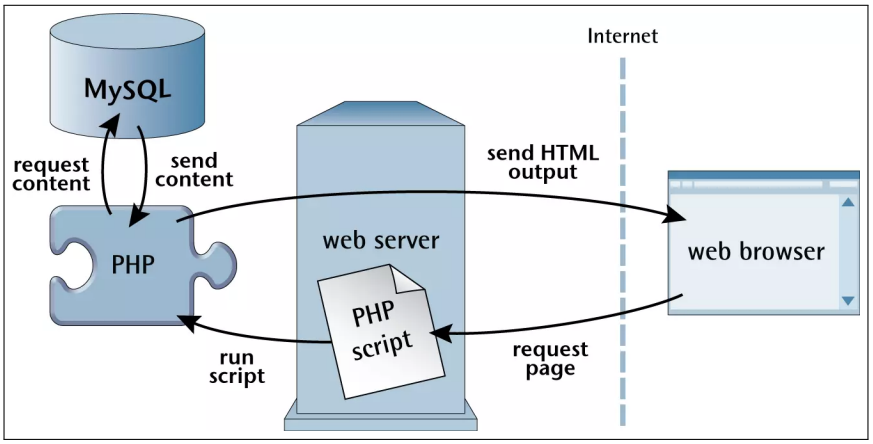

Figure 5.7: PHP and MySQL.

Thus, PHP acts as an intermediary between the client (browser) and the MySQL database, translating the developer's instructions into SQL queries and converting the results into dynamic web pages.

Note

Although PHP can communicate with several **DBMS**, the most common association remains undoubtedly with **MySQL**, due to its simplicity, popularity, and native integration with PHP. To interact with MySQL, PHP provides three main methods:

- **PDO (PHP Data Objects)**: an object-oriented, generic interface compatible with multiple DBMS;
- **MySQLi in procedural mode**: a simple approach, close to traditional functions;
- **MySQLi in object-oriented mode**: a modern and better-structured version of MySQLi.

In this course, we will focus on using the **MySQLi extension**, as it is specifically designed for MySQL and is very practical and efficient for handling databases in an educational context.

13 Creating a Database with phpMyAdmin

Creating a database is not done directly via PHP, but through the MySQL Database Management System (DBMS). To facilitate this task, environments like EasyPHP or XAMPP include the tool **phpMyAdmin**, which is a web-based graphical interface accessible through a browser.

phpMyAdmin provides a user-friendly interface that allows you to:

- create and delete databases;
- design and modify tables;
- enter and manage records;
- execute SQL queries.

Once created, the database is stored in the `mysql/data` directory located in the installation folder of the environment (for example, `EasyPHP` or `XAMPP`). This folder physically contains the files corresponding to the databases and their tables.

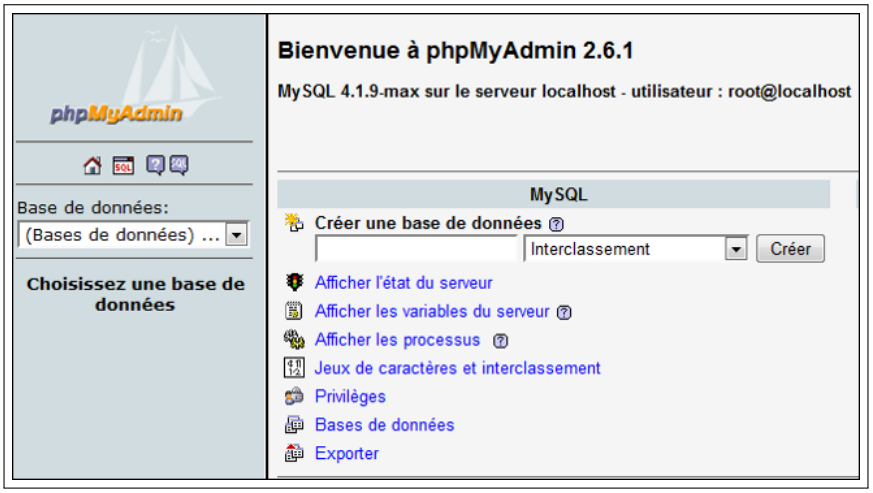

Figure 5.8: phpMyAdmin.

Thus, phpMyAdmin serves as an interface between the user and the MySQL server, translating actions performed through the graphical interface into SQL instructions executed by the DBMS.

14 Connecting to a Database in PHP

Before manipulating data, it is essential to establish a connection between PHP and the Database Management System (DBMS). PHP supports several methods to interact with databases, including:

- **MySQLi in object-oriented mode**
- **MySQLi in procedural mode** (rarely used)
- **PDO (PHP Data Objects)**

14.1 Connection with MySQLi (object-oriented)

The object-oriented method relies on using the `mysqli` class and its methods.

```
<?php
$servername = "localhost";
$username   = "root";
$password   = "";

// Create connection
$conn = new mysqli($servername, $username, $password);

// Check connection
if ($conn->connect_error) {
 die("Connection failed: " . $conn->connect_error);
}
echo "Connection successful";
?>
```

14.2 Connection with PDO

PDO is a more generic interface that allows access to multiple DBMS (MySQL, PostgreSQL, Oracle, etc.) using the same syntax. It is therefore more flexible and portable. It works only in object-oriented mode and uses exceptions for error handling.

```
<?php
$servername = "localhost";
```

```
$username   = "root";
$password   = "";

try {
 $conn = new PDO("mysql:host=$servername;dbname=testdb", $username, $password);
 // Set the error mode
 $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
 echo "Connection successful";
} catch(PDOException $e) {
 echo "Connection failed: " . $e->getMessage();
}
?>
```

This method provides more portable code and can be easily adapted to other database engines.

