



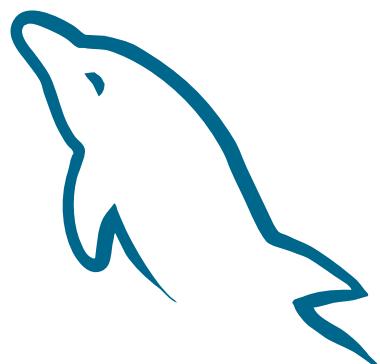
LYCÉE MONJI SLIM SBIBA
4 année Sciences de l'informatique



Cours STI



Année Scolaire : 2025-2026



< *Abidi* />
LYCÉE MONJI
SLIM SBIBA

Table des matières

Chapitre 1 : base de données

I.	Terminologie et les concepts fondamentaux.....	7
II.	Les anomalies :	8
1.	Absence ou duplication de clés primaires :	8
2.	Clés étrangères non conformes	9
3.	Redondance des données :	9
4.	Violation des contraintes de domaine :.....	11
III.	Manipulations sur la structure d'une base de données (Définition des données) :	12
1.	Langage SQL :.....	12
2.	Définition des bases :	12
3.	Définition des tables :.....	12
a.	Les types des données en SQL :.....	12
b.	Création d'une table :.....	13
c.	Contraintes définition complète :.....	14
d.	Supprimer une table :	18
4.	Définition des colonnes :	18
5.	Définitions des contraintes :	19
IV.	Manipulations sur les données d'une base de données (Manipulations des données) :	19
1.	Insertion des données :	19
2.	Suppression des données :	20
3.	Modifications des données :	20
4.	Sélection des données	20
a.	Filtrage des résultats :.....	21
b.	Jointure :	24
c.	Fonctions sur les types DATE en SQL.....	26
d.	Fonctions d'agrégation en SQL	26
e.	Groupement en SQL	27
f.	Tri en SQL.....	29
g.	Sous-requête	31

Chapitre 2 : HTML

I.	Définition et structure générale :	34
II.	Attributs globaux	34
III.	Eléments d'en-tête	35
1.	style:	35
2.	<link>	35
3.	<meta>	36
4.	<script>	36
5.	<title>	37
IV.	Eléments de structuration de texte	38
1.	Les entêtes :	38
2.	Les conteneurs du texte :	38
3.	Les séparateurs :	39
4.	Les liens hypertextes :	39
5.	Contenu pliable :	40
V.	Les éléments de structuration de media	41
1.	Les images :	41
2.	Les Vidéos :	41
3.	Les audios :	42
4.	Balise source :	43
5.	Les figures :	43
VI.	Les conteneurs :	43
VII.	Les tableaux :	44
VIII.	Les Listes :	46
IX.	Les formulaires :	46
1.	Les labels :	47
2.	Les zones de saisie :	47
3.	Zone multilignes textarea:	49
4.	Liste déroulante :	49
5.	fieldset :	49
6.	datalist :	50
X.	Eléments de structuration d'une page web	50
XI.	Les événements :	51

Chapitre 3 : CSS

I.	Introduction au CSS	52
1.	Définition d'une règle CSS	52
2.	Types de Sélecteurs CSS	53
a.	Sélecteur de type	53
b.	Sélecteur de classe :.....	53
c.	Sélecteur d'ID :.....	53
d.	Sélecteur universel :	54
e.	Sélecteur d'attribut :	54
f.	Sélecteurs combinés :	55
g.	Sélecteurs multiples :	55
h.	pseudo-classe	56
II.	Propriétés de mise en forme du texte en CSS.....	56
1.	font-family :	56
2.	font-weight :	57
3.	font-style :	58
4.	font-size :	58
5.	font :	58
6.	text-align :	59
7.	text-shadow :.....	59
8.	text-transform :	59
9.	color :	60
XII.	Propriétés de couleur et de fond :	61
1.	background-color	61
2.	background-image	61
3.	background-repeat :.....	62
4.	background-size :.....	63
5.	background	64
5.	1. border-style	64
6.	border-color	65
7.	border-width.....	66
4.	border-radius.....	66
8.	5. border	66
XIII.	Propriétés des listes	67

1. list-style-type.....	67
2. 2. list-style-position	68
3. 3. list-style-image	68
4. 4. list-style.....	69
XIV. Propriétés des tableaux	70
5. 1. table-layout	70
6. 2. border-collapse.....	71
XV. Propriétés des images :.....	72
1. invert()	72
2. grayscale().....	73
3. blur()	73
XVI. Propriétés des boîtes :	74
1. width et height	75
2. margin.....	75
3. padding.....	76
4. Display :	77
5. Position :	78
6. float :	79
7. overflow	80
8. box-shadow.....	80
9. Opacity	81
XVII.Transformation	82
1. rotate() : Rotation d'un élément.....	82
2. skew() : Inclinaison d'un élément	83
3. scale() : Redimensionnement d'un élément	83
4. translate() : Déplacement d'un élément.....	83
XVIII. Les transition	84
XIX. animations.....	85
Chapitre 4 : Javascript	
I. Introduction :	87
II. Intégration de JavaScript dans le HTML :.....	88
1. intégration interne.....	88
2. intégration externe	88
3. événements.....	89

III.	Affichage de données :	89
1.	alert	89
2.	document.write	89
IV.	Déclaration et lecture des variables :	90
1.	let	90
2.	const (pour les constantes)	90
3.	Lecture des variables avec prompt	90
V.	typage dans JavaScript	91
1.	Number (Nombres)	91
a.	Opérations arithmétiques :	91
b.	Objet Math :	91
c.	Conversion vers un nombre :	92
2.	String (Chaînes de caractères)	92
3.	Les booléens :	93
a.	Opérateurs de comparaison	93
b.	Opérateurs logiques :	94
4.	Objet Date :	94
5.	Array (Tableaux)	94
a.	Créer un tableau vide :	94
b.	Créer un tableau avec des valeurs initiales :	94
c.	accéder à un élément du tableau :	95
d.	taille d'un tableau (length)	95
VI.	Les structures de contrôle conditionnelles :	95
VII.	Structures itératives	95
VIII.	Les fonctions	96
IX.	Manipulations sur les éléments HTML	96
1.	Ciblage des éléments :	96
a.	Par identifiant :	96
b.	Par nom :	97
2.	Manipulation de contenu d'un élément innerHTML :	97
3.	Modification des attributs :	97
4.	Modification du style	98
5.	Méthodes principales pour contrôler une vidéo/audio	99

Chapitre 5 : PHP

I.	Introduction	100
II.	Fonctionnement d'un site web dynamique en PHP.....	100
III.	Environnement de travail	101
IV.	Les base d'un script PHP.....	101
1.	Déclaration du code PHP :	101
2.	Variables :	102
3.	typage et fonctions prédefinies :.....	102
a.	Les nombres	102
b.	Les chaînes de caractères en PHP	103
c.	Les booléens :	104
d.	Fonctions des date/heure :.....	105
e.	Les tableaux :.....	105
4.	Structures de contrôles conditionnelles	106
5.	Structures itératives :	106
6.	Les fonctions :	107
7.	Opérateurs de transtypage :	107
8.	Fonctions diverses :	107
a.	die().....	107
b.	isset()	108
c.	require().....	108
V.	Principe de transmission de données entre des pages web.....	108
1.	La méthode GET :.....	108
2.	La méthode POST :.....	109
VI.	Fonctions PHP pour MySQL.....	109



Chapitre I bases de données



I. Terminologie et les concepts fondamentaux

Une **base de données relationnelle (BDR)** est un système de gestion de données qui organise les informations sous forme de **tables**, où chaque table est constituée de lignes et de colonnes. Les lignes représentent des **enregistrements** (ou tuples) et les colonnes des **attributs** (ou champs). Le modèle relationnel repose sur des **relations** définies entre ces tables, souvent par l'intermédiaire de **clés primaires** et de **clés étrangères**.

Exemple : Voici les données d'une boutique de ventes des produits IT

Table client

idClient	Nom	Email
201	Ahmed Nebli	Nebli.ah@yahoo.fr
202	Layla Gafsi	Gafsi.m@gmail.com
203	Fatima Ayari	ayari@topnet.net
204	Omar Ben Ammar	bjAmmar@gmail.com
205	Salma Nasri	Nasri.salma@outlook.com

Table Article

ArticleID	Nom Article	Marque	Prix en dinar Tunisien
101	Ordinateur portable	HP	800
102	Smartphone	Huawei	500
103	Tablette	Infinix	300
104	Écouteurs Bluetooth	GBL	50
105	Clavier mécanique	Logitech	100

Table achat

idClient	ArticleID	Date d'Achat	Quantité
202	101	2024-01-15	1
201	102	2024-01-16	1
203	103	2024-02-10	2
202	104	2024-03-05	3
205	105	2024-04-20	1

On peut dégager la représentation textuelle suivante :

client(idClient, nom, email)

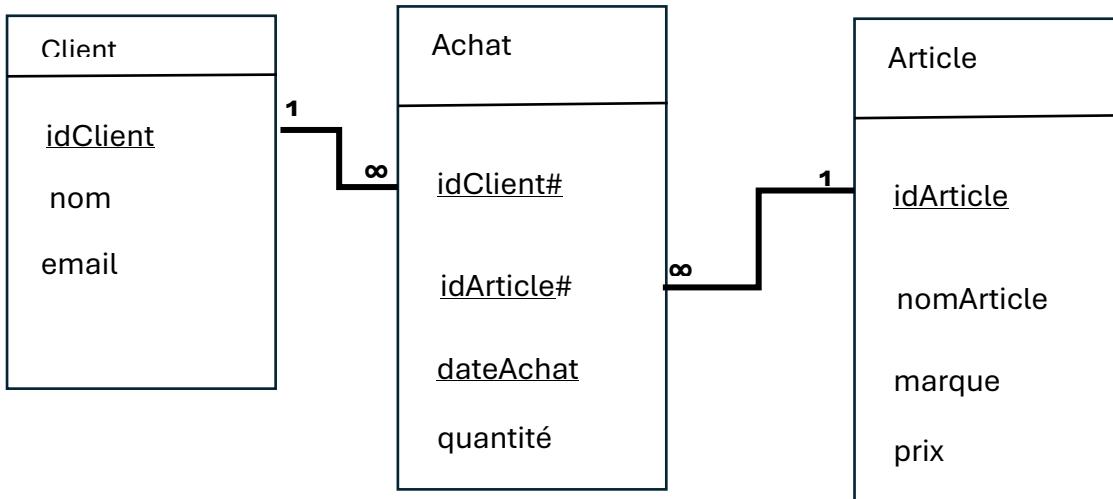
article(idArticle, nomArticle, marque, prix)

achat (idClient#, idArticle#, dateAchat, quantité)

Remarques :

- ❖ Les clés primaires sont soulignées
- ❖ Les clés étrangères sont identifiées par "#"
- ❖ La clé primaire de la table achat est composée par les champs (telClient, articleId et dateAchat)

La représentation graphique de cette base est :



II. Les anomalies :

1. Absence ou duplication de clés primaires :

Une **clé primaire** est un identifiant unique qui permet de distinguer chaque enregistrement d'une table dans une base de données relationnelle. Elle ne doit **jamais être dupliquée** ou laissée **vide** (valeur nulle).

L'**absence** de clé primaire ou la **duplication** de celle-ci dans une table provoque des **anomalies d'intégrité**. Cela empêche d'identifier de manière unique chaque enregistrement, rendant la manipulation des données incohérente. Ces erreurs peuvent entraîner des **incohérences**, des **problèmes de redondance** et des **difficultés lors des requêtes**.

Exemple1:

idClient	Nom	Email
1	Ahmed	Ahmed.ah@yahoo.fr
NULL	Layla	Leyla.gz@gmail.com
3	Omar	BejiOmar@outlook.com

Problème :

Ici, un enregistrement (Layla) n'a pas de idClient, ce qui signifie que cet enregistrement ne peut pas être identifié de manière unique. Cela peut poser des problèmes lorsqu'on veut mettre à jour ou supprimer cet enregistrement, car on ne pourra pas le retrouver de manière fiable.

Exemple2:

idClient	Nom	Email
1	Ahmed	ahmed@mail.com
2	Layla	layla@mail.com
2	Omar	omar@mail.com

Problème :

Ici, les deux enregistrements pour Layla et Omar ont le même ClientID = 2. Cela pose un problème, car il devient impossible **de distinguer ces deux enregistrements** lors d'une requête ou d'une manipulation de données.

2. Clés étrangères non conformes

Une **clé étrangère** (ou **foreign key**) est un champ dans une table qui fait référence à la **clé primaire** d'une autre table. Elle établit un lien entre deux tables et garantit l'intégrité référentielle dans une base de données relationnelle.

L'anomalie des **clés étrangères non conformes** se produit lorsque la valeur d'une clé étrangère fait référence à un enregistrement **qui n'existe pas** dans la table référencée (celle contenant la clé primaire). Cela brise l'intégrité référentielle, ce qui peut entraîner des **incohérences dans les données**.

Exemple

Supposons que nous avons deux tables dans une base de données : **Clients** et **Achats**. La table **Achats** contient une clé étrangère qui fait référence à la table **Clients**.

Données de la table Clients :

IdClient	Nom	Email
1	Ahmed	ahmed@mail.com
2	Layla	layla@mail.com
3	Omar	omar@mail.com

Données de la table Achats :

idClient	Date d'achat	Quantité
1	2024-01-15	1
2	2024-01-20	2
4	2024-02-10	1

Problème :

Dans la table **Achats**, la ligne 3 contient une valeur de **idClient = 4**, mais il n'y a **aucun enregistrement** avec **ClientID = 4** dans la table **Clients**. Cela signifie que cet achat fait référence à un client qui **n'existe pas** dans la base de données.

3. Redondance des données :

La **redondance des données** se produit lorsqu'une information est **répétée inutilement** dans une ou plusieurs tables d'une base de données. Cela entraîne une **duplication** des informations, ce qui est inefficace et peut causer des **incohérences** et des **anomalies de mise à jour**.

Les bases de données relationnelles visent à minimiser cette redondance en normalisant les données (en les organisant dans des tables distinctes reliées par des clés primaires et étrangères). Lorsque cette normalisation n'est pas respectée, les mêmes données peuvent apparaître plusieurs fois, ce qui crée des problèmes.

Exemple

Prenons l'exemple d'une base de données qui gère des informations sur des clients, des produits et des achats. Si toutes ces informations sont stockées dans une seule table, cela peut entraîner une redondance.

Table Achats:

NomClient	Email	NomProduit	PrixProduit	DateAchat
Ahmed	ahmed@mail.com	Ordinateur portable	800	2024-01-10
Layla	layla@mail.com	Smartphone	500	2024-01-15
Ahmed	ahmed@mail.com	Smartphone	500	2024-02-05

Problème de redondance :

- Les informations sur Ahmed (son nom et son email) sont répétées dans chaque ligne où il a effectué un achat.
- Chaque fois qu'Ahmed achète un produit, son nom et son email doivent être répétés, ce qui entraîne une duplication des informations.
- Cela peut poser des problèmes lorsque les informations de Ahmed changent (par exemple, s'il change d'email). Il faudra mettre à jour chaque ligne où ses informations apparaissent, ce qui peut entraîner des incohérences si certaines lignes ne sont pas mises à jour correctement.

Solution : Normalisation

Pour résoudre ce problème de redondance, la base de données doit être **normalisée**. La normalisation consiste à décomposer **une grande table en plusieurs tables plus petites** afin de minimiser la redondance.

Dans cet exemple, nous pouvons diviser la table Achats en plusieurs tables : Clients, Produits, et Achats, tout en reliant les tables par des clés étrangères.

Table Clients (normalisée) :

idClient	Nom	Email
1	Ahmed	ahmed@mail.com
2	Layla	layla@mail.com

Table Produits (normalisée) :

idProduit	NomProduit	Prix
101	Ordinateur portable	800
102	Smartphone	500
103	Clavier mécanique	100

Table Achats (normalisée) :

idClient#	idProduit#	Date d'achat
1	101	2024-01-10
2	102	2024-01-15
1	103	2024-02-05

Résultat :

- ❖ Les informations sur les clients sont stockées une seule fois dans la table Clients.
- ❖ Les informations sur les produits sont également stockées une seule fois dans la table Produits.
- ❖ La table Achats ne contient que des références aux idClient et idProduit, sans redondance des données.

4. Violation des contraintes de domaine :

Une **contrainte de domaine** est une règle qui impose des restrictions sur les valeurs pouvant être stockées dans une colonne d'une table. Ces contraintes assurent que les données respectent des critères spécifiques, tels que le type de données, la plage de valeurs, ou des formats spécifiques.

La **Violation des contraintes de domaine** se produit lorsqu'une valeur insérée dans une colonne ne respecte pas ces règles. Cela peut entraîner des incohérences dans la base de données et des erreurs lors des requêtes ou des mises à jour.

Exemple

Imaginons une base de données qui gère les informations sur des étudiants et qui contient une table appelée **Étudiants**. Supposons que nous avons les contraintes suivantes pour les colonnes de cette table :

- ❖ **ID** : entier positif.
- ❖ **Âge** : entier devant être compris entre 18 et 50 ans.
- ❖ **Email** : doit respecter un format d'email valide *****@****.*****

ID	Nom	Âge	Email
1	Ahmed	20	ahmed@mail.com
2	Layla	17	layla@mail.com
-3	Omar	30	omar@mail.com
4	Mounir	23	mounir@mail@com

Problèmes de violation des contraintes de domaine :

1. **Âge de Layla :**
 - o Layla a un âge de **17**, ce qui ne respecte pas la contrainte d'âge qui exige que les étudiants soient âgés de **18 à 25 ans**.
2. **ID d'Omar :**
 - o L'ID de Omar est **-3**, ne respecte pas la contrainte que l'ID doit être un entier positif.
3. **Email de Mounir :**

- L'email de Zara, **zara@mail@com**, ne respecte pas le format d'un email valide (il contient un @ supplémentaire).

III. Manipulations sur la structure d'une base de données (Définition des données) :

1. Langage SQL :

SQL (Structured Query Language) est un langage standard utilisé pour interagir avec les bases de données relationnelles. Il permet d'effectuer diverses opérations comme la création, la modification, la gestion et la requête des données stockées dans des bases de données sous forme de tables.

2. Définition des bases :

La commande **CREATE DATABASE** est utilisée pour **créer** une nouvelle base de données dans un système de gestion de bases de données (SGBD) comme MySQL

Exemple :

Si vous souhaitez créer une base de données nommée **Ecole**, la requête sera :

```
CREATE DATABASE Ecole;
```

La commande **DROP DATABASE** est utilisée pour **supprimer** une base de données entière, ainsi que toutes les tables et données qu'elle contient. Cette opération est **irréversible** et doit être utilisée avec précaution, car elle supprime définitivement toutes les données de la base de données.

