

Développement d'Applications Web

Leila BAGDADI

17 juin 2026

Syllabus

Unité d'enseignement méthodologique :	UEM
Matière :	Développement d'applications web
Domaine/Filière :	Informatique
Niveau/Semestre :	Licence 2, S4
Crédits/Coefficient :	4 / 2
Volume Horaire Hebdomadaire Total :	3h00
Enseignant responsable :	Dr. Leila BAGDADI
Email :	leila.bagdadi@univ-usto.dz

Contenu de la matière

Chapitre 1 : Introduction au World Wide Web

1. Définition et historique
2. Architecture Client/Serveur
3. Protocole HTTP.

Chapitre 2 : Langages de programmation pour le Web

1. Généralités : page statique, page dynamique et applications Web
2. Langages de balise : définition et historique
3. HTML
 - (a) Qu'est ce que le HTML ?
 - (b) Contexte d'exécution HTML
 - (c) HTML de base
4. Feuilles de style (CSS 3)
5. JavaScript
6. XML
7. DTD (Document Type Definition)
8. XML Schema
9. XSLT

Chapitre 3 : Langage de programmation côté serveur (PHP)

1. Introduction
2. Syntaxe de base
3. Types, variables et opérateurs
4. Structures de contrôles
5. Classes et objets
6. Caractéristiques

Chapitre 4 : Notions de base des services Web

1. Introduction
2. Architecture orientée services (SOA)
3. Caractéristiques des services Web
4. Standards pour les services Web
5. Plateformes de développement des services Web

Mode d'évaluation : Examen (60%) , contrôle continu (40%)

Table des matières

1	Avant-propos	1
2	Introduction au World Wide Web	2
2.1	Introduction	2
2.2	Définition	3
2.3	Évolution générationnelle du Web	3
2.4	Architecture Client/Serveur	4
2.5	Les trois piliers du Web	4
2.6	Du Web statique au Web dynamique	5
3	HyperText Markup Language	6
3.1	Généralités	6
3.1.1	Caractéristiques principales	7
3.2	HTML : définition et historique	8
3.2.1	Qu'est ce que le HTML ?	8
3.2.2	HTML de base	8
3.2.2.1	Ossature d'un document HTML	8
3.2.2.2	Les balises HTML	9
3.2.2.3	Les formulaires	22
3.3	Exercices	29
4	Cascading Style Sheet	31
4.1	Introduction	31
4.2	Principe de fonctionnement du CSS	32
4.3	Les différentes méthodes d'intégration du CSS	32

TABLE DES MATIÈRES

4.3.1	CSS en ligne (inline)	32
4.3.2	CSS interne (internal)	33
4.3.3	CSS externe (external)	33
4.4	Les sélecteurs CSS	34
4.4.1	Sélecteurs de base	34
4.4.2	Sélecteurs combinés	36
4.4.3	Sélecteurs d'attributs	36
4.4.4	Pseudo-classes	37
4.5	Formatage du texte	39
4.6	Listes	40
4.7	Tables	40
4.8	Modèle de boîtes	41
4.9	Positionnement des éléments	44
4.9.1	Display	44
4.9.2	Flex and grid	44
4.10	Responsive design and media queries	46
4.11	Exercices	47
5	Javascript	50
5.1	Introduction	50
5.2	Les différentes méthodes d'intégration d'un script	50
5.2.1	Insertion interne	50
5.2.2	Insertion externe	51
5.2.3	Insertion en ligne (inline)	51
5.3	Syntaxe de base	52
5.3.1	Typage faible et dynamique	52
5.3.2	Déclaration et portée des variables	52
5.3.3	Les fonctions	53
5.4	Boîtes de dialogue	53
5.5	Les événements	54
5.6	Manipulation par le DOM	55
5.6.1	Structure arborescente du DOM	56
5.6.2	Accès aux éléments HTML	56
5.6.3	Modification du contenu	57
5.6.4	Ajout et suppression d'éléments	58
5.6.5	Modification des attributs et du style	59
5.6.6	L'écoute d'événements avec addEventListener()	61
5.7	Contrôle des formulaires	61

TABLE DES MATIÈRES

6	PHP	65
6.1	Introduction	65
6.2	Environnement de développement	66
6.2.1	Installation et premiers pas	67
6.3	Syntaxe de base	68
6.3.1	Structures de contrôles	69
6.3.2	Tableaux	71
6.3.3	Fonctions PHP	72
6.4	Traitement des formulaires par PHP	74
6.5	Interaction avec MySQL	75
6.5.1	Connexion à une base de données	76
6.5.2	Manipulation d'une base de données	77
7	Web services : notions de base	80
7.1	Introduction	80
7.2	Définition	81
7.3	Caractéristiques des services Web	81
7.4	Fonctionnement des services Web	81
7.5	XML, DTD et XSD	82
7.5.1	XML	82
7.5.2	DTD	83
7.5.3	XSD	84
7.6	SOAP, WSDL et UDDI	85
7.6.1	SOAP	85
7.6.2	WSDL	86
7.6.3	UDDI	87
7.7	Plateformes de développement des services Web	87
7.8	Implémentation d'un service Web SOAP	87
8	Solutions	90
8.1	Exercice 3.3.1	90
8.2	Exercice 3.3.2	91
8.3	Exercice 3.3.3	92
8.4	Exercice 4.11.1	94
8.5	Exercice 4.11.2	94
	Bibliographie	96

Avant-propos

Le présent polycopié est destiné aux étudiants de deuxième et troisième année Licence en Informatique. Il constitue un support de cours pour l'unité d'enseignement « Développement d'applications web ».

L'objectif de ce document est de fournir aux apprenants les connaissances théoriques et pratiques nécessaires à la conception et à la réalisation d'applications web complètes. Il couvre l'ensemble de la chaîne de développement, depuis les technologies **front-end** (HTML, CSS, JavaScript) jusqu'aux technologies **back-end** (PHP, MySQL), en passant par les protocoles et architectures fondamentaux du World Wide Web.

La progression pédagogique adoptée suit une logique ascendante : après une introduction aux concepts du Web et à son architecture, les étudiants découvriront successivement les langages de structuration (HTML), de présentation (CSS) et d'interactivité côté client (JavaScript). Enfin, la partie serveur (PHP) et l'interaction avec les bases de données (MySQL) permettront aux apprenants de développer des applications web dynamiques complètes.

Chaque chapitre est structuré de manière homogène : objectifs pédagogiques, exposé théorique illustré d'exemples concrets, notes de synthèse et exercices d'application. Les codes source présentés sont volontairement simples et commentés afin de faciliter leur assimilation.

Nous remercions par avance les étudiants pour leur lecture attentive et les encourageons à une mise en pratique régulière, condition indispensable à la maîtrise des technologies web.

Introduction au World Wide Web



"The Web does not just connect machines, it connects people." -
Tim Berners-Lee

Objectifs du chapitre

À l'issue de ce chapitre, l'étudiant sera capable de décrire l'origine et l'évolution du World Wide Web, d'identifier les protocoles et concepts fondamentaux qui régissent son fonctionnement, ainsi que d'expliquer les mécanismes sous-jacents au chargement d'une page web.

2.1 Introduction

Conçu à l'origine sous le nom de « World Wide Web » pour permettre aux scientifiques des universités et instituts du monde entier d'échanger des informations en temps réel, le Web est une application d'Internet offrant la possibilité de partager des documents interconnectés appelés « pages Web ». Chaque page Web peut intégrer du texte, des images, des hyperliens ainsi que d'autres contenus multimédias.

Depuis les années 1990, le Web a connu une évolution remarquable : de sites essentiellement statiques et peu interactifs, il est devenu un espace dynamique permettant des échanges

riches entre l'utilisateur et le serveur.

2.2 Définition

Le World Wide Web (WWW ou Web) est un système hypertexte public fonctionnant sur Internet. Il permet de consulter, via des navigateurs, des pages contenant du texte, des images, des hyperliens et divers contenus multimédias. Contrairement à une idée répandue, le Web n'est pas Internet : ce dernier en est l'infrastructure réseau mondiale, tandis que le Web n'en est qu'un service parmi d'autres, au même titre que le courrier électronique ou le partage de fichiers.

Le Web a été inventé en **1989** par **Sir Tim Berners-Lee**, chercheur britannique au CERN (Organisation européenne pour la recherche nucléaire). L'objectif initial était de permettre aux scientifiques dispersés à travers le monde d'échanger instantanément des informations et des documents de recherche.

En 1990, les premiers navigateurs web firent leur apparition, dont Mosaic et Netscape}. Ces outils introduisirent la capacité d'afficher des images et des graphiques, rendant le contenu web plus attrayant et la navigation plus intuitive.

2.3 Évolution générationnelle du Web

L'évolution du Web se décompose en plusieurs grandes étapes, chacune associée à des usages et des technologies spécifiques :

Web 1.0 (années 1990 – début 2000) - Le Web statique : À ses débuts, le Web fonctionnait selon un modèle unidirectionnel : les sites, constitués de pages HTML statiques conçues par des développeurs professionnels, diffusaient passivement l'information. Les utilisateurs, simples consommateurs, ne pouvaient ni interagir ni contribuer. L'expérience restait limitée, sans retour ni personnalisation possible.

Web 2.0 (à partir du début des années 2000)- Le Web participatif : Le Web 2.0 a opéré une rupture majeure en transformant l'utilisateur de spectateur passif en acteur. L'interactivité et la collaboration sont devenues centrales, permettant à chacun de publier, commenter et partager du contenu (texte, image, vidéo).

Cette période a vu l'émergence des **blogs**, **forums**, réseaux sociaux (**MySpace**, **Facebook**) et plateformes vidéo tel que **YouTube**, ce qui a considérablement enrichi l'écosystème web.

Web 3.0 (Web sémantique) - Web intelligent : Le Web 3.0, ou Web sémantique, vise à rendre les données compréhensibles par les machines. Il ambitionne de créer un réseau où les informations interconnectées permettent une navigation plus intelligente et des résultats de recherche plus pertinents, améliorant ainsi la personnalisation de l'expérience utilisateur.

2.4 Architecture Client/Serveur

L'architecture **client-serveur** du Web se caractérise par une application répartie entre deux entités dont le dialogue s'effectue selon un protocole applicatif spécifique, **HTTP**, qui régit les échanges de requêtes et de réponses. Ces deux entités sont :

- **Le client** : navigateur web (**Chrome**, , **Safari**, etc.) qui initie l'échange en adressant une requête **HTTP** au **serveur**. Il sollicite ainsi ses services, gère l'interface utilisateur, puis interprète et affiche les ressources retournées par le serveur.
- **Le serveur** : application (**Apache**, **IIS**, etc.) qui répond à la requête initiée par le client en traitant la demande reçue. Il héberge les pages **Web**, fournit les services sollicités, puis renvoie les ressources appropriées au client via une réponse **HTTP**.

2.5 Les trois piliers du Web

Le fonctionnement du World Wide Web repose sur trois technologies fondamentales, souvent désignées comme les trois piliers étroitement liés du **Web** : l'**URL** désigne "**quoi**", **HTTP** définit "**comment**" et **HTML** décrit **quoi afficher**.

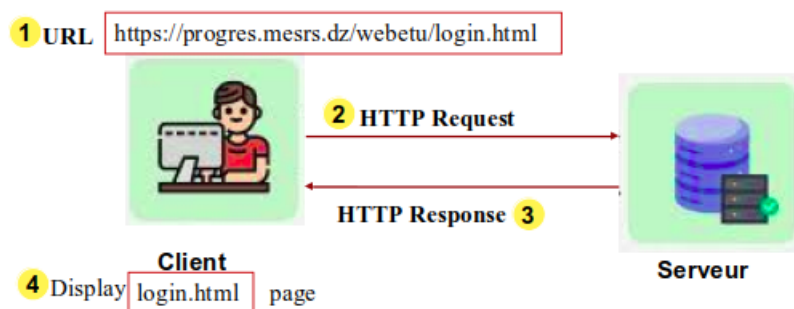
1. **URL** (Uniform Resource Locator) : est une adresse unique permettant de localiser une **ressource** sur le **Web**. Comme une adresse postale pour le facteur, elle indique exactement où trouver une page, une image ou un fichier sur Internet. Comme l'illustre l'exemple de la Figure 1.5.1, l'URL se compose de plusieurs parties :

FIGURE 2.5.1 – URL



2. **HTTP** (HyperText Transfer Protocol) : est le protocole de communication du Web. Il régit les échanges de requêtes et de réponses entre client et serveur, comme schématisé dans la Figure 1.5.2. Les clients HTTP les plus répandus sont les navigateurs web (**Chrome**, **Firefox**, **Safari**, etc.), qui permettent d'accéder aux ressources stockées sur un serveur. Ce dernier écoute généralement sur le **port 80** pour les connexions non sécurisées, et sur le **port 443** pour **HTTPS**. Le protocole propose plusieurs méthodes indiquant l'action souhaitée : **GET** (récupération d'une ressource), **POST** (envoi de données au serveur), ainsi que **PUT** (mise à jour) et **DELETE** (suppression).

FIGURE 2.5.2 – HTTP



3. **HTML** (HyperText Markup Language) : est le langage de balisage standard du Web. Il permet de structurer et de décrire le contenu d'une page web. Le navigateur analyse et interprète ce langage pour restituer la page à l'utilisateur, et le cas échéant, solliciter les ressources incorporées (images, feuilles de style, scripts, etc.).

2.6 Du Web statique au Web dynamique

Tous les sites web ne fonctionnent pas selon le même modèle. On distingue classiquement deux architectures : le Web statique, où les pages sont préfabriquées et identiques pour tous, et le Web dynamique, où les pages sont générées à la volée en fonction de la requête de l'utilisateur. Les schémas suivants en présentent le fonctionnement.

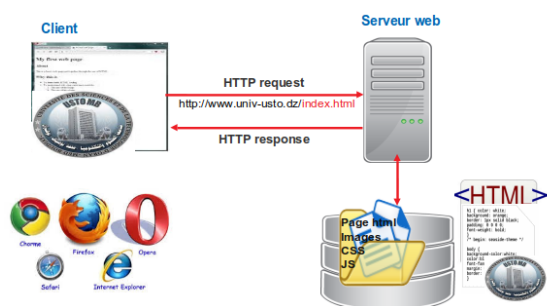


FIGURE 2.6.1 – Web statique

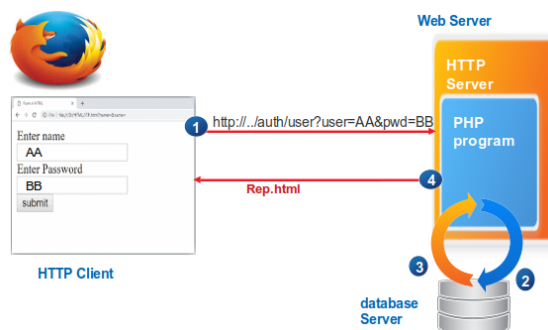


FIGURE 2.6.2 – Web dynamique

Les deux schémas ci-dessus mettent en évidence une différence fondamentale : dans le Web statique, le client reçoit directement une page HTML préexistante ; dans le Web dynamique, une couche supplémentaire (PHP) génère cette page à la demande. Avant d'aborder la programmation côté serveur avec PHP, il est essentiel de maîtriser le langage qui structure toute page web : le HTML.

HyperText Markup Language

Objectifs du chapitre

À l'issue de ce chapitre, vous serez capables de :

1. Rédiger un code HTML conforme aux standards, en utilisant des éléments textuels hiérarchisés.
2. Structurer le contenu à l'aide de listes (numérotées, à puces et descriptives).
3. Construire des tableaux de données accessibles avec une sémantique appropriée.
4. Insérer des images et des vidéos en veillant à inclure un texte alternatif et des légendes adéquates.
5. Développer des formulaires web interactifs avec des types de champs adaptés et une validation appropriée.
6. Utiliser les balises sémantiques HTML5 (`<header>`, `<main>`) pour organiser clairement le contenu.

3.1 Généralités

Le *HyperText Markup Language* (**HTML**) est le langage de balisage standard pour créer des pages web. Il permet de structurer le contenu d'un document en utilisant un système de balises (tags) qui délimitent et organisent les différents éléments de la page.

3.1.1 Caractéristiques principales

Le HTML repose sur des concepts structurants que tout développeur web doit connaître :

1. **Syntaxe basée sur les balises** : une balise ouvrante (`<balise>`) et une balise fermante (`</balise>`) pour délimiter le contenu d'un élément . Par exemple, voici le code HTML qui affiche la chaîne de caractères « Hello World ! » en italique.

Exemple 3.1

```
<i>Hello World!</i>
```

Dans cet exemple, la balise `<i>` définit le type de mise en forme « italique »

2. **Accessibilité et simplicité** : la création d'une page HTML repose sur deux éléments clés : un éditeur de texte pour écrire le code source, et un navigateur web pour l'interpréter et l'afficher. Les éditeurs vont du simple Bloc-notes aux outils avancés comme Visual Studio Code, tandis que les navigateurs modernes (Chrome, Edge, Firefox) assurent le rendu des pages HTML.
3. **Insensibilité à la casse** : bien que le langage HTML ne fasse pas de distinction entre majuscules et minuscules dans les balises (par exemple, `<body>` et `<BODY>` sont équivalents), il est vivement conseillé d'adopter l'usage systématique des minuscules. Cette convention favorise une meilleure lisibilité du code et une homogénéité dans les pratiques de développement.
4. **Imbrication** : la syntaxe HTML impose une structure arborescente où les éléments suivent un ordre d'imbrication hiérarchique où chaque balise ouvrante doit être fermée selon le principe *LIFO* (*Last In, First Out*). L'exemple 3.2 illustre cette conformité structurelle.

Exemple 3.2

✓ `<body><i>Hello World!</i></body>`

Le non respect de cette règle provoque une erreur d'interprétation par le navigateur comme illustré par l'exemple 3.3 où un chevauchement incorrect est effectué entre la balise `<i>` et la balise fermante `</body>`.

Exemple 3.3

✗ `<body><i>Hello World!</body></i>`

3.2 HTML : définition et historique

3.2.1 Qu'est ce que le HTML ?

Définition

HTML (HyperText Markup Language) est un langage de balisage standardisé, initialement développé par Tim Berners-Lee, qui permet de structurer et d'organiser le contenu des documents destinés au Web. Il définit la hiérarchie et la logique des éléments tels que les titres, paragraphes, listes et liens hypertextes. En tant que langage fondamental du Web, HTML facilite l'intégration et la présentation cohérente de textes, images et autres médias au sein des pages web.