15 Inserting Data in PHP

After establishing a connection with the database, the next step is to insert data into a table. This is done using an SQL `INSERT INTO` query executed from PHP.

15.1 Insertion with MySQLi (object-oriented)

```
<?php
$servername = "localhost";
$username   = "root";
$password   = "";
$dbname     = "testdb";

// Connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
 die("Connection failed: " . $conn->connect_error);
}

// Insert query
```

```
$sql = "INSERT INTO users (name, email)
VALUES ('Sara', 'sara@example.com')";

if ($conn->query($sql) === TRUE) {
 echo "New record added";
} else {
 echo "Error: " . $conn->error;
}

$conn->close();
?>
```

15.2 Insertion with PDO

```
<?php
$servername = "localhost";
$username   = "root";
$password   = "";
$dbname     = "testdb";

try {
 $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
 $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

 // Insert query
 $sql = "INSERT INTO users (name, email)
 VALUES ('Karim', 'karim@example.com')";
 $conn->exec($sql);

 echo "New record added";
} catch(PDOException $e) {
 echo "Error: " . $e->getMessage();
}

$conn = null;
?>
```

15.3 Using Prepared Statements

To secure insertions and prevent SQL injections, it is recommended to use **prepared statements**.

Example with MySQLi (object-oriented)

```
<?php
$conn = new mysqli("localhost", "root", "", "testdb");

$stmt = $conn->prepare("INSERT INTO users (name, email) VALUES (?, ?)");
$stmt->bind_param("ss", $name, $email);

// Values to insert
$name = "Amine";
$email = "amine@example.com";
$stmt->execute();

$name = "Leila";
$email = "leila@example.com";
$stmt->execute();

echo "Records inserted successfully";

$stmt->close();
$conn->close();
?>
```

Example with PDO

```
<?php
try {
 $conn = new PDO("mysql:host=localhost;dbname=testdb", "root", "");
 $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

 $stmt = $conn->prepare("INSERT INTO users (name, email)
 VALUES (:name, :email)");
```

```
 // Bind parameters
 $stmt->bindParam(':name', $name);
 $stmt->bindParam(':email', $email);

 // First record
 $name = "Sofia";
 $email = "sofia@example.com";
 $stmt->execute();

 // Second record
 $name = "Nabil";
 $email = "nabil@example.com";
 $stmt->execute();

 echo "Records inserted successfully";
} catch(PDOException $e) {
 echo "Error: " . $e->getMessage();
}
?>
```

16 Updating Data

Updating data in a MySQL table is done using the `UPDATE` statement. This operation allows modifying certain records based on a given condition (using the `WHERE` clause).

Example with MySQLi (object-oriented)

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "my_database";

// Connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
```

```
if ($conn->connect_error) {
 die("Connection failed: " . $conn->connect_error);
}

// Direct update query
$sql = "UPDATE users SET email='new@mail.com' WHERE id=1";

if ($conn->query($sql) === TRUE) {
 echo "Update successful";
} else {
 echo "Error: " . $conn->error;
}

$conn->close();
?>
```

Example with PDO

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "my_database";

try {
 $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
 $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

 // Direct update query
 $sql = "UPDATE users SET email='new@mail.com' WHERE id=1";
 $stmt = $conn->prepare($sql);
 $stmt->execute();

 echo "Update successful";
} catch(PDOException $e) {
 echo "Error: " . $e->getMessage();
}
```

```
$conn = null;
?>
```

Update with Prepared Statements

When dealing with user input, it is strongly recommended to use prepared statements to prevent SQL injection attacks.

With MySQLi (object-oriented):

```
<?php
$conn = new mysqli("localhost", "root", "", "my_database");

// Prepared statement
$stmt = $conn->prepare("UPDATE users SET email=? WHERE id=?");
$stmt->bind_param("si", $email, $id);

// Set variables
$email = "secure@mail.com";
$id = 2;

$stmt->execute();

echo "Update successful";

$stmt->close();
$conn->close();
?>
```

With PDO:

```
<?php
$conn = new PDO("mysql:host=localhost;dbname=my_database", "root", "");
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// Prepared statement
$stmt = $conn->prepare("UPDATE users SET email=:email WHERE id=:id");
$stmt->bindParam(':email', $email);
```

```
$stmt->bindParam(':id', $id);

// Set variables
$email = "secure@mail.com";
$id = 2;

$stmt->execute();

echo "Update successful";
?>
```

17 Deleting Data (DELETE)

Deleting data from a MySQL table is done using the `DELETE` statement. The operation targets specific records using a `WHERE` clause. Warning: without a `WHERE` clause, all records will be deleted.