Exemple :

Si vous voulez supprimer la base de données **Ecole**, la commande sera :

```
DROP DATABASE Ecole;
```

3. Définition des tables :

a. Les types des données en SQL :

Type SQL	Description	Exemple d'utilisation
INT	Entier sur 4 octets (32 bits), utilisé pour stocker des nombres entiers.	INT : Stocke des valeurs entières comme 100, -50, 2023. Exemple : AGE INT;
DECIMAL	Nombre décimal avec une précision fixe (nombre total de chiffres) et une échelle (chiffres après la virgule).	DECIMAL(10, 2) : Nombre avec jusqu'à 10 chiffres au total, dont 2 après la virgule. Exemple : PRIX DECIMAL(10, 2);
CHAR	Chaîne de caractères de longueur fixe. Si la longueur n'est pas atteinte, la chaîne est complétée par des espaces.	CHAR(10) : Chaîne fixe de 10 caractères. Exemple : CODE CHAR(10); pour des codes de produit de longueur fixe.
VARCHAR	Chaîne de caractères de longueur variable. La taille est flexible jusqu'à une limite définie.	VARCHAR(50) : Chaîne de longueur variable jusqu'à 50 caractères. Exemple : NOM VARCHAR(50); pour stocker des noms.

Type SQL	Description	Exemple d'utilisation
TEXT	Utilisé pour de longues chaînes de caractères, comme des paragraphes ou des descriptions.	TEXT : Texte long sans limite de taille prédéfinie. Exemple : DESCRIPTION TEXT; pour des descriptions détaillées.
DATE	Représente une date au format AAAA-MM-JJ (année-mois-jour).	DATE : Stocke une date. Exemple : DATE_NAISSANCE DATE; avec une valeur comme 1990-05-12.
TIME	Représente une heure au format HH:MM (heure:minute).	TIME : Stocke une heure. Exemple : HEURE_ENTRÉE TIME; avec une valeur comme 14:30:00.
DATETIME	Combine une date et une heure au format AAAA-MM-JJ HH:MM .	DATETIME : Stocke une date et une heure. Exemple : CREATION DATETIME; avec une valeur comme 2023-12-01 14:30:00.

b. Création d'une table :

La commande **CREATE TABLE** en SQL est utilisée pour créer une nouvelle table dans une base de données. Elle permet de définir la structure de la table en spécifiant les colonnes, leurs types de données et d'autres contraintes comme les clés primaires, les clés étrangères, les valeurs par défaut, etc.

Syntaxe de base :

```
CREATE TABLE NomTable (
    Colonne1 TypeDeDonnée(taille) [Contraintes],
    Colonne2 TypeDeDonnée(taille) [Contraintes],
    , [contraintes définition complète]
);
```

- ❖ **NomTable** : Le nom que vous donnez à la table.
- ❖ **Colonne1, Colonne2** : Les noms des colonnes que vous créez.
- ❖ **TypeDeDonnée** : Le type de données que chaque colonne contiendra (par exemple : INT, VARCHAR, DATE).
- ❖ **Contraintes** : Les règles que vous pouvez appliquer aux colonnes



Liste des contraintes :

Contrainte	Description	Exemple d'utilisation
PRIMARY KEY	Assure que chaque enregistrement dans une table est unique et non nul.	ClientID INT PRIMARY KEY
NOT NULL	Empêche une colonne de contenir des valeurs nulles (obligatoire).	Nom VARCHAR(50) NOT NULL
UNIQUE	Assure que toutes les valeurs d'une colonne (ou d'un ensemble de colonnes) sont uniques dans la table.	Email VARCHAR(100) UNIQUE
DEFAULT	Définit une valeur par défaut pour une colonne si aucune valeur n'est spécifiée lors de l'insertion.	DateInscription DATE DEFAULT "1990-01-01"
REFERENCES	Déclare une clé étrangère (foreign key) qui fait référence à une colonne d'une autre table pour garantir l'intégrité référentielle.	ClientID INT REFERENCES Client(ClientID);
AUTO_INCREMENT	Incrémente automatiquement la valeur à chaque nouvelle insertion. S'applique à une colonne de type entier (INT)	id INT AUTO_INCREMENT PRIMARY KEY

c. Contraintes définition complète :**c.1. FOREIGN KEY**

- ❖ La contrainte **FOREIGN KEY** (clé étrangère) est utilisée pour lier deux tables.
- ❖ Elle fait référence à une colonne ou à un ensemble de colonnes dans une autre table, généralement à une **PRIMARY KEY**.
- ❖ Elle garantit l'**intégrité référentielle** entre les tables. Cela signifie que la valeur de la clé étrangère dans la table enfant doit exister dans la table parent.

Syntaxe dans CREATE TABLE :

```
CREATE TABLE TableEnfant (
    Colonne1 TypeDeDonnée,
    ColonneFK TypeDeDonnée,
    FOREIGN KEY (ColonneFK) REFERENCES TableParent(ColonneParent)
);
```

Exemple :

```
CREATE TABLE Commande (
    CommandeID INT PRIMARY KEY,
    ClientID INT,
    DateCommande DATE,
    FOREIGN KEY (ClientID) REFERENCES Client(ClientID) );
```

Dans cet exemple, **ClientID** dans la table **Commande** est une **clé étrangère** qui fait référence à la colonne **ClientID** de la table **Client**. Cela garantit qu'une commande ne peut être liée qu'à un client existant.

Remarques :

Les clauses **ON UPDATE CASCADE** et **ON DELETE CASCADE** sont utilisées en conjonction avec les **contraintes de clé étrangère (FOREIGN KEY)** dans SQL. Elles définissent le comportement à adopter lorsqu'une ligne dans la table parent est mise à jour ou supprimée, en synchronisant automatiquement les modifications dans la table enfant.

Exemple :

```
CREATE TABLE Clients (
    ClientID INT PRIMARY KEY,
    Nom VARCHAR(100)
);

CREATE TABLE Commandes (
    CommandeID INT PRIMARY KEY,
    ClientID INT,
    FOREIGN KEY (ClientID) REFERENCES Clients(ClientID)
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

Explication :

- **ON UPDATE CASCADE** : Si l'ID d'un client dans **Clients** est mis à jour, la colonne **ClientID** dans **Commandes** sera automatiquement mise à jour.
- **ON DELETE CASCADE** : Si un client est supprimé de **Clients**, toutes ses commandes seront également supprimées de **Commandes**.

c.2. CHECK

- ❖ La contrainte **CHECK** impose une condition qui doit être vraie pour chaque enregistrement dans la colonne ou dans l'ensemble de colonnes spécifié.
- ❖ Elle permet de restreindre les valeurs qu'une colonne peut accepter.

Syntaxe dans CREATE TABLE

```
CREATE TABLE TableName (
    Colonne1 TypeDeDonnée CHECK (Condition),
    Colonne2 TypeDeDonnée,
    ...
);
```

Exemple :

```
CREATE TABLE Produit (
    ProduitID INT PRIMARY KEY,
    Nom VARCHAR(100) NOT NULL,
    Prix DECIMAL(10, 2),
    QuantitéStock INT CHECK (QuantitéStock >= 0)
    -- QuantitéStock doit être supérieure ou égale à 0
);
```

Liste opérateurs de comparaison SQL

Opérateur	Description	Exemple SQL avec CHECK	Explication de l'exemple
=	Vérifie l'égalité entre deux valeurs	CHECK (Age = 30)	Restreint l'âge à la valeur exacte de 30.
<> ou !=	Vérifie l'inégalité (différent de)	CHECK (Age <> 30)	Interdit les enregistrements où l'âge est exactement 30.
>	Vérifie si la valeur de gauche est supérieure à celle de droite	CHECK (Age > 18)	Permet uniquement des âges supérieurs à 18.
<	Vérifie si la valeur de gauche est inférieure à celle de droite	CHECK (Age < 65)	Permet uniquement des âges inférieurs à 65.
>=	Vérifie si la valeur de gauche est supérieure ou égale à celle de droite	CHECK (Salaire >= 3000)	Garantit que le salaire est d'au moins 3000.
<=	Vérifie si la valeur de gauche est inférieure ou égale à celle de droite	CHECK (Salaire <= 10000)	Garantit que le salaire ne dépasse pas 10 000.
BETWEEN	Vérifie si une valeur est comprise entre deux valeurs	CHECK (Age BETWEEN 18 AND 65)	Restreint l'âge à être entre 18 et 65 inclus.
IN	Vérifie si une valeur est présente dans une liste de valeurs	CHECK (Statut IN ('Actif', 'Inactif'))	Limite le statut aux valeurs "Actif" ou "Inactif".

Opérateurs logiques SQL

Opérateur	Description	Exemple	Explication de l'exemple
AND	Renvoie vrai si toutes les conditions sont vraies	CHECK (Age >= 18 AND Age <= 65)	Limite l'âge à être entre 18 et 65.
OR	Renvoie vrai si au moins une condition est vraie	CHECK (Sexe = 'F' OR Sexe = 'M')	Permet que le sexe soit soit "F" soit "M".
NOT	Renverse le résultat d'une condition	CHECK (NOT (Age < 18))	Interdit les enregistrements où l'âge est inférieur à 18.

c.3. Clé primaire

Une **clé primaire composée** est une clé primaire qui est définie sur plusieurs colonnes (ou champs) d'une table. Cela signifie que la combinaison de ces colonnes doit être **unique** et **non nulle** pour chaque enregistrement, mais aucune de ces colonnes ne peut, à elle seule, identifier de manière unique une ligne dans la table.

Pourquoi utiliser une clé primaire composée ?

- Une clé primaire composée est utilisée lorsque plusieurs colonnes ensemble définissent de manière unique un enregistrement, mais qu'aucune colonne prise individuellement ne le fait.
- C'est souvent utilisé dans les **tables de relation** pour modéliser des associations entre plusieurs tables.

Syntaxe de la clé primaire composée dans une requête CREATE TABLE :

```
CREATE TABLE NomTable (
    Colonne1 TypeDeDonnée,
    Colonne2 TypeDeDonnée,
    ...
    PRIMARY KEY (Colonne1, Colonne2)
);
```

Dans cet exemple, la clé primaire est définie sur plusieurs colonnes (**Colonne1** et **Colonne2**), et la combinaison de leurs valeurs doit être unique dans la table.

Exemple : Table Inscription

Imaginons une table **Inscription** qui enregistre l'association entre des étudiants et des cours. Chaque étudiant peut suivre plusieurs cours, et chaque cours peut être suivi par plusieurs étudiants. Pour garantir qu'un étudiant ne s'inscrit pas plusieurs fois au même cours, nous allons utiliser une clé primaire composée des colonnes **EtudiantID** et **CoursID**.

Requête :

```
CREATE TABLE Inscription (
    EtudiantID INT,
    CoursID INT,
    DateInscription DATE,
    PRIMARY KEY (EtudiantID, CoursID),
    FOREIGN KEY (EtudiantID) REFERENCES Etudiant(EtudiantID),
    FOREIGN KEY (CoursID) REFERENCES Cours(CoursID)
);
```

d. Supprimer une table :

La requête SQL **DROP TABLE** est utilisée pour **supprimer** une table existante dans une base de données, ainsi que toutes les données qu'elle contient. Une fois exécutée, cette opération est **irréversible**, c'est-à-dire que la table et ses données sont définitivement supprimées.

Syntaxe :

```
DROP TABLE NomTable;
```

Exemple :

Imaginons que nous avons une table appelée **Client** dans notre base de données, et que nous souhaitons la supprimer :

```
DROP TABLE Client;
```

Cette commande supprime la table **Client** ainsi que toutes les données et les contraintes associées à cette table.

Remarque :

Si la table a des relations (clés étrangères) avec d'autres tables, il peut être nécessaire de désactiver ou de supprimer ces relations avant d'exécuter la commande **DROP TABLE**.

4. Définition des colonnes :

La commande SQL **ALTER TABLE** permet de modifier la structure d'une table existante dans une base de données. Cela inclut l'ajout, la suppression, la modification, ou le renommage de colonnes. Voici une explication de ces fonctionnalités, accompagnée d'exemples :

Opération	Syntaxe	Exemple
Ajouter une colonne	ALTER TABLE NomTable ADD COLUMN NomColonne TypeDeDonnée;	ALTER TABLE Client ADD COLUMN Email VARCHAR(100);
Supprimer une colonne	ALTER TABLE NomTable DROP COLUMN NomColonne;	ALTER TABLE Client DROP COLUMN Adresse;
Modifier une colonne	ALTER TABLE NomTable ALTER COLUMN NomColonne NouveauType;	ALTER TABLE Client ALTER COLUMN Age VARCHAR(3);
Renommer une colonne	ALTER TABLE NomTable RENAME COLUMN AncienNom TO NouveauNom;	ALTER TABLE Client RENAME COLUMN Nom TO NomClient;

5. Définitions des contraintes :

La commande SQL **ALTER TABLE** permet de modifier la structure d'une table en ajoutant ou supprimant des contraintes, ou en activant/désactivant des contraintes existantes. Voici une explication de ces opérations, avec des exemples concrets :

Opération	Syntaxe	Exemple
Ajouter une contrainte	ALTER TABLE NomTable ADD CONSTRAINT NomContrainte TypeDeContrainte (NomColonne) ;	ALTER TABLE Commandes ADD CONSTRAINT fk_client FOREIGN KEY (ClientID) REFERENCES Clients(ClientID) ;
Supprimer une contrainte	ALTER TABLE NomTable DROP CONSTRAINT NomContrainte;	ALTER TABLE Commandes DROP CONSTRAINT fk_client;
Activer une contrainte	ALTER TABLE NomTable ENABLE CONSTRAINT NomContrainte;	ALTER TABLE Commandes ENABLE CONSTRAINT fk_client;
Désactiver une contrainte	ALTER TABLE NomTable DISABLE CONSTRAINT NomContrainte;	ALTER TABLE Commandes DISABLE CONSTRAINT fk_client;

Types de contraintes courantes :

- PRIMARY KEY** : Un identifiant unique pour chaque ligne de la table.
- FOREIGN KEY** : Crée un lien entre deux tables.
- CHECK** : Impose une règle sur les valeurs qu'une colonne peut accepter.

IV. Manipulations sur les données d'une base de données (Manipulations des données) :

1. Insertion des données :

La commande SQL **INSERT INTO** est utilisée pour insérer des nouvelles données dans une table d'une base de données. Vous pouvez insérer des données dans toutes les colonnes d'une table ou seulement dans certaines colonnes spécifiques.

Action	Syntaxe	Exemple
Insérer dans toutes les colonnes	INSERT INTO NomTable VALUES (valeur1, valeur2, ..., valeurN);	INSERT INTO Clients VALUES (1, 'Ahmed', 'ahmed@example.com');
Insérer dans certaines colonnes	INSERT INTO NomTable (Colonne1, Colonne2, ...) VALUES (valeur1, valeur2, ...);	INSERT INTO Clients (Nom, Email) VALUES ('Layla', 'layla@example.com');
Insérer plusieurs lignes	INSERT INTO NomTable (Colonne1, Colonne2, ...) VALUES (valeur1, valeur2), (valeur1, valeur2), ...);	INSERT INTO Clients (Nom, Email) VALUES ('Ahmed', 'ahmed@example.com'), ('Layla', 'layla@example.com');

2. Suppression des données :

La commande **DELETE** permet de supprimer des enregistrements dans une table. Elle peut être utilisée pour supprimer toutes les lignes d'une table ou seulement certaines lignes, en fonction d'une condition spécifiée dans la clause **WHERE**.

Action	Syntaxe	Exemple
Suppression simple	DELETE FROM NomTable WHERE condition;	DELETE FROM Clients WHERE ClientID = 1;
Suppression de tous les enregistrements	DELETE FROM NomTable;	DELETE FROM Clients;

3. Modifications des données :

La requête **UPDATE** permet de modifier les valeurs d'une table en fonction de conditions spécifiques. Sans jointure, elle ne concerne qu'une seule table.

Action	Syntaxe	Exemple
Mettre à jour une valeur	UPDATE table_name SET column_name = new_value WHERE condition;	<i>Modifie le salaire de l'employé dont l'ID est 1.</i> UPDATE Employes SET salaire = 4000 WHERE employe_id = 1;
Mettre à jour plusieurs colonnes	UPDATE table_name SET column1 = new_value1, column2 = new_value2 WHERE condition;	<i>Met à jour le salaire et le département de l'employé avec l'ID 3.</i> UPDATE Employes SET salaire = 4500 , département = 'Marketing' WHERE employe_id = 3;
Mettre à jour toutes les lignes (sans WHERE)	UPDATE table_name SET column_name = new_value;	<i>Modifie le salaire de tous les employés dans la table.</i> UPDATE Employes SET salaire = 5000;
Modifier une valeur existante	UPDATE table_name SET column_name = column_name + increment_value WHERE condition;	<i>Augmente le salaire des employés du département IT de 500.</i> UPDATE Employes SET salaire = salaire + 500 WHERE département = 'IT';

4. Sélection des données

La requête **SELECT** en SQL permet de **récupérer des données** dans une table de base de données. C'est l'une des opérations les plus basiques et les plus importantes en SQL.

Structure de base d'une requête SELECT

```
SELECT colonnes
FROM table;
```

LYCÉE MONJI
SLIM SBIBA

❖ SELECT

- Cela signifie : **choisir** les colonnes que tu veux afficher.
- Par exemple : SELECT nom, age signifie que tu veux récupérer les colonnes nom et age.
- SELECT * signifie sélectionner tous les colonnes.

❖ FROM

- C'est là où tu indiques **de quelle table(s)** tu veux récupérer les données. Par exemple : FROM utilisateurs.

Exemple:

Si tu as une table appelée utilisateurs avec ces données :

id	nom	age
1	Alice	25
2	Bob	30
3	Carol	22

Une requête pour afficher les noms et âges de tous les utilisateurs ressemble à ça :

```
SELECT nom, age
FROM utilisateurs;
```

Résultat :

nom	age
Alice	25
Bob	30
Carol	22

a. Filtrage des résultats :

La clause **WHERE** en SQL est utilisée pour filtrer les enregistrements dans une requête **SELECT** en fonction d'une ou plusieurs conditions. Elle permet de récupérer **uniquement** les lignes qui répondent aux **critères spécifiés**.

Syntaxe générale

```
SELECT colonne1, colonne2, ...
FROM table
WHERE condition;
```

LYCÉE MONJI
SLIM SBIBA

Fonctionnement de la clause WHERE

- **SELECT** : Utilisé pour spécifier les colonnes que vous souhaitez récupérer.

- **FROM** : Indiquer la(les) table(s) à partir de laquelle les données sont extraites.
- **WHERE** : Spécifie les conditions que les lignes doivent respecter pour être incluses dans les résultats.

Exemple1 :

Supposons que nous ayons une table employes comme ci-dessous :

id_employe	nom	age	salaire
1	Alice	30	50000
2	Bob	45	60000
3	Charlie	25	40000
4	David	35	55000

Si vous souhaitez récupérer les informations des employés qui ont un salaire supérieur à 50 000

```
SELECT nom, salaire
FROM employes
WHERE salaire > 50000;
```

Résultat :

nom	salaire
Bob	60000
David	55000

Ici, la condition `salaire > 50000` filtre les lignes pour ne conserver que celles où le salaire est supérieur à 50 000.

Exemple 2 : Utilisation avec plusieurs conditions (opérateur AND)

Si vous souhaitez récupérer les informations des employés âgés de plus de 30 ans et ayant un salaire supérieur à 50 000 :