3.2.2 HTML de base

3.2.2.1 Ossature d'un document HTML

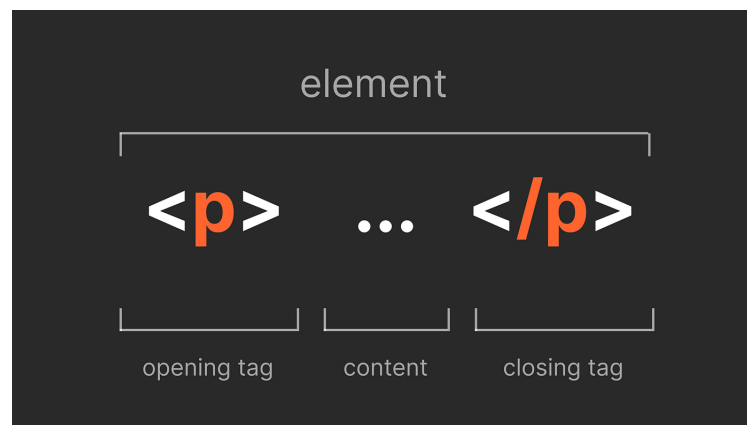
Avant d'aborder les éléments spécifiques du contenu HTML, il est crucial de comprendre la structure fondamentale qui constitue l'ossature de tout document HTML. Cette structure standardisée garantit une organisation cohérente et une interprétation correcte par les navigateurs web. Le listing suivant illustre les éléments fondamentaux qui définissent cette base essentielle : la déclaration de version *DOCTYPE*, l'élément conteneur `<html>`, la section `<head>` dédiée aux métadonnées et la section `<body>` réservée au contenu visible.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>My Page</title>  <!-- Title of the web page -->
5 </head>
6 <body>
7   <p>This is a text</p>  <!-- paragraph tag -->
8 </body>
9 </html>
```

Listing 3.1 – HTML

Chaque élément HTML est délimité par une balise ouvrante et une balise fermante. La balise ouvrante indique le début de l'élément et s'écrit entre chevrons, par exemple `<body>`, tandis que la balise fermante marque sa fin et se distingue par une barre oblique avant le nom, comme `</body>`. Entre ces deux balises se trouve le contenu de l'élément, qui peut être du texte, d'autres éléments HTML ou une combinaison des deux.

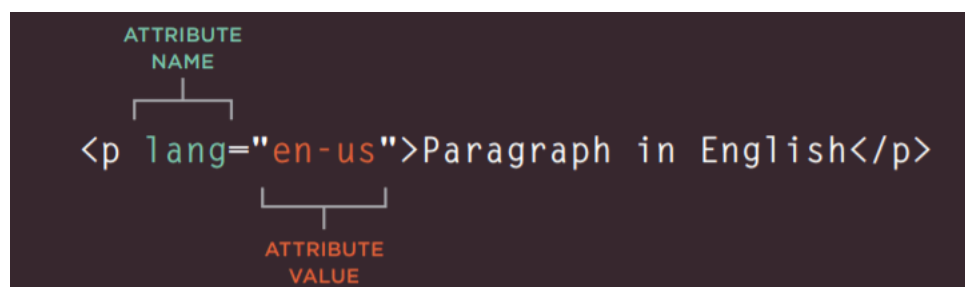
FIGURE 3.2.1 – opening & closing tag



Par exemple, dans `<p>This is a text</p>`, `<p>` est la balise ouvrante, `</p>` la balise fermante, et « This is a text » est le contenu de cette balise, ici un paragraphe.

Une balise HTML peut être enrichie par un ou des attributs fournissant des informations complémentaires relatives au contenu d'un élément. Ils s'inscrivent dans la balise ouvrante de l'élément et se composent de deux parties distinctes : un nom et une valeur.

FIGURE 3.2.2 – Attribute tag



Ici, un attribut nommé `lang` est utilisé pour indiquer la langue employée dans cet élément. La valeur de cet attribut sur cette page spécifie qu'il s'agit de l'anglais américain.

Cette syntaxe permet aux navigateurs d'interpréter correctement la portée et le rôle de chaque partie du document.

3.2.2.2 Les balises HTML

HTML se compose d'un ensemble de balises servant à organiser la structure ainsi que le contenu d'une page web. Selon leur fonction, les balises HTML se classent en plusieurs catégories :

1. **Balises de structure** : ces balises définissent l'architecture essentielle d'un document HTML en organisant son contenu selon une hiérarchie claire. Elles délimitent les principales zones logiques qui composent la page web. Le tableau ci-dessous présente les principales balises HTML utilisées pour structurer une page Web.

TABLE 3.1 – Balises de structure

Balise	Description
<html>	Élément racine qui englobe tout le document HTML
<head>	Section contenant les métadonnées (titre de la page, encodage, liens CSS...)
<body>	Contenu visible de la page

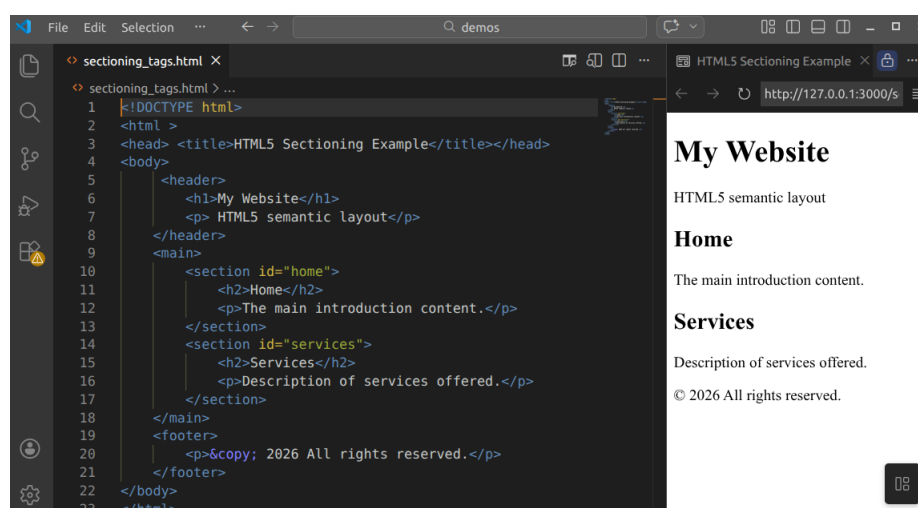
Introduites avec HTML5, les balises sémantiques structurales (Tableau 3.2) permettent d'organiser le contenu du <body> selon une logique sémantique claire, améliorant l'accessibilité.

TABLE 3.2 – Balises sémantiques de sectionnement

Balise	Description
<header>	Conteneur d'en-tête pour un document ou une section, regroupant le contenu introductif tel que les titres, le logo ou la navigation.
<footer>	Conteneur de pied de page pour un document ou une section, regroupant les informations de conclusion telles que les mentions légales, les coordonnées, les liens annexes ou les droits d'auteur.
<nav>	Conteneur de navigation regroupant les liens principaux permettant de se déplacer dans le site ou dans le document.
<section>	Conteneur regroupant un ensemble thématique de contenu, généralement accompagné d'un titre.
<article>	Conteneur regroupant un ensemble d'éléments désignant un contenu autonome, indépendant et réutilisable (comme un article de blog, une actualité ou une fiche produit).
<aside>	Conteneur désignant un contenu en marge, indirectement lié au contenu principal qui l'entoure.

Le code et le rendu de l'application de ces balises sont présentés par la Figure 3.2.3.

FIGURE 3.2.3 – Sectioning HTML5 tags



2. Balises de hiérarchie textuelle

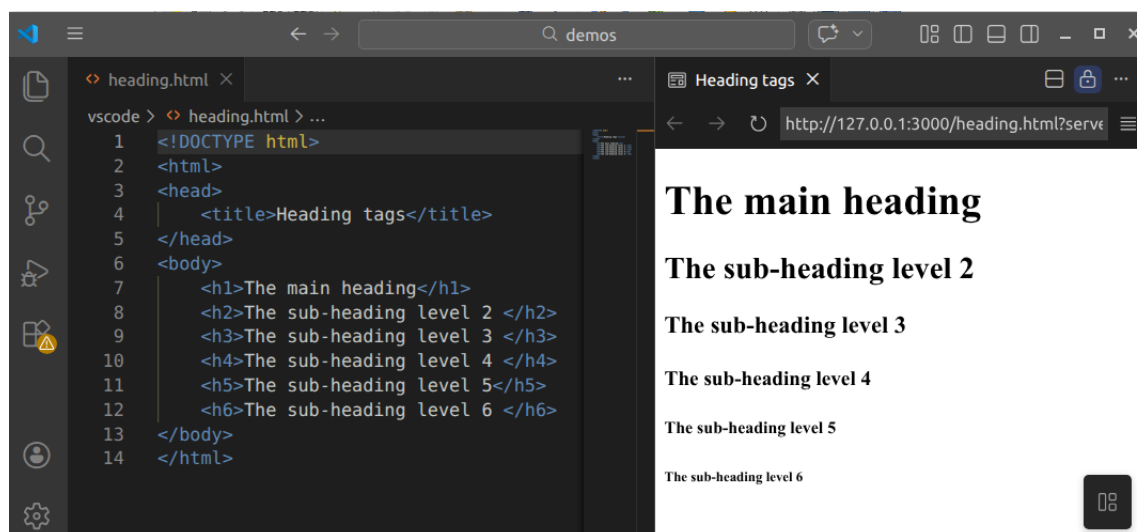
Ces balises de titres, hiérarchisées de `<h1>` à `<h6>` et présentées dans le tableau 3.3, permettent de structurer le contenu textuel d'un document.

TABLE 3.3 – Balises de hiérarchie textuelle

Balise	Meaning	Visual weight
<code><h1></code>	Main heading	Largest (32 px)
<code><h2></code>	Level 2 heading	Large (24 px)
<code><h3></code>	Level 3 heading	Medium-Large (18)
<code><h4></code>	Level 4 heading	Normal /base-size (16 px)
<code><h5></code>	Level 5 heading	Small (14 px)
<code><h6></code>	Level 6 heading	Smallest (10 px)

La Figure 3.2.4 illustre le code HTML correspondant aux balises de titre `<h1>` à `<h6>`, ainsi que le rendu visuel .

FIGURE 3.2.4 – Heading tags



3. Balises de formatage textuel

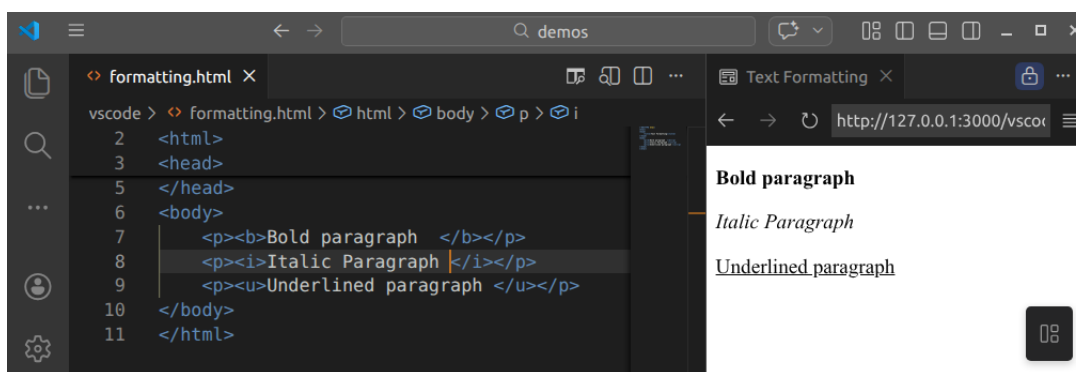
Les balises de formatage textuel, essentielles à la mise en forme du contenu, sont synthétisées dans le Tableau 3.4.

TABLE 3.4 – Balises de formatage

Balise	Meaning	Description
<code></code>	<u>bold</u>	L'encadrement du texte par les balises <code></code> et <code></code> permet d'afficher les caractères en gras.
<code><u></code>	<u>underline</u>	L'encadrement du texte par les balises <code><u></code> et <code></u></code> permet d'afficher les caractères soulignés.
<code><i></code>	<u>italic</u>	L'encadrement du texte par les balises <code><i></code> et <code></i></code> permet d'afficher les caractères en italique.
<code><p></code>	<u>paragraph</u>	L'encadrement du texte par les balises <code><p></code> et <code></p></code> permet de créer un paragraphe.
<code><sup></code>	<u>superscript</u>	La balise <code><sup></code> affiche le texte en exposant, utilisée notamment pour puissances mathématiques.
<code><sub></code>	<u>subscript</u>	La balise <code><sub></code> affiche le texte en indice, utilisée notamment pour les formules chimiques ou les notations mathématiques.

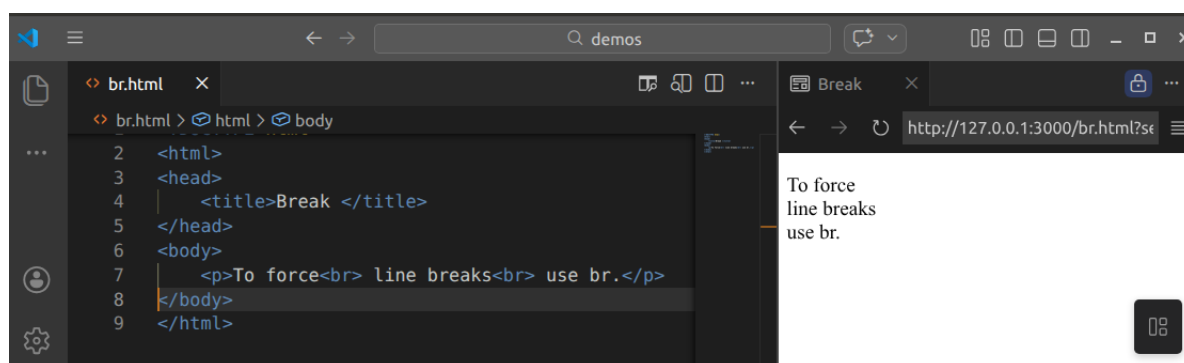
La Figure 3.2.5 illustre le code HTML correspondant aux balises `<p>` , `<b>`, `<i>` et `<u>` (voir Tableau 3.4), ainsi que le rendu visuel .

FIGURE 3.2.5 – Text formatting



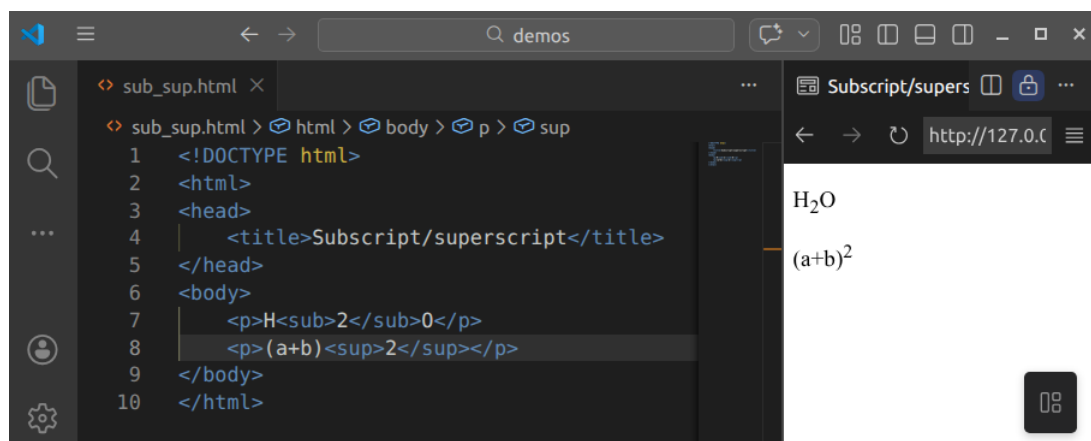
Comme vous l'avez déjà constaté, le navigateur affiche automatiquement chaque nouveau paragraphe dans une nouvelle ligne. Toutefois, si vous souhaitez insérer un saut de ligne au sein même d'un paragraphe, vous pouvez utiliser la balise de saut de ligne `<br/>`.

FIGURE 3.2.6 – Break Line



L'exemple suivant illustre l'utilisation des balises `<sub>` et `<sup>` (voir Tableau 3.4) pour afficher des indices et des exposants dans du texte. La balise `<hr/>`, quant à elle, permet d'insérer une ligne horizontale afin de marquer une séparation visuelle entre différentes sections d'un document, facilitant ainsi la structuration du contenu.

FIGURE 3.2.7 – Subscript & Superscript



Note

Les caractères spéciaux doivent être remplacés par leurs entités HTML correspondantes, présentées par le Tableau 3.5.

TABLE 3.5 – Entités HTML des caractères spéciaux

Symbole	>	<	≥	≤	&	"	espace
Code HTML	<code>&gt;</code>	<code>&lt;</code>	<code>&ge;</code>	<code>&le;</code>	<code>&amp;</code>	<code>&quot;</code>	<code>&nbsp;</code>

4. Balises de listes structurées

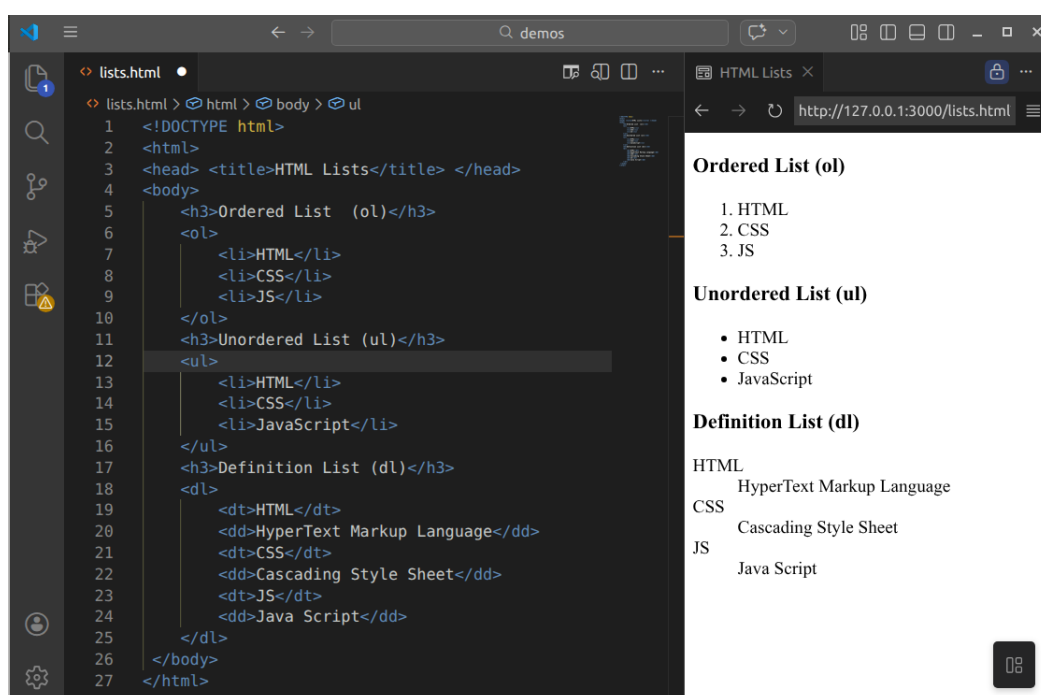
Les balises HTML dédiées à la création des listes offrent un moyen simple et efficace de présenter des informations sous forme ordonnée ou non ordonnée. Elles sont récapitulées dans le Tableau 3.6.

TABLE 3.6 – Balises de listes

Balise	Meaning	Description
<code></code>	<u>O</u> rd <u>e</u> red <u>L</u> ist	Les balises <code></code> et <code></code> délimitent une liste ordonnée dans laquelle les éléments sont automatiquement numérotés.
<code></code>	<u>U</u> n <u>o</u> rd <u>e</u> red <u>L</u> ist	Les balises <code></code> et <code></code> délimitent une liste non ordonnée (dite aussi une liste à puce) dont les éléments sont présentés sans numérotation.
<code></code>	<u>L</u> ist <u>I</u> tem	Les balises <code></code> et <code></code> délimitent un élément appartenant à une liste.
<code><dl></code>	<u>D</u> efinition <u>L</u> ist	Les balises <code><dl></code> et <code></dl></code> délimitent une liste de définitions associant des termes à leurs descriptions.
<code><dt></code>	<u>D</u> efinition <u>T</u> erm	Les balises <code><dt></code> et <code></dt></code> délimitent un terme à définir au sein d'une liste de définitions.
<code><dd></code>	<u>D</u> efinition <u>D</u> escription	Les balises <code><dd></code> et <code></dd></code> délimitent la description ou la définition associée à un terme.

L'exemple ci-après démontre l'usage des balises `<ol>`, `<ul>`, `<li>` et `<dl>`, `<dt>` et `<dd>`, ainsi que leur rendu visuel.

FIGURE 3.2.8 – HTML Lists



Note

Par défaut, une liste ordonnée commence à **1** et utilise une numérotation numérique. Toutefois, HTML permet de définir la valeur de départ de cette numérotation à l'aide de l'attribut **start**, applicable uniquement aux listes ordonnées (`<ol>`).

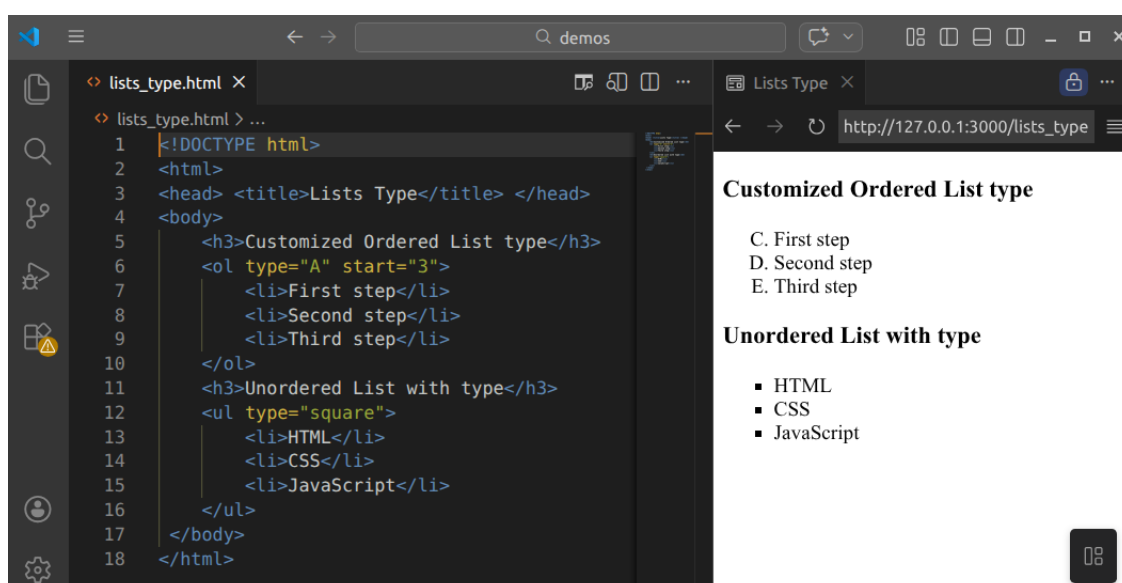
Certaines listes HTML telles que `<ol>` et `<ul>` disposent de l'attribut **type**, lequel permet de contrôler leur style de numérotation. Toutefois, les valeurs associées à cet attribut varient selon la nature de la liste utilisée, qu'il s'agisse d'une liste ordonnée ou non ordonnée.

TABLE 3.7 – Attribut type appliqué à `<ol>` et `<ul>`

Balise	Rôle de l'attribut type	valeurs possibles
<code></code>	Définit le style de numérotation des éléments	1 (numérotation par défaut). A (lettre majuscule) a (lettres minuscule) I ou i (chiffres romains)
<code></code>	Définit la forme des puces des éléments	disc (puce pleine par défaut) circle (cercle vide) square (carré)

L'exemple présenté ci-après met en évidence l'application des attributs `type` et `start` aux listes ordonnées, tout en illustrant l'utilisation de l'attribut `type` dans le cas des listes non ordonnées.

FIGURE 3.2.9 – Customized List



Important

Les listes HTML peuvent être imbriquées, qu'elles soient du même type ou de types différents, afin de structurer l'information de manière hiérarchique.

5. Balises de données tabulaires

Les tableaux constituent un moyen efficace pour organiser des données en lignes et en colonnes. Le langage HTML met à disposition des balises spécifiques permettant de structurer ces données de manière claire et cohérente. Les principales balises utilisées à cet effet sont synthétisées dans le Tableau 3.8.

TABLE 3.8 – Balises HTML de tableau

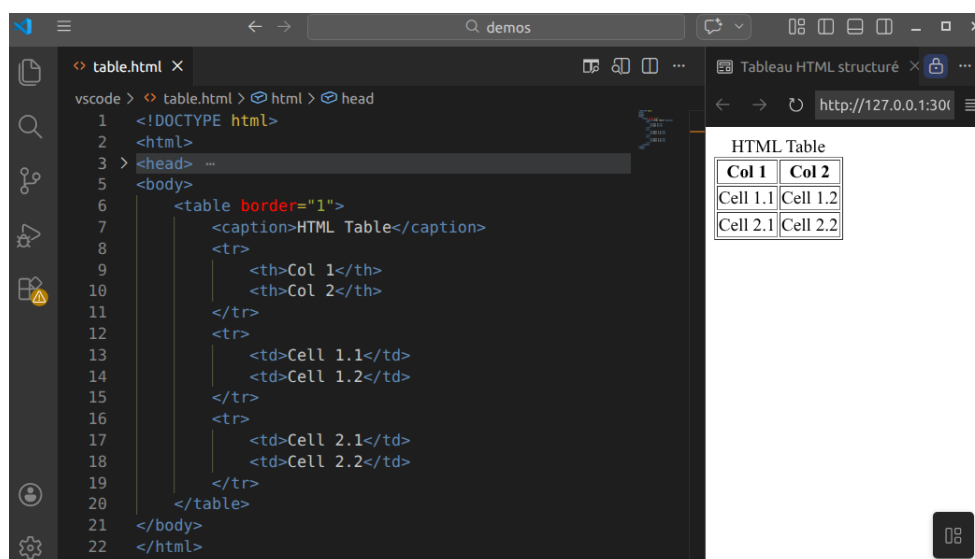
Balise	Description
<table>	Définit le tableau dans son ensemble
<caption>	Donne un titre au tableau (placé juste après <table>)
<tr>	<u>T</u> able <u>R</u> ow :Représente une ligne du tableau
<td>	<u>T</u> able <u>D</u> ata : Définit une cellule de données
<th>	<u>T</u> able <u>H</u> ead : Définit une cellule d'en-tête.
<thead>	<u>T</u> able <u>H</u> ead : regroupe les lignes d'en-tête du tableau. Il contient généralement des cellules <th> et décrit les colonnes.
<tbody>	<u>T</u> able <u>B</u> ody : contient l'ensemble des lignes de données du tableau. Il regroupe les cellules <td>.
<tfooter>	Table Footer : regroupe les lignes de pied de tableau, souvent utilisées pour afficher des informations récapitulatives.

Note

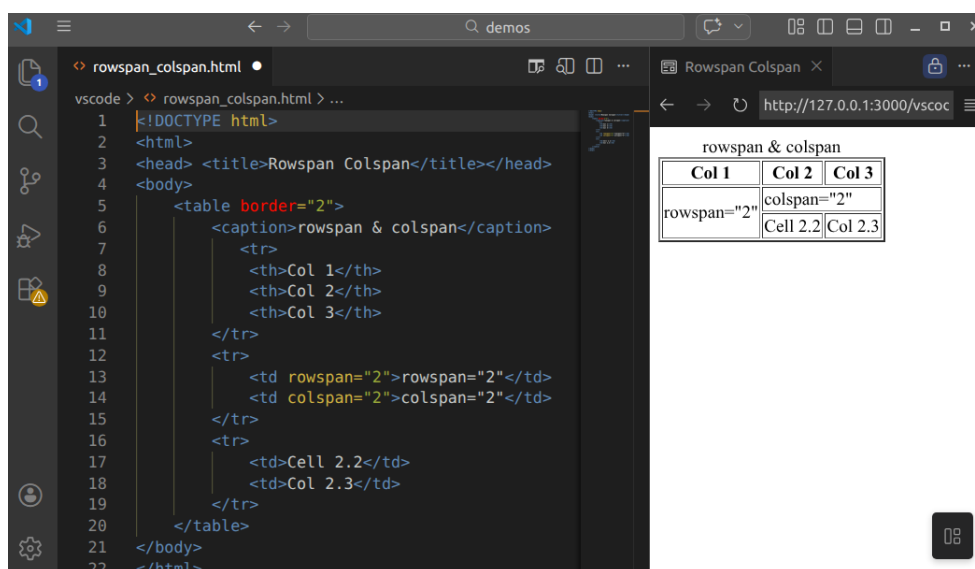
Les balises <thead>, <tbody> et <tfoot> sont particulièrement utiles pour structurer des tableaux volumineux ou complexes.

L'exemple présenté ci-après montre la création d'un tableau HTML en utilisant les balises <table>, <tr>, <th>, <td> et <caption>. Dans cet exemple, l'attribut border de la balise <table> a été fixé à 2 pour rendre visible la bordure, qui est par défaut nulle.

FIGURE 3.2.10 – Code HTML et rendu visuel du tableau



Il arrive parfois qu'une cellule doit s'étendre sur plusieurs colonnes ou plusieurs lignes. Pour cela, on utilise les attributs `colspan` et `rowspan` de la balise `<td>`. Les cellules recouvertes par cette cellule étendue n'ont alors pas besoin d'être définies.

FIGURE 3.2.11 – Fusion de cellules avec `rowspan` et `colspan`

Comme illustré dans l'exemple ci-dessus, les attributs `rowspan` et `colspan` permettent de fusionner des lignes et des colonnes, rendant les tableaux HTML plus structurés.

6. Balises de liens et navigation

L'attribut href est un composant essentiel en HTML qui permet de créer des liens hypertextes, appelés aussi hyperliens, facilitant la navigation entre les pages web ou vers différentes ressources en ligne. On crée un hyperlien grâce à la balise `<a>` (voir Figure 3.2.12), qui entoure le texte cliquable du lien.

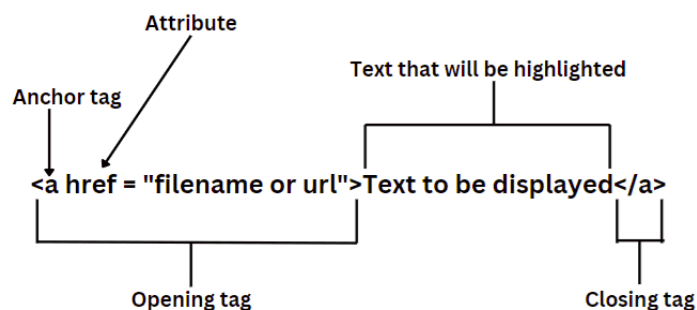
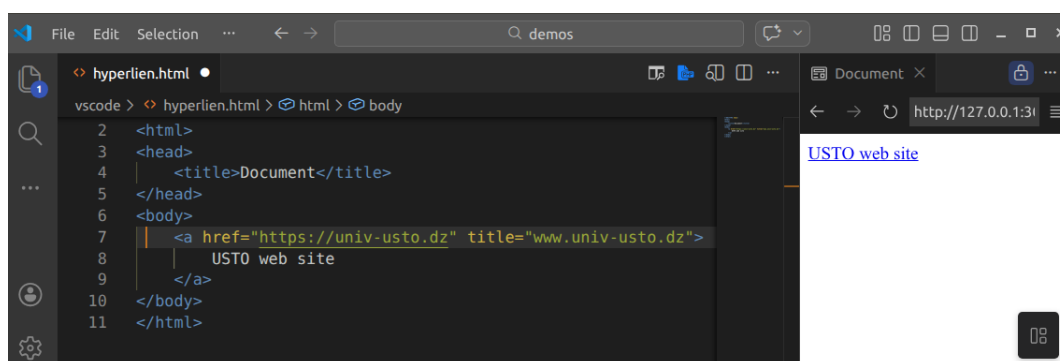


FIGURE 3.2.12 – Hyperlink tag

Cet élément inclut l'attribut href, qui spécifie l'adresse de destination vers laquelle l'utilisateur est redirigé lorsqu'il clique sur le texte. Cette adresse peut correspondre à une page web interne (une autre page du même site), une page web externe (un site différent), un document (tel qu'un PDF ou une image) ou une ancre située dans la même page. L'URL définie dans href détermine donc la cible du lien, tandis que le texte visible entre les balises constitue le contenu cliquable pour l'utilisateur comme illustré dans l'exemple ci-dessous

FIGURE 3.2.13 – Exemple d'hyperlien



Il est également possible d'ajouter l'attribut title à la balise `<a>` pour fournir une description supplémentaire visible au survol du lien. Les principaux types de liens hypertextes ainsi que leur syntaxe correspondante sont synthétisés dans le Tableau 3.9.

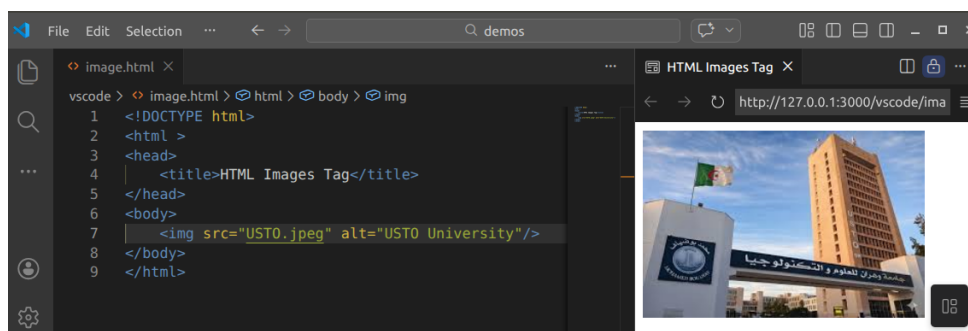
Lien	Exemple
Vers un fichier dans le même dossier	<pre> go to index</pre> Permet d'accéder à la page « index.html » qui se trouve dans le même dossier que la page appelante
Vers un fichier dans un sous-dossier	<pre>usto logo</pre> Permet d'accéder à l'image « usto-logo.jpg » qui se trouve dans un sous-dossier nommé « images »
Navigation vers un dossier parent	<pre>go to index</pre> Permet d'accéder au fichier « index.html » situé dans le dossier parent
Vers une URL externe	<pre>www.univ-usto.dz</pre> Mène vers « www.univ-usto.dz »
mailto	<pre>Send an email</pre> Permet d'ouvrir le client de messagerie de l'utilisateur pour envoyer un e-mail à l'adresse spécifiée.
Ancre (Lien vers une section spécifique d'une page)	<pre>Go to Section 2</pre> Dirige l'utilisateur vers une section spécifique d'une même page identifiée par « section2 ».

TABLE 3.9 – Exemples d'hyperliens

7. Balises de contenu graphique

Pour insérer une image dans une page Web, on utilise la balise `<img>`, qui est une balise autofermante, c'est-à-dire qu'elle ne possède ni contenu ni balise de fermeture. Elle requiert au minimum l'attribut `src`, qui indique le chemin vers l'image à afficher. Il est également fortement recommandé d'ajouter l'attribut optionnel `alt`, qui fournit une description textuelle de l'image pour les navigateurs en cas de non-chargement. L'exemple présenté ci-après montre le code HTML permettant l'insertion d'une image en utilisant la balise `<img>`.

FIGURE 3.2.14 – HTML Image



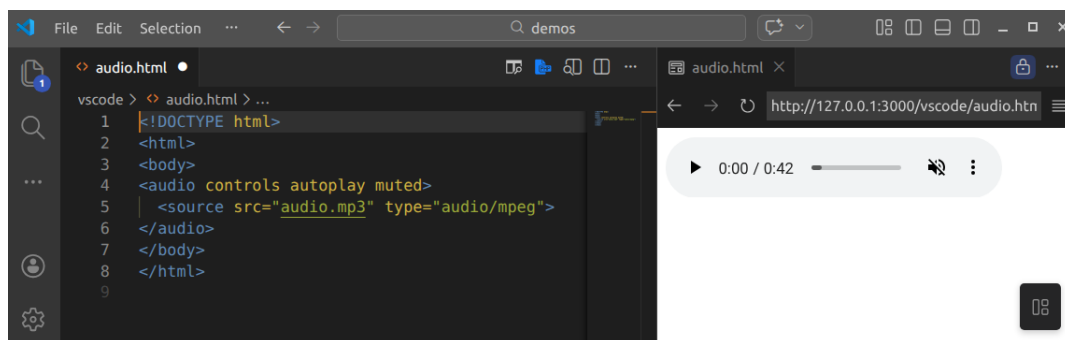
La taille de l'image peut être définie à l'aide des attributs `width` et `height`, qui spécifient respectivement sa largeur et sa hauteur. L'attribut `title` permet d'afficher une information complémentaire lors du survol de l'image. Enfin, l'attribut `align` peut être utilisé pour préciser l'alignement horizontal de l'image (`left` ou `right`), tandis que l'alignement vertical peut être défini à l'aide de l'attribut `valign`, qui accepte les valeurs `top`, `middle` ou `bottom`.

8. Balises de contenu multimédia

Les balises multimédias permettent d'intégrer des contenus dynamiques tels que des fichiers audio, des vidéos, ou des contenus externes dans une page Web. Contrairement aux images, ces éléments nécessitent un lecteur intégré ou un accès à des ressources externes pour leur affichage.

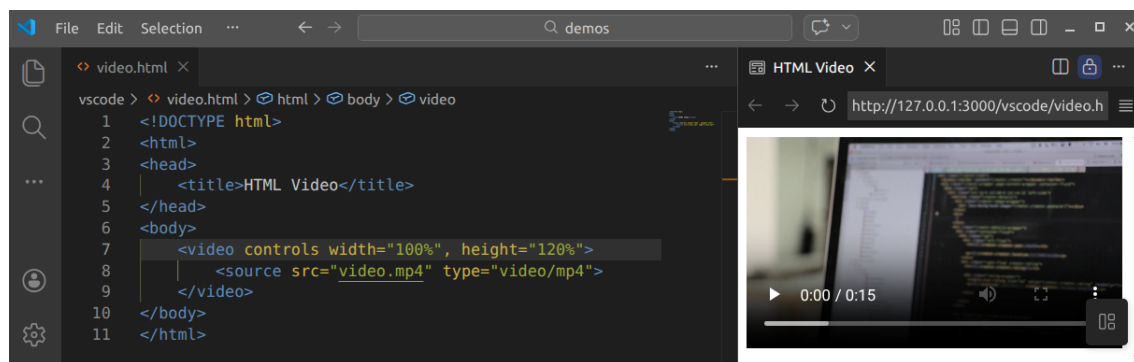
- La balise `<audio>` permet d'intégrer un fichier sonore au sein d'une page Web. Elle dispose de plusieurs attributs destinés à contrôler le comportement du média : `controls` affiche les commandes de lecture, `autoplay` déclenche automatiquement la lecture lors du chargement de la page, `loop` assure une lecture en continu, tandis que `muted` désactive le son par défaut.

FIGURE 3.2.15 – HTML audio example



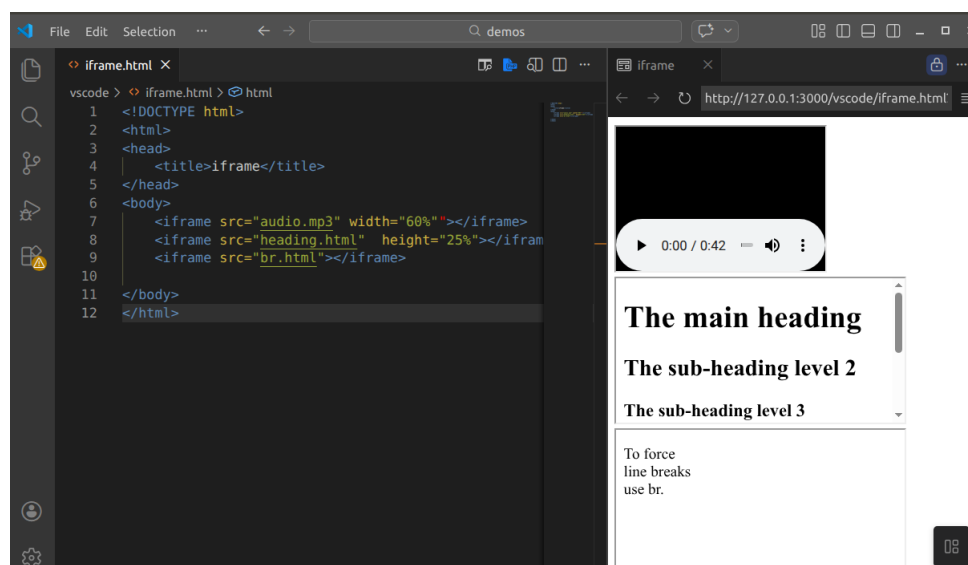
- La balise `<video>` permet d'intégrer une vidéo directement dans la page Web. Elle possède plusieurs attributs similaires à ceux de la balise `<audio>` (`controls`, `autoplay`, `loop`, `muted`), auxquels s'ajoutent des attributs spécifiques tels que `width`, `height` et `poster`, permettant de définir respectivement les dimensions de la vidéo et l'image affichée avant sa lecture.

FIGURE 3.2.16 – HTML video example



- La balise `<iframe>` permet d'intégrer des contenus externes à une page Web, comme des vidéos YouTube, des cartes Google Maps, ou des formulaires provenant d'autres sites. La balise `<iframe>` comporte plusieurs attributs permettant de contrôler l'intégration du contenu externe. L'attribut `src` spécifie l'URL de la ressource à afficher, tandis que `width` et `height` définissent respectivement la largeur et la hauteur du cadre intégré. L'attribut `frameborder`, lorsqu'il est fixé à la valeur "0", supprime la bordure. Enfin, l'attribut `allowfullscreen` autorise l'affichage du contenu en mode plein écran, notamment dans le cas d'une vidéo.

FIGURE 3.2.17 – iframe example



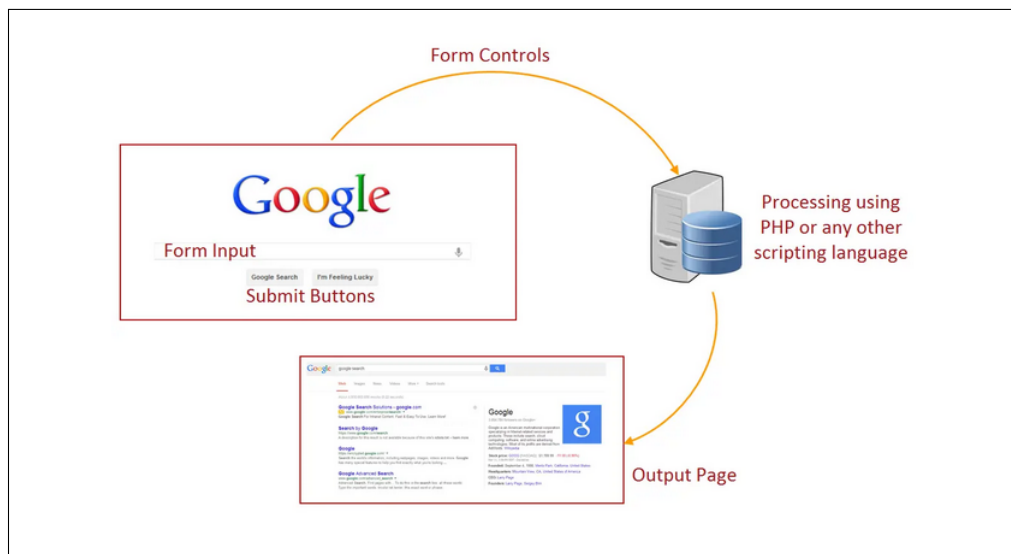
9. **Éléments conteneurs** : ces éléments jouent un rôle fondamental dans la structuration d'une page web. Ils permettent d'organiser, de regrouper et de styliser le contenu de manière logique et visuelle. Parmi eux, `<div>` et `<span>` sont les plus utilisés.

- **<div>** : conteneur générique de type *bloc*, il s'étend sur la totalité de la largeur de son parent. Il est employé pour délimiter et organiser les grandes structures de la page en-têtes, barres de navigation, sections principales ou pieds de page et constitue ainsi l'ossature layout du document.
- **** : conteneur générique de type *inline*, il n'induit aucun saut de ligne et n'occupe que l'espace strictement nécessaire à son contenu. Il est utilisé pour cibler et appliquer un style ou un comportement à une portion restreinte de texte à l'intérieur d'un bloc, sans altérer la structure globale du document.

3.2.2.3 Les formulaires

Les formulaires HTML constituent un pilier interactif du Web en permettant à l'utilisateur d'interagir avec un serveur Web. Le traitement des données saisies via un formulaire consiste généralement à les transmettre à une application exécutée côté serveur, souvent développée à l'aide du langage PHP ou un autre langage de script. Le processus de traitement mis en oeuvre lors de la soumission d'un formulaire est illustré par la Figure suivante.

FIGURE 3.2.18 – Form process



Comme l'illustre l'exemple de la Figure 3.2.18, l'utilisateur saisit une requête de recherche puis valide sa demande. Les informations fournies sont alors transmises au serveur de Google, qui les traite à l'aide d'un programme exécuté côté serveur, avant de renvoyer au navigateur une page de réponse. Enfin, l'utilisateur visualise la page contenant les résultats de recherche.

En HTML, tout formulaire est structuré autour de la balise `<form>`, qui sert à la fois de conteneur et de mécanisme de transmission des données saisies par l'utilisateur.