Delete with MySQLi (object-oriented)

```
<?php
$servername = "localhost";
$username   = "root";
$password   = "";
$dbname     = "myDB";

// Connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
 die("Connection failed: " . $conn->connect_error);
}

// Direct DELETE query
$sql = "DELETE FROM students WHERE id=3";

if ($conn->query($sql) === TRUE) {
```

```
 echo "Record deleted successfully";
} else {
 echo "Error deleting record: " . $conn->error;
}

$conn->close();
?>
```

Delete with PDO

```
<?php
$servername = "localhost";
$username   = "root";
$password   = "";
$dbname     = "myDBPDO";

try {
 $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
 $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

 // Direct DELETE query
 $sql = "DELETE FROM students WHERE id=3";
 $conn->exec($sql);

 echo "Record deleted successfully";
} catch (PDOException $e) {
 echo "Error deleting record: " . $e->getMessage();
}

$conn = null;
?>
```

Delete with Prepared Statements

For enhanced security (especially against SQL injections), it is recommended to use **prepared statements**, particularly when the deletion criteria come from user input.

With MySQLi (object-oriented):

```
<?php
$conn = new mysqli("localhost", "root", "", "myDB");

// Prepared statement
$stmt = $conn->prepare("DELETE FROM students WHERE id = ?");
$stmt->bind_param("i", $id);

// Value to delete
$id = 3;

$stmt->execute();

echo "Record deleted successfully";

$stmt->close();
$conn->close();
?>
```

With PDO:

```
<?php
$conn = new PDO("mysql:host=localhost;dbname=myDBPDO", "root", "");
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// Prepared statement
$stmt = $conn->prepare("DELETE FROM students WHERE id = :id");
$stmt->bindParam(':id', $id);

// Value to delete
$id = 3;

$stmt->execute();

echo "Record deleted successfully";
?>
```

18 Selecting Data (SELECT)

Selecting data from a MySQL database allows retrieving information stored in tables. In SQL, the basic query is the `SELECT` command, followed by the desired columns and the table being queried.

Syntax: `SELECT column1, column2, ... FROM table_name;` (To select all columns, you can use SELECT *.)

Example with MySQLi (object-oriented)

```
<?php
$servername = "localhost";
$username   = "root";
$password   = "";
$dbname     = "myDB";

// Connection
$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) {
 die("Connection failed: " . $conn->connect_error);
}

// Direct SELECT query
$sql    = "SELECT id, firstname, lastname FROM students";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
 // Loop through results
 while ($row = $result->fetch_assoc()) {
  echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]
 }
} else {
 echo "No results";
}

$conn->close();
?>
```

Example with PDO

```
<?php
$servername = "localhost";
$username   = "root";
$password   = "";
$dbname     = "myDBPDO";

try {
 $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
 $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

 // Direct SELECT query
 $stmt = $conn->query("SELECT id, firstname, lastname FROM students");
 while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
  echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]
 }
} catch (PDOException $e) {
 echo "Error: " . $e->getMessage();
}

$conn = null;
?>
```

In a real application, the rows of this table would be generated dynamically from the results returned by the SQL query.

ID	Last Name	First Name
1	Mohamed	Ben ali
2	Fatima	Saidi
3	Mohamed	Benyoussef
4	Yacine	Merabet

19 Session Management in PHP

A **session** in PHP is a mechanism that allows storing user-specific data temporarily on the server throughout their browsing session. Unlike cookies, session data is stored on the server.

Sessions are used to retain information across multiple web pages, such as a username after login, a shopping cart, or any data that needs to persist during navigation.

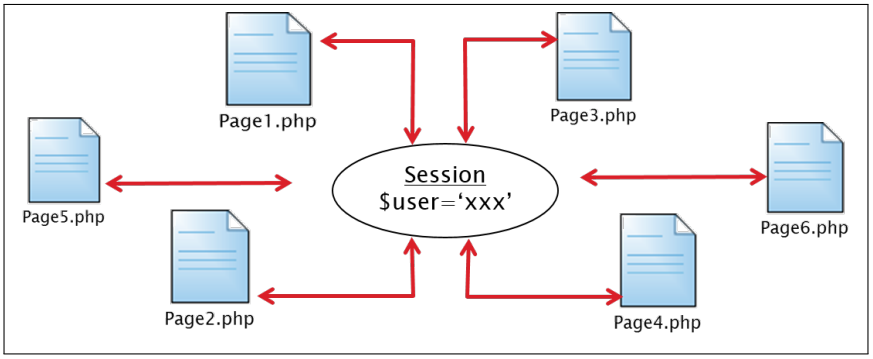

Figure 5.9: Sessions in PHP.

Starting a Session

Before manipulating session variables, you must call `session_start()`, and this must be done before any HTML output:

```
<?php
session_start();  // Start or resume a session
?>
<!DOCTYPE html>
<html>
<body>
...
```

Storing and Retrieving Session Data

You can store variables on the server using the superglobal array `$_SESSION`. These data will be accessible on all pages of the application as long as the session is active.

```
<?php
session_start();

// Store data
$_SESSION['username'] = 'nabila';
$_SESSION['email']    = 'nabila@example.com';
```

```
// Retrieve data
echo $_SESSION['username'];
echo '<br>';
echo $_SESSION['email'];
?>
```

A practical example: a page view counter:

```
<?php
session_start();
if (isset($_SESSION['views'])) {
 $_SESSION['views'] = $_SESSION['views'] + 1;
} else {
 $_SESSION['views'] = 1;
}
echo "Number of pages viewed: " . $_SESSION['views'];
?>
```

Deleting Data or Destroying the Session

To remove a single session variable:

```
<?php
session_start();
unset($_SESSION['views']);
?>
```

To completely destroy the session (all data and the associated ID):

```
<?php
session_start();
session_destroy();
?>
```

20 Conclusion

The **PHP** language is an essential tool for developing server-side web applications. It offers a wide range of features, including form handling, database interaction, session

management, and more. In this chapter, we limited our presentation to the fundamental elements necessary for understanding the basics of the language and for implementing initial practical developments.