```
SELECT nom, age, salaire
FROM employes
WHERE age > 30 AND salaire > 50000;
```

Résultat :

nom	age	salaire
Bob	45	60000
David	35	55000

Ici, les deux conditions doivent être vraies (`age > 30` et `salaire > 50000`) pour que l'enregistrement soit inclus dans le résultat.

Exemple 3 : Utilisation de l'opérateur OR

Si vous souhaitez récupérer les informations des employés qui ont soit plus de 30 ans soit un salaire supérieur à 50 000 :

```
SELECT nom, age, salaire
FROM employes
WHERE age > 30 OR salaire > 50000;
```

Résultat :

nom	age	salaire
Alice	30	50000
Bob	45	60000
David	35	55000

Avec l'opérateur **OR**, il suffit qu'une seule des conditions soit vraie pour que la ligne soit incluse.

Exemple 4 : Utilisation de l'opérateur LIKE

Si vous souhaitez récupérer les noms des employés dont le nom commence par "A" :

```
SELECT nom
FROM employes
WHERE nom LIKE 'A%';
```

Résultat :

nom
Alice

Ici, l'opérateur **LIKE** est utilisé pour rechercher des correspondances partielles. '**A%**' signifie "commence par A".

Voici un tableau des jokers principaux utilisés avec l'opérateur LIKE en SQL

Joker	Description	Exemple de motif	Explication du motif
%	Représente zéro, un ou plusieurs caractères	C%	Correspond à toute chaîne commençant par "C"
-	Représente exactement un caractère	C_____	Correspond à toute chaîne commençant par "C" suivie de 7 autres caractères

Exemple 5 : IS NULL IS NOT NULL :

En SQL, les opérateurs `IS NULL` et `IS NOT NULL` sont utilisés pour vérifier la présence ou l'absence de valeurs `NULL` dans une colonne. Voici leurs fonctions et des exemples d'utilisation.

Soit table appelée employes suivant :

id_employe	nom	age	salaire
1	Alice	30	50000
2	Bob	NULL	60000
3	Charlie	25	NULL
4	David	35	55000

Requête avec IS NULL :

```
SELECT nom, age
FROM employes
WHERE age IS NULL;
```

Résultat :

nom	age
Bob	NULL

Requête avec IS NOT NULL :

```
SELECT nom, salaire
FROM employes
WHERE salaire IS NOT NULL;
```

Résultat :

nom	salaire
Alice	50000
Bob	60000
David	55000

b. Jointure :

En SQL, une **jointure** est utilisée pour combiner des lignes provenant de deux ou plusieurs tables en fonction d'une condition qui lie les tables, souvent basée sur une relation entre une clé primaire et une clé étrangère. Pour faire la jointure les tables sont listées dans la clause `FROM`, et la condition de jointure est définie dans la clause `WHERE`.

Syntaxe :

```
SELECT Colonnes
FROM Table1, Table2
WHERE Table1.Colonne = Table2.Colonne;
```

Exemple :

Soient deux tables : **Clients** et **Commandes**. La table **Clients** contient les informations des clients, et la table **Commandes** contient les commandes effectuées par ces clients. Les deux tables sont reliées par la colonne **ClientID** (la clé primaire dans **Clients** et la clé étrangère dans **Commandes**).

Clients :

ClientID	Nom
1	Ahmed
2	Layla

Commandes :

CommandeID	ClientID	Produit
101	1	Ordinateur
102	2	Smartphone
103	1	Imprimante

Requête avec jointure:

```
SELECT A.Nom AS Client, C.Produce
FROM Clients AS A, Commandes AS C
WHERE A.ClientID = C.ClientID;
```

Résultat :

Client	Produit
Ahmed	Ordinateur
Layla	Smartphone
Ahmed	Imprimante

Explication :

- ✓ La requête sélectionne les noms des clients dans la table **Clients** et les produits commandés dans la table **Commandes**.
- ✓ La condition **WHERE Clients.ClientID = Commandes.ClientID** relie les deux tables en fonction du **ClientID**.
- ✓ Le résultat montre les produits commandés par chaque client.
- ✓ L'opérateur **AS** permet de créer des alias

c. Fonctions sur les types DATE en SQL

SQL fournit plusieurs fonctions pour manipuler les dates, permettant d'extraire des informations spécifiques comme le jour, le mois, l'année, ou encore la date actuelle. Voici un aperçu des fonctions couramment utilisées sur les types **DATE**.

Liste des fonctions :

Fonction	Description	Exemple de requête	Résultat
DAY()	Renvoie le jour d'une date.	SELECT DAY('2024-10-06');	6
MONTH()	Renvoie le mois d'une date.	SELECT MONTH('2024-10-06');	10
YEAR()	Renvoie l'année d'une date.	SELECT YEAR('2024-10-06');	2024
NOW()	Renvoie la date et l'heure actuelles.	SELECT NOW();	2025-07-03 14:30:15
DATE()	Renvoie la partie date (sans l'heure).	SELECT DATE(NOW());	2024-10-06
ADDDATE()	Ajoute un intervalle de temps (jours, mois, etc.) à une date donnée	SELECT ADDDATE("2025-10-10", 15)	2025-10-25
ADDDATE(date, INTERVAL valeur unité)	✓ valeur : un nombre entier (positif ou négatif). ✓ unité : l'unité de temps (par exemple : DAY, MONTH, YEAR)	SELECT ADDDATE("2025-10-10" , INTERVAL 3 MONTH)	2026-01-10
DATEDIFF()	Calcule la différence (en jours) entre deux dates. Il retourne un entier : $\text{DATEDIFF(date1, date2)} = \text{date1} - \text{date2}$	SELECT DATEDIFF('2025-07-15', '2025-07-01')	14

d. Fonctions d'agrégation en SQL

Les **fonctions d'agrégation** en SQL sont utilisées pour effectuer des calculs sur un ensemble de lignes et renvoyer une valeur unique.

Voici les principales fonctions d'agrégation en SQL

Fonction	Description	Exemple de requête	Résultat
COUNT()	Compte le nombre de lignes.	SELECT COUNT(*) FROM Commandes;	Nombre total de lignes
SUM()	Calcule la somme d'une colonne numérique.	SELECT SUM(Prix) FROM Commandes;	Somme des prix

Fonction	Description	Exemple de requête	Résultat
AVG()	Calcule la moyenne d'une colonne numérique.	SELECT AVG(Prix) FROM Commandes;	Moyenne des prix
MAX()	Renvoie la valeur maximale d'une colonne.	SELECT MAX(Prix) FROM Commandes;	Prix maximum
MIN()	Renvoie la valeur minimale d'une colonne.	SELECT MIN(Prix) FROM Commandes;	Prix minimum

e. Groupement en SQL

Le **groupement** en SQL est utilisé pour regrouper les lignes d'une table qui partagent des valeurs communes dans une ou plusieurs colonnes, puis appliquer des **fonctions d'agrégation** (comme COUNT(), SUM(), AVG(), etc.) sur chaque groupe. La clause utilisée pour effectuer ce regroupement est **GROUP BY**.

Syntaxe de base :

```
SELECT colonne1, fonction_agrégation(colonne2)
FROM table
GROUP BY colonne1;
```

Fonctionnement du groupement avec GROUP BY :

- Regroupement des données** : Le **GROUP BY** regroupe les lignes ayant des valeurs identiques dans les colonnes spécifiées.
- Application des fonctions d'agrégation** : Les fonctions d'agrégation sont appliquées aux groupes formés.
- Obtenir un résultat pour chaque groupe** : La requête renvoie une seule ligne par groupe avec les résultats des fonctions d'agrégation.

Exemple simple

Supposons qu'on ait une table **Ventes** avec les données suivant :

id	Produit	Quantite	Prix
1	Ordinateur	1	1200
2	Souris	3	15
3	Clavier	2	50
4	Ordinateur	1	1200
5	Souris	2	15
6	Casque Audio	1	30
7	Clavier	1	50

On souhaite connaître la **quantité totale vendue par produit**.

```
SELECT Produit, SUM(Quantite) AS QuantiteTotale
FROM Ventes
GROUP BY Produit;
```

Résultat :

Produit	QuantiteTotale
Ordinateur	2
Souris	5
Clavier	3
Casque Audio	1

Utilisation de HAVING avec GROUP BY

La clause **HAVING** est utilisée pour filtrer les résultats après le **groupement**. Contrairement à **WHERE**, qui filtre les lignes avant le groupement, **HAVING** filtre les groupes une fois qu'ils sont formés.

Exemple avec HAVING

Supposons que nous voulons obtenir uniquement les produits dont la **quantité totale vendue** est supérieure ou égal à **3**.

```
SELECT Produit, SUM(Quantite) AS QuantiteTotale
FROM Ventes
GROUP BY Produit
HAVING SUM(Quantite) >= 3;
```

Résultat :

Produit	QuantiteTotale
Souris	5
Clavier	3

Groupement sur plusieurs colonnes

Il est possible de faire un **groupement sur plusieurs colonnes**. Cela permet de créer des sous-groupes selon plusieurs critères.

Exemple

Une entreprise de livraison stocke les **commandes livrées** dans une table Livraisons. Chaque livraison est associée à un **gouvernorat**, une **ville**, et une **quantité de colis livrés**.

id	gouvernorat	ville	nb_colis
1	Tunis	La Marsa	10
2	Tunis	Ariana Ville	8
3	Sfax	Sfax Ville	15
4	Sfax	Sakiet Ezzit	7
5	Tunis	La Marsa	5
6	Sfax	Sfax Ville	12
7	Tunis	Ariana Ville	4

Soit la requête SQL suivante :

```
SELECT gouvernorat, ville, SUM(nb_colis) AS total_colis
FROM Livraisons
GROUP BY gouvernorat, ville;
```

Résultat:

gouvernorat	ville	total_colis
Tunis	La Marsa	15
Tunis	Ariana Ville	12
Sfax	Sfax Ville	27
Sfax	Sakiet Ezzit	7

f. Tri en SQL

La clause **ORDER BY** en SQL permet de trier les résultats d'une requête **par ordre croissant ou décroissant** en fonction d'une ou plusieurs colonnes. Par défaut, les résultats sont triés en **ordre croissant**. On peut également spécifier un **ordre décroissant** à l'aide de la commande **DESC**.

Syntaxe de base :

```
SELECT colonne1, colonne2, ...
FROM table
ORDER BY colonne1 [ASC|DESC], colonne2 [ASC|DESC];
```

- **ASC** : Trie par ordre croissant (par défaut, si aucun ordre n'est spécifié).
- **DESC** : Trie par ordre décroissant.

Exemples d'utilisation

Exemple1 : Trier par une seule colonne (ordre croissant)

Supposons que nous avons une table **Produits** suivante :

refProduit	nomProduit	quantité	prix
P001	Ordinateur Portable	10	1500.00
P002	Souris Sans Fil	50	25.00
P003	Clavier Mécanique	30	70.00
P004	Écran 24 pouces	20	300.00
P005	Disque SSD 1To	15	120.00

Nous voulons obtenir les produits **triés par prix** (ordre croissant).

```
SELECT NomProduit, Prix
FROM Produits
ORDER BY Prix;
```

Résultat:

NomProduit	Prix
Souris Sans Fil	25.00
Clavier Mécanique	70.00
Disque SSD 1To	120.00
Écran 24 pouces	300.00
Ordinateur Portable	1500.00

Exemple 2 : Trier par numéro de colonne

Si nous voulons obtenir la même liste **triée par prix décroissant**, on utilise **DESC**.

```
SELECT NomProduit, Prix
FROM Produits
ORDER BY 2 DESC;
```

Résultat:

NomProduit	Prix
Ordinateur Portable	1500.00
Écran 24 pouces	300.00
Disque SSD 1To	120.00
Clavier Mécanique	70.00
Souris Sans Fil	25.00

Exemple 3 : Trier par plusieurs colonnes

On peut trier les résultats par plusieurs colonnes. Par exemple, si nous voulons **trier les produits par prix croissant et, en cas d'égalité, trier par nom de produit en ordre alphabétique**.

Soit la table livraisons suivante :

id	gouvernorat	ville	nb_colis
1	Tunis	La Marsa	10
2	Tunis	Ariana Ville	8
3	Sfax	Sfax Ville	15
4	Sfax	Sakiet Ezzit	7
5	Tunis	La Marsa	5
6	Sfax	Sfax Ville	12
7	Tunis	Ariana Ville	4

Soit la requête SQL permettant de lister les gouvernorats et les villes (tri par **gouvernorat (A→Z)** puis par **ville (A→Z)**)

```
SELECT gouvernorat, ville, nb_colis
FROM Livraisons
ORDER BY gouvernorat ASC, ville ASC;
```

Résultat :

gouvernorat	ville	nb_colis
Sfax	Sakiet Ezzit	7
Sfax	Sfax Ville	15
Sfax	Sfax Ville	12
Tunis	Ariana Ville	8
Tunis	Ariana Ville	4
Tunis	La Marsa	10
Tunis	La Marsa	5

g. Sous-requête :

Une sous-requête est une requête imbriquée utiliser dans la clause **WHERE** pour filtrer les résultats de la requête principale en fonction du résultat d'une autre requête. Cela permet de traiter des conditions complexes dans le filtrage des données.

Syntaxe de base :

```
SELECT colonne1, colonne2, ...
FROM table
WHERE colonne opérateur (sous-requête);
```

- La **sous-requête** retourne une valeur ou un ensemble de valeurs que la requête principale utilise pour filtrer les résultats.
- Les **opérateurs** couramment utilisés avec les sous-requêtes sont des opérateurs de comparaison =, IN, NOT IN , >, <, etc.

Exemple :Sous-requête avec l'opérateur =

Trouver les employés dont le **salaire** est égal au **salaire minimum** de la table **employées** suivante :

ID	Nom	Salaire
1	Amal	1200
2	Karim	1800
3	Sana	1200
4	Mehdi	2000

Requête :

```
SELECT Nom, Salaire
FROM Employes
WHERE Salaire = (SELECT MIN(Salaire) FROM Salaries);
```

Résultat :

Nom	Salaire
Amal	1200
Sana	1200

Explication : La sous-requête (**SELECT MIN(Salaire) FROM Salaries**) renvoie le salaire minimum, et la requête principale sélectionne les employés ayant ce salaire.

Exemple 2 Sous-requête avec l'opérateur IN

Trouver les produits qui ont été **vendus en 2023**.

Données de la table Produits :

IDProduit	NomProduit
P001	Ordinateur
P002	Souris
P003	Clavier
P004	Écran

Données de la table Ventes :

IDVente	IDProduit	Annee
V01	P001	2023
V02	P003	2023
V03	P004	2022

Requête :

```
SELECT NomProduit
FROM Produits
WHERE IDProduit IN (SELECT IDProduit FROM Ventes WHERE Annee = 2023);
```

Résultat :

NomProduit
Ordinateur
Clavier

Explication : La sous-requête (**SELECT IDProduit FROM Ventes WHERE Annee = 2023**) renvoie les identifiants des produits vendus en 2023. La requête principale sélectionne uniquement les produits dont l'ID se trouve dans cette liste.



Chapitre II HTML



I. Définition et structure générale :

HTML (HyperText Markup Language) est le langage standard utilisé pour structurer et afficher le contenu des pages web. Il utilise des **balises** pour organiser le texte, les images, les vidéos, les liens, et d'autres éléments multimédias sur un site. Chaque élément sur une page web est défini par une balise HTML, permettant aux navigateurs de comprendre comment présenter le contenu. HTML5 est la dernière version du langage HTML utilisé pour structurer le contenu des pages web. Il introduit de nouvelles balises sémantiques et des API pour rendre les pages web plus interactives et accessibles.

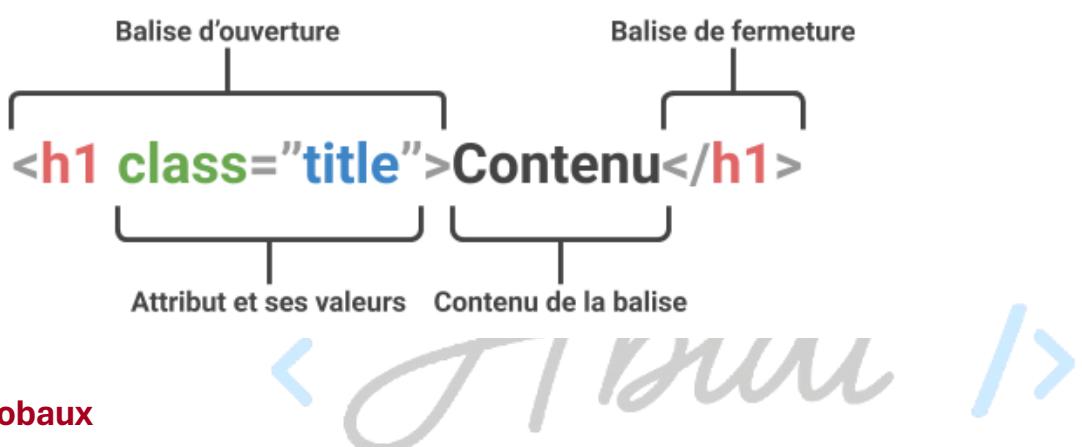
Squelette d'une page HTML5 :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Titre de la page</title>
</head>
<body>
Bonjour HTML
</body>
</html>
```

Balises de structure :

- <html> : Définit le début du document HTML.
- <head> : Contient les informations sur le document (métadonnées, titre, liens vers CSS, etc.).
- <body> : Contient le contenu visible de la page.

Structure d'une balise html :



II. Attributs globaux

Un **attribut global** est un attribut que l'on peut appliquer à **toutes** les balises HTML, quelle que soit leur nature (<div>, <p>, <input>, etc.).

Les principaux attributs globaux

Attribut	Rôle
class	Attribue une ou plusieurs classes CSS à un élément.
id	Donne un identifiant unique à un élément. Permet de le cibler précisément (CSS ou JavaScript).
hidden	Masque l'élément dans la page (il n'est pas visible à l'affichage).
lang	Indique la langue du contenu de l'élément (utile pour l'accessibilité et le SEO).
style	Permet d'écrire des règles CSS directement dans l'élément (à éviter en grande quantité).
title	Affiche un infobulle au survol de l'élément avec la souris.

III. Eléments d'en-tête

1. **style:**

La balise <style> permet d'**écrire du code CSS directement dans un fichier HTML**, généralement dans la section <head>. Elle sert à **appliquer des styles** (couleur, taille, marges, etc.) à des éléments HTML.

```
<head>
  <meta charset="UTF-8">
  <title>Exemple avec style</title>
  <style>
    h1 {
      color: red;
      font-family: cursive;
    }
  </style>
</head>
<body>
  <h1>Bienvenue sur mon site</h1>
</body>
```

Bienvenue sur mon site

2. **<link>**

La balise <link> est utilisée pour établir une connexion entre le document HTML et des ressources externes, telles que des feuilles de style CSS. Elle se place dans l'élément <head> du document.