```

1 <form method="....." action=".....">
2 <!-- champs du formulaire -->
3 </form>
  
```

L'attribut `method` définit la méthode de transmission des données d'un formulaire. Il peut prendre les valeurs `GET` ou `POST`.

Dans le cas de la méthode `GET`, les données sont envoyées de manière visible dans l'URL, tandis que pour la méthode `POST`, elles sont transmises dans le corps de la requête. Cette dernière option offre une meilleure confidentialité et permet d'envoyer des volumes de données plus importants.

L'attribut `action` précise le script côté serveur chargé du traitement de la requête.

Dans un formulaire, les éléments de saisie peuvent être regroupés en trois catégories : `<input>`, `<select>` et `<textarea>`.

1. **L'élément `input`** : permet de créer différents contrôles de formulaire. La valeur de l'attribut `type` détermine la nature du champ généré. Les principaux types sont récapitulés dans le tableau ci-dessous.

Note

Chaque champ permettant la saisie ou la sélection d'une information doit être associé explicitement à une étiquette descriptive (`label`). Cette association permet d'identifier clairement la fonction du champ

TABLE 3.10 – Les différents types de l'élément `<input>`

Type	Exemple	Apparence
text	<pre><label for="nom">Nom :</label> <input type="text" id="nom" name="nom"></pre>	Nom :
number	<pre><label for="age">Age :</label> <input type="number" id="age" name="age"></pre>	Age : <input type="number" value="18"/>
password	<pre><label for="pass">Password :</label> <input type="password" id="pass" name="pass"></pre>	Password :
hidden	<pre><input type="hidden" name="nationalité" value=" Algérienne"></pre>	
radio	<pre><input type="radio" id="licence" name="niveau" value="licence" checked> <label for="licence">Licence </label> <input type="radio" id="master" name="niveau" value="master"> <label for="master">Master</label></pre>	☒ Licence ☐ Master

checkbox	<pre> <input type="checkbox" id="html" name=" competences[]" value="HTML"> <label for="html">HTML</label> <input type="checkbox" id="css" name=" competences[]" value="CSS"> <label for="css">CSS</label> <input type="checkbox" id="js" name=" competences[]" value="JavaScript"> <label for="js">JavaScript</label> </pre>	☒ HTML ☐ CSS ☒ JavaScript
file	<pre><input type="file" accept=".pdf"/></pre>	Choose File No file chosen
search	<pre> <input type="search"/> <input type="submit" value="Search"/> </pre>	Search
tel	<pre> <label for="phone">Telephone :</label> <input type="tel" id="pone" name="phone" placeholder="213xxxxxxxxxx"> </pre>	Telephone : 213xxxxxxxxxx
email	<pre> <label for="mail">Email :</label> <input type="email" id="mail" name="mail" placeholder="studentID@univ-usto.dz"> </pre>	Email : studentID@univ-usto.dz
date	<pre><input type="date" id="date" name="date"></pre>	mm/dd/yyyy
month	<pre><input type="month" id="month" name="month"></pre>	-----
week	<pre><input type="week" id="week" name="week"></pre>	Week --, ----
time	<pre><input type="time" id="time" name="time"></pre>	--:-- --
datetime-local	<pre><input type="datetime-local" id="dateTime" name="dateTime"></pre>	mm/dd/yyyy, --:-- --
URL	<pre><input type="url" id="url" name="url" placeholder="URL"></pre>	URL
image	<pre><input type="image" id="img" name="img"></pre>	
color	<pre><input type="color" id="color" name="color"></pre>	<input type="color" value="black"/>
button	<pre><input type="button" id="btn" value="Click me"></pre>	Click me
submit	<pre><input type="submit"></pre>	Submit
reset	<pre><input type="reset"></pre>	Reset

Les principaux attributs de la balise `<input>` permettent de préciser le comportement et les contraintes de saisie d'un champ.

- ▶ L'attribut `name` identifie la donnée transmise au serveur, tandis que l'attribut `value` définit sa valeur. Par ailleurs, l'attribut `id` (identifiant) permet d'attribuer un identifiant unique à un élément.
- ▶ L'attribut `required` impose une saisie obligatoire.
- ▶ Les attributs `readonly` et `disabled` contrôlent respectivement la possibilité de modification et l'activation du champ.
- ▶ L'attribut `accept` limite les types de fichiers pouvant être importés par le champ `input`.
- ▶ L'attribut `multiple` autorise des sélections multiples, notamment pour les fichiers ou les adresses électroniques.

L'exemple ci-dessous montre l'utilisation du champ de saisie `input` de type `text`, `email`, `tel`, `hidden`, `submit` et `reset` dans la construction d'un formulaire HTML.

FIGURE 3.2.19 – Exemple d'input