21 Practical Work: Developing a PHP Website

This practical session is conducted over several classes. The objective is to show students how to develop a **complete PHP website**, applying the most important aspects of the language in practice.

The website will consist of several main pages:

- **index.php**: home page.
- **ajouter.php**: page to add a new student.
- **afficher.php**: page to display the list of students.

All these pages form a small application for **managing students in the Computer Science Department**.

21.1 Creating the Database with phpMyAdmin

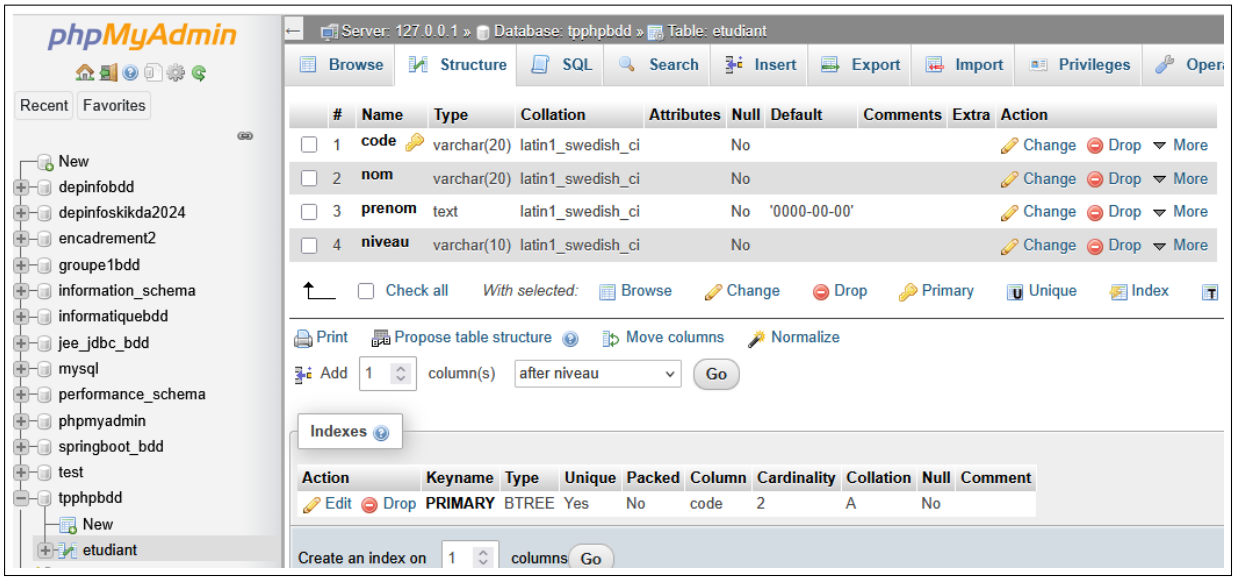

Figure 5.10: Creating the database with phpMyAdmin

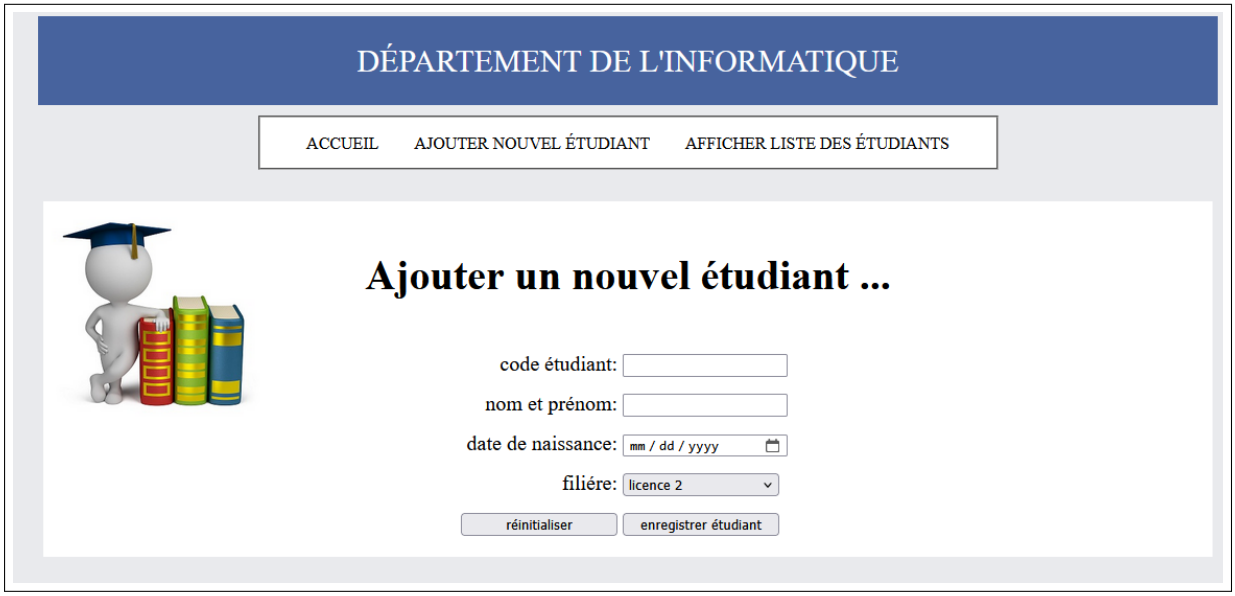

Figure 5.11: Inserting a new student into the database

21.2 Inserting a New Student into the Database

21.3 Displaying the List of Students

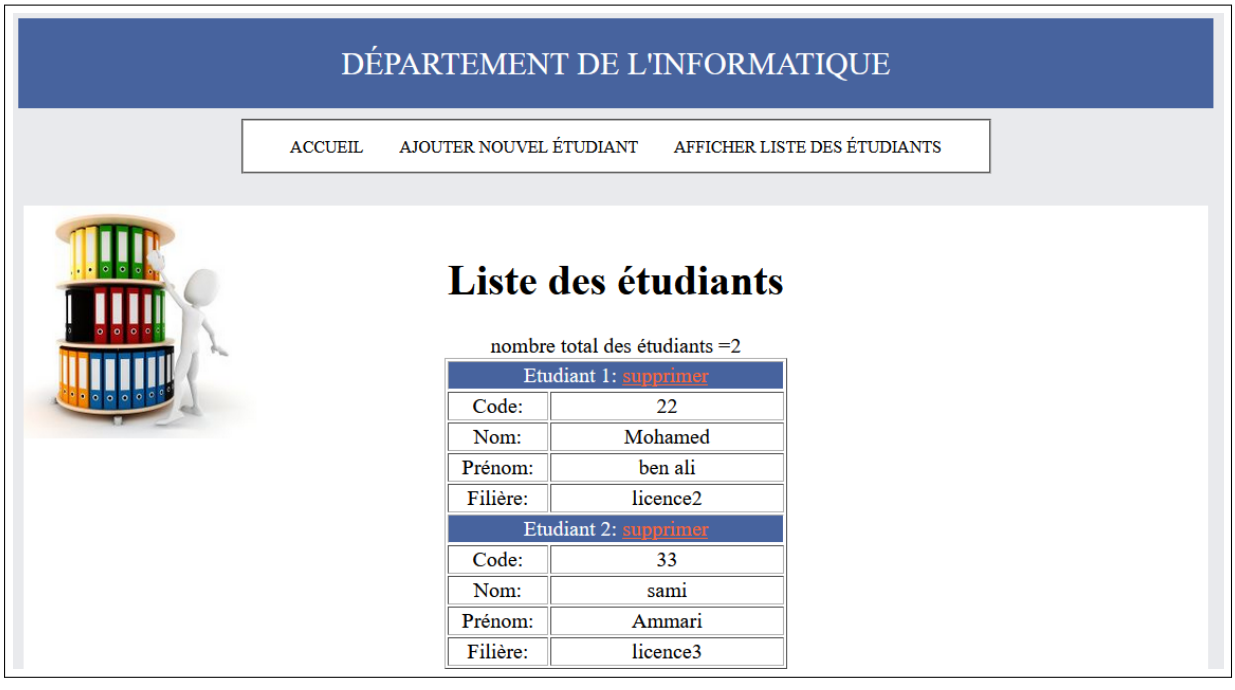

Figure 5.12: Displaying the list of students

CHAPTER

6

XML — EXTENSIBLE MARKUP LANGUAGE

1 Introduction

XML (Extensible Markup Language) is a text-based markup language designed for data transport and storage. Unlike HTML, XML does not define predefined elements: it allows the creation of custom tags adapted to the data being represented. The main goal of XML is *portability* and *readability*, both for humans and machines.

Example of unstructured content:
*The movie **Gladiator**, produced in the United States and directed by Ridley Scott, was released in the year 2000.*

This is a simple, unstructured text from which no automated application can extract information meaningfully.

Possible representation of the same content in XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<film>
<titre>Gladiator</titre>
```

```
<pays>United States</pays>
<realisateur>Ridley Scott</realisateur>
<annee>2000</annee>
</film>
```

This time, opening and closing tags separate the different parts of the content (title, release year, producing country, director, etc.). It then becomes (relatively) easy to analyze and exploit this content, provided, of course, that the meaning of these tags is known [8].

2 Basic Structure and Syntax