Exemple :

```

<!DOCTYPE html>
<html lang="fr">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Exemple de Balise Link</title>
<!-- Lien vers une feuille de style CSS -->
<link rel="stylesheet" href="styles.css">
</head>
<body>
<h1>Bienvenue sur ma page</h1>
</body>
</html>

```

- ⇒ Dans cet exemple, la balise `<link>` est utilisée pour relier le document HTML à une feuille de style CSS externe nommée "**styles.css**".

3. <meta>

La balise `<meta>` est utilisée pour fournir des métadonnées au document HTML. Elle peut définir des informations sur le caractère encodé, la description, les mots-clés, et d'autres paramètres importants pour le référencement et le fonctionnement du document.

Exemple :

```

<!DOCTYPE html>
<html lang="fr">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Un exemple de page HTML avec des balises meta.">
<meta name="keywords" content="HTML, CSS, exemple">
<meta name="author" content="Auteur du Document">
<title>Exemple de Balise Meta</title>
</head>
<body>
<h1>Page d'exemple</h1>
</body>
</html>

```

- ⇒ Ici, la balise `<meta>` est utilisée pour définir l'encodage des caractères, la description, les mots-clés, et l'auteur de la page.

4. <script>

La balise `<script>` est utilisée pour inclure du code JavaScript dans un document HTML. Elle peut être utilisée pour incorporer des scripts directement dans le document ou pour référencer des fichiers JavaScript externes.

Exemple :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Exemple de Balise Script</title>
<!-- Inclusion d'un script JavaScript externe -->
<script src="script.js"></script>
</head>
<body>
<h1>Bienvenue sur ma page</h1>
<!-- Code JavaScript en ligne -->
<script>
alert('La page est chargée !');
</script>
</body>
</html>
```

- ⇒ Dans cet exemple, la balise `<script>` est utilisée pour inclure un fichier JavaScript externe (`script.js`) ainsi que pour insérer un script JavaScript en ligne qui affiche une alerte lorsque la page est chargée.

5. `<title>`

La balise `<title>` est utilisée pour définir le titre du document HTML. Ce titre est affiché dans la barre de titre du navigateur ou l'onglet de la page. Il est essentiel pour le référencement et l'expérience utilisateur.

Exemple :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Exemple de Balise Title</title>
</head>
<body>
<h1>Contenu de la Page</h1>
</body>
</html>
```



- ⇒ Dans cet exemple, la balise `<title>` définit le titre de la page comme "Exemple de Balise Title", qui sera affiché dans la barre de titre du navigateur.

IV. Eléments de structuration de texte

1. Les entêtes :

En HTML, les **en-têtes** (ou **titres**) sont représentés par les balises <h1> à <h6>. Ce sont des balises **sémantiques** utilisées pour structurer hiérarchiquement le contenu d'une page web. Les balises d'en-têtes sont utilisées pour la **structure logique** du contenu, **pas pour la mise en forme visuelle** (on utilise le CSS pour cela).

Balise	Signification	Usage
<h1>	Titre principal	souvent le titre de la page
<h2>	Sous-titre du <h1>	Sections principales
<h3>	Sous-sous-titre du <h2>	Sous-sections
<h4> à <h6>	Niveaux inférieurs	Pour structurer en profondeur

Exemple d'utilisation :

```
<h1>Centre de formation  
SUFAS</h1>
```

```
<h2>Nos formations</h2>  
<h3>Développement web</h3>  
<h4>HTML, CSS,  
JavaScript</h4>  
<h4>PHP & MySQL</h4>  
  
<h3>Réseaux  
informatiques</h3>  
<h4>CCNA</h4>  
<h4>Sécurité réseau</h4>  
  
<h2>À propos de nous</h2>  
<h3>Notre mission</h3>  
<h3>Nos formateurs</h3>
```

Centre de formation SUFAS

Nos formations

Développement web

HTML, CSS, JavaScript

PHP & MySQL

Réseaux informatiques

CCNA

Sécurité réseau

À propos de nous

Notre mission

Nos formateurs

2. Les conteneurs du texte :

En HTML, certaines balises sont utilisées pour **structurer, afficher et mettre en valeur du texte**. Le tableau ci-dessous présente quelques-unes de ces balises :

Balise	Rôle	Exemple d'utilisation
<p>	Conteneur de paragraphe. Elle permet d'écrire des blocs de texte.	<p>Bienvenue sur notre site de formation.</p>
		Bienvenue sur notre site de formation.
<cite>	Sert à citer le titre d'une œuvre (livre, film, article...).	<cite>Le Petit Prince</cite>
		Le Petit Prince
<address>	Utilisée pour afficher une adresse postale ou électronique .	<address>contact@ecole.tn</address>
		contact@ecole.tn
<output>	Affiche le résultat d'un calcul ou d'une interaction (souvent avec JS).	<output>42</output> (résultat d'un formulaire)
		42
<mark>	Met en surbrillance une portion de texte (souvent pour attirer l'attention).	<mark>Urgent</mark> OU <mark>Examen final</mark>
		Urgent ou Examen final

3. Les séparateurs :

En HTML, deux balises sont utilisées pour **séparer visuellement ou logiquement le texte** sans créer de nouvelle section :
 pour un **saut de ligne**, et <hr> pour une **ligne de séparation horizontale**.

Exemple d'utilisation :

```

<h2>Programme de la journée</h2>
<p>Accueil des
participants<br>Présentation du
formateur<br>Pause café</p>

<hr>

<p>Début des ateliers à 10h00</p>

```

Programme de la journée

Accueil des participants
Présentation du formateur
Pause café

Début des ateliers à 10h00

4. Les liens hypertextes :

La balise <a> (pour *anchor* = ancre) est utilisée pour créer un **lien hypertexte**, qui permet de :

- ✓ Naviguer vers une autre page,
- ✓ Se déplacer à l'intérieur de la même page.

Attributs essentiels :

Attribut	Rôle	Exemple
href	(<i>hypertext reference</i>) : définit l' URL vers laquelle le lien pointe	href="https://example.com"
target	Détermine où le lien s'ouvrira	target="_blank"

Valeurs possibles de target :

Valeur	Effet
_self	Ouvre dans la même fenêtre/onglet (valeur par défaut)
_blank	Ouvre dans un nouvel onglet ou fenêtre
_parent	Ouvre dans le cadre parent (rarement utilisé)

Exemple d'utilisation :

```
<a href="https://www.sufas.tn"
target="_blank"> Visitez notre
site</a>
```

[Visitez notre site](#)

- ⇒ Le clic sur le lien ouvre le site indiquer en href dans **un nouvel onglet** sans quitter la page actuelle

```
<h1>Bienvenue</h1>
<p><a href="#contact">Aller à
la section Contact</a></p>

<p>Lorem ipsum dolor sit
amet... (beaucoup de texte)</p>

<!-- Cible -->
<h2 id="contact">Contact</h2>
<p>Email : contact@sufas.tn</p>
```

Bienvenue

[Aller à la section Contact](#)

Lorem ipsum dolor sit amet... (beaucoup de texte)

Contact

Email : contact@sufas.tn

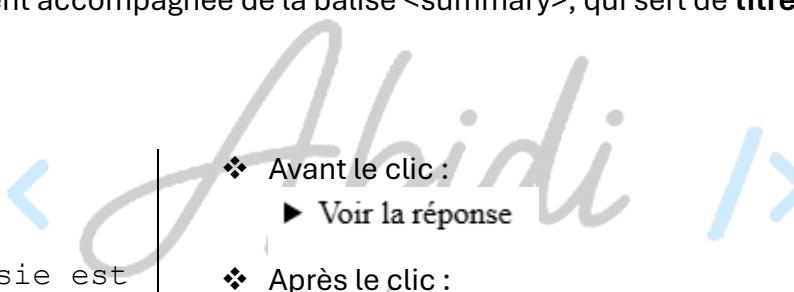
- ⇒ Le lien [cherche un élément avec id="contact". Quand on clique, le navigateur **défile automatiquement jusqu'à cet élément.**](#contact)

5. Contenu pliable :

La balise `<details>` permet de créer un **contenu repliable** (rétractable) que l'utilisateur peut **déployer ou masquer**. Elle est souvent utilisée pour afficher des informations supplémentaires sans encombrer la page. Elle est généralement accompagnée de la balise `<summary>`, qui sert de **titre cliquable**.

Exemple d'utilisation :

```
<details>
  <summary>Voir la
réponse</summary>
  <p>La capitale de la Tunisie est
Tunis.</p>
</details>
```

- 
- ❖ Avant le clic :
 - ▶ Voir la réponse
 - ❖ Après le clic :
 - ▼ Voir la réponse

La capitale de la Tunisie est Tunis.

V. Les éléments de structuration de media

1. Les images :

La balise `` sert à **insérer une image** dans une page web. Elle est **auto-fermante** (pas besoin de ``).

Attributs principaux :

Attribut	Rôle
src	(source) Chemin de l'image à afficher
alt	(texte alternatif) Description affichée si l'image ne s'affiche pas

Exemple d'utilisation :

```
<h2> image de l'école </h2>

```

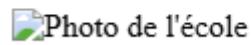
- ❖ Si l'image est bien chargée :

image de l'école



- ❖ En cas de problème de chargement :

image de l'école



2. Les Vidéos :

La balise `<video>` permet d'**intégrer une vidéo** directement dans une page web, avec ou sans contrôles (lecture, pause, volume...).

Attributs principaux

Attribut	Rôle
src	Spécifie le chemin de la vidéo
controls	Affiche les contrôles de lecture (obligatoire pour interaction)

Exemple d'utilisation :

LYCÉE MONJI
SLIM SBIBA

```
<video
src="videos/presentation.mp4">
Votre navigateur ne supporte pas
la balise vidéo.
</video>
```



```
<video controls
src="videos/presentation.mp4" >
Votre navigateur ne supporte pas
la balise vidéo.
</video>
```



3. Les audios :

La balise **<audio>** permet d'**insérer un fichier audio** (comme une musique, un podcast ou un message vocal) dans une page web.

Attributs principaux

Attribut	Rôle
src	Spécifie le chemin du fichier audio
controls	Affiche les contrôles de lecture (lecture, pause, volume...)

Exemple d'utilisation :

```
<audio src="sons/theme.mp3"
controls>
Votre navigateur ne supporte pas
la balise audio.
</audio>
```

LYCÉE MONJI
SLIM SBIBA

4. Balise source :

La balise `<source>` est utilisée à l'intérieur de balises multimédias comme `<audio>`, `<video>` pour **spécifier plusieurs sources alternatives**. Le navigateur choisit la première source qu'il peut lire selon le type et les conditions.

Attributs principaux :

Attribut	Rôle	Exemple
src	Chemin du fichier à charger (audio, vidéo ou image)	src="media/video.mp4"
type	Type du fichier (video/mp4, video/avi, audio/mp3, ...)	type="video/mp4"

Exemple d'utilisation :

```
<audio controls>
  <source src="sons/theme.mp3" type="audio/mpeg">
  <source src="sons/theme.ogg" type="audio/ogg">
    Votre navigateur ne supporte pas l'audio HTML5.
</audio>
```

5. Les figures :

La balise `<figure>` sert à **regrouper un contenu autonome illustratif**, comme :

- une image
- une vidéo
- un extrait de code, etc.

Elle est souvent accompagnée de la balise `<figcaption>` pour fournir une **légende descriptive**.

Exemple d'utilisation :

```
<figure>
  
  <figcaption>Un beau paysage de
  montagne au coucher du
  soleil.</figcaption>
</figure>
```



Un beau paysage de montagne au coucher du soleil.

VI. Les conteneurs :

Les balises `<div>`, `` et `<iframe>` sont des **conteneurs** utiles pour organiser le contenu dans une page web.

Balise	Rôle	Type	Exemple d'utilisation
<div>	Conteneur générique en bloc pour structurer la page	Bloc	<div class="boite">...</div>
	Conteneur en ligne pour styliser une partie de texte	En ligne	mot rouge
<iframe>	Affiche une page web externe intégrée dans la page actuelle	Bloc	<iframe src="https://example.com"></iframe>

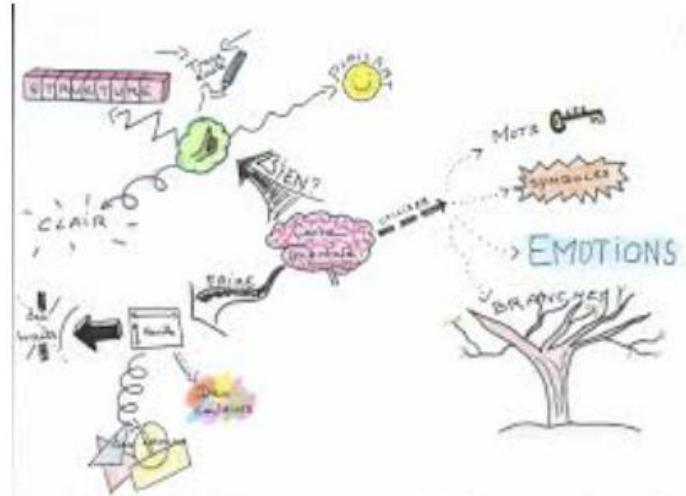
Exemple d'utilisation :

```
<div class="encadré">
    <h2>Actualité du jour</h2>
    <p>Bienvenue sur notre
    <span style="color:red">site
    web éducatif</span> !</p>
</div>

<h3>Carte interactive</h3>
<iframe src="carte.html"
name="if"></iframe>
```

Actualité du jour

Bienvenue sur notre **site web éducatif** !

Carte interactive**Remarque :**

On peut utiliser l'attribut target pour ouvrir le lien dans une <iframe> nommée. Par exemple :

```
<a href="info.html" target="cadre1">Voir plus d'information </a>
```

Permet de charger la page "info.html" dans une iframe nommée "cadre1".

VII. Les tableaux :

En HTML, on utilise plusieurs balises spécifiques pour structurer un tableau de façon claire et sémantique.



Balise	Rôle
<table>	Définit l'ensemble du tableau.
<caption>	Ajoute un titre ou une légende au tableau.
<thead>	Contient les lignes de l'en-tête du tableau.
<tbody>	Contient les lignes principales (corps du tableau).
<tfoot>	Contient les lignes de fin, souvent pour des totaux.
<tr>	Définit une ligne du tableau.
<th>	Définit une cellule d'en-tête (texte généralement en gras et centré).
<td>	Définit une cellule de données normale dans une ligne.

Exemple d'utilisation :

```

<table>
  <caption>Notes des
élèves</caption>
  <thead>
    <tr>
      <th>Nom</th>
      <th>Maths</th>
      <th>Physique</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Amine</td>
      <td>14</td>
      <td>16</td>
    </tr>
    <tr>
      <td>Sonia</td>
      <td>17</td>
      <td>15</td>
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <td>Moyenne</td>
      <td>15.5</td>
      <td>15.5</td>
    </tr>
  </tfoot>
</table>

```

Notes des élèves		
Nom	Maths	Physique
Amine	14	16
Sonia	17	15
Moyenne	15.5	15.5



VIII. Les Listes :

En HTML, les listes permettent de structurer des éléments de manière ordonnée ou non ordonnée, en utilisant les balises :

- ✓ pour les listes à puces
- ✓ pour les listes numérotées
- ✓ pour créer un élément de liste.

Attributs principaux :

Liste	Attribut	Rôle
ul	type	Change le style des puces (disc, circle, square).
ol	type	Change le style des numéros : <ul style="list-style-type: none"> ❖ "1 Nombres (1, 2, 3...) — par défaut ❖ "A" Lettres majuscules (A, B, C...) ❖ "I" Chiffres romains majuscules (I, II...) ❖
	start	Définit à quel numéro la liste commence.
	reversed	Inverse l'ordre de numérotation (ex. de 5 à 1 au lieu de 1 à 5).

Exemple d'utilisation listes imbriquées :

```
<ol type="I" start="5" reversed>
  <li>Introduction</li>
  <li>Partie théorique
    <ul type="circle">
      <li>Définitions</li>
      <li>Notions de base</li>
    </ul>
  </li>
  <li>Partie pratique
    <ul type="square">
      <li>Exercice 1</li>
      <li>Exercice 2</li>
    </ul>
  </li>
</ol>
```

- V. Introduction
- IV. Partie théorique
 - Définitions
 - Notions de base
- III. Partie pratique
 - Exercice 1
 - Exercice 2

IX. Les formulaires :

La balise <form> permet de **créer un formulaire** pour recueillir des données saisies par l'utilisateur (nom, email, mot de passe, etc.). Ces données peuvent ensuite être envoyées à un serveur pour traitement.



Attributs principaux :

Attribut	Rôle	Valeurs possibles
<code>action</code>	URL où les données du formulaire sont envoyées	Une URL ou page valide
<code>method</code>	Méthode d'envoi des données	<code>get</code> : ajoute les données à l'URL (claire) <code>post</code> : envoie les données dans le corps de la requête.
<code>target</code>	Indique où afficher la réponse après envoi	<code>_self</code> : même onglet (par défaut) <code>_blank</code> : nouvel onglet <code>_parent</code> : dans le cadre parent (si dans une iframe) <code>_top</code> : dans la fenêtre complète (brise tous les cadres) Nom d'iframe : dans une iframe spécifique

Exemple d'utilisation :

```
<form action="noter.php"
method="get">
  <h3>Noter un restaurant</h3>
  <label for="nom">Nom du restaurant
  :</label><br>
  <input type="text" id="nom"
name="nom" required><br><br>

  <label for="note">Note (1 à 5)
  :</label><br>
  <input type="number" id="note"
name="note" min="1" max="5"
required><br><br>

<input type="submit"
value="Envoyer">
</form>
```

Noter un restaurant

Nom du restaurant :

Sodi

Note (1 à 5) :

3

Envoyer

⇒ url obtenu :

/noter.php?nom=Sodi&note=3

1. Les labels :

La balise HTML `<label>` est utilisée pour **associer une étiquette à un champ de formulaire** (comme `<input>`, `<textarea>`, etc.). L'utilisateur peut cliquer sur le texte du `<label>` pour activer le champ associé.

Exemple d'utilisation :

```
<label for="id_du_champ">Texte
visible</label>
<input type="text" id="id_du_champ"
name="nom_du_champ">
```

Texte visible

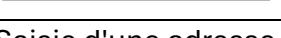
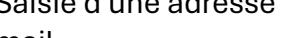
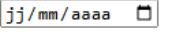
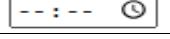
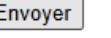
⇒ L'attribut for de `<label>` doit correspondre à l'attribut id du champ concerné.**2. Les zones de saisie :**

En HTML, les zones de saisie sont créées à l'aide de la balise `<input>` avec différents types (type) selon la nature des données attendues. Elles permettent à l'utilisateur de fournir des informations via un formulaire.