```
<html lang="fr">
</head>
<body>
<form action="#" method="post">
<fieldset>
  <legend>Informations personnelles</legend>
  <div>
    <label for="nom">Nom :</label>
    <input type="text" id="nom" name="nom" required>
  </div>
  <div>
    <label for="email">Adresse mail :</label>
    <input type="email" id="email" name="email" required>
  </div>
  <div>
    <label for="tel">Téléphone :</label>
    <input type="tel" id="tel" name="tel">
  </div>
  <input type="hidden" name="statut" value="candidat">
</fieldset>
<input type="submit">
<input type="reset">
</form>
```

Informations personnelles

Nom :

Adresse mail :

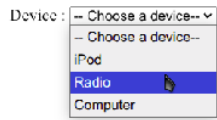
Téléphone :

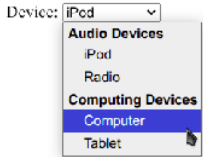
Important

- (a) L'association entre un champ de saisie ou de sélection et son libellé s'opère via les attributs `id` (sur l'élément `<input>`) et `for` (sur l'élément `<label>`). Ces deux attributs doivent porter une valeur identique, établissant ainsi une liaison fonctionnelle qui permet d'activer le champ en cliquant sur son libellé.
- (b) Pour les boutons radio (`<input type="radio">`), l'ensemble des options appartenant à un même groupe doivent partager une valeur commune pour l'attribut `name`. Cette contrainte assure le comportement d'exclusivité propre à ce type de contrôle, limitant la sélection à une seule option parmi le groupe.
- (c) S'agissant des cases à cocher (`<input type="checkbox">`), il est vivement recommandé d'utiliser la notation par crochets pour l'attribut `name` (exemple : `name="interets[]"`). Cette convention permet au serveur de recevoir l'ensemble des valeurs sélectionnées sous la forme d'un tableau, facilitant ainsi leur traitement côté backend.

2. **L'élément `<select>...</select>`** : permet de créer un menu déroulant offrant plusieurs choix à l'utilisateur, chaque choix étant défini par un élément `<option>`, et les options pouvant être regroupées en catégories logiques grâce à l'élément `<optgroup>`. L'exemple et l'apparence de ces éléments sont synthétisés dans le tableau suivant.

TABLE 3.11 – L'élément `<select>`

Élément	Exemple	Apparence
select avec avec option	<pre> <label for="device">Device :</label> <select id="device" name="devices"> <option value="">Choose a device</option> <option value="ipod">iPod</option> <option value="radio">Radio</option> <option value="computer">Computer</option> </select> </pre>	

select avec avec optgroup et option	<pre> <label for="device">Device :</label> <select id="device" name="devices"> <option value="">Choose a device</option> <optgroup label="Audio Devices"> <option value="ipod">iPod</option> <option value="radio">Radio</option></ <optgroup label="Computing Devices"> <option value="computer">Computer</option> <option value="tablet">Tablet</option> </optgroup> </select> </pre>	
--	--	---

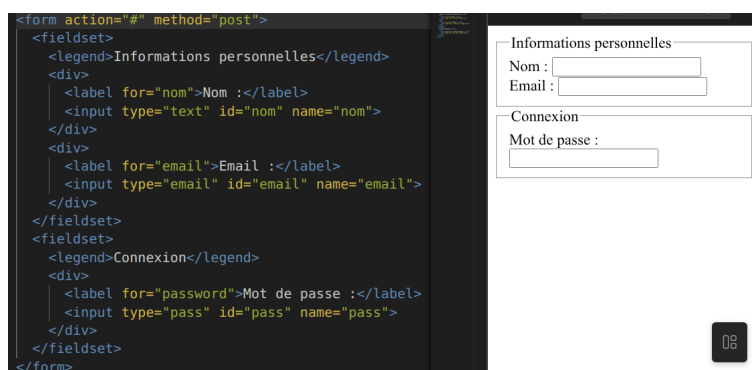
3. L'élément `<textarea>...</textarea>` : permet de créer un champ de saisie multi-ligne pour que l'utilisateur puisse entrer un texte long. L'exemple et l'apparence de cet élément sont synthétisés dans le tableau suivant.

TABLE 3.12 – L'élément `<select>`

Type	Exemple	Apparence
cols et rows	<pre> <label for="comments">Votre commentaire :</ label> <textarea id="comments" name="comments" rows ="4" cols="30" placeholder="Écrivez votre message ici..."> </textarea> </pre>	

4. Les éléments **Fieldset** et **legend** : l'élément `<fieldset>` permet de regrouper des champs qui appartiennent à une même catégorie, tandis que l'élément `<legend>` sert à donner un titre à ce groupe de champs. L'exemple suivant illustre l'utilisation des éléments `<fieldset>` et `<legend>` pour structurer les champs d'un formulaire

FIGURE 3.2.20 – Fieldset et legend



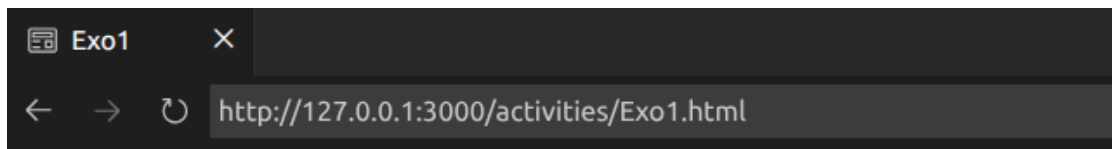
Dans ce chapitre, nous avons étudié les principales balises du langage HTML permettant de structurer et d'organiser le contenu d'une page Web. Nous avons notamment abordé l'utilisation des titres pour hiérarchiser les informations, des paragraphes pour présenter le texte, des listes pour énumérer des éléments, des tableaux pour représenter des données sous forme tabulaire, des liens hypertextes pour relier différentes pages, ainsi que des formulaires pour permettre aux utilisateurs de saisir et d'envoyer des informations.

Toutefois, le langage HTML est principalement dédié à la structuration du contenu d'un document Web. La gestion de l'apparence visuelle d'une page, telle que les couleurs, les polices de caractères, les espacements ou la disposition des éléments, est assurée par les feuilles de style en cascade.

3.3 Exercices

Exercice 3.3.1

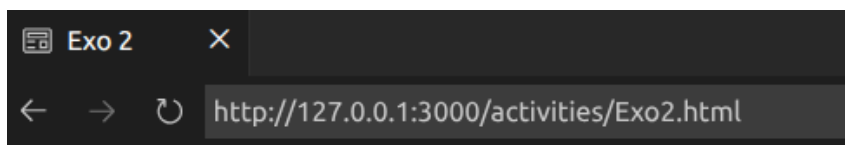
Donner le code HTML permettant d'obtenir le tableau illustré par la figure suivante.



Technologie	Niveau	Ressource
HTML	Débutant	w3school
CSS	Intermédiaire	
Javascript	Avancé	
Remarque : Ces technologies constituent les bases du développement web.		

Exercice 3.3.2

Donner le code HTML permettant de créer le rendu tel qu'illustré par la figure suivante.



Mes compétences

- HTML
- CSS
- Javascript
- PHP
- MYSQL

Les langues maîtrisées

1. Arabe
2. Anglais
3. Français

Exercice 3.3.3

Donner le code HTML permettant de créer le rendu du formulaire illustré par la figure suivante.

NB : les options offertes par le service sont administration, informatique et ressources humaines.

Exo 3

http://127.0.0.1:3000/activities/Exo3.html

Informations personnelles

Numéro de sécurité sociale : 000-00-0000

Email :

Téléphone :

Date d'embauche : mm/dd/yyyy

Sexe : ☐ Homme ☐ Femme

Fonction et service

Fonction :

Service : --Choisir une fonction--

Compétences et notes

☐ Communication ☐ Gestion ☐ Informatique

Remarques : Notes complémentaires...

Envoyer

Note

Les solutions des exercices sont disponibles en annexe à la fin de ce document.

Cascading Style Sheet

Objectifs du chapitre

At the end of this chapter, you will understand :

1. xxxx
2. yyyyyy
3. zzzz

4.1 Introduction

Le **CSS** (Cascading Style Sheets) est un langage de feuilles de style utilisé pour définir la présentation visuelle des documents web écrits en HTML. Il constitue un élément fondamental du développement d'applications web modernes en permettant de dissocier clairement la structure du contenu de sa mise en forme.

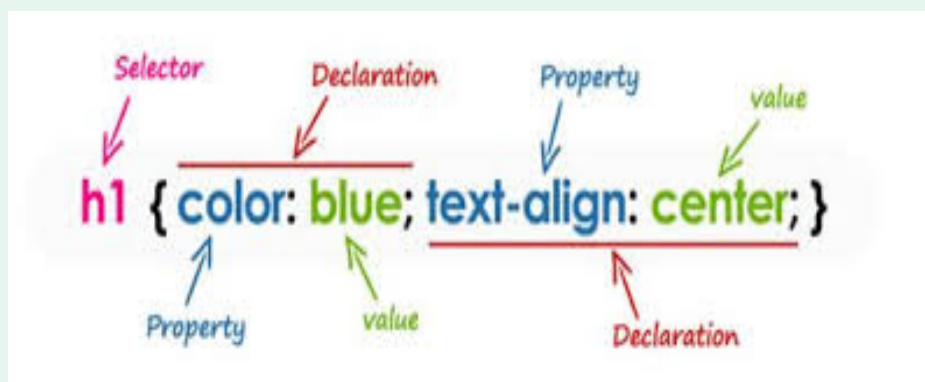
Grâce au CSS, l'apparence d'une page web peut être modifiée sans affecter son contenu. Ce langage offre un ensemble de règles permettant de contrôler avec précision l'affichage des éléments HTML, notamment la typographie, les couleurs, les marges, les bordures, le positionnement et le recouvrement des composants graphiques tels que le texte et les images.

Dans ce chapitre, nous présenterons les principes fondamentaux du CSS, ses mécanismes de fonctionnement et ses principales applications dans le développement d'interfaces web.

4.2 Principe de fonctionnement du CSS

Le CSS repose sur des règles constituant une feuille de style, qui définissent la mise en forme des éléments HTML. Comme illustré dans l'exemple 3.1, chaque règle associe un sélecteur identifiant l'élément ciblé aux déclarations précisant les styles à appliquer. Chaque déclaration est composée d'une propriété et d'une valeur, séparées par deux points.

Exemple 4.1



Tel qu'illustré par l'exemple 3.1, cette règle permet de modifier l'apparence des titres `<h1>` en leur appliquant une couleur bleu et en les alignant au centre de la page.

4.3 Les différentes méthodes d'intégration du CSS

Il existe plusieurs méthodes permettant d'intégrer les feuilles de style CSS dans un document HTML. Ces méthodes diffèrent par leur mode d'utilisation, leur portée et leur impact sur l'organisation du code.

4.3.1 CSS en ligne (inline)

Le style CSS inline est déclaré à l'intérieur d'une balise HTML à l'aide de l'attribut `style`, et sa portée est limitée uniquement à la balise où il a été inséré .

```
1 <p>a default color paragraph</p>
2 <p style="color:blue;">A Blue paragraph</p>
```

Dans cet exemple, seul le deuxième paragraphe `<p>` est impacté par le `style` bleu.

Note

L'insertion de styles inline complique la maintenance du code et compromet la séparation entre le contenu et la présentation ; son usage est donc généralement déconseillé.

4.3.2 CSS interne (internal)

Le style CSS interne est applicable à tout le document et il est déclaré dans le bloc `<style>` placé dans la section `<head>`. Cette méthode permet d'appliquer des règles de mise en forme à plusieurs éléments du document.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     p { color: blue; }
6   </style>
7 </head>
8 <body>
9   <p>A Blue paragraph</p>
10  <p>An other blue paragraph !</p>
11 </body>
12 </html>
```

Dans cet exemple et contrairement au style inline qui n'affecte qu'une seule balise, dans le style interne, tous les paragraphes `<p>` du document auront une couleur bleue.

Note

L'utilisation de styles internes centralise la mise en forme au sein d'un même document HTML, mais limite la réutilisation des styles et complique la maintenance lorsque le site comporte plusieurs pages.

4.3.3 CSS externe (external)

Le style est déclaré dans un fichier externe ayant l'extension ".css" et est référencé dans la section `<head>` par la balise `<link>` qui utilise trois attributs principaux :

- ▶ `rel="stylesheet"` : précise la relation entre le document HTML et la feuille de style.
- ▶ `type="text/css"` : indique le type du document lié correspondant à une feuille de style CSS.
- ▶ `href="/.../.../file_name.css"` : permet de spécifier le chemin d'accès et le nom du fichier CSS.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <link rel="stylesheet" type="text/css"
5     href="style.css">
6 </head>
7 <body>
8   <p>A blue paragraph</p>
9   <p>An other blue paragraph</p>
10 </body>
</html>
```

```
1 p{color:blue;}
```

Listing 4.1 –
style.css

Dans cet exemple, la feuille de style externe (à droite) `style.css` est liée au document HTML via la balise `<link>`. Elle contient une règle CSS qui applique la couleur bleue à tous les paragraphes (`<p>`) du document.

Important

L'utilisation de feuilles de style externes favorise la séparation entre le contenu et la présentation, améliore la maintenabilité du code et permet la réutilisation cohérente des styles sur l'ensemble des pages d'un site web.

4.4 Les sélecteurs CSS

Les sélecteurs CSS constituent un mécanisme fondamental permettant d'identifier les éléments HTML auxquels s'appliquent les règles de mise en forme.

4.4.1 Sélecteurs de base

Les sélecteurs de base permettent de cibler des éléments selon leur nature ou leurs caractéristiques spécifiques. Ils se déclinent en quatre catégories : le sélecteur universel, le sélecteur de type (balise), le sélecteur d'identifiant (ID) et le sélecteur de classe.

- ▶ le sélecteur universel, représenté par `*`, cible tous les éléments du document HTML.
- ▶ le sélecteur de balise cible les éléments du document HTML par leur nom de balise.
- ▶ le sélecteur ID (noté `#id_name`) cible un élément HTML, cela est nécessaire pour styliser un élément de manière exclusive.
- ▶ le sélecteur de classe (noté `.class_name`) cible plusieurs éléments HTML partageant le même attribut `class`.

L'exemple de code suivant illustre l'application des styles précédemment présentés dans une page HTML liée à une feuille de style externe.

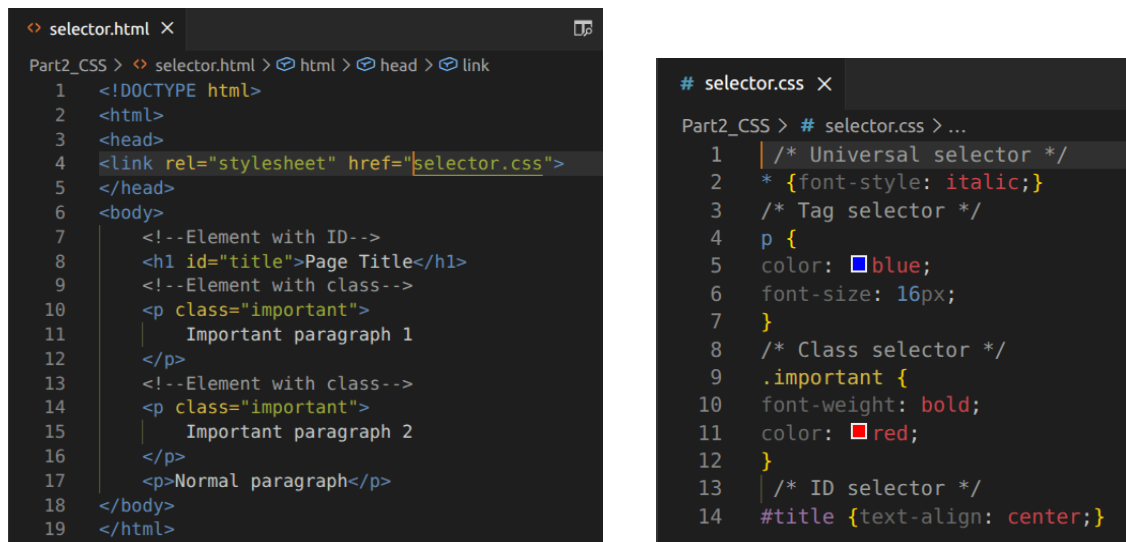
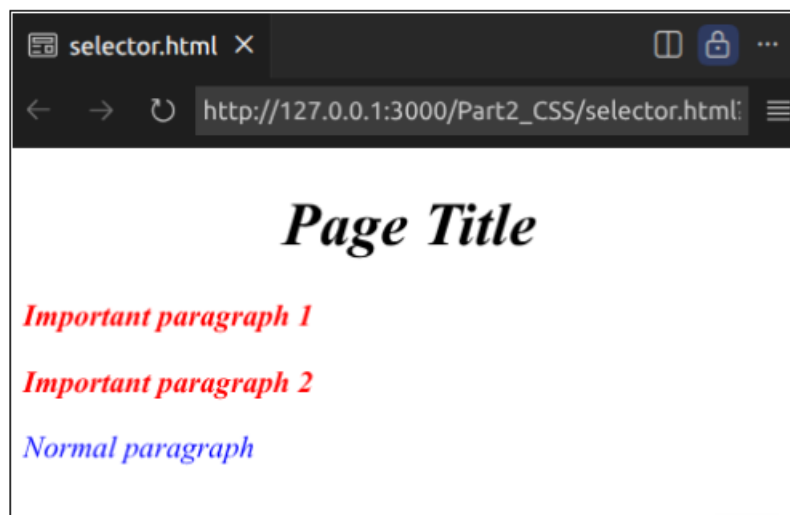


FIGURE 4.4.1 – Selectors



Cet exemple met en évidence l'application des règles CSS à l'aide de différents types de sélecteurs. Le sélecteur universel (*) permet d'appliquer un style italique à l'ensemble des éléments de la page. Le sélecteur de balise (p) est utilisé pour associer une couleur bleue et une taille de police de 16 px à tous les paragraphes. Le sélecteur de classe (.important) cible uniquement les paragraphes marqués comme importants afin de les distinguer visuellement. Enfin, le sélecteur d'identifiant (#title) permet de styliser spécifiquement le titre principal en modifiant son alignement et sa taille de police.

Important

En CSS, les règles sont appliquées selon le principe de la cascade : lorsqu'un même élément est concerné par plusieurs règles, la règle la plus spécifique ou définie plus tard dans la feuille de style est prioritaire, ce qui explique la couleur rouge des paragraphes de la classe "important".

4.4.2 Sélecteurs combinés

Il est également possible de cibler des éléments en fonction de leur position ou de leurs relations au sein de la hiérarchie HTML. Ces sélecteurs permettent d'appliquer des styles avec précision, en s'appuyant uniquement sur la structure du document, sans recourir à des classes ou identifiants spécifiques. Les principaux cas d'usage sont synthétisés dans le tableau suivant.

TABLE 4.1 – Les sélecteurs combinés

Sélecteur	Description	Exemple
child <code>tag > tag</code>	Cible l'élément fils d'un élément donné.	<code>div > p{.....}</code> Cible tous les éléments <code><p></code> dont l'élément parent est un <code><div></code> .
Descendant <code>tag tag</code>	Cible tous les descendants d'un élément donné.	<code>div p{.....}</code> Cible tous les éléments <code><p></code> descendants des éléments <code><div></code> .
next sibling <code>tag+tag</code>	Cible tous les éléments qui suivent immédiatement (frère) un élément donné.	<code>div+p{.....}</code> Cible tous les éléments <code><p></code> placés immédiatement après le <code><div></code> .
Subsequent-sibling <code>tag ~ tag</code>	Cible tous les éléments qui suivent un élément donné et qui partagent le même parent.	<code>div ~ p{.....}</code> Cible tous les <code><p></code> qui suivent un <code><div></code> et qui partagent le même parent

Note

Les sélecteurs combinés présentés dans ce tableau ne constituent pas une liste exhaustive de l'ensemble des sélecteurs; ils correspondent aux sélecteurs de combinaison les plus couramment utilisés.

4.4.3 Sélecteurs d'attributs

Les sélecteurs d'attributs permettent de cibler des éléments en fonction de la présence ou de la valeur de leurs attributs. Par exemple, le sélecteur `input[type="text"]` permet de sélectionner uniquement les champs de saisie de type text et de leur appliquer un style spécifique. Les principaux sélecteurs d'attributs sont synthétisés dans le tableau suivant.

TABLE 4.2 – Les sélecteurs d'attributs

Sélecteur	Description	Exemple
<code>[attribut]</code>	Sélectionne tous les éléments possédant l'attribut spécifié	<code>input[disabled]</code> cible tous les inputs désactivés
<code>[attribut="valeur"]</code>	Sélectionne les éléments dont la valeur de l'attribut correspond exactement à la valeur spécifiée	<code>input[type="text"]</code> cible tous les inputs de type text
<code>[attribut^="valeur"]</code>	Sélectionne les éléments dont la valeur de l'attribut commence par la valeur spécifiée	<code>[id^="section"]</code> cible les éléments dont l'ID commence par la chaîne « section »
<code>[attribut\$="valeur"]</code>	Sélectionne les éléments dont la valeur de l'attribut se termine par la valeur spécifiée	<code>[src\$=".jpg"]</code> cible toutes les images portant l'extension « jpg »
<code>[attribut*="valeur"]</code>	Sélectionne les éléments dont la valeur contient la chaîne spécifiée	<code>[title*="important"]</code> cible tous titres contenant la chaîne « important »

4.4.4 Pseudo-classes

Les pseudo-classes permettent de cibler des éléments en fonction de leur état, de leur position ou de conditions spécifiques, sans nécessiter de classes ou d'identifiants supplémentaires dans le HTML, selon la syntaxe `élément:pseudo-classe`. Les principaux pseudo-classes sont synthétisés dans le tableau suivant.

TABLE 4.3 – Les pseudo-classes

<code>:hover</code>	Cible un élément lors du survol de la souris.	<code>a:hover { color: blue; }</code>
Sélecteur	Description	Exemple
<code>:active</code>	Cible un élément pendant l'action de clic.	<code>button:active { background: lime; }</code>

:visited	Cible un lien qui a été déjà visité	<pre>a:active { background-color:blue; }</pre>
:focus	Cible un élément qui reçoit le focus (clavier ou clic).	<pre>input:focus { border: 2px solid; }</pre>
:first-child	Cible le premier enfant d'un parent.	<pre>li:first-child {font-weight: bold ; }</pre>
:nth-child(n)	Cible le n-ième enfant d'un parent (n = odd, even).	<pre>tr:nth-child(even) { background: WhiteSmoke; }</pre>
:valid	Cible les champs valides d'un formulaire .	<pre>input:valid {border-color: green; }</pre>
:invalid	Cible les champs invalides d'un formulaire.	<pre>input:invalid {border-color: red; }</pre>
:required	Cible les champs obligatoires d'un formulaire.	<pre>input:required{font-style: italic ; }</pre>
:checked	Cible les éléments sélectionnés à option	<pre>input:checked+label{color: green ;}</pre>

4.5 Formatage du texte

Le formatage du texte en CSS regroupe l'ensemble des propriétés permettant de contrôler la typographie, l'espacement, l'alignement et la transformation visuelle des contenus textuels, les principales d'entre elles étant présentées dans le Tableau 3.4.

TABLE 4.5 – Formatage du texte

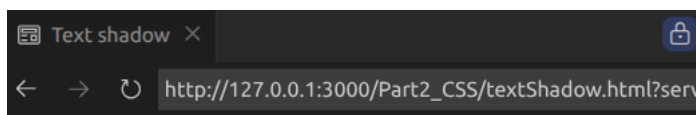
Propriété	Valeurs possibles
font-style	normal,italic
font-weight	normal,bold
font-size	valeur en px, em, cm
font-family	Times New Roman, Courier New, ...
color	red, blue, green, ...
background-color	yellow, red, green, ...
text-decoration	underline, overline, none
text-transform	uppercase, lowercase, capitalize
text-align	left, right, center, justify
word-spacing	valeur en px, em, mm, cm
letter-spacing	valeur en px, em, mm, cm
text-shadow	h-shadow v-shadow blur-radius color

La propriété text-shadow permet d'appliquer des effets d'ombre au texte où h-shadow représente le décalage horizontal, v-shadow pour le décalage vertical, blur-radius pour le rayon de flou et color pour la couleur de l'ombre. L'utilisation est illustrée par l'exemple suivant.

```
Part2_CSS > <> textShadow.html > ...
1 <!DOCTYPE html>
2 <head>
3   <title>Text shadow</title>
4   <link rel="stylesheet" href="textShadow.css">
5 </head>
6 <body>
7   <h1>The text-shadow property</h1>
8   <div id="myDIV">
9     <h1>The text-shadow property</h1>
10 </div>
11 </body>
12 </html>
```

```
Part2_CSS > # textShadow.css > ...
1 #myDIV {
2   height:300px;
3   background-color: #FFFFFF;
4   text-shadow: 5px 5px 5px red;
5 }
```

FIGURE 4.5.1 – Border style



The text-shadow property

The text-shadow property

4.6 Listes

Le CSS dispose offre la possibilité de styliser les listes HTML grâce aux propriétés synthétisées dans le Tableau 3.5

TABLE 4.6 – Propriétés de mise en forme d’une liste

Propriété	Valeurs possibles	Exemple
list-style-type	disc, circle, square upper-roman, lower-roman, lower-alpha, upper-alpha, decimal	<pre>ul {list-style-type: square;}</pre>
list-style-image	url('img_name.extension')	<pre>list-style-image: url(image.gif);</pre>
list-style-position	outside, inside	<pre>list-style-position: outside;</pre>
list-style	Times New Roman, Courier New, ...	<pre>list-style-image: url(image.gif);</pre>

- list-style-type : définit le type de marqueur des éléments de liste
- list-style-image : définit une image comme marqueur des éléments de liste
- list-style-position : définit la position des marqueurs des éléments de liste
- list-style : une propriété abrégée regroupant les propriétés ci-dessus

4.7 Tables

Le CSS permet de mettre en forme les tableaux HTML à l’aide de plusieurs propriétés destinées à améliorer leur apparence, dont les principales sont synthétisées dans le Tableau 3.6.

TABLE 4.7 – Propriétés de mise en forme d'un tableau

Propriété	Valeurs possibles	Exemple
caption-side	top,bottom	<pre>caption{caption-side: bottom;}</pre>
border-collapse	separate, collapse	<pre>table { border-collapse: collapse; }</pre>
border-spacing	valeur en pixel	<pre>table { border-collapse: separate; border-spacing: 15px; }</pre>
empty-cells	show, hide	<pre>table{empty-cells: hide;}</pre>

- caption-side : définit la position de la légende d'un tableau.
- border-collapse : définit le mode d'affichage des bordures d'un tableau.
- border-spacing : définit l'espacement entre les bordures des cellules d'un tableau généralement exprimé en pixels.
- empty-cells : définit si les bordures des cellules vides d'un tableau doivent être affichées ou non.

Note

La propriété "empty-cells" n'a aucun effet lorsque la valeur de "border-collapse" est définie sur "collapse".

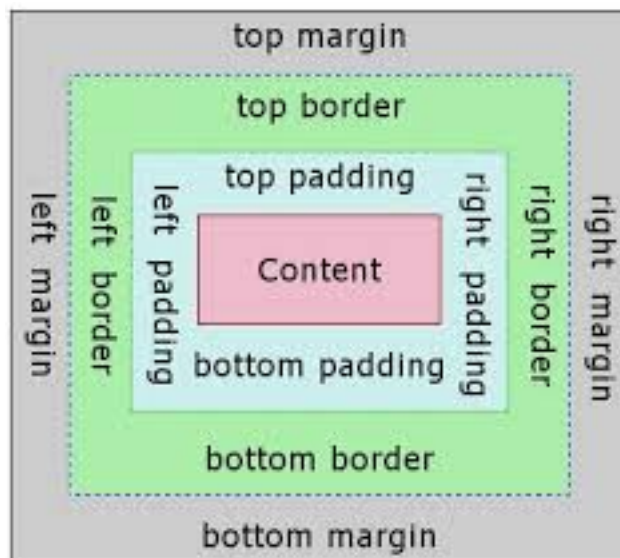
4.8 Modèle de boîtes

Le modèle de boîtes (ou box model) en CSS constitue le fondement du calcul des dimensions et de la disposition des éléments HTML dans une page web. Chaque élément y est traité comme une boîte subdivisée en quatre zones distinctes : content (contenu), padding (marge intérieure), border (bordure) et le margin (marge extérieure).

- Content : le contenu de la boîte, où apparaissent le texte et les images
- padding : crée un espace transparent autour du contenu.
- border : une bordure qui entoure le padding et le content
- margin : crée un espace transparent à l'extérieur du border.

L'illustration ci-dessous permet de visualiser de manière explicite la structure du modèle de boîtes en CSS.

FIGURE 4.8.1 – Box model



Note

Les propriétés `border`, `margin` et `padding` constituent des formes abrégées pour désigner simultanément leurs variantes directionnelles (`border-left`, `border-right`, `border-top`, `border-bottom`; de même pour `margin-` et `padding-`), permettant une déclaration concise des styles.

L'exemple ci-après illustre l'utilisation des propriétés `border`, `padding` et `margin`.