Let us start with a simple example of an XML document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cours titre="XML">
<intervenant nom="alexandre brillant">
</intervenant>
<plan>
Introduction to XML and document composition
</plan>
</cours>
```

We can already analyze that an XML document is a readable text document. Without necessarily understanding the entire syntax, we can deduce that it describes a course whose instructor is the author of this book. The course outline is also apparent from the document [1].

2.1 XML Prologue

An XML document usually starts with a declaration (prologue) indicating the version and encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
```

2.2 Elements (tags)

An XML element is delimited by an opening tag and a closing tag:

```
<personne>
<nom>Mohamed</nom>
<prenom>Ben ali</prenom>
</personne>
```

Elements can be nested, and the nesting must be well-formed: each opened tag must be properly closed and nesting must not overlap.

2.3 Attributes

Elements can have attributes:

```
<Realisateur
nom="Scott"
prenom="Ridley"
annee_naissance="1937"/>
```

Note

Prefer to store "heavy" data as element text rather than overusing attributes, to preserve semantics.

2.4 Text and Comments

Comments: `<!-- This is a comment -->`

Text: this is the content of the tags.

3 Important Rules

- **Well-formedness**: the document must follow XML syntax rules.
- **Case-sensitive**: XML distinguishes between uppercase and lowercase (e.g., `<Nom>` ≠ `<nom>`).
- **Encoding**: specify the encoding (UTF-8 recommended) if necessary.

4 Namespaces

Namespaces help avoid name collisions when multiple vocabularies are combined:

```
<racine xmlns:ex="http://example.org/schema">
<ex:element>...</ex:element>
</racine>
```

Prefixes are declared using the `xmlns[:prefix]` attribute.

5 DTD — Document Type Definition

A *DTD* (Document Type Definition) defines the structure of an XML document: which elements and attributes are allowed, in what order, and under which rules. An XML document is **valid** when it conforms to the DTD.

5.1 Types of DTD

- **Internal**: included directly in the XML file.