Attributs principaux :

Attribut	Rôle
name	Identifie le champ dans les données envoyées au serveur
placeholder	Affiche un texte indicatif dans le champ avant que l'utilisateur n'écrive (exemple ou aide)
required	Rend le champ obligatoire à remplir avant de pouvoir soumettre le formulaire.
readonly	Rend le champ non modifiable par l'utilisateur
value	valeur par défaut

Liste de balise input

Élément (balise)	Rôle	Attributs
<input type="text">	Champ de texte simple 	name, value, placeholder, readonly, required, disabled
<input type="email">	Saisie d'une adresse e-mail 	name, value, placeholder, readonly, required, disabled
<input type="tel">	Saisie d'un numéro de téléphone 	name, value, placeholder, readonly, required, disabled,
<input type="password">	Masque les caractères saisis 	name, value, placeholder, readonly, required, disabled,
<input type="number">	Saisie de valeurs numériques 	name, value, readonly, required, disabled ✓ min : Définit la valeur minimale que l'utilisateur peut entrer. ✓ max : Définit la valeur maximale que l'utilisateur peut entrer.
<input type="range">	Curseur pour sélectionner une valeur 	min, max, name, value, readonly, required, disabled
<input type="date">	Sélection d'une date 	min, max, name, value, readonly, required, disabled
<input type="time">	Sélection d'une heure 	min, max, name, value, readonly, required, disabled
<input type="checkbox">	cases à cocher 	name, value, disabled ✓ checked : coché par défaut lorsque la page est chargée
<input type="radio">	Choix unique dans un groupe 	name, value, disabled, checked
<input type="submit">	Bouton pour envoyer le formulaire 	name, value, disabled
<input type="reset">	Réinitialise les champs du formulaire	name, value, disabled
<button>	Crée un bouton personnalisable	name, value, disabled

3. Zone multilignes textarea:

La balise `<textarea>` permet à l'utilisateur de saisir **plusieurs lignes de texte**, contrairement à un champ `<input>` qui ne gère qu'une seule ligne.

Avec les attributs `name` , `readonly` , `required` , `disabled` , `placeholder` textarea possède aussi :

- ✓ **cols:** Définit la **largeur visible** de la zone de texte, en **nombre de caractères**.
- ✓ **rows:** Définit la **hauteur visible** de la zone de texte, en **nombre de lignes**.
- ✓ **maxlength:** Limite le **nombre maximum de caractères** que l'utilisateur peut taper.

Exemple d'utilisation :

```
<label for="commentaire">Laissez  
un commentaire : </label><br>  
<textarea id="commentaire"  
name="commentaire" rows="5"  
cols="50" maxlength="200">  
</textarea>
```

Laissez un commentaire :

4. Liste déroulante :

Une **liste déroulante** en HTML est créée à l'aide des balises `<select>` (conteneur) et `<option>` (éléments de la liste). Elle permet à l'utilisateur de **choisir une seule option parmi plusieurs**.

Exemple d'utilisation :

```
<label for="langue">Choisissez une  
langue :</label><br>  
<select id="langue" name="langue">  
  <option value="fr"  
selected>Français</option>  
  <option value="en">Anglais</option>  
  <option value="es">Espagnol</option>  
  <option value="de">Allemand</option>  
</select>
```

Choisissez une langue :

Français ▾

⇒ `selected` : Définit l'**option sélectionnée par défaut**.

5. fieldset :

La balise `<fieldset>` est utilisée en HTML pour **regrouper plusieurs éléments d'un formulaire** dans un même bloc logique, généralement entouré d'un **cadre visuel**. Elle est souvent accompagnée de la balise `<legend>` pour donner un titre à ce groupe.

```
<form>  
  <fieldset>  
    <legend>Informations  
personnelles</legend>  
  
    <label for="nom">Nom :</label>  
    <input type="text" id="nom"  
name="nom"><br><br>  
  
    <label for="email">Email :</label>
```

Informations personnelles

Nom :

Email :

```
<input type="email" id="email"
name="email">
</fieldset>
</form>
```

6. **datalist** :

La balise `<datalist>` est utilisée pour fournir une **liste de suggestions** à un champ `<input>`. L'utilisateur peut taper librement, mais il voit apparaître des propositions en fonction de ce qu'il écrit.

Exemple d'utilisation :

```
<label for="ville">Choisissez
une ville :</label>
<input list="villes" id="ville"
name="ville">

<datalist id="villes">
  <option value="Paris">
  <option value="Lyon">
  <option value="Marseille">
  <option value="Toulouse">
  <option value="Nice">
</datalist>
```

Choisissez une ville : 

Paris

Lyon

Marseille

Toulouse

Nice

⇒ L'élément `<input>` est lié à la liste via l'attribut `list="villes"`.

X. Eléments de structuration d'une page web

Les éléments de structuration d'une page web permettant principalement de diviser et organiser logiquement et sémantiquement le contenu.

Balise HTML	Rôle
<code><header></code>	En-tête de page ou de section (titre, logo, menu de navigation...)
<code><nav></code>	Regroupe les liens de navigation (menu, liens internes, etc.)
<code><main></code>	Contenu principal de la page (unique par page)
<code><section></code>	Regroupe un ensemble logique de contenu (ex : services, articles)
<code><article></code>	Contenu autonome (ex : un article de blog, une fiche produit)
<code><aside></code>	Contenu complémentaire (ex : pub, encadré, liens secondaires)
<code><footer></code>	Pied de page ou de section (infos légales, liens, contacts...)

XI. Les événements :

Les événements sont utilisés pour déclencher des **actions** (en Javascript) lorsqu'un utilisateur interagit avec la page (clics, saisie, chargement, etc.).

Exemple d'utilisation :

```
<button onclick="alert('Bouton cliqué !')> Cliquez-moi</button>
```

Clique-moi

Lorsque l'utilisateur clic sur le bouton

html.onlineviewer.net indique

Bouton cliqué !

OK

Principaux événements HTML

Événement	Déclenché quand...
onclick	L'utilisateur clique sur un élément.
onsubmit	Un formulaire est soumis (via un bouton ou Enter).
onchange	La valeur d'un champ change , et l'utilisateur quitte le champ.
onfocus	Un élément reçoit le focus (ex. : on clique dans un champ).
onblur	Un élément perd le focus (on clique ailleurs).
oninput	La valeur d'un champ change (détecté en temps réel).
onload	La page ou une ressource est entièrement chargée .
onmouseover	Le curseur survole un élément.
onmouseout	Le curseur quitte un élément.
onplay	Une vidéo ou un audio commence à jouer .
onpause	Une vidéo ou un audio est mis en pause .
onkeydown	Une touche du clavier est enfoncée .
onkeyup	Une touche du clavier est relâchée .

LYCÉE MONJI
SLIM SBIBA

Chapitre III CSS



I. Introduction au CSS

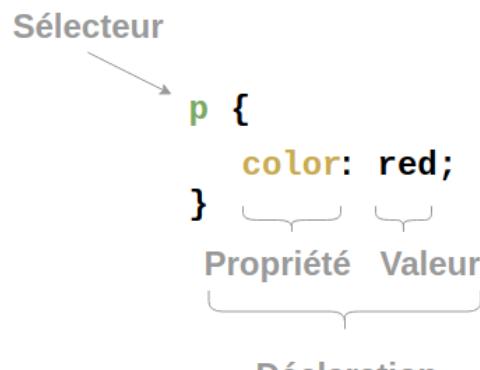
Le **CSS** (Cascading Style Sheets) est un langage utilisé pour décrire la présentation visuelle des pages web. Il permet de contrôler la mise en forme des éléments HTML, comme les couleurs, les polices, les marges, et la disposition. CSS sépare le contenu (HTML) de la présentation, facilitant la maintenance et la modification de l'apparence d'un site web.



1. Définition d'une règle CSS

Une **règle CSS** se compose de deux parties principales :

- ✓ **Sélecteur** : Il identifie les éléments HTML à styliser.
- ✓ **Déclaration** : Elle contient les propriétés CSS et leurs valeurs à appliquer aux éléments sélectionnés.



Exemple d'utilisation :

html

```
<p> Hello world ! </p>
```

css

```
p {
  color: blue;
  font-size: 16px;
}
```



Hello world !

2. Types de Sélecteurs CSS

a. Sélecteur de type

Cible un élément HTML par son nom de balise. Par exemple utiliser le sélecteur **p** pour tous les paragraphes.

Exemple d'utilisation :

html

```
<p> Hello world ! </p>
<span> Bonjour </span>
```

css

```
span {
  color: red;
}
```



Hello world !

Bonjour

b. Sélecteur de classe :

Cible un ou plusieurs éléments avec une classe spécifique, précédé d'un point (.).

Exemple d'utilisation :

html

```
<p> Hello </p>
<p> Bonjour </p>
<p class="esp"> hola </p>
```

css

```
.esp {
  color: red;
}
```



Hello

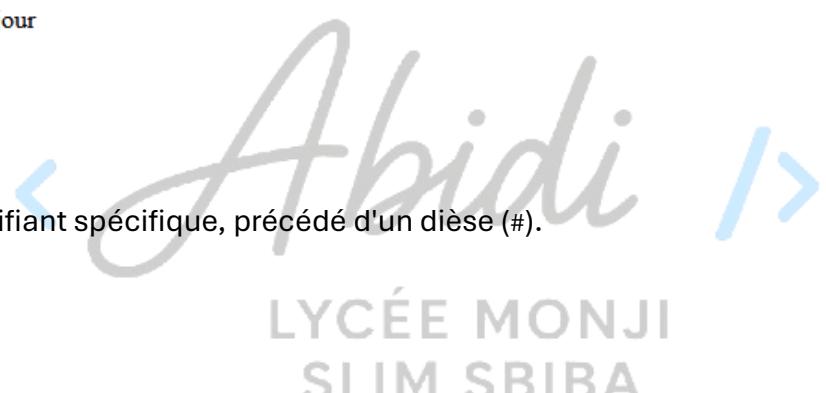
Bonjour

hola

c. Sélecteur d'ID :

Cible un élément unique avec un identifiant spécifique, précédé d'un dièse (#).

Exemple d'utilisation :



html

```
<p> Hello </p>
<p id="esp"> Bonjour </p>
<p class="esp"> hola </p>
```

css

```
#esp {
  color: red;
}
```



Hello
Bonjour
hola

d. Sélecteur universel :

Cible tous les éléments (*).

Exemple d'utilisation :**html**

```
<span> Hello </span>
<p> Bonjour </p>
<div> hola </div>
```

css

```
* {
  color: red;
}
```



Hello
Bonjour
hola

e. Sélecteur d'attribut :

Cible les éléments avec un attribut spécifique. (balise[attribut="valeur"])

Exemple d'utilisation :**html**

Nom :<input type="text">
Tel :<input type="tel">

css

```
input[type=text] {
  color: red;
}
```



SLIM SBIBA

Nom : Tel :

f. Sélecteurs combinés :

Il s'agit de la combinaison d'un sélecteur de balise avec un sélecteur de classe afin de cibler avec plus de précision certains éléments HTML.

Exemple d'utilisation :

html	css
<pre><p class="en"> Hello </p> <p> Bonjour </p> <p> hola </p> <div class="en"> reste ... </p></pre>	<pre>p.en { color: red; }</pre>
	
Hello Bonjour hola reste ...	

g. Sélecteurs multiples :

Les **sélecteurs multiples** permettent d'appliquer les mêmes styles à plusieurs éléments différents en les séparant par une **virgule (,)** dans la feuille de style CSS. Cela évite de répéter plusieurs règles identiques.

Exemple d'utilisation :

html	css
<pre><h1>Bienvenue </h1> <p>paragraphe</p> <div> une division.</div></pre>	<pre>p,h1 { color: red; }</pre>
	
	

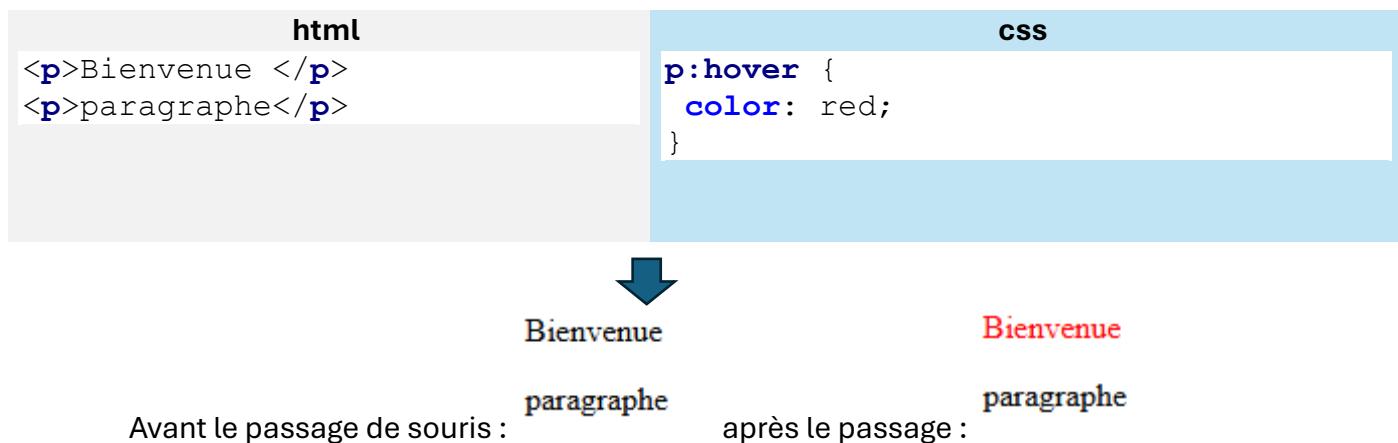
h. pseudo-classe

Une **pseudo-classe** est un mot-clé ajouté à un sélecteur, précédé d'un deux-points : qui permet de définir un **état particulier** d'un élément HTML **sans ajouter de classe ou d'attribut**.

Liste de pseudo-classes :

Pseudo-classe	Élément concerné	Moment où elle s'applique
:link	Lien <a>	Lorsqu'un lien n'a pas encore été visité
:visited	Lien <a>	Lorsqu'un lien a déjà été visité
:hover	Tout élément interactif	Lorsqu'on passe la souris dessus
:active	Bouton ou lien	Lors d'un clic actif (appui sans relâche)
:focus	Champs de formulaire	Lorsqu'un élément reçoit le focus

Exemple d'utilisation :



II. Propriétés de mise en forme du texte en CSS

Le CSS offre un large éventail de propriétés permettant de styliser le texte dans les pages web. Voici quelques-unes des plus courantes, avec des exemples :

1. font-family :

Cette propriété permet de définir la police de caractères utilisée pour le texte.



Exemple d'utilisation :

html <pre><p>Bienvenue </p></pre>	css <pre>p { font-family: arial, sans-serif ; }</pre>
---	---

↓

Bienvenue

Ici, le texte des paragraphes utilisera la police **Arial** avec une alternative **sans-serif** si Arial n'est pas disponible. Voici les 5 familles de polices avec des exemples des fonts :

Generic Font Family	Examples of Font Names
Serif	Times New Roman Georgia Garamond
Sans-serif	Arial Verdana Helvetica
Monospace	Courier New Lucida Console Monaco
Cursive	<i>Bush Script MT</i> <i>Lucida Handwriting</i>
Fantasy	Copperplate Papyrus

2. font-weight :

Elle permet de définir l'épaisseur du texte. Les valeurs possibles incluent normal, bold, ou des valeurs numériques comme 100 à 900.

Exemple d'utilisation :

html <pre><p>Bienvenue </p> <div> division </div></pre>	css <pre>p { font-weight: bold ; }</pre>
---	--

↓

Bienvenue

division

ÉCOLE MONT ST SLIM SBIBA

3. font-style :

Utilisée pour mettre le texte en italique. Les valeurs possibles sont **normal**, **italic**, ou **oblique**.

Exemple d'utilisation :

html	css
<pre><p>Bienvenue dans notre plateforme de jeux en ligne </p></pre>	<pre>span { font-style : italic ; }</pre>



 Bienvenue dans notre *plateforme* de jeux en ligne

4. font-size :

Elle permet de définir la taille du texte. Les valeurs peuvent être en pixels (px).



Exemple d'utilisation :

html	css
<pre><p>Bienvenue </p> <p class= "A"> division </p></pre>	<pre>p { font-size : 16px ; } .A { font-size : 32px ; }</pre>

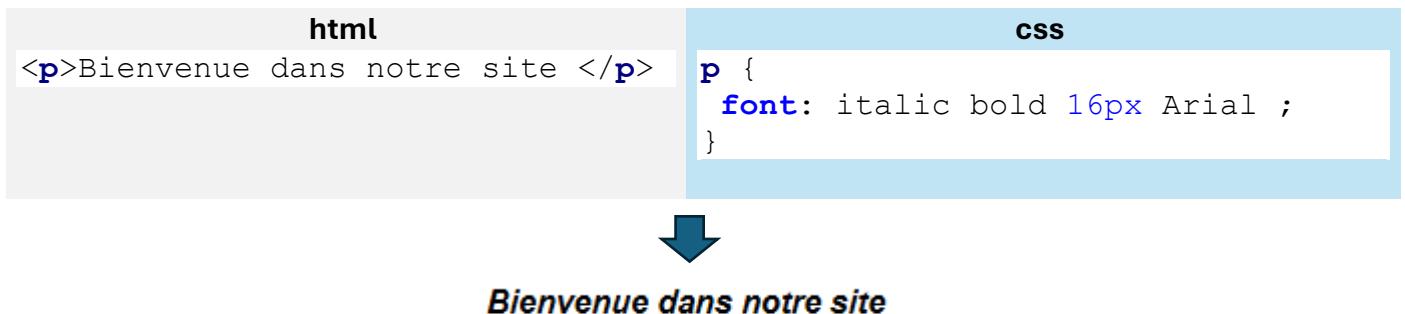


 Bienvenue

5. font :

Cette propriété regroupe plusieurs propriétés liées aux polices, comme font-style, font-weight, font-size, et font-family dans une seule déclaration.



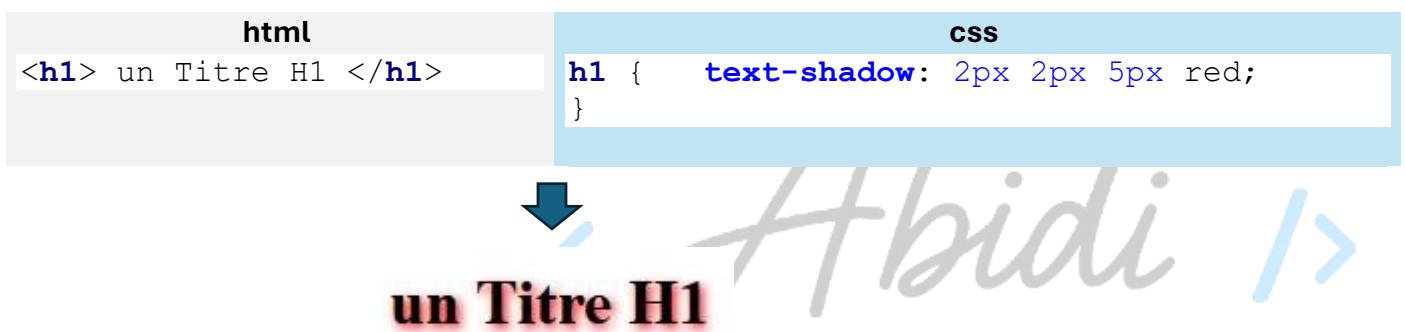
Exemple d'utilisation :**6. text-align :**

permet le contrôle l'alignement horizontal du texte. Les valeurs incluent **left**, **right**, **center**, et **justify**.