```

boxModel.html
Part2_CSS > boxModel.html > html
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <link rel="stylesheet" href="boxModel.css">
5  </head>
6  <body>
7
8  <h2>Exemple de modèle de boîte</h2>
9  <div>
10   <ul>
11     <li>Content : le contenu de la boîte,
12     où apparaissent le texte et les images</li>
13     <li>padding: crée un espace transparent
14     autour du contenu. </li>
15     <li>border : une bordure qui entoure le
16     padding et le content </li>
17     <li> margin : crée un espace transparent à
18     l'extérieur du border.</li>
19   </ul>
20 </div>
21 </body>
22 </html>

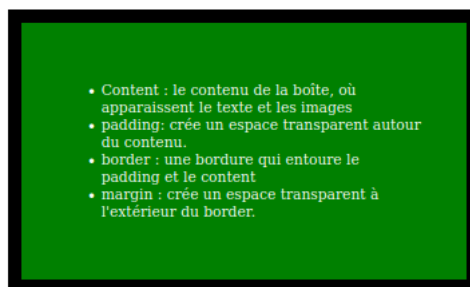
```

```

# boxModel.css
Part2_CSS > # boxModel.css > div
1  div {
2    background-color: green;
3    color: white;
4    width: 400px;
5    border: 15px solid black;
6    padding: 50px;
7    margin: 20px;
8  }
9

```

FIGURE 4.8.2 – Exemple d'un modèle de boîte

Exemple de modèle de boîte

Les différents styles de bordures (solid, dotted, dashed, double, ridge, groove, inset et outset) sont illustrés par l'exemple suivant, qui présente le code et son rendu visuel.

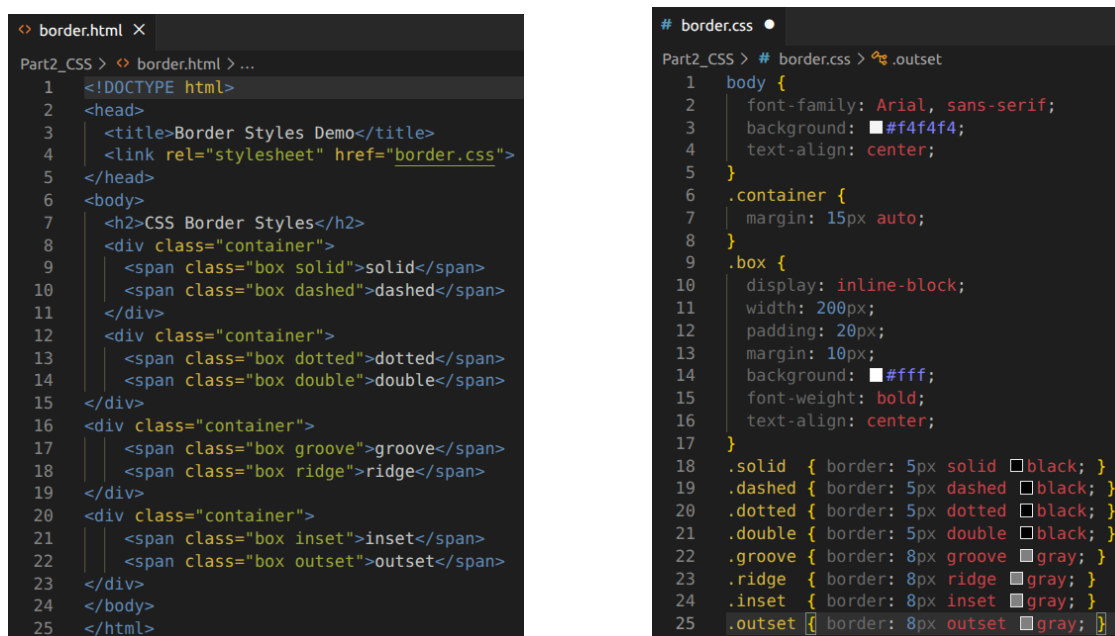
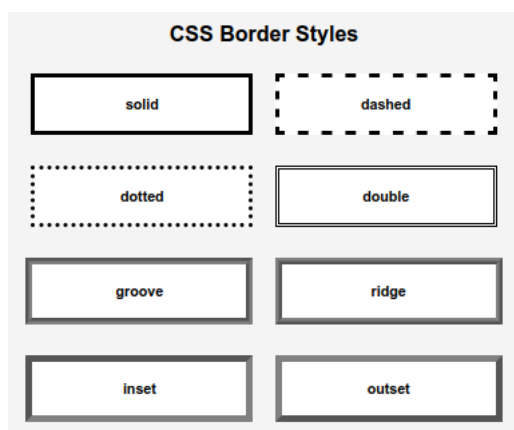


FIGURE 4.8.3 – Border style



Note

La propriété `box-shadow` crée des ombres portées autour des boîtes CSS, suivant une syntaxe identique à `text-shadow`.

4.9 Positionnement des éléments

4.9.1 Display

Tous les éléments HTML se voient attribuer une valeur de propriété `display` qui est soit `inline`, soit `block`. Les éléments `inline` s'affichent horizontalement dans le navigateur, tandis que les éléments `block` s'affichent verticalement dans le navigateur. Le tableau suivant récapitule les principales balises classées en `inline` ou `inblock` par défaut.

TABLE 4.8 – Les éléments `inline` et `inblock`

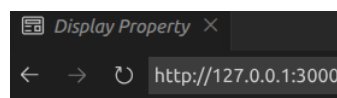
Inline	inblock
<code></code> , <code><a></code> , <code><sub></code> et <code><sup></code> , <code><input></code> et <code></code>	<code><div></code> , <code><h1></code> , <code><h2></code> ... <code><h6></code> , <code><p></code> , <code></code> , <code></code> , <code></code> , <code><table></code> , <code><tr></code> , <code><td></code>

Le comportement par défaut d'un élément HTML peut être modifié grâce à la propriété `display`. L'exemple suivant illustre la transformation d'un lien `<a>` (`inline` par défaut) en élément `block`.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Display Property</title>
5      <style>
6          a { display: block;}
7      </style>
8  </head>
9  <body>
10     <h4>Changing default display value</h4>
11     <h4>&lt;a href&gt;to in block</h4>
12     <a href="#">link 1</a>
13     <a href="#">link 2</a>
14     <a href="#">link 3</a>
15 </body>
16 </html>

```



Changing default display value

<a href=to in block

[link 1](#)
[link 2](#)
[link 3](#)

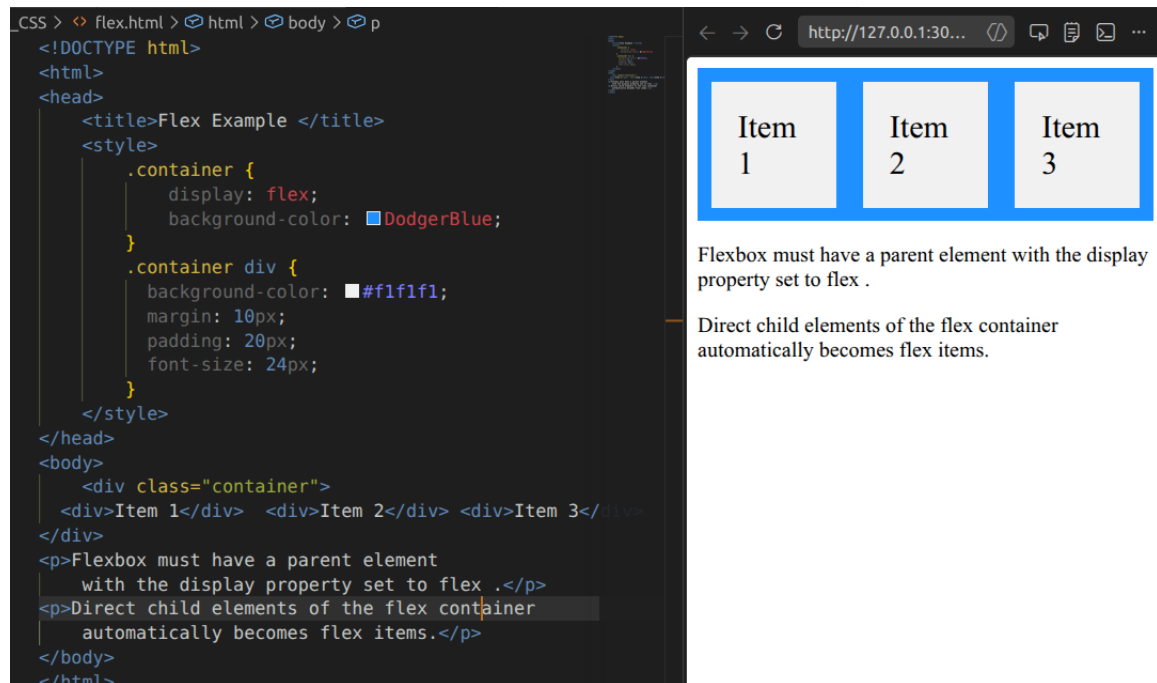
4.9.2 Flex and grid

Les propriétés `display : inline` et `display : block`, bien qu'utiles pour contrôler le flux d'affichage, montrent leurs limites pour des mises en page exigeant un centrage vertical, des hauteurs égales entre colonnes ou une répartition flexible des espaces. C'est pour remédier à ces limites que *flexbox* et *Grid* ont été développés.

- *FlexBox* (*Flexible Box Layout module*) est une méthode d'agencement unidirectionnel selon l'axe horizontal ou vertical. Les éléments d'un conteneur se dilatent ou se

rétractent pour occuper l'espace disponible de manière flexible et responsive. L'agencement en ligne se définit par *display : flex* ; l'agencement en colonne nécessite l'ajout de *flex-direction : column*, comme présenté dans l'exemple 3.9.1.

FIGURE 4.9.1 – Flex example



- *Grid* est une méthode d'agencement bidimensionnelle (horizontal et vertical) qui permet de créer des mises en page avec des lignes et des colonnes. On définit un conteneur avec *display : grid*, puis on précise la structure de la grille à l'aide de *grid-template-columns* (nombre et taille des colonnes) et *grid-template-areas* (nombre de lignes et zones où placer les éléments dans la grille). L'exemple 3.10 illustre également cette mise en page.

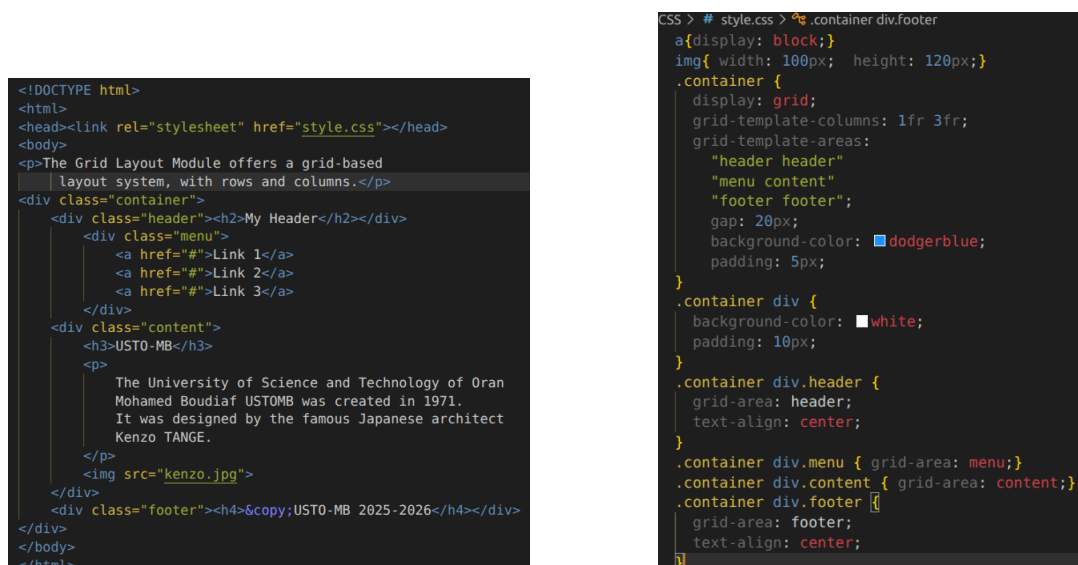
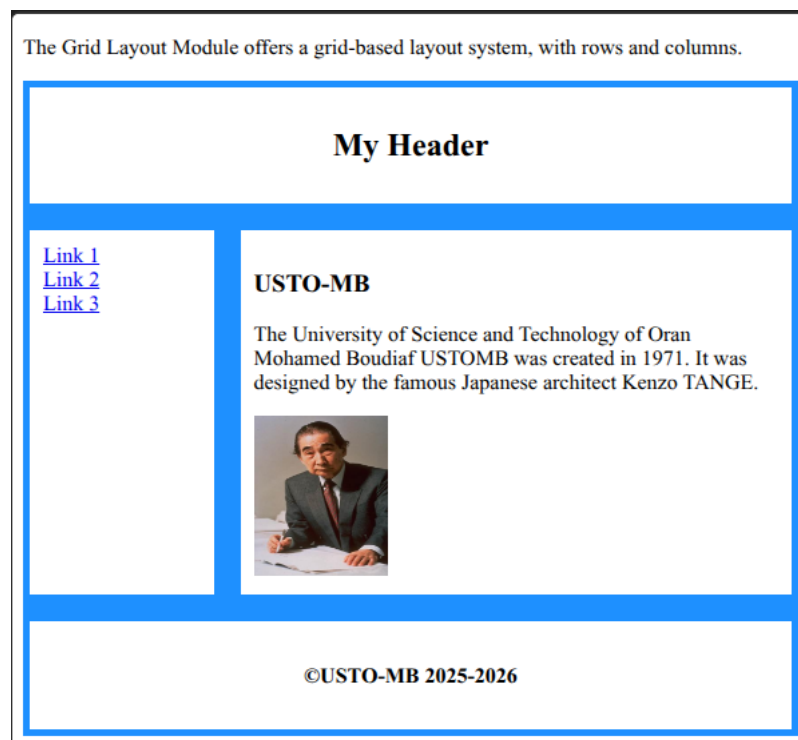


FIGURE 4.9.2 – grid example

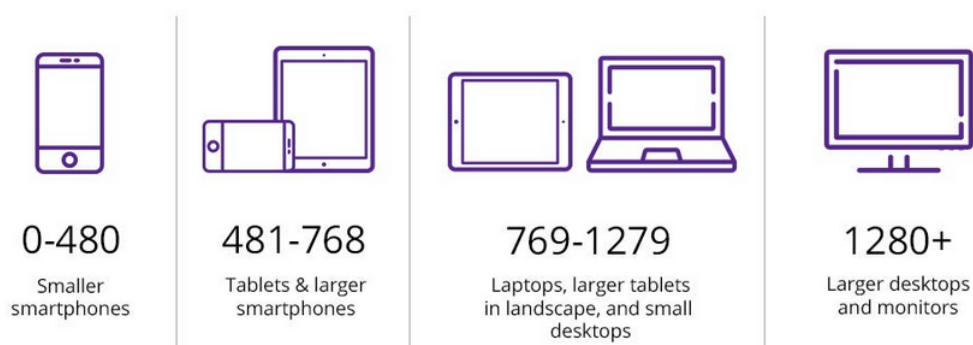


4.10 Responsive design and media queries

Le responsive design désigne l'ensemble des techniques permettant de concevoir un site web capable de s'adapter dynamiquement à la taille et aux caractéristiques de l'écran des utilisateurs. À l'heure actuelle, trois stratégies principales s'offrent aux développeurs pour assurer l'adaptabilité d'un site à divers terminaux :

- La création de sites dédiés pour chaque type de terminal (un site spécifique pour les mobiles, un autre pour les tablettes, un pour les ordinateurs, etc.).
- Le développement d'applications mobiles natives (pour Android, iOS, etc.).
- La conception d'une version responsive du site, intégrant des mécanismes d'adaptation automatique.

FIGURE 4.10.1 – responsive design



4.11 Exercices

Exercice 4.11.1

On donne ci-après le code HTML "*verbe.html*" ainsi que le rendu visuel attendu . Rédigez la feuille de style "*verbe.css*" correspondante, puis liez-la au document HTML.

IRREGULAR VERBS			
Base	Past	Participle	Image
go	went	gone	LET'S GO 
eat	ate	eaten	
be	was / were	been	
Liste non exhaustive			

```

1 <!DOCTYPE html>
2 <html lang="fr">
3 <head>
4   <title>CSS</title>
5   <link rel="stylesheet" type="text/css" href="verbe.css">
6 </head>
7 <body>
8   <table>
9     <caption>Irregular verbs </caption>
10    <tr><th>Base</th> <th>Past</th> <th>Participle</th> <th>Image</th>
11    <tr>
12      <td>go</td>
13      <td>went</td>
14      <td>gone</td>
15      <td>
16        <a href="https://irregularverbs.org/"></a>
18      </td>
19    </tr>
20    <tr>
    <td>eat</td>
    <td>ate</td>

```

```

21     <td>eaten</td>
22     <td>
23         <a href="https://irregularverbs.org/"></a>        </td>
25     </tr>
26     <tr>
27         <td rowspan="2">be</td>
28         <td>was / were</td>
29         <td>been</td>
30         <td>
31             <a href="https://irregularverbs.org/"></a>
33         </td>
34     </tr>
35     <tr>
36         <td colspan="3">Liste non exhaustive</td> </tr>
37 </table>
38 </body>
39 </html>

```

Exercice 4.11.2

Donner le code CSS permettant de styliser le formulaire (rendu ci-après) donné par le code HTML suivant

NB : L'utilisation des attributs "id" et "class" n'est autorisée que pour les deux champs "authorName" et "documentTitle"

Author Name

Document Title

Author Position:

☐ First Author ☐ Second Author

After Form Submit:

☐ Show thank you message ☐ Save form submission

Validate

```

1 <html>
2 <head>
3   <title>CSS</title>
4   <link rel="stylesheet" type="text/css" href="form.css">

```

```
5 </head>
6 <body>
7   <form id="submissionForm">
8     <label for ="authorName">Author Name </label>
9     <input type="text" id="authorName" name="authorName" required />
10    <label for="documentTitle">Document Title </label>
11    <input type="text" id="documentTitle" name="documentTitle"
      required />
12    <fieldset>
13      <legend>Author Position</legend>
14      <input type ="radio" name="authorPosition" value="first"/> <
        span>First Author</span>          <input type ="radio"
        name="authorPosition" value="second" /> <span>Second Author
        </span>
15    </fieldset>
16    <fieldset>
17      <legend>After Form Submit</legend>
18      <input type="checkbox" name="afterSubmit[]" value="thanks"/><span>
        Show thank you message</span>          <input type="checkbox" name
        ="afterSubmit[]" value="save"/> <span>Save form submission</span
        >
19    </fieldset>
20    <input type="submit" value="Validate"/>
21  </form>
22 </body>
23 </html>
```

Javascript

Objectifs du chapitre

À la fin de ce chapitre, vous serez capable d'ajouter de l'interactivité à une page web en manipulant son contenu et en réagissant aux actions de l'utilisateur.

5.1 Introduction

JavaScript est un langage de script interprété directement par le navigateur web. Il constitue, avec HTML et CSS, l'un des langages standards du développement web. Toutefois, contrairement à HTML qui structure le contenu et à CSS qui en assure la présentation, JavaScript permet d'en assurer une interactivité avec l'utilisateur.

5.2 Les différentes méthodes d'intégration d'un script

Nous distinguons trois méthodes principales pour insérer un script dans une page HTML : interne, externe ou en ligne.

5.2.1 Insertion interne

Le script est placé directement dans la page HTML à l'intérieur d'une balise `<script>` comme l'illustre l'exemple ci-dessous.

```
1 <!DOCTYPE html>  
2 <html>
```

```
3 <head>
4   <title>Internal JavaScript Example</title>
5 </head>
6 <body>
7   <h1>My Web Page</h1>
8   <script> alert("Hello!"); </script>
9 </body>
10</html>
```

5.2.2 Insertion externe

Le script est rédigé dans un fichier séparé portant l'extension `.js`, puis lié à la page HTML via l'attribut `src` de la balise `<script>`, comme l'illustre l'exemple ci-dessous.

```
1 <!-- HTML file : index.html -->
2 <!DOCTYPE html>
3 <html>
4   <head>
5     <title>External JavaScript Example</title>
6   </head>
7   <body>
8     <h1>My Web Page</h1>
9     <script src="myScript.js"></script>
10  </body>
11</html>
```

```
1 // Fichier myScript.js
2 alert("Bonjour !");
```

5.2.3 Insertion en ligne (inline)

Le script est placé directement dans une balise HTML.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Inline JavaScript Example</title>
5   </head>
6   <body>
7     <h1>My Web Page</h1>
8     <button onclick="alert('Hello!')">Click me</button>
9   </body>
10</html>
```

Comme le montre l'exemple, le script est inséré directement via l'attribut événementiel `onclick` au sein même de la balise HTML.

Note

L'insertion d'un script JavaScript en ligne est à éviter car elle mélange structure et comportement. En revanche, l'insertion externe est vivement recommandée du fait qu'elle favorise une meilleure organisation du code et permet une réutilisabilité parfaite sur plusieurs pages.

5.3 Syntaxe de base

Du point de vue de sa syntaxe, JavaScript s'inspire du langage C. Il s'agit donc d'un langage de programmation structuré.

5.3.1 Typage faible et dynamique

En JavaScript, le type d'une variable est déterminé par la valeur qui lui est affectée, ce qui explique la qualification de ce langage comme faiblement typé. Contrairement au langage C où chaque variable possède un type fixe (`int`, `float`, `char`), une variable y peut changer de type en cours d'exécution, comme l'illustre l'exemple ci-dessous.

```
1 let valeur = 42;           // type : number
2 valeur = "Bonjour";       // type : string
3 valeur = true;            // type : boolean
```

JavaScript permet également de déclarer des constantes à l'aide du mot-clé `const` comme le montre l'exemple ci-dessous. Une constante, comme son nom l'indique, ne peut pas être modifiée après son initialisation.

```
1 const PI = 3.14159;
```

5.3.2 Déclaration et portée des variables

La portée d'une variable dépend à la fois du mot-clé utilisé pour la déclarer et de l'endroit où elle est définie dans le code.

- **var** : une variable déclarée avec le mot-clé `var` est accessible dans toute la fonction où elle est définie. Si elle est déclarée en dehors de toute fonction, elle est accessible dans l'ensemble du script.
- **let** : une variable déclarée avec le mot-clé `let` n'est accessible que dans le bloc où elle est définie (par exemple, à l'intérieur d'une boucle `for` ou d'un bloc `if`). On parle de portée de bloc.
- **const** : une constante déclarée avec `const` ne peut pas être modifiée après son initialisation. Comme `let`, sa portée est limitée au bloc.

- Sans mot-clé : si l'on affecte une valeur à une variable sans utiliser `var`, `let` ou `const`, celle-ci est automatiquement ajoutée à l'espace global.

Les différences de portée mentionnées ci-dessus sont illustrées par l'exemple suivant :

```
1 let x = 5;    // portée de bloc
2 var y = 3;    // portée de fonction (ou globale selon le contexte)
3 z = 1;        // variable globale implicite
4 const PI = 3.14; // constante
```

5.3.3 Les fonctions

À l'instar du langage C, une fonction JavaScript se déclare avec le mot-clé **function**. Elle peut recevoir des paramètres et retourner une valeur, comme l'illustre l'exemple ci-dessous.