  ```
  <!DOCTYPE note [
  <!ELEMENT note (to,from,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
  ]>
  ```

- **External**: defined in a separate file and referenced.

  ```
  <!DOCTYPE note SYSTEM "note.dtd">
  ```

5.2 Main Declarations

- `<!ELEMENT>`: defines the structure of an element.
- `<!ATTLIST>`: defines the attributes of an element.
- `<!ENTITY>`: defines shortcuts for text or symbols.

5.3 Purpose

- Standardize the structure of XML documents.
- Allow automatic validation by a parser.
- Facilitate data exchange between applications.

5.4 Limitations

- Not very expressive (no complex types).
- Syntax not based on XML.
- Often replaced by XML Schemas (XSD).

6 XML Schema (XSD)

An **XML Schema**, also called **XSD**, is a language used to describe the **structure** and **constraints** of an XML document.

6.1 Advantages over DTDs

- XML-based syntax, making it more readable and consistent.
- Allows defining **data types** (string, integer, date, etc.).
- Provides better control over **structure** and **content** (occurrences, restrictions, allowed values).

6.2 Purpose

- Ensures that XML documents follow a specific format.
- Promotes interoperability between systems.

6.3 Example

```
<xs:element name="note">
<xs:complexType>
```

```
<xs:sequence>
<xs:element name="to" type="xs:string"/>
<xs:element name="from" type="xs:string"/>
<xs:element name="heading" type="xs:string"/>
<xs:element name="body" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

7 XSLT (Extensible Stylesheet Language Transformations)

XSLT is an XML-based language that allows transforming an XML document into another format, such as **HTML**, **text**, or another **XML** document.

7.1 Principle

- XSLT uses **stylesheets** (XSL) to define **transformation rules**.
- These rules are expressed using **XPath expressions** to select parts of the XML document.
- The XSLT processor applies the stylesheet to the XML document and generates the output document.

7.2 Example

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>Student List</h2>
<ul>
<xsl:for-each select="class/student">
<li><xsl:value-of select="name"/></li>
</xsl:for-each>
```

```
</ul>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

7.3 Purpose

- Allows presenting XML data in a readable format (HTML).
- Separates data logic (XML) from presentation (XSLT).
- Facilitates multi-platform publishing (web, text, other XML).

8 XPath (XML Path Language)

XPath is a language used to navigate an XML document and select elements or attributes using **paths**.

8.1 Basic Principles

- XPath uses a path-like syntax (similar to file paths).
- Paths allow selecting **nodes** (elements, attributes, text).
- It is widely used in **XSLT** to apply transformation rules.

8.2 Examples of XPath Queries

- `/library/book` : selects all `<book>` elements that are children of `<library>`.
- `//title` : selects all `<title>` elements in the document, anywhere they appear.
- `/library/book[1]` : selects the first `<book>`.
- `//book[@category='novel']` : selects all books with the attribute `category="novel"`.
- `//author/name/text()` : retrieves the text contained in `<name>` for each author.

8.3 Purpose

- Select specific parts of an XML document.
- Filter data based on conditions (e.g., attributes, values).
- Serve as a basis for other XML technologies such as XSLT and XQuery.

9 Conclusion

XML remains a robust standard for transporting and describing structured data, particularly when formal schemas, transformations (XSLT), or interoperability with existing systems are required. This chapter presented the basic concepts needed for a student to understand the syntax, validation, and methods to query and transform XML documents.

10 Exercises

10.1 Creating a Book in XML

We want to write a book using the `XML` format. The document must follow the following structure:

- The book is composed of **sections** (at least two).
- Each section contains **chapters** (at least two).
- Each chapter contains **paragraphs** (at least two).
- The book includes a list of **authors** (with first and last names).
- All elements have a **title**, except for paragraphs which contain text directly.

Tasks:

1. Propose an `XML` structure for this book containing:
    - 2 authors;
    - 2 sections;
    - 2 chapters per section;
    - 2 paragraphs per chapter.

2. Verify, using an editor, that the document is well-formed.
3. Follow these constraints:
    - No attributes are allowed;
    - The file must be saved as `book1.xml`.

10.2 Exercise 2: Define a DTD

Based on the `book1.xml` document created in the previous exercise:

1. Write an **internal DTD** that describes the structure of the book (authors, sections, chapters, paragraphs, and titles).
2. Associate this DTD with the file `book1.xml`.
3. Verify that the document is **valid**.

CHAPTER

7

WEB SERVICES

1 Introduction

The emergence of **Web Services** has marked a decisive step in the evolution of distributed architectures. Based on **open standards** such as XML, SOAP, WSDL, and UDDI, Web Services enable **interoperability** between heterogeneous applications, regardless of programming languages, platforms, or operating systems. This chapter presents the **fundamental principles** of Web Services, their characteristics, architecture, as well as the tools and platforms that enable their implementation.