Alice opened the door and found that it led into a small passage, not much larger than a rat hole: she knelt down and looked along the passage into the loveliest garden you ever saw.	Alice opened the door and found that it led into a small passage, not much larger than a rat hole: she knelt down and looked along the passage into the loveliest garden you ever saw.	Alice opened the door and found that it led into a small passage, not much larger than a rat hole: she knelt down and looked along the passage into the loveliest garden you ever saw.	Alice opened the door and found that it led into a small passage, not much larger than a rat hole: she knelt down and looked along the passage into the loveliest garden you ever saw.
left	center	right	justify

7. text-shadow :

Cette propriété permet d'ajouter une ombre au texte. Elle prend 4 valeurs : l'offset horizontal et vertical, le flou, et la couleur.

Exemple d'utilisation :**8. text-transform :**

Elle permet de modifier la casse du texte. Les valeurs incluent uppercase, lowercase, et capitalize.

Exemple d'utilisation :

html <pre><h1>Titre principal </h1> <p>ceci est un texte.</p></pre>	css <pre>h1 { text-transform: uppercase; } p { text-transform: capitalize; }</pre>
---	--



TITRE PRINCIPAL

Ceci Est Un Texte.

9. color :

Cette propriété définit la couleur du texte. Les couleurs peuvent être spécifiées en nom (comme red), en hexadécimal .

Maroon #800000	Red #FF0000	Orange #FFA500	Yellow #FFFF00	Olive #808000
Purple #800080	Fuchsia #FF00FF	White #FFFFFF	Lime #00FF00	Green #008000
Navy #000080	Blue #0000FF	Aqua #00FFFF	Teal #008080	
Black #000000	Silver #C0C0C0	Gray #808080		

HEXADECIMAL COLOR CODES

Red Color Code	Green Color Code	Blue Color Code
----------------------	------------------------	-----------------------

RR GG BB

Signifies Hex Color Code

Each of the two-digit color codes are in the range 00...FF hexadecimal

EXAMPLES

#FFCC99 #3C00A6

Exemple d'utilisation :

html <pre><h1>Titre principal </h1></pre>	css <pre>h1 { color: lime; }</pre>
---	--



LYCÉE MONJI
M SBIBA
Titre principal

XII. Propriétés de couleur et de fond :

1. background-color

La propriété **background-color** définit la couleur de fond d'un élément. Elle peut être spécifiée à l'aide de noms de couleurs, de valeurs hexadécimales,

Exemple d'utilisation :

html <pre><h1>Titre </h1> <p>Paragraphe</p></pre>	css <pre>body {background-color: #f0f0f0;} h1 {background-color: lightblue;}</pre>
---	--

↓

Titre

Paragraphe

2. background-image

La propriété **background-image** permet d'appliquer une image en tant qu'arrière-plan d'un élément. L'image peut être spécifiée par une URL.

Exemple d'utilisation :

html <pre><h1>Titre </h1> <p>Paragraphe</p></pre>	css <pre>body { background-image: url("backg.png"); color:white; }</pre>
---	--

↓

Titre

Paragraphe



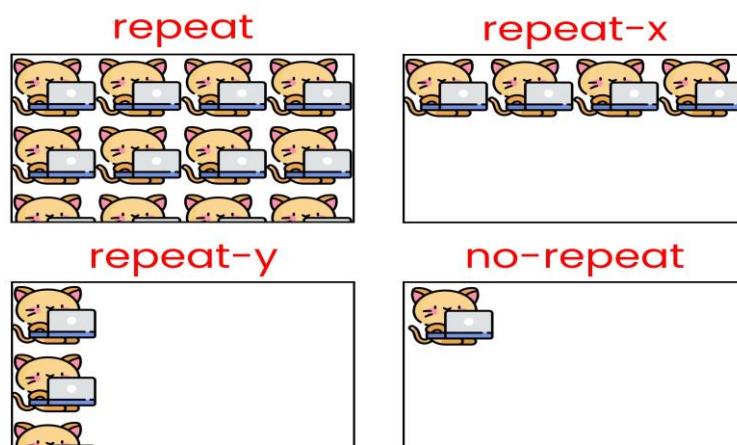
/idi />
 E MONJI
 SBIBA

3. background-repeat :

La propriété **background-repeat** contrôle la répétition de l'image de fond. Par défaut, une image de fond se répète pour couvrir l'élément. Vous pouvez choisir de désactiver la répétition ou de répéter l'image horizontalement ou verticalement.

Valeurs :

- ❖ **repeat** (par défaut) : L'image de fond se répète dans les deux directions.
- ❖ **repeat-x** : L'image se répète uniquement horizontalement.
- ❖ **repeat-y** : L'image se répète uniquement verticalement.
- ❖ **no-repeat** : L'image ne se répète pas.



Exemple d'utilisation :

html

```
<h1>Titre </h1>
<p>Paragraphe</p>
```

css

```
body {
    background-color: silver;
    background-image: url("backg2.png");
    background-repeat: repeat-y;
}
```

↓

Titre

Sun

Paragraphe





4. background-size :

La propriété **background-size** définit la taille de l'image d'arrière-plan. Vous pouvez définir des dimensions précises, utiliser des pourcentages, ou employer des mots-clés comme **cover** ou **contain**.

Valeurs :

- ❖ **auto** (par défaut) : La taille de l'image de fond reste à sa taille d'origine.
- ❖ **cover** : L'image couvre entièrement l'élément tout en gardant ses proportions.
- ❖ **contain** : L'image est redimensionnée pour être complètement visible à l'intérieur de l'élément.
- ❖ **Valeurs libre** : comme 100px 50px, pour des largeurs et hauteurs spécifiques.



Exemple d'utilisation :

```
html
<h1>Titre </h1>
<p>Paragraphe</p>
```

```
css
body {
    background-color:silver;
    background-image: url("backg2.png");
    background-repeat: no-repeat;
    background-size :cover ;
}
```



5. background

La propriété **background** est une propriété **raccourcie** qui permet de définir plusieurs propriétés de fond en une seule déclaration, comme `background-color`, `background-image`, `background-repeat`, `background-position`, et `background-size`.

Syntaxe :

```
background: [background-color] [background-image] [background-repeat]
[position / taille];
```

Exemple d'utilisation :

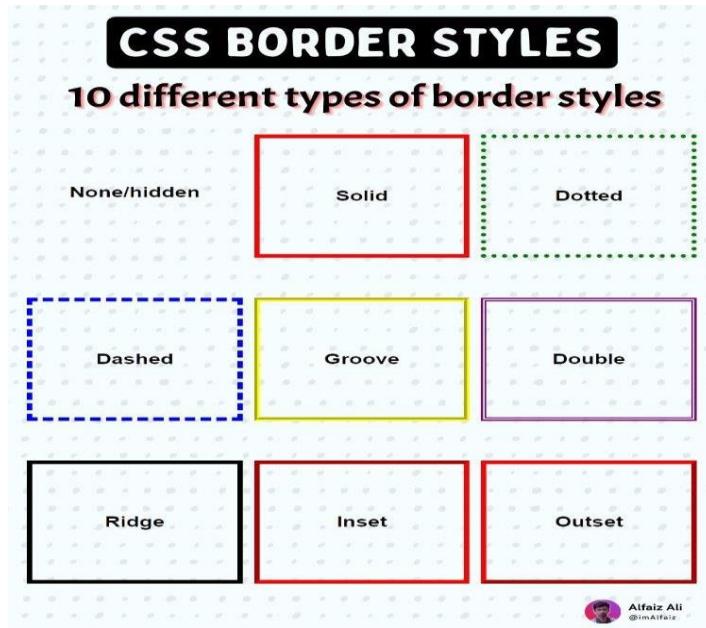
html	css
<pre><h1>Titre </h1> <p>Paragraphe</p></pre>	<pre>body { background: lightblue url('background.jpg') no-repeat center / cover; }</pre>



Propriétés des bordures :

5. 1. border-style

La propriété **border-style** définit le style de la bordure d'un élément. Elle peut prendre différentes valeurs pour spécifier l'apparence de la bordure (solide, en pointillés, en tirets, etc.).

Exemple d'utilisation :

html <pre><h1>Titre principal </h1> <p> paragraphe </p></pre>	css <pre>h1 { border-style: solid;</pre>
---	--



Titre principal

paragraphe

6. border-color

La propriété **border-color** définit la couleur de la bordure. Elle peut être spécifiée en utilisant un nom de couleur, une valeur hexadécimale, RGB,

Exemple d'utilisation :

html <pre><h1>Titre principal </h1></pre>	css <pre>h1 { border-style: solid; border-color :red ;</pre>
---	--



LYCÉE MONJI

Titre principal

7. border-width

La propriété **border-width** contrôle l'épaisseur de la bordure. Vous pouvez définir une épaisseur spécifique en pixels ou utiliser des valeurs prédéfinies telles que **thin**, **medium**, ou **thick**.

Exemple d'utilisation :

```
html
<h1>Titre principal </h1>
<p> un paragraphe</p>
```

```
css
h1, p {
    border-style: solid;
    border-color :red ;
}
h1{border-width :10px ; }
```



Titre principal

un paragraphe

4. border-radius

La propriété **border-radius** permet de créer des coins arrondis pour les bordures. Vous pouvez définir un rayon spécifique pour les coins en pixels ou en pourcentage.

Exemple d'utilisation :

```
html
<h1>Titre principal </h1>
```

```
css
h1 { border-style: solid;
      text-align:center;
      border-color :red ;
      border-radius :50% ;
}
```



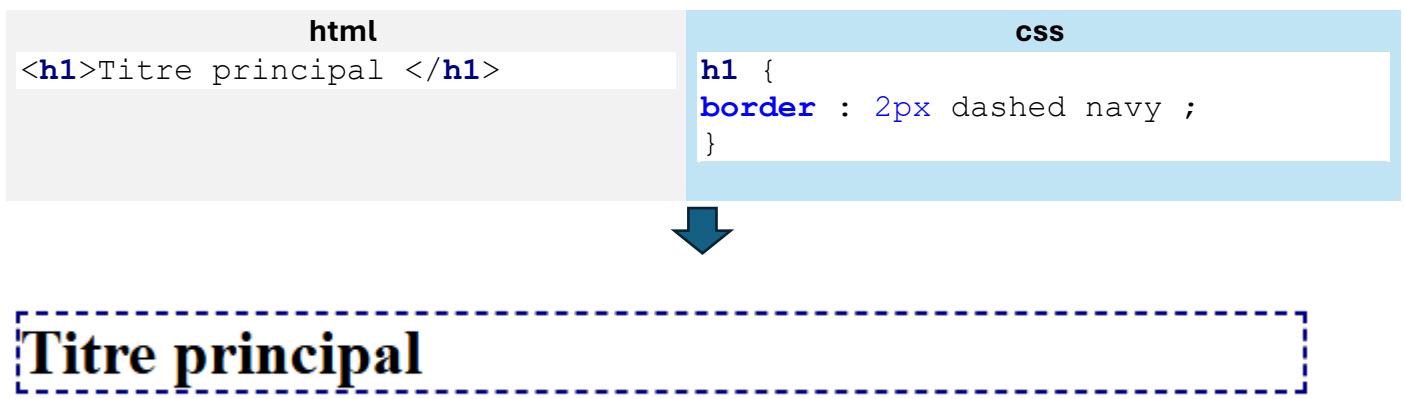
Titre principal

8. 5. border

La propriété **border** est une **propriété raccourcie** qui permet de définir en une seule déclaration le style, la couleur et l'épaisseur de la bordure.

Syntaxe :

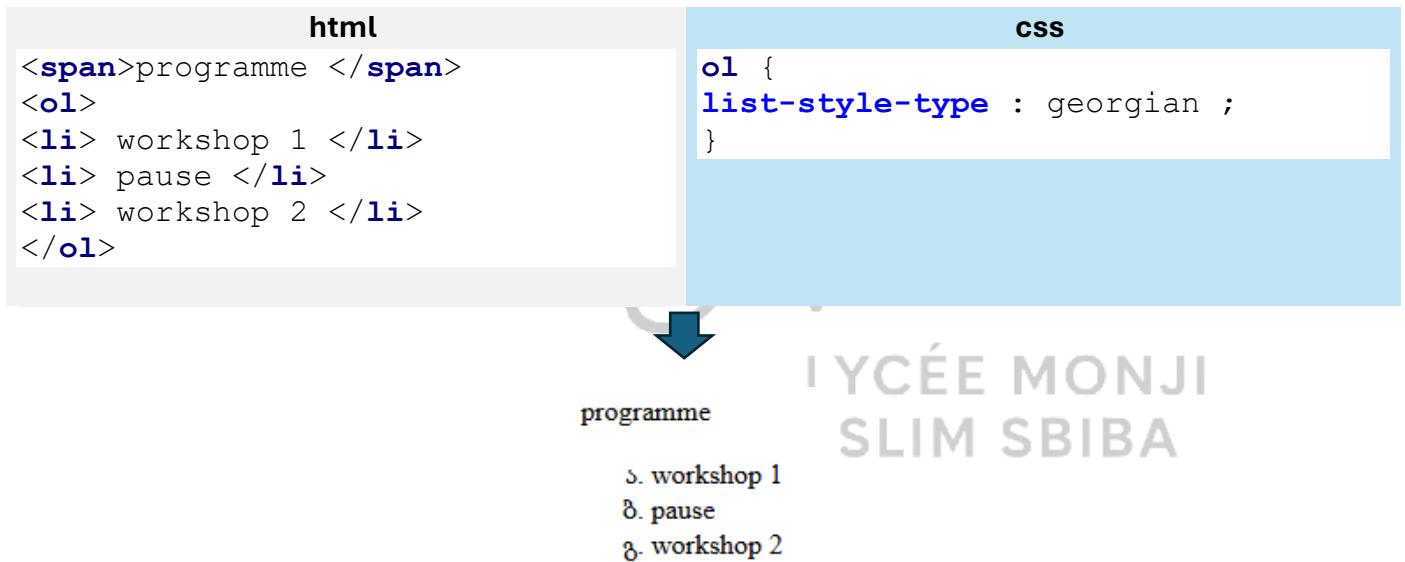
```
border: [épaisseur] [style] [couleur];
```

Exemple d'utilisation :**XIII. Propriétés des listes****1. list-style-type**

La propriété **list-style-type** définit le type de puce ou de numérotation à utiliser pour les éléments d'une liste. Elle peut être appliquée à des listes ordonnées (****) et non ordonnées (****).

Valeurs possibles :

armenian	circle	decimal	georgian	decimal-leading-zero	lower-alpha	lower-greek	lower-roman	square	upper-alpha	upper-roman
Ա.	○	1.	Ճ.	01.	ա.	Ճ.	ի.	■	Ա.	I.
Բ.	○	2.	Ճ.	02.	բ.	Ճ.	ii.	■	Բ.	II.
Գ.	○	3.	Ճ.	03.	շ.	Ճ.	iii.	■	Յ.	III.
Դ.	○	4.	Ճ.	04.	դ.	Ճ.	iv.	■	Դ.	IV.
Ե.	○	5.	Ճ.	05.	է.	Ճ.	v.	■	Ե.	V.

Exemple d'utilisation :

2. 2. list-style-position

La propriété **list-style-position** détermine la position de la puce ou du numéro par rapport au contenu de l'élément de la liste.

Valeurs possibles :

- **inside** : la puce ou le numéro est à l'intérieur du bloc, donc aligné avec le texte de l'élément.
- **outside** (par défaut) : la puce ou le numéro est à l'extérieur du bloc, décalé à gauche du texte.

list-style-position: inside

- Item 1
- Item 2

list-style-position: outside

- Item 1
- Item 2

Exemple d'utilisation :

html	css
<pre>programme workshop 1 pause workshop 2 </pre>	<pre>ol { list-style-position: outside; } li{ background-color:pink; }</pre>

↓

programme

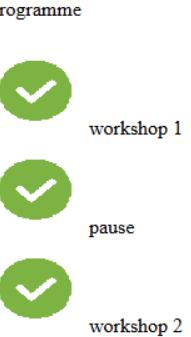
- 1. workshop 1
- 2. pause
- 3. workshop 2

3. 3. list-style-image

La propriété **list-style-image** permet d'utiliser une image en tant que puce dans une liste.



Exemple d'utilisation :

html	css
<pre>programme workshop 1 pause workshop 2 </pre>	<pre>ol { list-style-image: url("motif.jpg"); }</pre>
	
	

4. 4. list-style

La propriété **list-style** est une propriété **raccourcie** qui permet de définir simultanément **list-style-type**, **list-style-position**, et **list-style-image** en une seule déclaration.

Syntaxe :

```
list-style: [type] [position] [image];
```

Exemple d'utilisation :

html	css
<pre>programme workshop 1 pause workshop 2 </pre>	<pre>ol { list-style: decimal inside url("motif.jpg"); }</pre>

Dans cet exemple, la liste utilise une image **motif.png**, avec un style de puce décimale comme secours, et les puces sont alignées à l'intérieur du texte.

XIV. Propriétés des tableaux

5. 1. table-layout

La propriété **table-layout** détermine la manière dont la largeur des colonnes d'un tableau est calculée. Elle influence la rapidité du rendu du tableau et la manière dont les colonnes sont redimensionnées.

Valeurs possibles :

table-layout: auto

ID	Description	Prix
1	Petit texte	10€
2	Texte beaucoup plus long qui élargit la colonne	20€

table-layout: fixed

ID	Description	Prix
1	Petit texte	10€
2	Texte beaucoup plus long qui va être coupé ou passer à la ligne	20€

Exemple d'utilisation :

html

```
<table>
  <tr>
    <th>Colonne courte</th>
    <th>Colonne avec un texte beaucoup plus long qui ne devrait pas élargir</th>
  </tr>
  <tr>
    <td>123</td>
    <td> texte pour voir l'effet.</td>
  </tr>
</table>
```

css

```
table {
  table-layout: fixed;
}
td, th{
  border: 1px solid black;
}
```



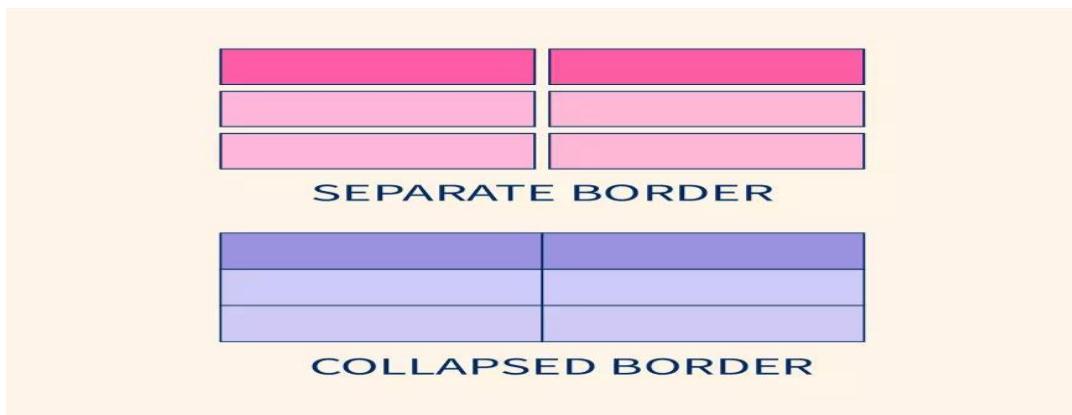
Colonne courte	Colonne avec un texte beaucoup plus long qui ne devrait pas élargir
123	Beaucoup de texte pour voir l'effet.