```
1 function add(a, b) {
2     return a + b;
3 }
4 let somme = add(5, 3);
```

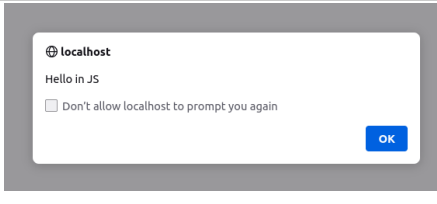
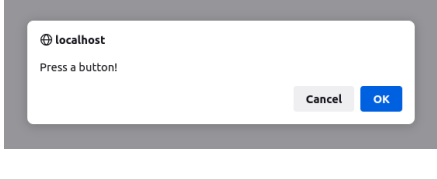
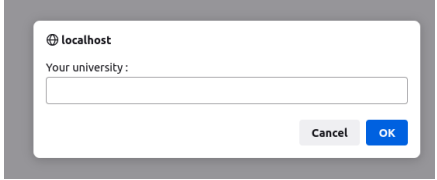
JavaScript permet également de définir des fonctions anonymes, c'est-à-dire des fonctions sans nom. Elles sont fréquemment utilisées pour gérer les événements et peuvent être stockées dans une variable ou passées en argument à d'autres fonctions.

```
1 // Fonction classique (comme en C)
2 function add(a, b) {
3     return a + b;
4 }
5 // Fonction anonyme (spécificité JavaScript)
6 let carre = function(x) {
7     return x * x;
8 };
9 console.log(carre(5));
```

5.4 Boîtes de dialogue

JavaScript met à disposition trois fonctions permettant d'interagir avec l'utilisateur par l'intermédiaire de fenêtres de dialogue (connues aussi sous le nom de pop-up) : `alert()`, `confirm()` et `prompt()`. Le Tableau ci-dessous illustre l'usage de ces trois fonctions.

TABLE 5.1 – alert(), confirm() et prompt()

Fonction	Rôle	Exemple	Rendu
alert()	Affiche un message d'information	<pre> 1 <script> 2 alert("Hello in 3 JS "); </script> </pre>	
confirm()	Demande une confirmation	<pre> 1 <script> 2 confirm("Press a 3 button"); </script> </pre>	
prompt()	Demande une saisie à l'utilisateur	<pre> 1 <script> 2 prompt("Your 3 university :"); </script> </pre>	

Comme illustré par le Tableau 4.1, la fonction `alert()` affiche une fenêtre de dialogue contenant le message transmis en paramètre ainsi qu'un unique bouton "OK". La fonction `confirm()` présente quant à elle une fenêtre comportant le message spécifié et deux boutons, "OK" et "Annuler"; elle retourne une valeur booléenne : `true` si l'utilisateur valide, `false` dans le cas contraire. Enfin, la fonction `prompt()` produit une fenêtre invitant l'utilisateur à saisir une valeur : elle retourne la chaîne de caractères entrée si l'utilisateur clique sur "OK" et `null` s'il clique sur "Annuler".

5.5 Les événements

Un événement désigne toute action initiée par l'utilisateur, telle qu'un clic sur un élément, une saisie dans un champ, le déplacement du pointeur ou la soumission d'un formulaire. JavaScript offre la capacité de détecter ces événements et d'y associer l'exécution d'un script, permettant ainsi de rendre la page interactive. Le principe général de ce mécanisme est illustré par la figure ci-dessous.

Comme l'illustre la Figure 4.5.1, lorsqu'un événement `onClick` se produit, le navigateur déclenche l'exécution de la fonction `btnClick()` qui lui est associée affichant ainsi une fenêtre `alert()` à l'attention de l'utilisateur.

L'événement `onClick` a été choisi dans cet exemple car il s'agit généralement de l'événement le plus utilisé en pratique. En revanche, de nombreux autres événements existent dont les principaux sont répertoriés ci-après.

FIGURE 5.5.1 – Du clic utilisateur à la modification de la page

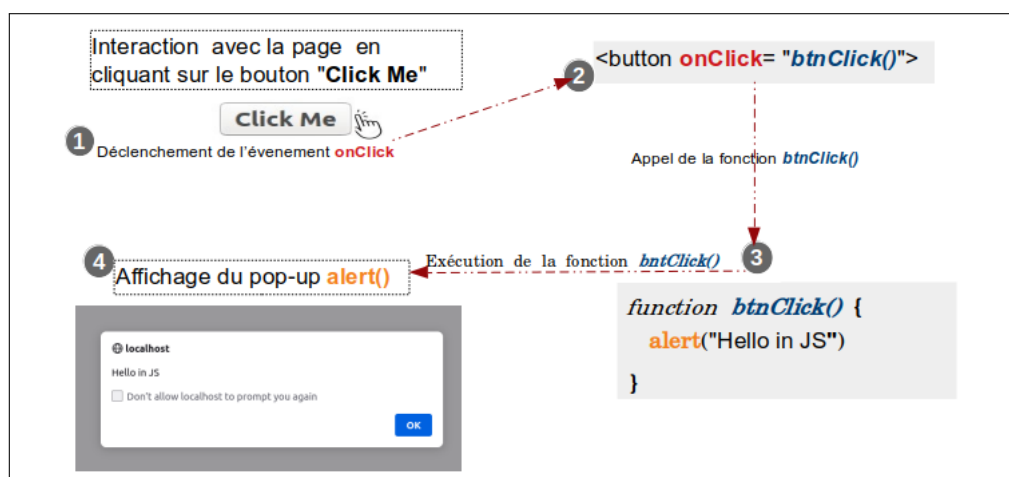


TABLE 5.2 – Liste des principaux événements

Événement	Se produit lorsque
load	Le navigateur a fini le chargement de la page
unload	Le navigateur quitte la page
click	L'utilisateur clique sur un élément
dblclick	L'utilisateur double-clique sur un élément
mouseover	Le curseur de la souris survole un élément
mouseout	Le curseur de la souris quitte un élément
change	L'utilisateur modifie la valeur d'un champ (input, select, etc.)
keydown	Une touche est appuyée
keyup	Une touche est relâchée
focus	L'élément reçoit le focus (clic ou tabulation)
blur	L'élément perd le focus
submit	Le formulaire est soumis (généralement par un clic sur un <code>input</code> de <code>type="submit"</code>)
reset	Le formulaire est réinitialisé (généralement par un clic sur un <code>input</code> de <code>type="reset"</code>)

5.6 Manipulation par le DOM

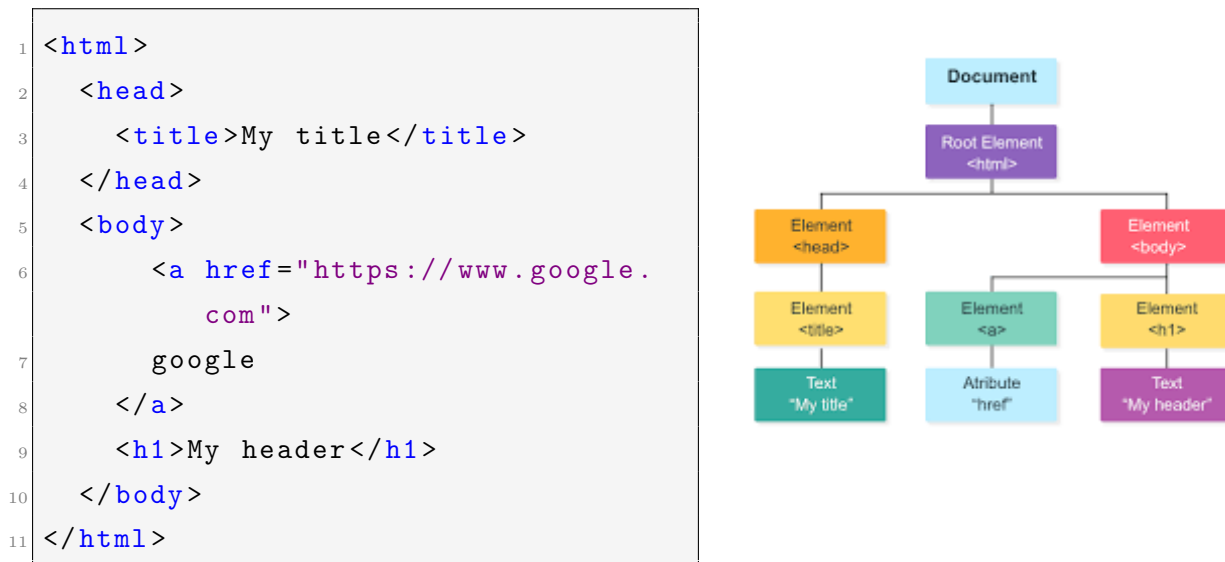
Pour qu'une page web soit interactive, JavaScript doit être capable d'accéder aux éléments HTML et de les modifier. C'est précisément le rôle du DOM (Document Object Model), une interface de programmation (API) standardisée par le W3C.

Le DOM représente le document HTML sous forme d'une structure arborescente, où chaque élément devient un noeud manipulable par JavaScript. Grâce à cette API, il est

possible d'ajouter, supprimer ou modifier dynamiquement le contenu et l'apparence d'une page, sans avoir à la recharger.

5.6.1 Structure arborescente du DOM

Chaque page HTML est modélisée sous forme d'un arbre dont son noeud racine correspond à `document` comme illustré ci-dessous.



5.6.2 Accès aux éléments HTML

L'accès aux éléments HTML d'une page nécessite leur sélection préalable. Pour ce faire, le DOM met à disposition plusieurs méthodes de sélection, dont les principales sont synthétisées dans le tableau ci-dessous.

TABLE 5.3 – Liste des principales méthodes DOM

Méthode	Description	Retour
<code>getElementById(id)</code>	Sélectionne l'élément dont l'id correspond au paramètre.	Un seul élément
<code>getElementsByTagName(tag)</code>	Sélectionne les éléments dont le nom de la balise correspond au paramètre.	Collection HTML
<code>getElementsByClassName(class)</code>	Sélectionne les éléments dont la valeur de l'attribut <code>class</code> correspond au paramètre.	Collection HTML
<code>getElementsByName(name)</code>	Sélectionne les éléments dont la valeur de l'attribut <code>name</code> correspond au paramètre.	Collection HTML
<code>querySelector(selector)</code>	Sélectionne le premier élément correspondant au sélecteur CSS.	Un seul élément
<code>querySelectorAll(selector)</code>	Sélectionne tous les éléments correspondant au sélecteur CSS.	NodeList

 Note

Le DOM est construit selon le principe orienté objet. L'objet `document` représente la page web et contient toutes les méthodes de sélection. Elles doivent donc toujours s'écrire sous la forme `document.methodName()`.

5.6.3 Modification du contenu

Une fois un élément sélectionné, on peut lire ou modifier son contenu grâce aux propriétés offertes par l'API DOM dont les principales sont synthétisées dans le Tableau ci-dessous :

TABLE 5.4 – Propriétés

Propriété	Utilisation	Exemple
<code>innerHTML</code>	Lit ou modifie le contenu d'un élément.	<code>element.innerHTML = "Texte"</code>
<code>textContent</code>	Lit ou modifie le texte brut d'un élément.	<code>element.textContent = "Texte simple"</code>
<code>value</code>	Lit ou modifie la valeur d'un champ de formulaire	<code>input.value = "modified value"</code>

 Note

Toutes ces propriétés ne s'appliquent pas à tous les types d'éléments. Les propriétés `innerHTML` et `textContent` concernent les éléments conteneurs (comme `<div>`, `<p>`, `<span>`, `<section>`), c'est-à-dire ceux qui possèdent une balise ouvrante et une balise fermante avec du contenu à l'intérieur.

En revanche, pour les éléments de formulaire (`<input>`, `<textarea>`, `<select>`), la propriété à utiliser est `.value`, qui permet de lire ou de modifier la valeur saisie ou sélectionnée par l'utilisateur.

Un exemple illustrant l'utilisation des propriétés `innerHTML` et `.value` est donné ci-dessous :

```

1 <html>
2   <head>
3     <title>Exemple DOM</title>
4   </head>
5   <body>
6     <div id="zone">Contenu initial</div>

```

```

7      <input type="text" id="nom" value="Anonyme">
8      <script>
9          document.getElementById("zone").innerHTML = "<i>Contenu modifié
              </i>";
10         document.getElementById("nom").value = "Mohamed";
11     </script>
12 </body>
13 </html>

```

5.6.4 Ajout et suppression d'éléments

Le DOM permet de créer et de supprimer dynamiquement des éléments HTML, dont les principales méthodes offertes sont répertoriées dans le Tableau 4.5.

TABLE 5.5 – Les méthodes d'ajout et de suppression

Méthode	Description
<code>createElement("tag")</code>	Crée un nouvel élément <code>tag</code>
<code>createTextNode("text")</code>	Crée un noeud texte avec comme valeur <code>text</code>
<code>appendChild(elt)</code>	Ajoute l'élément <code>elt</code> comme dernier enfant à l'élément courant
<code>insertBefore(elt_1, elt_2)</code>	Insère l'élément <code>elt_1</code> avant l'élément <code>elt_2</code>
<code>removeChild(elt)</code>	Supprime l'élément <code>elt</code> , passé en paramètre, de son parent.
<code>remove()</code>	Supprime l'élément courant.

L'exemple ci-dessous illustre la création d'un nouvel élément `<p>` et son insertion dynamique au `<body>` existante à l'aide des méthodes `createElement()` et `appendChild()`.

```

1 <!DOCTYPE html>
2 <html>
3   <body>
4     <h2>The createElement() Method</h2>
5     <p>Create a p element with some text:<
6       /p>
7     <button onclick="create();">Create</
8       button>
9     <script>
10      function create(){
11        const para = document.createElement(
12          "p");
13        para.innerText = "This is a
14          paragraph.";
15        document.body.appendChild(para);
16      }
17    </script>
18  </body>
19 </html>

```

The createElement() Method

Create a p element with some text:

Create

This is a paragraph.

5.6.5 Modification des attributs et du style

Lorsqu'un élément est sélectionné, le DOM permet d'agir sur ses attributs, son style ou ses classes grâce à des méthodes et propriétés dédiées, dont les principales sont présentées dans le Tableau suivant.

Note

Les opérations présentées dans le tableau ci-dessous supposent que l'élément cible a été sélectionné au préalable. Sans cette étape, le code provoquera une erreur JavaScript.

TABLE 5.6 – Modification des attributs, du style et des classes

Opération	Syntaxe	Exemple
Lire un attribut	getAttribute("att")	<pre>let src = img.getAttribute("src")</pre>

Modifier un attribut	setAttribute("att_name", "att_val")	link.setAttribute("href", "https://exemple.com")
Supprimer un attribut	removeAttribute("att_name")	img.removeAttribute("alt")
Modifier le style	element.style.propriete = "val"	title.style.color = "red"
Ajouter une classe	classList.add("classe")	field.classList.add("error")
Supprimer une classe	classList.remove("classe")	field.classList.remove("error")

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <style>
5       .erreur { border: 2px solid red; }
6       .succes { border: 2px solid green; }
7     </style>
8   </head>
9   <body>
10    <h2 id="titre">Validation</h2>
11    <input type="text" id="nom" placeholder="Votre nom">
12    <button id="valider">Valider</button>
13    <script>
14      let titre = document.getElementById("titre");
15      let champ = document.getElementById("nom");
16      let btn = document.getElementById("valider");
17      btn.onclick = function() {
18        if (champ.value == "") {

```

```
19     champ.setAttribute("placeholder", "Obligatoire");
20     titre.style.color = "red";
21     champ.classList.add("erreur");
22     champ.classList.remove("succes");
23     } else {
24     champ.removeAttribute("placeholder");
25     titre.style.color = "green";
26     champ.classList.add("succes");
27     champ.classList.remove("erreur");
28     }
29 };
30 </script>
31 </body>
32 </html>
```

5.6.6 L'écoute d'événements avec addEventListener()

`addEventListener()` est une méthode du DOM qui permet d'exécuter une fonction lorsqu'un événement précis se produit sur un élément HTML (clic, saisie, soumission, etc.). Comme l'illustre l'exemple ci-dessous, l'événement `click` est attaché à l'élément identifié par `id="paragraph"` : au clic, une fonction anonyme modifie le contenu de l'élément `id="demo"` via la propriété `innerHTML`

```
1 <body>
2   <h2>The addEventListener() Method</h2>
3   <p>Attach a click event to a paragraph:</p>
4   <p id="paragraph">Try to click me</p>
5   <p id="demo"></p>
6   <script>
7     const element = document.getElementById("paragraph");
8     element.addEventListener("click", function() {
9       document.getElementById("demo").innerHTML = "Hello World";
10    });
11  </script>
12 </body>
```

5.7 Contrôle des formulaires

HTML5 offre des attributs de validation natifs (`required`, `min`, `max`, `type="email"`, `pattern`). L'API DOM et la gestion d'événements en JavaScript (`blur`, `click`, `focus`, ...etc) permettent d'aller plus loin. Ensemble, ils forment une base solide pour contrôler les données saisies. Pour bloquer l'envoi d'un formulaire en cas d'erreur, la méthode `preventDefault()`

joue ici un rôle essentiel.

FIGURE 5.7.1 – Contrôle des formulaires

L'exemple illustré par la Figure 4.7.1 a pour but de déterminer si un étudiant est admissible dans une filière donnée. L'utilisateur renseigne sa moyenne (comprise entre 0 et 20) et sélectionne une filière parmi : Informatique (seuil ≥ 13), Mathématiques (seuil ≥ 12) ou Biologie (seuil $\geq 10,5$). L'accès est autorisé si la moyenne atteint ou dépasse le seuil requis.

```

1 <body>
2 <h1>Vérifier l'éligibilité à une
   filière</h1>
3 <form id="orientationForm"
   method="POST">
4   <div>
5     <label for="moyenne">
6       Moyenne (0 à 20) :
7     </label>
8     <input type="number" id="
       moyenne" name="
       moyenne" min="0" max="20
       " required>
9   </div>
10  <div>
11    <label for="filiere">
12      Filière souhaitée :
13    </label>
14    <select id="filiere" name="
       filiere" required>
15      <option value="">-- Choisissez
        une filière --</option>
16      <option value="info">
        Informatique (min 13)</option>
17      <option value="math">
        Mathématiques (min 12)</
        option>
18      <option value="biologie">
        Biologie (min 10.5)</option>
19    </select>
20  </div>
21  <div>
22    <input type="button" value="
       verif" onclick="verifier();"
       >
23  </div>
24 </form>
25 <div id="resultat"></div>
26 </body>

```

Les attributs HTML5 `required`, `min` et `max` ont été utilisés pour valider les champs moyenne et filière, tandis que la fonction JavaScript `verifier()` valide l'admissibilité en vérifiant si la moyenne saisie atteint ou dépasse le seuil exigé par la filière choisie.

```

1 function verifier() {
2     let moyenne = parseFloat(
3         document.getElementById("
4             moyenne").value);
5     let filiere = document.
6         getElementById("filiere").
7         value;
8     let resultatDiv = document.
9         getElementById("resultat");
10    let seuils = ['info', 'math',
11        'biologie'];
12    let minimas = [13, 12, 10.5];
13    let intitules = ['Informatique
14        ', 'Mathématiques', '
15        Biologie'];
16    let indice = -1;
17    for (let i = 0; i < 3; i++) {
18        if (seuils[i] === filiere) {
19            indice = i; }
20    }
21    if (isNaN(moyenne) || indice ===
22        -1) {

```

```

13    resultatDiv.innerHTML = "<p
14        style='color:orange'>
15        Veuillez saisir une moyenne
16        valide et choisir une filière
17        . </p>";
18    }
19    else if (moyenne < minimas[
20        indice]) {
21        resultatDiv.innerHTML = "<
22            p style='color:red'>
23            Désolé, votre moyenne
24            de "
25            + moyenne + " est insuffisante
26            pour " + intitules[indice] + "
27            (minimum requis : " +
28            minimas[indice] + ").</p>";
29    } else {
30        resultatDiv.innerHTML = "<p
31            style='color:green'>
32            Félicitations, vous êtes
33            admissible en " + intitules[
34            indice] + " avec une
35            moyenne de " + moyenne + "
36            .</p>";
37    }
38    }

```

Exercice 5.7.1

On donne ci-dessous le code HTML d'un formulaire d'authentification ainsi que les règles de validation à respecter.

```

1 <!DOCTYPE html>
2 <html>
3     <head>
4         <title>Authentification</title>
5     </head>
6     <body>
7         <h2>Formulaire d'accès</h2>
8         <form method="GET" action="#">
9             <div>
10                 <label for="user">Nom d'utilisateur :</label>

```

```
11     <input type="text" id="user" name="user">
12 </div>
13     <div>
14         <label for="pass">Mot de passe :</label>
15         <input type="password" id="pass" name="pass">
16     </div>
17         <input type="button" value="Se connecter" onclick="
18             checkAccess()">
19         <input type="number" placeholder="attempt number" id="nbr"
20             >
21     <p id="resultat"></p>
22 </form>
</body>
</html>
```

Règles à implémenter :

1. Les identifiants valides sont : `user = "guest1234"` et `pass = "pass1234"`.
2. Si l'authentification est réussie, afficher le message `"successfully connected"` dans l'élément dont l'id est `result`.
3. En cas d'échec, afficher le message `"user name or password incorrect"` et incrémenter le compteur de tentatives affiché dans l'élément avec `id='nbr'`.
4. Après 3 échecs successifs, désactiver les champs `user` et `pass`, leur appliquer une bordure rouge (`2px solid red`), puis afficher le message `"Access denied !"` dans l'élément avec `id= 'result'`.

Travail demandé :

Donner le script js de la fonction `checkAccess()` qui respecte les règles ci-dessus.

Objectifs du chapitre

Ce chapitre vise à acquérir les bases du langage PHP pour développer dynamiquement des applications web côté serveur, en interaction avec une base de données MySQL.

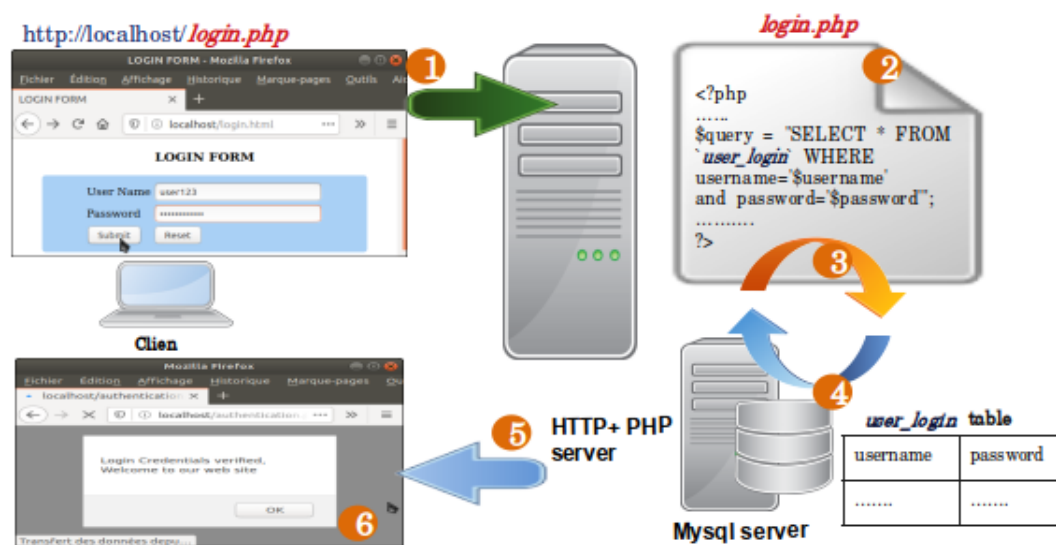
6.1 Introduction

Après avoir découvert les bases du développement web avec HTML/CSS pour la structure et la mise en forme des pages, puis JavaScript et le DOM pour l'interactivité côté client, il devient nécessaire d'introduire une technologie capable de gérer les traitements côté serveur.