2 Service-Oriented Architecture (SOA)

Service-Oriented Architecture (SOA) is a software architecture style that organizes an information system into a set of **independent and reusable services**. A *service* represents a well-defined business function, accessible through a standardized interface and usable by different applications, regardless of language or platform [3].

2.1 Objectives of SOA

- **Interoperability**: enable communication between heterogeneous applications (Java, .NET, etc.).
- **Reusability**: promote the multiple use of services in different contexts.
- **Flexibility and scalability**: facilitate the composition of new business processes from existing services.
- **Standardization**: rely on open protocols and languages (XML, SOAP, WSDL, UDDI).

Thus, SOA provides an approach to building modular, flexible systems oriented towards business needs.

3 Definition of a Web Service

A **Web Service** is an application accessible over a network that exposes *well-defined functionalities* through a standardized interface. It enables communication and data exchange between heterogeneous applications, regardless of the platform, programming language, or operating system used.

Web Services rely on **open standards** such as:

- **XML** for data representation and exchange,
- **SOAP** (Simple Object Access Protocol) for the communication protocol,
- **WSDL** (Web Services Description Language) for service description,
- **UDDI** (Universal Description, Discovery and Integration) for service publication and discovery.

Thus, a Web Service constitutes a reusable software component, enabling interoperability and integration of distributed systems.

4 Web Services Architecture

The **Web Services architecture** is based on a simple model, centered on the interaction between three main roles and the use of open standards:

- **Service Provider**: provides the service, defines its interface, and publishes its description (WSDL).
- **Service Registry**: acts as a directory where providers publish their services and clients can search for them (often via UDDI).
- **Service Consumer**: searches for a service in the registry, retrieves its description, and invokes it via the appropriate communication protocols.

The consumer discovers the service through the registry, reads its description (WSDL), and then communicates with the provider using SOAP over HTTP or other protocols. Thus, the Web Services architecture relies on the triad: **publish – discover – invoke**.

5 WSDL (Web Services Description Language)

WSDL is an XML-based language that allows for the **formal description** of a Web Service. It provides all the necessary information for a client to *discover*, *understand*, and *consume* a service without needing to know its internal implementation.

A WSDL document mainly defines:

- **The operations** offered by the service (available methods).
- **The messages** exchanged: input data, output data, and possible errors.
- **The data types** used, described using XML Schema (XSD).
- **The communication protocols** and access points (bindings), such as SOAP/HTTP.
- **The service address** (endpoint), indicating where the service is accessible on the network.

Thus, WSDL serves as a **contract** between the service provider and the client. Thanks to this standardized description, a Web Service can be used by applications developed in different environments (Java, .NET, etc.).

6 UDDI (Universal Description, Discovery and Integration)

UDDI is a specification that defines a **Web Services registry**, allowing their *publication*, *discovery*, and *integration*. It acts as a **global directory** where providers can list their services and consumers can search for the services they need.

6.1 Main Objectives

- **Publication**: allow providers to register their services (name, description, access point, associated WSDL).
- **Discovery**: give consumers the ability to search for a service based on various criteria (category, provider, service type).
- **Interoperability**: provide a standardized and universal mechanism to share services across heterogeneous environments.

6.2 Structure of a UDDI Registry

A UDDI registry typically contains:

- **Business information** (data about the service provider company),
- **Service descriptions** (offered functionalities),
- **Technical references** pointing to WSDL documents and communication protocols used.

Thus, UDDI complements SOAP and WSDL by providing the essential building block of the Web Services architecture: **dynamic service discovery**.

7 Web Services Development Platforms

Web Services can be developed and deployed on different **technology platforms**, with the two main ones being **J2EE** (Java 2 Enterprise Edition) and **.NET**. These environments offer tools and libraries that facilitate the creation, description, and consumption of Web Services.

7.1 J2EE Platform

The **Java 2 Enterprise Edition** platform provides a set of technologies and APIs to build distributed, service-oriented applications:

- Use of **servlets** and **JSP** for the web layer.
- Support for **EJB** (Enterprise JavaBeans) for business logic.

- Specific libraries for **SOAP**, **WSDL**, and **UDDI**.
- Application servers (Tomcat, JBoss, WebSphere, etc.) ensuring deployment.

7.2 .NET Platform

Microsoft's **.NET** platform also provides a rich infrastructure for developing Web Services:

- Native support for Web Services via **ASP.NET**.
- Automatic **WSDL** description of exposed services.
- Integration with .NET languages (C#, VB.NET, etc.).
- Development tools like **Visual Studio .NET** facilitating service generation and consumption.

7.3 Comparison and Interoperability

- Both J2EE and .NET rely on open standards (**XML, SOAP, WSDL, UDDI**) ensuring interoperability.
- Differences mainly lie in development environments and programming models.
- Thanks to these standards, a service developed in J2EE can be consumed by a .NET application, and vice versa.

Thus, J2EE and .NET represent the two major Web Services development platforms, each offering a set of tools and components suited to enterprise needs. However, the ecosystem has significantly evolved over the years, and many other frameworks and environments are now widely used:

- **Open-source solutions**: Apache Axis, Apache CXF, gSOAP.
- **Modern frameworks**: Spring Web Services (Java), Node.js (JavaScript), Flask or Django (Python).
- **Cloud environments**: AWS (API Gateway), Microsoft Azure (API Management), Google Cloud (Endpoints).