6. 2. border-collapse

La propriété **border-collapse** détermine si les bordures des cellules adjacentes dans un tableau sont fusionnées en une seule bordure ou séparées.

Valeurs possibles :

- **separate** (valeur par défaut) : Les bordures des cellules sont séparées
- **collapse** : Les bordures adjacentes des cellules fusionnent en une seule bordure. Cela donne une apparence plus propre au tableau avec des bordures continues.



Exemple d'utilisation :

```
html
<table>
  <tr>
    <th>Colonne courte</th>
    <th>Colonne avec un texte  
beaucoup plus long qui ne devrait  
pas élargir</th>
  </tr>
  <tr>
    <td>123</td>
    <td>texte pour voir  
l'effet.</td>
  </tr>
</table>
```

```
css
table {
  table-layout: auto;
  border-collapse: collapse;
}
td, th{
  border: 1px solid black;
}
```

Colonne courte	Colonne avec un texte beaucoup plus long qui ne devrait pas élargir
123	Beaucoup de texte pour voir l'effet.

XV. Propriétés des images :

La propriété **filter** en CSS permet d'appliquer des effets graphiques (comme un flou ou un changement de couleurs) aux éléments tels que des images, du texte, ou même des blocs. C'est un moyen simple de modifier l'apparence visuelle d'un élément sans avoir à modifier l'image ou l'élément lui-même.

1. invert()

La fonction **invert()** inverse les couleurs d'un élément. Cela signifie que les couleurs claires deviennent sombres et les couleurs sombres deviennent claires. La valeur est un pourcentage entre **0%** (aucune inversion) et **100%** (couleurs complètement inversées).



Exemple d'utilisation :

html	css
<pre> avant après </pre>	<pre>.f1 { filter: invert(100%); }</pre>
 avant	 après

2. grayscale()

La fonction **grayscale()** applique un effet de désaturation, transformant une image en niveaux de gris. La valeur est un pourcentage entre **0%** (pas de changement) et **100%** (image entièrement en niveaux de gris).



Exemple d'utilisation :

html <pre> avant après </pre>	css <pre>.f1{ filter: grayscale(70%); }</pre>
--	---



3. blur()

La fonction **blur()** applique un flou à l'élément. La valeur spécifie le rayon du flou, exprimé en pixels.



Exemple d'utilisation :

html <pre> avant après </pre>	css <pre>.f1{ filter: blur(5px); }</pre>
--	--



avant

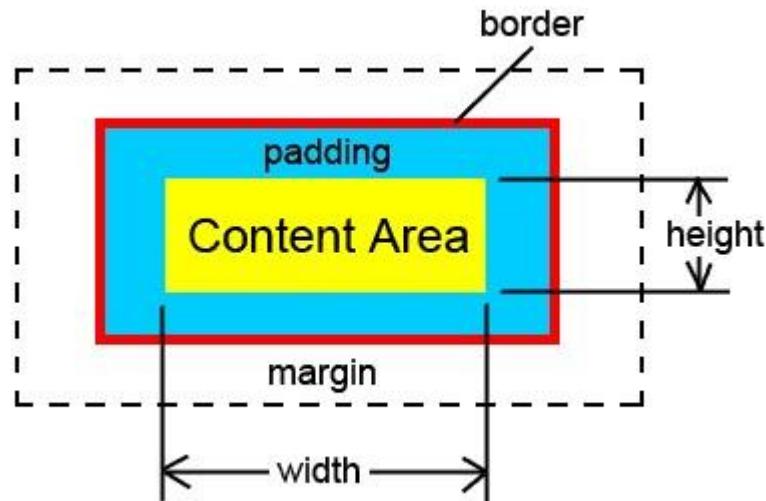


après

XVI. Propriétés des boîtes :

Les propriétés CSS des boîtes permettent de gérer la taille, la position, les marges, les bordures, le remplissage et d'autres aspects des éléments HTML.



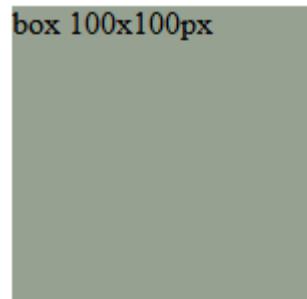


1. width et height

- ❖ **width** : définit la largeur d'un élément.
- ❖ **height** : définit la hauteur d'un élément.

Exemple d'utilisation :

html	css
<pre><div> box 150x150 px</div></pre>	<pre>div{ width :150px ; height : 150px ; background-color: #97A191;</pre>



2. margin

La propriété **margin** ajoute de l'espace autour d'un élément. Elle peut être définie pour les quatre côtés individuellement ou globalement.



Syntaxe :

Syntaxe	Exemple	Effet
1 valeur	<code>margin: 20px;</code>	20px en haut, droite, bas, gauche
2 valeurs	<code>margin: 20px 50px;</code>	Haut et Bas = 20px Droite et Gauche = 50px
3 valeurs	<code>margin: 10px 30px 50px;</code>	Haut = 10px Droite et Gauche = 30px Bas = 50px
4 valeurs	<code>margin: 5px 10px 15px 20px;</code>	Haut = 5px Droite = 10px Bas = 15px Gauche = 20px
Propriétés individuelles	<code>margin-top: 10px; margin-right: 20px; margin-bottom: 30px ; margin-left: 40px;</code>	Définit chaque côté séparément
Valeurs spéciales	<code>margin: auto ;</code>	centre un bloc horizontalement
	<code>margin: 0;</code>	aucune marge

3. padding

La propriété **padding** ajoute de l'espace à l'intérieur de l'élément, entre son contenu et sa bordure.

Exemple d'utilisation :

html

```
<p> un paragraphe avec  
espace intérieur à gauche  
</p>
```

css

```
p {  
    width: 300px;  
    border: 2px solid red;  
    padding-left: 100px;  
}
```

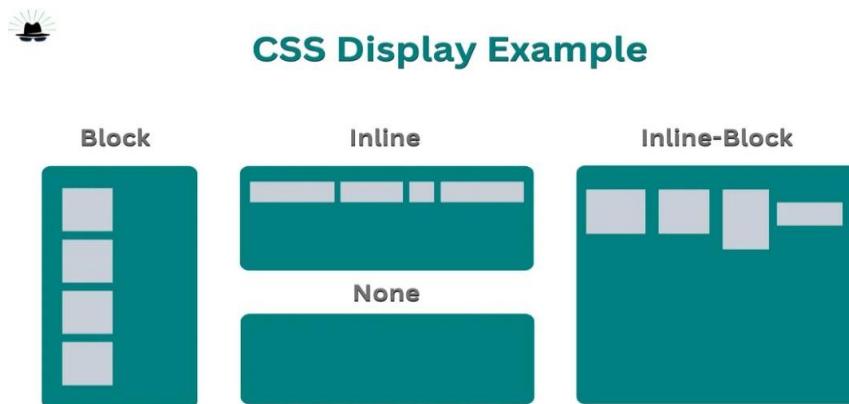
SLIM SBIBA

un paragraphe avec espace intérieur à gauche

4. Display :

En CSS, la propriété **display** détermine la manière dont un élément est rendu à l'écran :

- ❖ **none** : l'élément est totalement masqué et n'occupe aucun espace dans la page.
- ❖ **block** : l'élément s'affiche sous forme de bloc, prend toute la largeur disponible et commence toujours sur une nouvelle ligne (ex. : <div>, <p>).
- ❖ **inline** : l'élément s'intègre directement dans le flux du texte, à la suite des autres, sans retour à la ligne. On ne peut pas lui appliquer de dimensions (width, height) (ex. : , <a>).
- ❖ **inline-block** : l'élément reste sur la même ligne que les autres, mais il conserve les possibilités de mise en forme d'un bloc (dimensions, marges, espacements).



Exemple d'utilisation :

```
html
<div> box 1 </div>
<div> box 2 </div>
```

```
css
div{
  display : inline-block ;
  width : 200px ;
  background-color : #FAB9C2 ;
  height : 150px ;
}
```

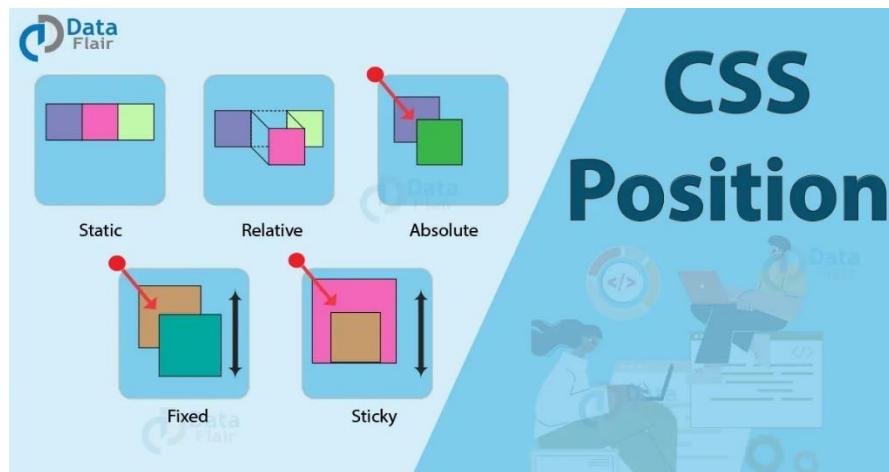


5. Position :

Elle permet de contrôler **l'emplacement** d'un élément dans la page.

Valeurs communes :

- ❖ **static** (par défaut) : l'élément suit le flux normal du document.
- ❖ **relative** : l'élément est positionné par rapport à sa position normale.
- ❖ **absolute** : l'élément est retiré du flux et positionné par rapport à son conteneur positionné le plus proche.
- ❖ **fixed** : l'élément est positionné par rapport à la fenêtre et ne bouge pas lors du défilement.
- ❖ **sticky** : permet à un élément de rester positionné normalement dans la page, mais de devenir fixe (collant) lorsqu'on fait défiler la page .



La propriété **position** est utilisé souvent avec les propriétés CSS :

- ❖ **top** : décale l'élément vers le bas à partir du haut.
- ❖ **bottom** : décale l'élément vers le haut à partir du bas.
- ❖ **left** : décale l'élément vers la droite à partir de la gauche.
- ❖ **right** : décale l'élément vers la gauche à partir de la droite.

Remarque : Ces propriétés fonctionnent seulement si **position ≠ static**.



Exemple d'utilisation :

html	css
<pre><div> <p> un paragraphe </p> </div></pre>	<pre>div{ width :200px ; background-color : #FAB9C2 ; height :150px ; } p{ background-color : red ; position :relative ; left : 180px ; }</pre>



un paragraphe

6. float :

La propriété **float** en CSS permet de positionner un élément à gauche ou à droite de son conteneur, de façon que le texte et les autres éléments s'enroulent autour.

Valeurs principales :

- ❖ left : l'élément flotte à gauche.
- ❖ right : l'élément flotte à droite.
- ❖ none : valeur par défaut, pas de flottement.
- ❖ **Exemple d'utilisation :**

html	css
<pre><div> <p> un paragraphe </p> </div></pre>	<pre>div{ width :300px ; border : 2px solid red; height :40px ; } p{float :right ;}</pre>



un paragraphe

7. overflow

La propriété **overflow** contrôle comment le contenu débordant d'un élément est géré.

Valeurs :

- ❖ **visible** (par défaut) : le contenu déborde de la boîte.
- ❖ **hidden** : le contenu qui dépasse est masqué.
- ❖ **scroll** : des barres de défilement sont ajoutées si nécessaire.
- ❖ **auto** : des barres de défilement apparaissent seulement si le contenu déborde.
- ❖ **clip**: semble fonctionner exactement comme overflow: hidden. La différence est que vous pouvez utiliser d'autre propriétés pour faire défiler quelque chose avec **hidden**, mais pas si l'élément a **clip**.

Guide to CSS Overflow



Exemple d'utilisation :

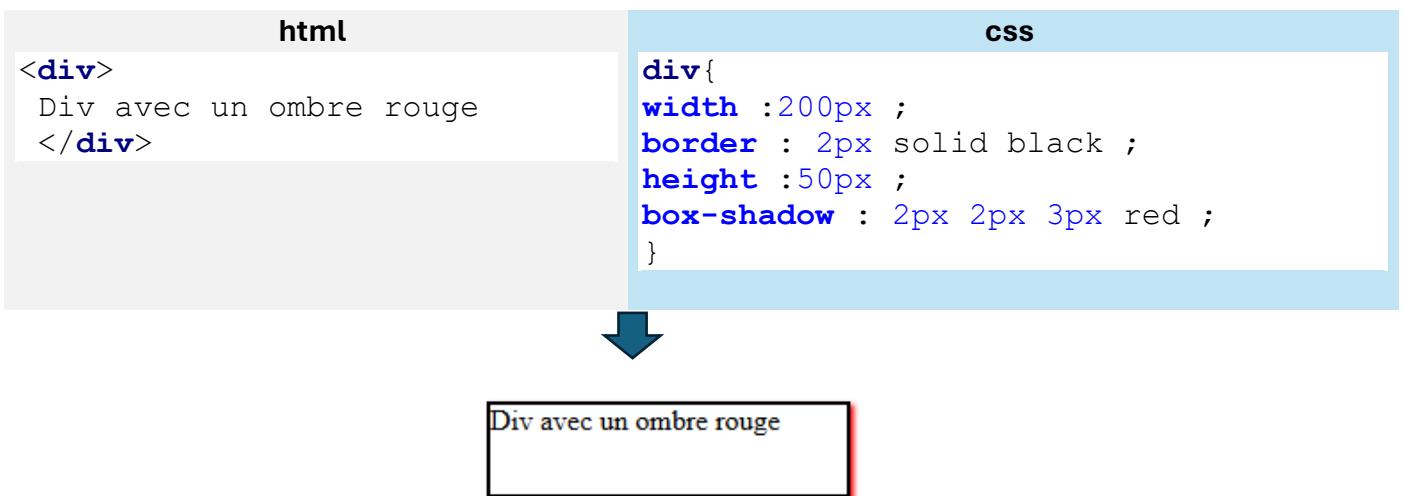
html	css
<pre><div> Contenu trop grand pour le container </div></pre>	<pre>div{ width :50px ; background-color : #FAB9C2 ; height :50px ; overflow :clip ; }</pre>

8. box-shadow

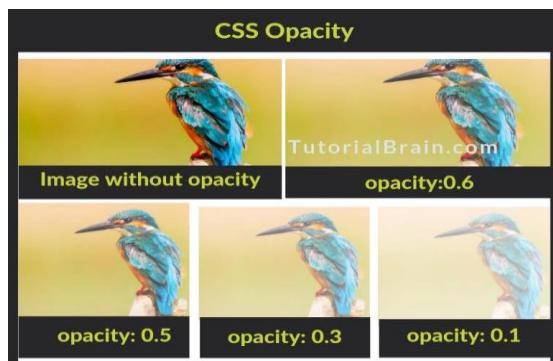
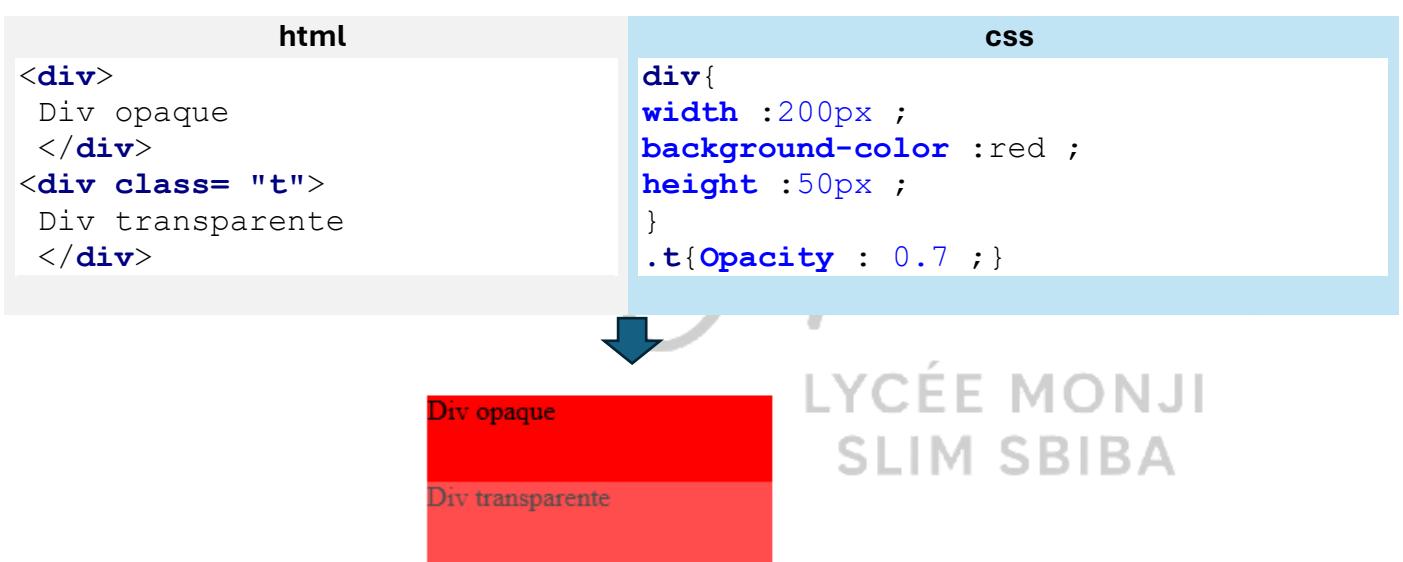
La propriété **box-shadow** ajoute une ombre autour d'un élément.

Syntaxe :

```
box-shadow: offsetX offsetY taille color;
```

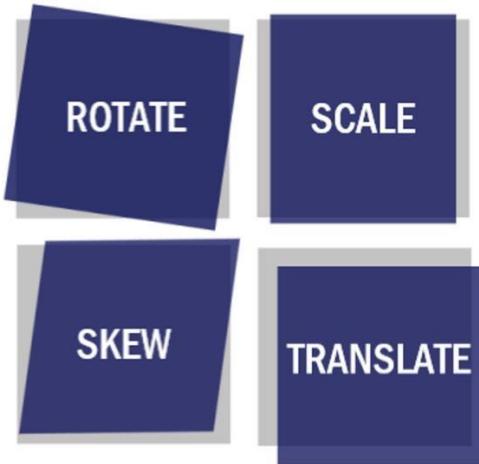
Exemple d'utilisation :**9. Opacity**

La propriété **opacity** définit la transparence d'un élément. La valeur varie de **0** (entièrement transparent) à **1** (entièrement opaque).