C'est dans ce contexte que le langage PHP intervient. Contrairement à JavaScript (exécuté côté client), PHP s'exécute sur le serveur web. Il permet de traiter les données des formulaires, de gérer des sessions et d'interagir avec des bases de données MySQL.

La Figure 5.1 illustre ce mécanisme à travers un exemple d'authentification

FIGURE 6.1.1 – Fonctionnement PHP



Le schéma représente un scénario d'authentification. Il permet de visualiser concrètement comment les données circulent entre le client (navigateur) et le serveur, et comment PHP intervient dans ce processus.

Le cycle de traitement se décompose en cinq étapes :

1. L'utilisateur saisit ses identifiants dans un formulaire en renseignant les champs `username` et `password`.
2. Le navigateur envoie les données au serveur via le script `login.php` suite à l'activation du bouton `submit`.
3. Le serveur exécute le script PHP pour vérifier les identifiants en interrogeant la base de données des utilisateurs autorisés.
4. En cas d'identifiants valides, le script `login.php` produit une page de bienvenue et la transmet au navigateur.
5. Le navigateur affiche enfin le contenu personnalisé à l'utilisateur.

En résumé, PHP agit comme le moteur de l'échange : il reçoit les données, les traite, et renvoie une réponse personnalisée.

6.2 Environnement de développement

Le développement d'applications web dynamiques nécessite la mise en place d'un environnement capable d'exécuter les scripts PHP, de gérer les échanges avec une base de données et de fournir les pages web aux utilisateurs à travers un serveur web. Pour cela, plusieurs solutions logicielles regroupant les outils essentiels sont utilisées par les développeurs.

Parmi les environnements les plus connus, on trouve WAMP, LAMP et XAMPP :

WAMP (Windows, Apache, MySQL, PHP) est destiné aux systèmes Windows ; LAMP (Linux, Apache, MySQL, PHP) fonctionne sous Linux ; XAMPP est une solution multiplateforme intégrant Apache, MariaDB / MySQL, PHP et d'autres outils complémentaires.

Ces environnements rassemblent généralement trois composants principaux :

1. **Apache** : serveur web chargé de recevoir les requêtes des utilisateurs et d'envoyer les pages web au navigateur
2. **PHP** : langage de programmation exécuté côté serveur pour générer des pages dynamiques
3. **MySQL** (ou MariaDB/SQLite) : système de gestion de base de données permettant de stocker et d'organiser les informations de l'application. Grâce à l'interface graphique **phpMyAdmin**, il est possible de créer, modifier et administrer facilement des bases de données MySQL sans écrire de commandes SQL. Cet outil est généralement inclus dans les environnements WAMP, LAMP et XAMPP et accessible via l'URL <http://localhost/phpmyadmin>.

L'utilisation de ces environnements simplifie considérablement le développement local des applications web, car elles permettent d'installer et de configurer rapidement tous les outils nécessaires dans un seul environnement cohérent.

6.2.1 Installation et premiers pas

Choix de l'environnement : comme vu précédemment, vous pouvez utiliser WAMP (Windows), XAMPP (multiplateforme) ou un environnement LAMP natif (Linux). Dans ce document, nous prendrons comme exemple WampServer sous Windows, mais la logique est similaire pour les autres.

Téléchargement et installation : rendez-vous sur <http://www.wampserver.com> et téléchargez la version adaptée à votre système (64 bits ou 32 bits). Lancez l'installateur. Acceptez les conditions d'utilisation et choisissez le dossier d'installation (par défaut C:\wamp). Pendant l'installation, vous pouvez cocher l'option permettant d'installer Apache et PHP comme services Windows, ce qui les lancera automatiquement au démarrage de l'ordinateur.

Vérification du bon fonctionnement : une fois l'installation terminée, lancez WampServer. L'icône dans la barre des tâches devient verte si tout fonctionne. Ouvrez votre navigateur et tapez : <http://localhost/>

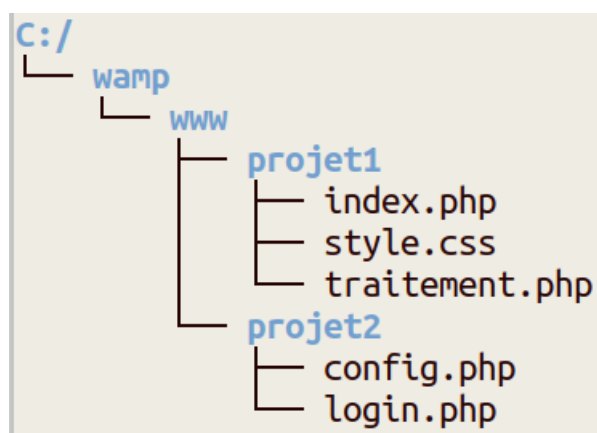
Une page d'accueil WampServer s'affiche. Pour vérifier que PHP est bien interprété, créez un fichier "info.php" dans le dossier C:\wamp\www\ contenant :

```
<?php phpinfo(); ?>
```

Accédez-y via <http://localhost/info.php>. Une page détaillant la configuration PHP apparaît.

Organisation et accès aux projets : l'ensemble de vos projets doit être placé dans le dossier C:\wamp\www\ (ou htdocs pour XAMPP et organisé dans des sous-dossiers par projet comme illustré ci-dessous :

FIGURE 6.2.1 – Structure d'un projet



Accès au projet depuis le navigateur : pour exécuter un projet, ouvrez votre navigateur et tapez : `http://localhost/nom_du_dossier/nom_fichier.php`. Par exemple :

`http://localhost/projet1/index.php`

Note

`localhost` désigne votre propre ordinateur. L'URL `http://localhost/` permet d'accéder au serveur web (Apache) installé localement via WampServer ou XAMPP.

Administration de la base de données : si vous avez besoin d'une base de données MySQL, utilisez phpMyAdmin accessible à : `http://localhost/phpmyadmin`. Vous pourrez y créer des bases de données, des tables, importer ou exporter des données.

Important

Il n'est pas nécessaire de redémarrer votre machine après l'installation. Si le serveur ne démarre pas, vérifiez qu'aucun autre programme (Skype, Teams, etc.) n'utilise le port 80.

6.3 Syntaxe de base

PHP est un langage interprété : son code est exécuté par le serveur web à chaque requête, sans compilation préalable.

Tout script PHP doit être encadré par les balises `<?php` et `?>`. Ces scripts doivent ainsi porter l'extension `.php` (et non `.html` ni `.txt`) pour être interprétés par le serveur web. Le code PHP s'écrit donc entre ces deux délimiteurs, comme dans l'exemple suivant :

```
<?php echo "Hello , World!"; ?>
```

Pour afficher du texte ou des variables à l'écran, PHP propose deux constructions : `echo` et `print`. Toutes deux produisent une sortie, mais `echo` est légèrement plus rapide et accepte plusieurs arguments séparés par des virgules.

```
1 <?php
2     echo "Bonjour ", "le ", "monde !" ;// plusieurs arguments
3     print "Hello"; // un seul argument
4 ?>
```

Comme en JavaScript, PHP est également un langage à typage dynamique (faiblement typé) : le type d'une variable est déterminé par la valeur qui lui est affectée et peut changer en cours d'exécution. De plus, les variables sont toujours précédées du symbole \$. L'exemple ci-dessous en est une illustration :

```
1 <?php
2     $valeur = 42; /* integer */
3     $valeur = "Bonjour"; /* string */
4     $valeur = true; /* boolean */
5 ?>
```

Pour concaténer plusieurs chaînes ou variables, on utilise le point `..`. Cette opération est très courante.

```
1 <?php
2     $nom = "Mohamed";
3     echo "Bonjour " . " Mohamed" . "<br>";
4 ?>
```

Important

On peut combiner PHP et HTML de deux manières :

- ▶ en incluant des balises HTML dans les chaînes affichées par `echo` ou `print` ;
- ▶ en alternant directement des blocs `<?php ... ?>` avec du code HTML statique.

Dans les deux cas, le navigateur interprétera les balises HTML.

6.3.1 Structures de contrôles

Comme tout langage de programmation, PHP dispose de structures de contrôle (conditions et boucles) dont la syntaxe est très proche de celle du langage C.

Important

Une différence syntaxique notable est l'écriture de `elseif` (en un mot) en PHP, alors qu'en langage C on écrit `else if` (deux mots). Les deux fonctionnent en PHP, mais `elseif` est la forme recommandée.

Le Tableau 5.1 en rappelle les principales formes, sans que nous ayons besoin de les détailler davantage.

TABLE 6.1 – Structures de contrôle PHP

Structure	Syntaxe
Condition simple	<pre> if (condition) { bloc d'instructions } </pre>
Condition avec alternative	<pre> if (condition) { bloc d'instructions 1 } else { bloc d'instructions 2 } </pre>
Condition multiple	<pre> if (condition1) { bloc d'instructions 1 } elseif (condition2) { bloc d'instructions 2 } else { bloc d'instructions 2 } </pre>
Condition par cas	<pre> switch (\$variable) { case valeur1: ... break; case valeur2: ... break; default: ... } </pre>
Boucle while	<pre> while (condition) { bloc d'instructions } </pre>

Boucle do...while	<pre> 1 do { 2 bloc d'instructions 3 } while (\$condition); </pre>
Boucle for	<pre> 1 for(init; condition; increment) { 2 bloc d'instructions 3 } </pre>
Boucle foreach	<pre> 1 foreach (\$tableau as \$valeur) { 2 bloc d'instructions 3 } </pre>

Note

La boucle **foreach** est particulièrement utile pour parcourir les tableaux, y compris les tableaux associatifs (présentés plus loin). Les autres structures s'utilisent comme en C.

6.3.2 Tableaux

PHP offre trois types de tableaux : indexé, associatif et multidimensionnel.

Tableau indexé : PHP permet de créer des tableaux indexés numériquement à partir de l'indice 0 comme illustré dans l'exemple ci-dessous :

```

1 <?php
2 $tab = [12, 15, 19];
3 for ($i = 0; $i < count($tab); $i++) {
4     echo $tab[$i]. " ";
5 }
6 ?>

```

Tableaux associatifs : Les tableaux associatifs constituent une particularité importante de PHP pour manipuler facilement des données structurées. Ils utilisent des clés nommées au lieu d'indices numériques. En PHP, cela signifie que l'on peut associer à chaque valeur une clé personnalisée (souvent une chaîne de caractères), comme illustré par l'exemple ci-dessous :

```
1 <?php
2     $notes = ["matière" => "Compilation", "note" => 14];
3     foreach ($notes as $cle => $valeur) {
4         echo "Clé : $cle, Valeur : $valeur <br>";
5     }
6 ?>
```

Tableau multidimensionnel : est un tableau qui contient d'autres tableaux (eux-mêmes pouvant être indexés ou associatifs). Cela permet de représenter des données structurées comme des listes d'étudiants, des grilles, ou des résultats de requêtes. L'exemple suivant illustre un tableau multidimensionnel associatif, où chaque élément du tableau principal représente un étudiant avec ses propres clés (nom, note).

```
1 $etudiants = [
2     ["nom" => "etudiant 1", "note" => 16],
3     ["nom" => "etudiant 2", "note" => 12],
4     ["nom" => "etudiant 3", "note" => 18]
5 ];
```

L'affichage de tous les éléments du tableau peut se faire simplement avec une boucle `foreach`, comme illustré ci-dessous :

```
1 foreach ($etudiants as $etudiant) {
2     echo $etudiant["nom"] . " a obtenu " . $etudiant["note"] . "<br>";
3 }
```

6.3.3 Fonctions PHP

La syntaxe des fonctions en PHP est strictement identique à celle du langage C et donc très proche de JavaScript. L'exemple ci-dessous illustre la définition et l'appel d'une fonction en PHP.

```
1 <?php
2     function addition($a, $b) {
3         return $a + $b;
4     }
5     $resultat = addition(5, 3); /* $resultat vaut 8 */
6 ?>
```

PHP propose plusieurs fonctions natives facilitant l'interrogation, la modification et la fusion des tableaux. (voir Tableau 5.2). Il propose en outre des fonctions facilitant la manipulation des chaînes de caractères (voir Tableau 5.3)

TABLE 6.2 – Fonctions de manipulation des tableaux

Fonction	Effet
<code>count(\$tab)</code>	Retourne le nombre d'éléments du tableau <code>\$tab</code> .
<code>in_array(\$val, \$tab)</code>	Retourne <code>true</code> si la valeur <code>\$val</code> est présente dans le tableau <code>\$tab</code> , <code>false</code> sinon.
<code>array_merge(\$tab1,\$tab2)</code>	Fusionne <code>\$tab1</code> et <code>\$tab2</code> en ajoutant les éléments de <code>\$tab2</code> à la suite de ceux de <code>\$tab1</code> .

L'exemple ci-dessous illustre l'utilisation de ces fonctions

```

1 <?php
2   $couleurs = ["rouge", "vert", "bleu"];
3   echo count($couleurs); // Affiche 3
4   if (in_array("vert", $couleurs)) {
5       echo "Le vert est bien dans la liste.";
6   }
7   $autres = ["jaune", "noir"];
8   $tout   = array_merge($couleurs, $autres);
9   foreach ($tout as $color) {
10      echo $color." "; // Affiche le tableau fusionné
11  }
12 ?>

```

TABLE 6.3 – Fonctions de manipulation des chaînes de caractères

Fonction	Effet
<code>strlen(\$str)</code>	Retourne la longueur de <code>\$str</code> .
<code>strtoupper(\$str)</code>	Transforme <code>\$str</code> en majuscules.
<code>strtolower(\$str)</code>	Transforme <code>\$str</code> en minuscules.
<code>trim(\$str)</code>	Supprime les espaces, tabulations et retours à la ligne situés au début et à la fin de <code>\$str</code> .
<code>substr(\$str, \$i, \$j)</code>	Retourne une sous-chaîne de <code>\$str</code> débutant à la position <code>\$i</code> et de longueur <code>\$j</code> .

<pre>strcmp(\$str1, \$str2)</pre>	Compare \$str1 et \$str2 en retournant 0 si elles sont identiques, une valeur négative si \$str1 est inférieure à \$str2, positive dans le cas contraire.
-----------------------------------	---

Une illustration de quelques fonctions est donnée par l'exemple suivant

```

1 <?php
2     $texte = "  Bonjour le monde  ";
3     echo strlen($texte); // Longueur 19 (les espaces comptent)
4     $nettoye = trim($texte); // Suppression des espaces (début et fin)
5     echo $nettoye;        // "Bonjour le monde"
6     echo substr($nettoye, 0, 5); // Sous-chaîne (5 premiers caractères)
7     $chaine1 = "PHP";
8     $chaine2 = "php";
9     if (strcmp($chaine1, $chaine2) == 0) {
10         echo "Les chaînes sont identiques";
11     } else {
12         echo "Les chaînes sont différentes";
13     }
14 ?>

```

6.4 Traitement des formulaires par PHP

Dans le cas de la transmission de valeurs uniques par un formulaire au serveur, celles-ci sont contenues dans des tableaux associatifs dits superglobaux appelés `$_POST` ou `$_GET` selon la méthode choisie. Les clés de ces tableaux sont les noms associés aux champs par l'attribut `name`.

Avant d'utiliser ces données, il est indispensable de vérifier leur existence et leur validité. PHP fournit pour cela deux fonctions essentielles :

- `isset($variable)` : retourne true si une variable existe et n'est pas null.
- `empty($variable)` : retourne true si une variable est vide.

L'exemple suivant montre comment utiliser `isset()` et `empty()` pour vérifier l'existence et le contenu d'une donnée transmise par `$_POST` :

```

1 <html>
2 <body>
3     <form action="welcome.php" method="post">
4         <div>Name: <input type="text" name="name"></div>
5         <div>E-mail: <input type="text" name="email"></div>

```

```

6     <input type="submit">
7 </form>
8 </body>
9 </html>

```

```

1 <html>
2 <body>
3   Welcome <?php echo $_POST["name"]; ?>
4   <div> Your email address is: <?php echo $_POST["email"]; ?> </div>
5 </body>
6 </html>

```

Lorsqu'un champ peut transmettre plusieurs valeurs (par exemple des cases à cocher), on ajoute des crochets [] à la valeur de l'attribut `name` ; PHP regroupe alors automatiquement ces valeurs dans un tableau indexé accessible via la même superglobale (`$_POST` ou `$_GET`)

```

1 <html>
2 <body>
3   <form action="lang.php" method="post">
4     <div>
5       <input type="checkbox" name="lang[]" value="html">HTML
6       <input type="checkbox" name="lang[]" value="css">CSS
7       <input type="checkbox" name="lang[]" value="php">PHP
8     </div>
9     <input type="submit">
10  </form>
11 </body>
12 </html>

```

```

1 <html>
2 <body>
3   <?php foreach ($_POST['lang'] as $choose) {
4     echo "<br> Languages selected: ".$choose;
5   }
6   ?>
7 </body>
8 </html>

```

6.5 Interaction avec MySQL

Après avoir analysé le mécanisme d'interception des données soumises via les formulaires HTML , mécanisme fondé sur les superglobales `$_POST` et `$_GET` , nous abordons à présent l'interface entre PHP et le SGBD MySQL. L'objectif est d'explicitier comment PHP, agissant

comme une couche applicative médiane, assure l'échange de données entre la couche client (navigateur) et la couche de stockage relationnelle (MySQL), dont les fondements sont supposés maîtrisés.

6.5.1 Connexion à une base de données

Pour commencer, il convient d'établir une connexion au serveur MySQL via l'instanciation d'un objet `mysqli` respectant la syntaxe ci-dessous :

```
1 $connection = mysqli_connect($dbhost, $dbuser, $dbpass);
```

où `$connection` représente l'objet `mysqli` retourné.

Pour des raisons de sécurité, il est vivement recommandé de stocker ces paramètres de connexion dans un fichier distinct. Ce fichier sera ensuite inclus dans les scripts nécessitant une connexion à la base de données par la directive `require`.

L'exemple suivant illustre le script permettant la connexion à la base de données (supposée créée) `db_contacts`.

```
1 <?php
2 $dbhost = "localhost"; // Serveur MySQL (local)
3 $dbuser = "root"; // Nom d'utilisateur
4 $dbpass = ""; // Mot de passe (vide par défaut sur serveur local)
5 $dbname = "db_contacts"; // Nom de la base de données cible
6 $connection = mysqli_connect($dbhost, $dbuser, $dbpass);
7 if (!$connection) {
8     die("Database Connection Failed: " . mysqli_error($connection));
9 }
10 $select_db = mysqli_select_db($connection, $dbname);
11 if (!$select_db) {
12     die("Database Selection Failed: " . mysqli_error($connection));
13 }
14 ?>
```

Ce script procède en plusieurs étapes :

- **Définition des paramètres** : les variables `$dbhost`, `$dbuser`, `$dbpass` et `$dbname` stockent respectivement l'adresse du serveur, le nom d'utilisateur, le mot de passe et le nom de la base de données cible.
- **Connexion au serveur** : la fonction `mysqli_connect()` établit la connexion au serveur MySQL et retourne un objet de connexion stocké dans la variable `$connection`.
- **Vérification de la connexion** : si `$connection` vaut `false` (échec de connexion), la fonction `die()` interrompt le script et affiche un message d'erreur obtenu via `mysqli_error(connection)`.
- **Sélection de la base** : la fonction `mysqli_select_db()` choisit la base de données `db_contacts` sur laquelle les futures requêtes SQL s'exécuteront.

- **Vérification de la sélection** : si la sélection échoue (par exemple si la base n'existe pas), le script s'arrête avec un message explicite.

Une fois ce script placé dans un fichier séparé (par exemple `connexion.php`), il pourra être inclus dans tout `script` nécessitant un accès à la base de données de la manière suivante :

```
1 require "connexion.php";
```

6.5.2 Manipulation d'une base de données

Afin d'illustrer l'interaction entre PHP et MySQL, prenons l'exemple d'une application simple nommée **contacts** qui permet d'extraire, à partir de leur **genre**, les informations des contacts (nom, mail, téléphone) stockées dans la base **db_contacts**. La structure de la table est présentée dans la figure ci-dessous :

FIGURE 6.5.1 – Structure de la table `contacts`

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
nom	varchar(30)	YES		NULL	
telephone	varchar(20)	YES		NULL	
mail	varchar(30)	YES	UNI	NULL	
genre	char(1)	YES		NULL	

L'application se présente sous forme d'un formulaire proposant à l'utilisateur un choix de genre (M/F) via un champ `radio` dont le code est donné par le fichier `"select.html"` ci-dessous :

```
1 <html>
2   <body>
3     <form method="GET" action="select.php">
4       <div>
5         <input type="radio" name="genre" value="M">
6         <label>Masculin</label>
7       </div>
8       <div>
9         <input type="radio" name="genre" value="F">
10        <label>Féminin</label>
11      </div>
12      <input type="submit" name="submit" value="Submit Query">
13    </form>
14  </body>
15</html>
```

La validation du formulaire par l'utilisateur via le bouton `submit` provoque la transmission de la valeur sélectionnée au script `"select.php"`. Ce dernier procède d'abord à

l'inclusion du fichier "connexion.php" pour établir la connexion à la base `contacts_db`, puis extrait la donnée transmise en utilisant `$_GET`. La variable `$genre` ainsi renseignée, le script construit et exécute la requête SQL suivante :

```

1 <?php
2     require "connexion.php";
3     $genre = $_GET['genre'];
4     $sql_query = "SELECT * FROM contacts where genre='$genre'";
5     $result = mysqli_query($connection, $sql_query)
6         or die(mysqli_error($connection));
7 ?>

```

La requête ayant retourné au moins un résultat, le script construit un tableau HTML. La boucle `while`, associée à `mysqli_fetch_assoc()`, parcourt chaque ligne du jeu de résultats et l'insère dynamiquement dans le tableau :

```

1 if (mysqli_num_rows($result) > 0) {
2 ?>
3 <table border="1">
4     <tr>
5         <th>Nom</th>
6         <th>Email</th>
7         <th>Téléphone</th>
8     </tr>
9 <?php
10 while ($row = mysqli_fetch_assoc($result)) {
11 ?>
12     <tr>
13         <td><?php echo $row['nom']; ?></td>
14         <td><?php echo $row['mail']; ?></td>
15         <td><?php echo $row['telephone']; ?></td>
16     </tr>
17 <?php
18     }
19 ?>

```

Après l'affichage (voir Figure 5.5.2), le script referme le tableau et libère la connexion avec `mysqli_close()` :

```

1 </table>
2 <?php
3     }
4     mysqli_close($connection);
5 ?>

```

FIGURE 6.5.2 – Affichage des enregistrements

Nom	Email	Telephone
Mohamed	mohamed@gmail.com	06 66 66 66 66
Ahmed	ahmed@gmail.com	05 55 55 55 55

Exercice 6.5.1

Créez une nouvelle page "insert_contact.html" contenant un formulaire permettant de saisir un nouveau contact (nom, email, téléphone, genre). Le formulaire enverra les données au script "insert_contact.php" qui les insérera dans la table contacts. En cas de succès, affichez un message de confirmation ; dans le cas contraire affichez un message d'erreur.

Web services : notions de base

Objectifs du chapitre

Ce chapitre vise à acquérir les bases des services Web, à savoir

7.1 Introduction

L'essor des services Web constitue aujourd'hui une évolution majeure dans le développement d'applications Web. Pourquoi un tel engouement ? La réponse tient à l'évolution des besoins : les systèmes d'information modernes ne sont plus des entités isolées.

Considérons quelques situations concrètes : une application de réservation doit communiquer avec un système de paiement ; une plateforme e-commerce doit interroger un service de livraison ; un site météo doit récupérer des données auprès de capteurs distants. Partout, les besoins d'intégration et d'interopérabilité se multiplient. Or, ces applications sont rarement construites sur les mêmes bases. Elles utilisent des langages différents (Java, C, PHP, Python), tournent sur des systèmes d'exploitation variés (Windows, Linux, macOS) et sont hébergées sur des infrastructures diverses (**serveurs physiques**, cloud, etc). C'est ici qu'interviennent les services Web. Ils offrent une solution universelle permettant à des applications **hétérogènes** de dialoguer via **Internet**, **indépendamment** des **plateformes** et des **langages** sur lesquelles elles reposent. Grâce à un ensemble de standards ouverts (HTTP, XML, SOAP, WSDL), ils assurent cette interopérabilité.