These developments show that although J2EE and .NET were historically dominant, Web Services can now be developed on a wide variety of platforms, depending on technological and business requirements.

8 Web Services Development (Provider Side)

The **service provider** is responsible for implementing and publishing the Web Service. The main steps are:

1. **Implementing business logic**: developing the code corresponding to the functionalities offered by the service.
2. **Encapsulation as a Web Service**: exposing this logic through standardized interfaces (for example using SOAP/HTTP).
3. **Service description**: generating a **WSDL** document specifying operations, messages, and the access endpoint.
4. **Publication**: making the service available in a **UDDI registry** or directly sharing the URL with the consumer.

Thus, on the provider side, development consists of **transforming an internal functionality** into a **remotely accessible service**, documented and publicly available.

9 Web Services Development (Consumer Side)

The **service consumer** is the client application that uses the Web Service published by a provider. The main steps are:

1. **Service discovery**: querying a UDDI registry or directly using the URL provided by the service provider.
2. **Retrieving the description**: reading the **WSDL** document to understand the available operations, expected parameters, and returned messages.
3. **Generating client code**: creating stubs or proxies from the WSDL, often automated by development environments.
4. **Invoking the service**: sending a SOAP message (or REST in modern evolutions) to the specified address and processing the response.

Thus, on the consumer side, development consists of **integrating an external service** into the application, leveraging standards to ensure interoperability and transparent execution.

10 RESTful Web Services

RESTful Web Services represent an alternative to traditional Web Services based on SOAP and WSDL. They rely on the REST (Representational State Transfer) architectural style and use the HTTP protocol directly through its main methods (GET, POST, PUT, DELETE). Unlike SOAP, which relies on XML messages, REST uses various data formats, including JSON, which is lighter and widely adopted in modern applications.

The main characteristics of RESTful Web Services are:

- **Simplicity**: they use the Web mechanisms directly.
- **Flexibility**: data can be exchanged in XML, JSON, or any other format.
- **Performance**: messages are often lighter than SOAP.
- **Wide adoption**: they are widely used for public APIs and microservices architectures.

Thus, RESTful does not completely replace SOAP, but establishes itself as a simpler solution better suited to the needs of modern Web and mobile applications.

11 Conclusion

In this chapter, we presented **Web Services** as an integration solution based on **open standards** ensuring interoperability between heterogeneous systems. We discussed their **principles**, **characteristics**, and the essential roles of **SOAP**, **WSDL**, and **UDDI** in Web Services architecture. Therefore, Web Services appear as a **fundamental technological building block** in implementing **Service-Oriented Architecture (SOA)**, providing reliable mechanisms for the publication, discovery, and invocation of distributed services.

GENERAL CONCLUSION

This course material aims to present the essential foundations of web application development across several chapters: a general introduction to the WWW, HTML, CSS, JavaScript, PHP, XML, and Web Services. It is not a comprehensive course, but a summary document providing students with a global and coherent overview of the various technologies involved in creating modern web applications.

Each chapter could, on its own, be the subject of an in-depth and detailed study. Here, we have chosen to highlight the fundamental concepts and basic examples to introduce students and give them the initial guidance needed to progress in the field of web development.

Thus, this material serves as a starting point: it allows students to acquire a first understanding of the languages and tools, preparing them to later explore more advanced aspects such as server-side programming, security, frameworks, and web application architecture.

In summary, this document serves as an entry point into web development, offering an overview and initial practical experience upon which students can build more specialized skills.

BIBLIOGRAPHY

[1] Alexandre Brillant. *XML : Cours et exercices.* Dunod, 2e édition edition, 2005.

[2] Raphaël Goetter and Hugo Giraudel. *CSS3 : Pratique du design web.* Éditions Eyrolles, Paris, 4 edition, 2019.

[3] Libero Maesano, Christian Bernard, and Xavier Le Galles. *Services Web avec J2EE et .NET – Conception et implémentations.* Éditions Eyrolles, Paris, 2003. Collection Blanche.

[4] Mozilla Developer Network. Introduction au html. `https://developer.mozilla.org/fr/docs/Web/HTML`, 2025. Consulté le 14 février 2025.

[5] Mozilla Developer Network. tutoriel javascript. `https://developer.mozilla.org/fr/docs/Web/JavaScript`, 2025. Consulté le 20 Aout 2025.

[6] Mathieu Nebra and Denis Laurent. *Réussir son site web avec XHTML et CSS.* Eyrolles, Paris, 3 edition, 2010.

[7] Emmanuel Puybaret. *Les Cahiers du Programmeur Java 1.4 et 5.0, subtitle = Les Cahiers du Programmeur, edition = 2, publisher = Eyrolles, address = Paris, year = 2004, isbn = 2-212-11478-8.*

[8] Philippe Rigaux. *Pratique de MySQL et PHP : Conception et réalisation de sites web dynamiques.* Dunod, Paris, 4 edition, 2009.

[9] Thierry Templier and Arnaud Gougeon. *JavaScript pour le Web 2.0 : Programmation objet, DOM, Ajax, Prototype, Dojo, Script.aculo.us, Rialto...*. Éditions Eyrolles, 2007.

[10] W3Schools. Html tutorial. `https://www.w3schools.com/html/`, 2025.