7.2 Définition

« Les services web sont des applications **auto-descriptives**, **modulaires**, et **faiblement couplées** qui fournissent un modèle de programmation et de déploiement d'applications, basé sur des normes, et s'exécutent au travers de l'infrastructure web » (Les Web Services, H.Kadima, Dunod 2003)

7.3 Caractéristiques des services Web

Un service Web se distingue par un ensemble de propriétés qui le rendent adapté à l'intégration d'applications hétérogènes. Voici ces principales caractéristiques :

1. **Accessibilité via le réseau** : le service Web est accessible depuis Internet ou un intranet via des protocoles standards, généralement HTTP.
2. **Interface publique décrite en WSDL** : tout service Web expose une interface qui décrit ses fonctionnalités, les paramètres d'appel et les données renvoyées. Cette description est formalisée dans un langage standardisé appelé WSDL (Web Services Description Language). Grâce au WSDL, un développeur sait précisément comment interagir avec le service.
3. **Publication dans un annuaire (UDDI)** : les services Web peuvent être répertoriés dans un annuaire. Le standard UDDI (Universal Description, Discovery and Integration) permet de stocker et de rechercher des services Web en fonction de leurs fonctionnalités, de leur localisation ou de leur fournisseur. Bien que moins utilisé aujourd'hui, UDDI illustre le principe de découvrabilité des services.
4. **Communication par messages XML sur HTTP** Les échanges entre clients et services Web s'effectuent au moyen de messages structurés en XML (eXtensible Markup Language). Ces messages sont transportés par des protocoles Internet, le plus souvent HTTP ou HTTPS. Pour les services Web dits "traditionnels" (style SOAP), la structure des messages suit le standard SOAP (Simple Object Access Protocol).
5. **Systèmes faiblement couplés** : l'intégration d'applications via des services Web produit des systèmes faiblement couplés, ce qui signifie que chaque application peut évoluer indépendamment puisqu'une modification apportée à un service n'impacte pas nécessairement les autres.

7.4 Fonctionnement des services Web

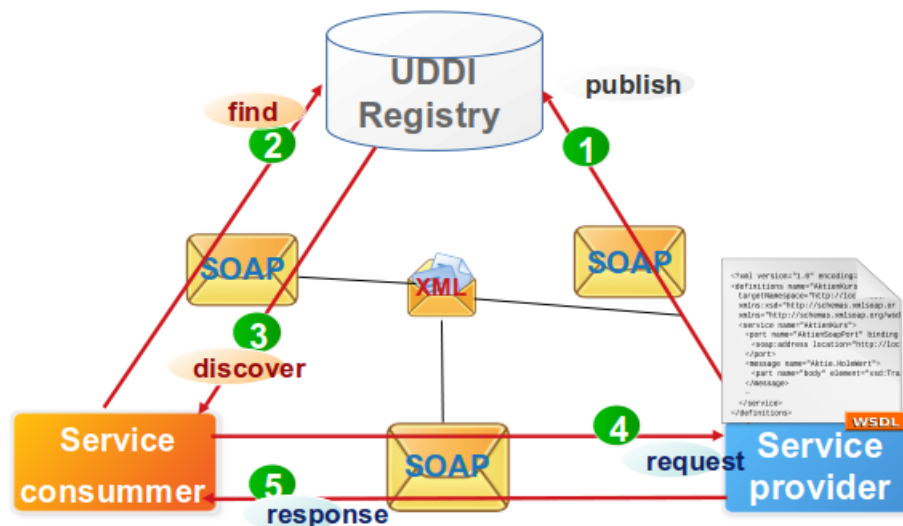
L'architecture des services Web s'articule autour de trois composantes principales :

Le fournisseur de service (service provider) : développe, héberge et expose le service

Le consommateur (service consumer) : recherche et utilise le service ;

L'annuaire (registry) : référence les services disponibles et permet leur découverte. Ces trois composantes interagissent selon un cycle de **publication**, **découverte** et **invocation**, illustré par le schéma ci-dessous :

FIGURE 7.4.1 – Fonctionnement des Web services



Comme le montre la Figure 6.4.1, l'invocation d'un service Web repose sur un ensemble d'étapes préalables qui le rendent opérationnel. Le fournisseur publie son service dans l'annuaire UDDI. Le consommateur le découvre via cet annuaire et récupère sa description WSDL. Une fois ces étapes accomplies, il peut l'invoquer en échangeant des requêtes et réponses SOAP, autrement dit des messages XML enfermés dans des enveloppes SOAP et transportées par HTTP.

Note

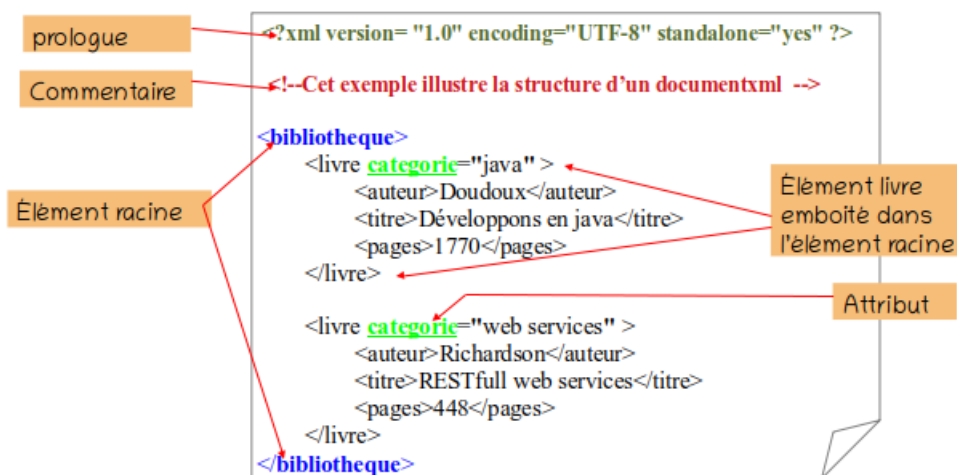
Il existe deux grandes familles de services Web : les services Web **SOAP** (basés sur XML, SOAP, WSDL et UDDI) et les services Web **REST** (basés sur HTTP et le concept de ressources). Bien que REST soit un concurrent potentiel des services Web SOAP, ce polycopié se concentre uniquement sur les principes des services Web SOAP, comme prévu par les objectifs pédagogiques.

7.5 XML, DTD et XSD

7.5.1 XML

XML est un langage de balisage extensible, issu du langage **SGML** (textttStandard Generalized Markup Language) et standardisé par le consortium *W3C*. Il s'agit d'un langage auto-

FIGURE 7.5.1 – Exemple d'un document XML



descriptif qui permet de structurer les données. La Figure 6.5.11 illustre un exemple simple de document XML décrivant une bibliothèque universitaire.

Mais qu'est-ce qui garantit qu'un tel document est syntaxiquement correct ? C'est la notion de **bien formé** (well-formed) qui le définit.

Un document XML est donc dit **bien formé** (well-formed) lorsque sa syntaxe respecte un ensemble de règles fondamentales :

1. Tout d'abord, le prologue doit être renseigné en précisant la version de XML utilisée.
2. Ensuite, le document ne doit posséder qu'une seule balise racine.
3. Les noms des balises et des attributs doivent respecter les règles de nommage (pas d'espaces, ne pas commencer par un chiffre, etc.).
4. Par ailleurs, toutes les balises ouvertes doivent être correctement fermées, et les valeurs des attributs doivent être placées entre guillemets (simples ou doubles).
5. Enfin, les balises du document ne doivent pas se chevaucher.

Tout document XML doit d'abord être bien formé (syntaxe correcte) ; sa conformité à une structure prédéfinie peut ensuite être validée par une DTD (Document Type Definition) ou un XSD (XML Schema Definition).

7.5.2 DTD

DTD est une grammaire que doit respecter un document XML pour être considéré comme valide. Elle permet de structurer un document XML en spécifiant précisément la nature des éléments qui le composent, leur ordre d'apparition, ainsi que leur imbrication. Pour chaque élément, la DTD peut également définir quels attributs sont obligatoires et lesquels sont optionnels. En d'autres termes, la DTD contient la description complète des éléments et des attributs autorisés dans le document XML, garantissant ainsi que celui-ci respecte un modèle prédéfini.

L'exemple suivant illustre une DTD à laquelle le document XML de la bibliothèque donné précédemment doit se conformer.

```
1 <!ELEMENT bibliotheque (livre+)>
2 <!ELEMENT livre (auteur, titre, pages)>
3 <!ELEMENT auteur (#PCDATA)>
4 <!ELEMENT titre (#PCDATA)>
5 <!ELEMENT pages (#PCDATA)>
6 <![ATTLIST livre categorie CDATA #REQUIRED]>
```

7.5.3 XSD

XSD est, tout comme DTD, une grammaire que doit respecter un document XML pour être considéré comme valide. Cependant, XSD offre une expressivité bien plus riche que DTD. Il permet non seulement de structurer un document XML en spécifiant la nature des éléments, leur ordre d'apparition et leur imbrication, mais aussi de définir précisément le **type** des données contenues (chaînes de caractères, entiers, dates). Pour chaque élément, XSD peut indiquer quels attributs sont obligatoires ou optionnels, et contraindre leurs valeurs (par exemple, une année comprise entre 2000 et 2025). En d'autres termes, XSD constitue une description complète et typée des éléments et attributs autorisés dans un document XML, garantissant à la fois sa structure et la nature de son contenu.

L'exemple suivant illustre un XSD auquel le document XML de la bibliothèque donné précédemment doit se conformer.

```
1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
2 <xs:element name="bibliotheque">
3 <xs:complexType>
4   <xs:sequence>
5     <xs:element name="livre" maxOccurs="unbounded">
6       <xs:complexType>
7         <xs:sequence>
8           <xs:element name="auteur" type="xs:string"/>
9           <xs:element name="titre" type="xs:string"/>
10          <xs:element name="pages" type="xs:positiveInteger"/>
11        </xs:sequence>
12        <xs:attribute name="categorie" type="xs:string" use="
13          required"/>
14        </xs:complexType>
15      </xs:element>
16    </xs:sequence>
17  </xs:complexType>
18 </xs:element>
19 </xs:schema>
```

Important

Les échanges entre les différents composants des services Web reposent sur des messages structurés en XML. C'est d'ailleurs sur **XSD** que repose la plupart des standards modernes de services Web pour décrire la structure des messages.

7.6 SOAP, WSDL et UDDI

7.6.1 SOAP

SOAP est un protocole d'invocation d'opérations distantes fondé sur XML. Il s'agit plus précisément d'un protocole d'échange de messages structurés destiné à l'invocation de services Web, conçu pour être indépendant de tout langage de programmation et de toute plateforme. Dans SOAP, les messages échangés sont des documents XML encapsulés dans une enveloppe SOAP, laquelle est généralement transportée par le protocole HTTP.

Comme l'illustre l'ossature d'un message SOAP donnée ci-dessous, cette structure est elle-même définie en XML.

```
1 <soap:Envelope
2   xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
3   <soap:Header>
4     <!-- Informations optionnelles -->
5   </soap:Header>
6   <soap:Body>
7     <!-- Corps du message : requête ou réponse -->
8     ....
9     <soap:Fault>
10      <!-- Erreur, optionnel -->
11      ...
12    </soap:Fault>
13  </soap:Body>
14</soap:Envelope>
```

Dans cette structure, l'élément racine est toujours `<soap:Envelope>`. Celui-ci se compose de deux parties distinctes. La première, `<soap:Header>`, est facultative : elle peut contenir des métadonnées, par exemple des informations d'authentification ou de transaction. La seconde, `<soap:Body>`, est obligatoire et constitue le coeur du message : il contient soit la requête ou la réponse, soit en cas d'erreur, un élément `<soap:Fault>` signalant le problème.

 Note

Dans un message SOAP, l'élément `<soap:Fault>` est optionnel. Il apparaît uniquement dans le corps (Body) d'une réponse lorsqu'une erreur s'est produite lors du traitement de la requête.

7.6.2 WSDL

WSDL est un langage de description de services Web basé sur XML. Il constitue le contrat entre le fournisseur du service et le client. Le WSDL doit contenir toutes les informations nécessaires au client pour consommer le service : les opérations proposées, les formats des messages échangés (entrée, sortie, erreur), les protocoles de transport utilisés (généralement SOAP/HTTP), ainsi que l'adresse où le service est accessible. L'élément racine d'un document WSDL est `<definitions>`. Il contient l'ensemble des descriptions du service, comme illustré par l'ossature suivante :

```
1 <definitions>
2   <types>
3     <!-- Définitions des types de données -->
4   </types>
5   <message>
6     <!-- Définition des messages échangés -->
7   </message>
8   <portType>
9     <!-- Ensemble des opérations proposées -->
10  </portType>
11  <binding>
12    <!-- Spécification du protocole et du format des données -->
13  </binding>
14 </definitions>
```

Un fichier WSDL se décompose en deux grandes parties :

Une partie abstraite : elle décrit le service indépendamment de toute implémentation. Elle regroupe notamment les définitions des types (`<types>`), des messages (`<message>`) et des opérations (`<portType>`).

Une partie concrète : elle spécifie les liaisons (`<binding>`) entre l'interface abstraite, le protocole de transport (généralement SOAP/HTTP) et l'adresse réseau (URL) du service.

Ainsi, le fichier WSDL joue le rôle de contrat entre le fournisseur et le consommateur du service : il décrit tout ce que le client doit savoir pour invoquer le service, sans avoir besoin d'en connaître les détails internes.

7.6.3 UDDI

UDDI est un annuaire de services Web. Il permet de publier des services et de les rechercher. Concrètement, UDDI offre un cadre pour présenter les entreprises, définir les services Web qu'elles proposent, décrire comment s'y connecter, et fournir les données techniques nécessaires à leur localisation et à leur invocation par d'autres applications. En d'autres termes, UDDI joue le rôle de "pages jaunes" des services Web, facilitant ainsi leur découverte et leur réutilisation.

7.7 Plateformes de développement des services Web

La mise en oeuvre concrète des services Web SOAP repose sur des API et des outils qui masquent la complexité des échanges XML sous-jacents. L'API **JAX-WS** (Java API for XML Web Services) constitue le standard de référence dans l'écosystème Java pour le développement et la consommation de services Web SOAP.

Avec JAX-WS, deux stratégies de développement s'offrent au développeur comme illustré par la Figure 6.7.1

FIGURE 7.7.1 – Approches de développement de Web services



Comme le montre la Figure 6.7.1, l'approche **code-first** (dite aussi *contract-last* ou *bottom-up*) génère le contrat WSDL à partir d'un code Java existant. À l'inverse, l'approche **code-last** (dite aussi *contract-first* ou *top-down*) part d'un fichier WSDL préalablement défini pour générer les squelettes de code Java correspondants.

7.8 Implémentation d'un service Web SOAP

L'implémentation d'un service Web SOAP avec l'API JAX-WS se décompose en deux grandes phases : la mise en place du côté serveur (fournisseur du service) et celle du côté client (consommateur du service).

Le développement du côté serveur suit trois étapes :

1. Définition de l'interface du service web qui déclare les méthodes accessibles à distance.

```

1 package dz.univ;
2 import javax.jws.*;
3 import javax.jws.soap.SOAPBinding.*;
4 //Service Endpoint Interface
5 @WebService
6 @SOAPBinding(style = Style.RPC)
7 public interface CompteInterface{
8     @WebMethod void credit(double montant);
9     @WebMethod void debit(double montant);
10    @WebMethod double solde();
11 }

```

2. Définition de la classe qui implémente l'interface augmentée par des annotations de l'API JAX-WS (notamment @WebService et @WebMethod)

```

1 package dz.univ;
2 import javax.jws.WebService;
3 @WebService(endpointInterface = "dz.univ.CompteInterface")
4 public class CompteImpl implements CompteInterface{
5     private double compte;
6     public CompteImpl(){ compte=0;}
7     public void credit(double montant) {compte+=montant;}
8     public void debit(double montant) {compte-=montant;}
9     public double solde() {return compte;}
10 }

```

3. Définition de la classe qui publie et lance le service via l'API Endpoint.publish()

```

1 package dz.univ;
2 import javax.xml.ws.Endpoint;
3 public class ComptePublisher{
4     public static void main(String[] args) {
5         Endpoint.publish("http://localhost:7777/ws/compte",new
5             CompteImpl());
6     }
7 }

```

Côté client, une fois le service publié et son fichier WSDL accessible, le développement consiste à générer automatiquement les stubs (squelettes) à partir du WSDL, puis à invoquer les opérations distantes via ces objets générés, comme s'il s'agissait d'appels locaux.

```

1 package dz.univ;
2 import java.net.URL;
3 import javax.xml.namespace.QName;
4 import javax.xml.ws.Service;

```

```
5 public class CompteClient{
6     public static void main(String[] args) throws Exception {
7         URL url = new URL("http://localhost:7777/ws/compte?wsdl");
8         QName qname = new QName("http://univ.dz/", "CompteImplService");
9         Service service = Service.create(url, qname);
10        CompteInterface cpt = service.getPort(CompteInterface.class);
11        cpt.credit(2000.00);
12        System.out.println(cpt.solde());
13        cpt.debit(500.00);
14        System.out.println(cpt.solde());
15        cpt.add(10,30);
16        System.out.println(cpt.add(10,30));
17    }
18 }
```

Solutions

8.1 Exercice 3.3.1

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Exo1</title>
6 </head>
7 <body>
8   <table border="1px">
9     <caption>Tableau des technologies Web</caption>
10    <thead>
11      <tr>
12        <th>Technologie</th>
13        <th>Niveau</th>
14        <th>Ressource</th>
15      </tr>
16    </thead>
17    <tbody>
18      <tr>
19        <td>HTML</td>
20        <td>Débutant</td>
21        <td rowspan="3">
```

```
22     <a href="https://www.w3schools.com/" title="ressource">w3school
23     </a>
24 </td>
25 </tr>
26 <tr>
27     <td>CSS</td>
28     <td>Intermédiaire</td>
29 </tr>
30 <tr>
31     <td>Javascript</td>
32     <td>Avancé</td>
33 </tr>
34 </tbody>
35 <tfoot>
36 <tr>
37     <td colspan="3">
38         <b>Remarque</b> : Ces technologies constituent les bases du
39         développement web.
40     </td>
41 </tr>
42 </tfoot>
43 </table>
44 </body>
45 </html>
```

8.2 Exercice 3.3.2

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <title>Exo 2</title>
6   </head>
7   <body>
8     <h3>Mes compétences</h3>
9     <ul>
10       <li>HTML</li>
11       <li>CSS</li>
12       <li>Javascript</li>
13       <li>PHP</li>
14       <li>MYSQL</li>
```

```
15     </ul>
16     <h3>Les langues maîtrisées</h3>
17     <ol>
18         <li>Arabe</li>
19         <li>Anglais</li>
20         <li>Français</li>
21     </ol>
22 </body>
23 </html>
```

8.3 Exercice 3.3.3

```
1 <!DOCTYPE html>
2 <html lang="fr">
3 <head>
4     <meta charset="UTF-8">
5     <title>Exo 3</title>
6 </head>
7 <body>
8     <form action="#" method="post">
9         <fieldset>
10             <legend>Informations personnelles</legend>
11             <div class="form-group">
12                 <label for="nss">Numéro de sécurité sociale :</label>
13                 <input type="text" id="nss" name="nss" placeholder="
14                     000-00-0000" required>
15             </div>
16             <div class="form-group">
17                 <label for="email">Email :</label>
18                 <input type="email" id="email" name="email" required>
19             </div>
20             <div class="form-group">
21                 <label for="telephone">Téléphone :</label>
22                 <input type="tel" id="telephone" name="telephone">
23             </div>
24             <div class="form-group">
25                 <label for="dateEmbauche">Date Embauche :</label>
26                 <input type="date" id="dateEmbauche" name="dateEmbauche">
27             </div>
28             <div class="radio-group">
29                 <label>Sexe :</label>
```

```

29     <input type="radio" name="sexe" value="Homme">
30     <label for="homme">Homme</label>
31     <input type="radio" id="femme" name="sexe" value="Femme">
32     <label for="femme">Femme</label>
33 </div>
34     </fieldset>
35     <fieldset>
36     <legend>Fonction et service</legend>
37         <div class="form-group">
38             <label for="fonction">Fonction :</label>
39             <input type="text" id="fonction" name="fonction">
40         </div>
41         <div class="form-group">
42             <label for="service">Service :</label>
43             <select id="service" name="service">
44                 <option value="">--Choisir une fonction--</option>
45                 <option value="administration">Administration</option>
46                 <option value="informatique">Informatique</option>
47                 <option value="ressources_humaines">Ressources Humaines</
48                 option>
49             </select>
50         </div>
51     </fieldset>
52 <fieldset>
53     <legend>Compétences et notes</legend>
54     <div class="checkbox-group">
55         <input type="checkbox" id="communication" name="competences[]"
56             " value="Communication">
57         <label for="communication">Communication</label>
58         <input type="checkbox" id="gestion" name="competences[]"
59             value="Gestion">
60         <label for="gestion" >Gestion</label>
61         <input type="checkbox" id="informatique" name="
62             competences[]" value="Informatique">
63         <label for="informatique">Informatique</label>
64     </div>
65     <div class="form-group">
66         <label for="notes">Remarques :</label>
67         <textarea id="notes" name="notes" placeholder="Notes
68             complémentaires...">
69     </textarea>

```

```
65     </div>
66   </fieldset>
67     <input type="submit" value="Envoyer"/>
68   </form>
69 </body>
70 </html>
```

8.4 Exercice 4.11.1

```
1 table {
2   margin: auto;
3   border-spacing: 15px;
4 }
5 caption {
6   caption-side: top;
7   font-weight: bold;
8   padding: 12px;
9   background-color: lightcyan;
10  border-bottom: 5px solid blue;
11  text-transform: uppercase;
12 }
13 td, th {
14   border: 1px solid black;
15   padding: 10px;
16   text-align: center;
17 }
18 th{background-color: whitesmoke;}
19 img {width: 25%; }
20 /* attributes */
21 td[colspan]:hover {
22   background-color: lightyellow;
23   font-size: 30px;
24   word-spacing: 10px;
25 }
26 td[rowspan]:hover {
27   background-color: lightgray;
28   font-size: 30px;
29 }
```

8.5 Exercice 4.11.2

```
1 label {
2     display: block;
3     margin-top: 15px;
4     font-weight: bold;
5 }
6 input [type="text"],input[type= "submit"] {
7     width: 100%;
8 }
9 input [type="text"],fieldset,input[type= "submit"]{
10     margin-top: 15px;
11     border-radius: 10px;
12     border: 2px solid black ;
13     padding: 10px;
14 }
15 legend {font-weight:bold;}
16 span { margin-right: 50px; }
17 input [type="radio"]:checked +span ,
18 input[type="checkbox"]:checked +span{
19     font-style: italic;
20     text-transform: uppercase;
21 }
```

Bibliographie

- [1] Eric A. Meyer and Estelle Weyl. *CSS : The Definitive Guide : Web Layout and Presentation*. O'Reilly Media, Sebastopol, CA, 5th edition, 2023.
- [2] Jean Engels. *HTML5 et CSS3 : Cours et exercices corrigés*. Eyrolles, 2012.
- [3] Refsnes Data. W3schools online web tutorials. <https://www.w3schools.com/>, 1998–17 juin 2026. Consulté le 17 juin 2026.
- [4] Ethan Cerami. *Web Services Essentials : Distributed Applications with XML-RPC, SOAP, UDDI & WSDL*. O'Reilly Media, Sebastopol, CA, 2002. Ouvrage de référence sur les fondements des services Web XML.
- [5] David Griffiths and Elisabeth Robson. *Head First HTML5 Programming : Building Web Apps with JavaScript*. O'Reilly Media, 1st edition, 2019.
- [6] David Flanagan. *JavaScript : The Definitive Guide*. O'Reilly Media, 7th edition, 2021.
- [7] Kevin Tatroe and Peter MacIntyre. *Programming PHP : Creating Dynamic Web Pages*. O'Reilly Media, 4th edition, 2020.
- [8] Jon Duckett. *HTML and CSS : Design and Build Websites*. Wiley, 2011.