**Exemple d'utilisation :**

XVII. Transformation

Les **transformations CSS** permettent de modifier l'apparence des éléments en les faisant pivoter, déplacer, redimensionner ou déformer. Les fonctions principales utilisées pour ces transformations sont : **rotate**, **skew**, **scale**, et **translate**. Voici les définitions et exemples pour chacune d'elles.



1. **rotate()** : Rotation d'un élément

La fonction **rotate()** permet de faire pivoter un élément autour de son point d'origine. La valeur est exprimée en degrés (**deg**).

Exemple d'utilisation :

html	css
<pre><div> Une division </div></pre>	<pre>div{ border : 4px solid red ; } div:hover{ transform : rotate(45deg) ; }</pre>

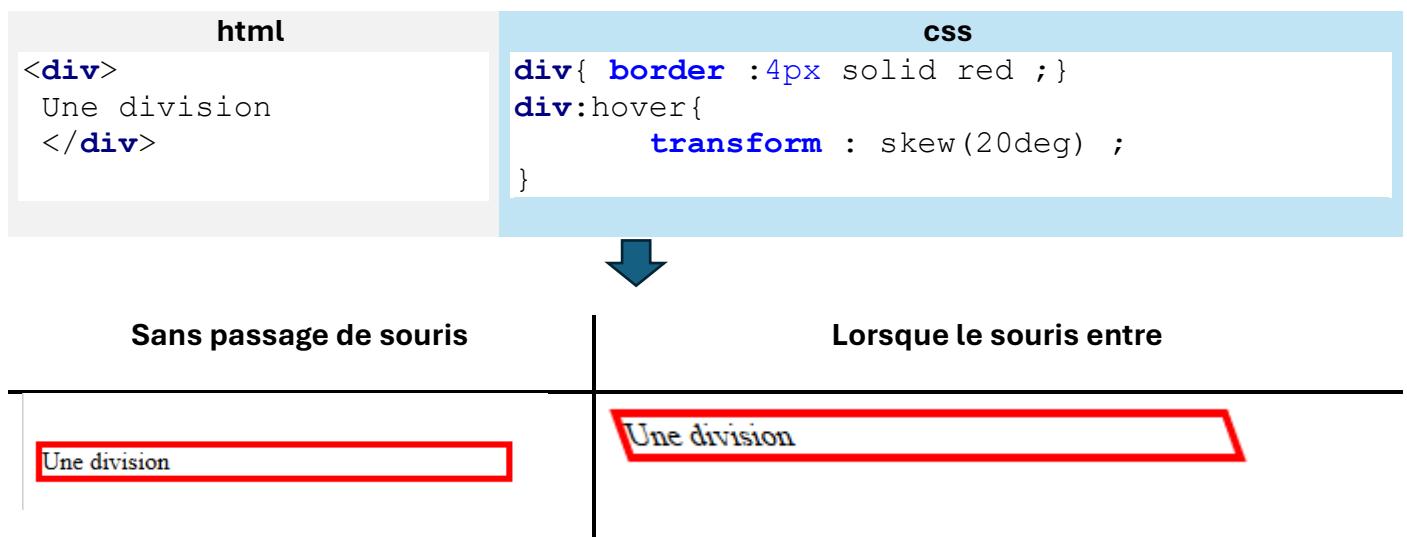
↓

Sans passage de souris	Lorsque le souris entre

2. skew() : Inclinaison d'un élément

La fonction **skew()** permet d'incliner un élément le long de l'axe X ou Y. Les valeurs sont exprimées en degrés.

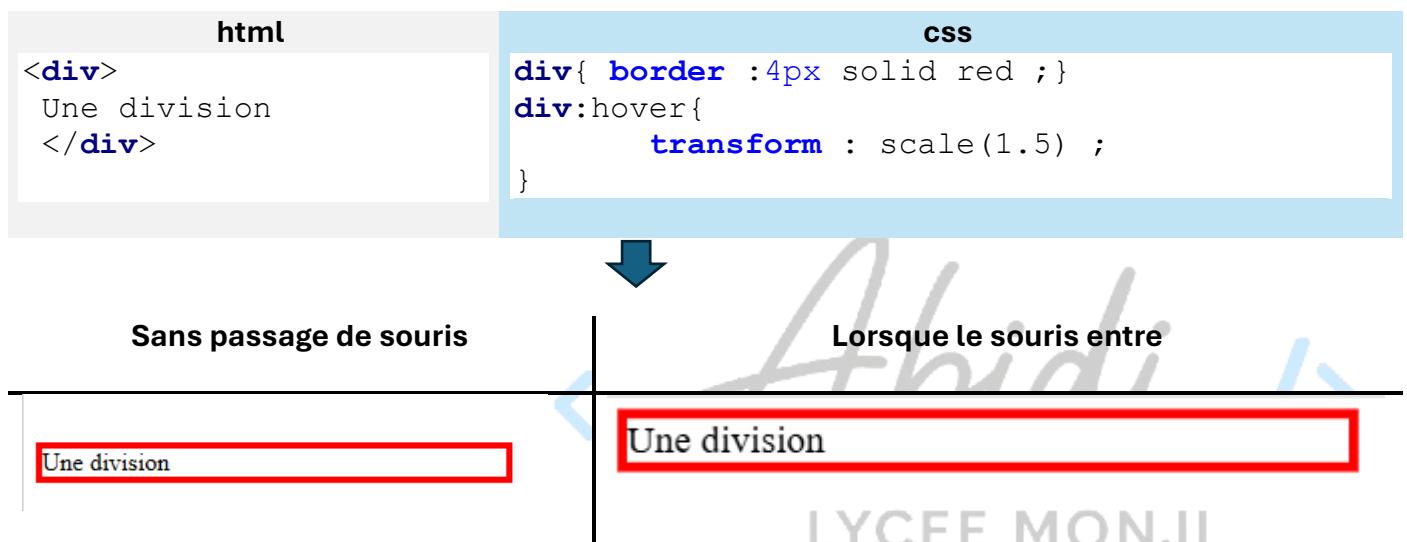
Exemple d'utilisation :



3. scale() : Redimensionnement d'un élément

La fonction **scale()** modifie la taille d'un élément en fonction des facteurs de redimensionnement donnés. Un facteur de **1** signifie que l'élément conserve sa taille d'origine.

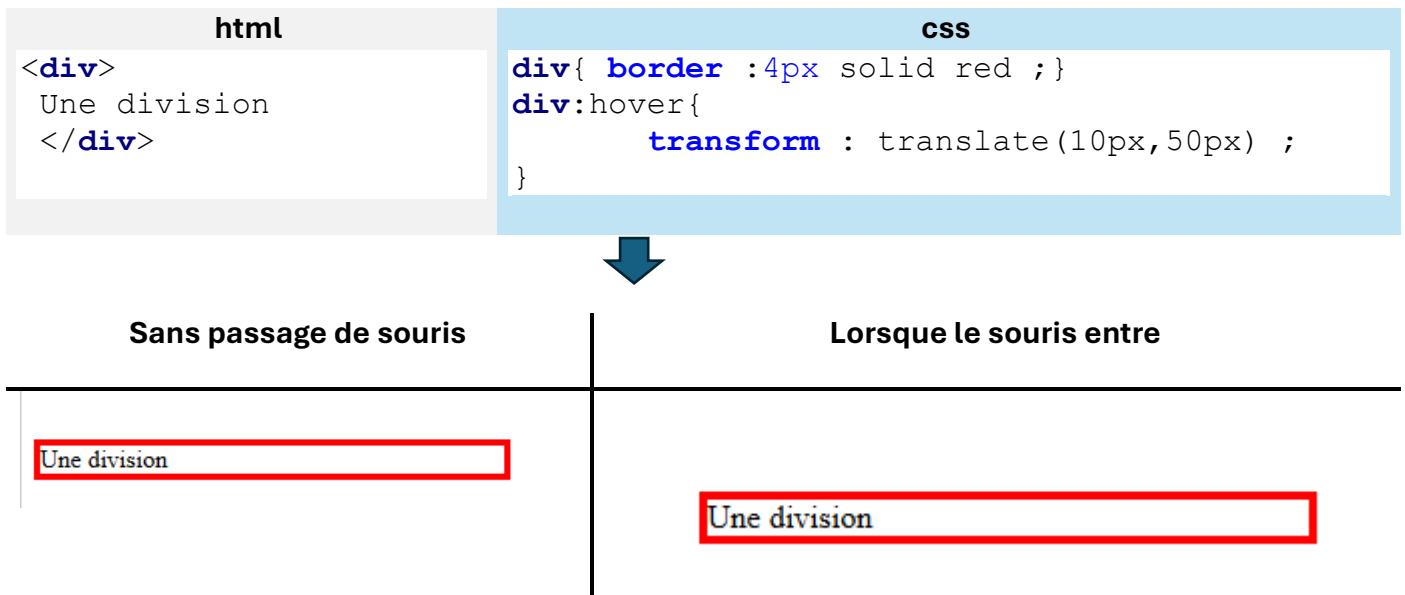
Exemple d'utilisation :



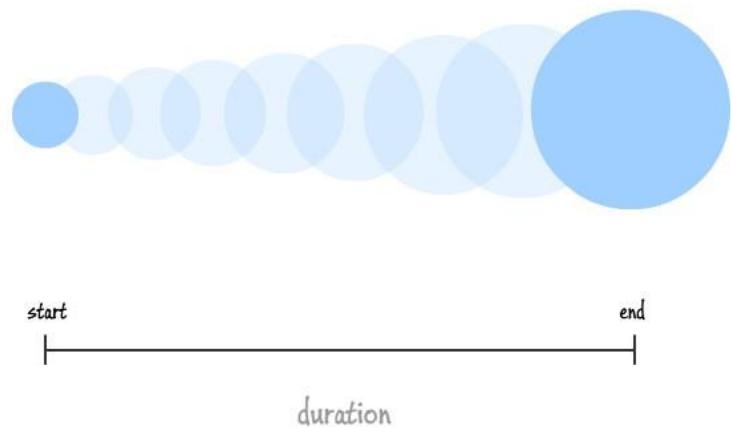
4. translate() : Déplacement d'un élément

La fonction **translate()** déplace un élément sur l'axe X ou Y. Les valeurs sont exprimées en pixels (**px**) ou en pourcentages (%).

Exemple d'utilisation :



XVIII. Les transitions



Les **transitions en CSS** permettent d'ajouter des animations simples et fluides lorsque des propriétés d'un élément changent d'état, par exemple lorsqu'un élément est survolé, cliqué, ou lorsqu'une classe lui est ajoutée. Les transitions facilitent le passage progressif d'une propriété CSS à une autre.

LYCÉE MONJI
SLIM SBIBA

Propriétés de transition :

Propriété	Rôle	Exemple
transition-property	Définit quelle(s) propriété(s) CSS vont être animées.	transition-property: width;
transition-duration	Définit la durée de la transition (combien de temps elle dure).	transition-duration: 2s;
transition-delay	Définit le délai avant le début de la transition.	transition-delay: 1s;

Syntaxe de propriété transition :

transition: <property> <duration> <delay>;

Exemple :

orsqu'on passe la souris sur le div, la couleur d'arrière-plan change vers le bleu. Ce changement commence après **1 seconde de retard** et la transition dure **3 secondes**.

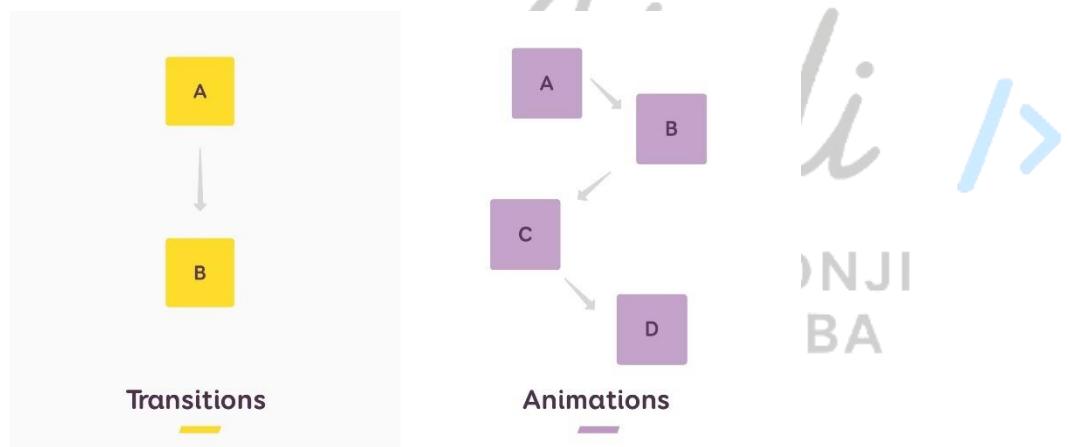
Méthode 1:

```
div:hover {
    background-color: blue;
    transition-property: background-color;
    transition-duration: 3s;
    transition-delay: 1s;
}
```

Méthode 2:

```
div:hover {
    background-color: blue;
    transition: background-color 3s 1s;
}
```

XIX. animations



Les **animations en CSS** permettent de créer des mouvements complexes et dynamiques en spécifiant une série d'étapes à travers les valeurs des propriétés CSS. Contrairement aux transitions qui ne se produisent que lorsque l'état d'un élément change (comme lors d'un survol), les animations peuvent être exécutées en continu ou selon un cycle défini. Voici une description détaillée des principales propriétés d'animations CSS :

Propriété	Rôle	Exemple
@keyframes + nom	Définit les étapes de l'animation (nom qu'on réutilise dans animation-name).	<code>@keyframes slide { from { left: 0; } to { left: 100px; }}</code>
animation-name	Indique quel nom d'animation utiliser (défini avec @keyframes).	<code>animation-name: slide;</code>
animation-duration	Durée de l'animation (obligatoire).	<code>animation-duration: 3s;</code>
animation-delay	Délai avant le démarrage de l'animation.	<code>animation-delay: 1s;</code>
animation-iteration-count	Nombre de répétitions de l'animation (1, 2, infinite).	<code>animation-iteration-count: infinite;</code>
animation-direction	Sens de l'animation : (normal, reverse, alternate, alternate-reverse).	<code>animation-direction: alternate;</code>

Syntaxe générale de propriété animation :

`animation: <name> <duration> <delay> <iteration-count> <direction>;`

Exemple :

Créer un carré rouge qui change de couleur en **4 étapes**. L'animation doit durer **4 secondes**, commencer après **1 seconde**, se répéter à l'infini et aller dans **les deux sens** à chaque cycle.

Réponse :

div	keyframes
<pre>div { width: 100px; height: 100px; background-color: red; /* animation */ animation-name: moveColor; animation-duration: 4s; animation-delay: 1s; animation-iteration-count: infinite; animation-direction: alternate; }</pre>	<pre>@keyframes moveColor { 0% { background-color: red; } 33% { background-color: yellow; } 66% { background-color: green; } 100% { background-color: blue; } }</pre>

Chapitre IV JavaScript



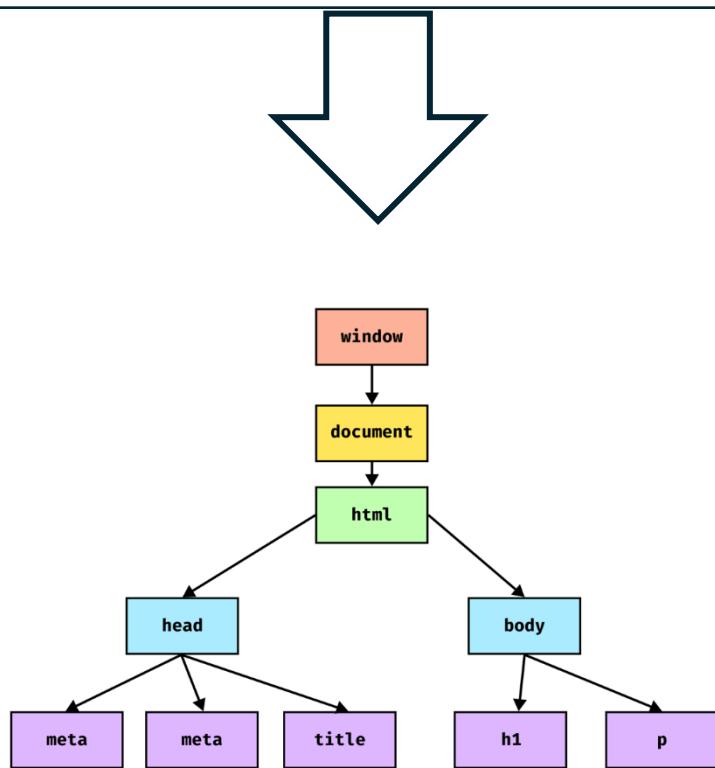
I. Introduction :

JavaScript est un langage de programmation essentiel pour le développement web. Il est principalement utilisé pour ajouter de l'interactivité aux sites web, comme les animations, la validation de formulaires, ou les mises à jour dynamiques sans recharger la page.

Avec JavaScript, on peut manipuler le DOM (Document Object Model) pour modifier dynamiquement le contenu et le style d'une page et créer des applications web interactives.

Exemple arbre DOM :

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
<h1>Hello World!</h1>
<p>This is a paragraph element</p>
</body>
</html>
```



II. Intégration de JavaScript dans le HTML :

Il existe plusieurs façons d'intégrer JavaScript dans un fichier HTML. Voici les méthodes principales :

1. intégration interne

Vous pouvez écrire du code JavaScript directement dans une balise `<script>` à l'intérieur du fichier HTML, généralement placée dans la section `<head>` ou avant la fermeture de `<body>`.

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemple JavaScript</title>
    <script>
        // Exemple de code JavaScript
        alert("Bonjour depuis JavaScript !");
    </script>
</head>
<body>
    <h1>Intégration interne de JavaScript</h1>
</body>
</html>
```

2. intégration externe

Vous pouvez écrire le code JavaScript dans un fichier séparé (par exemple, `script.js`) et le lier au fichier HTML avec une balise `<script>` contenant l'attribut `src`.

Fichier HTML

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemple JavaScript Externe</title>
    <script src="script.js"></script>
</head>
<body>
    <h1>Intégration externe de JavaScript</h1>
</body>
</html>
```

Fichier `script.js`

```
alert("Bonjour depuis un fichier externe !");
```

3. événements

Vous pouvez également écrire du JavaScript directement dans certains attributs HTML, comme onclick, onmouseover, etc. Cependant, cette méthode est moins recommandée, car elle mélange le code HTML et JavaScript.

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemple JavaScript Inline</title>
</head>
<body>
    <h1>Exemple avec un événement JavaScript</h1>
    <button onclick="alert('Bonjour !')">Cliquez-moi</button>
</body>
</html>
```

III. Affichage de données :

JavaScript offre plusieurs méthodes pour afficher du contenu, chacune adaptée à un usage spécifique. Voici les trois principales : alert, document.write, et console.log.

1. alert

La méthode alert affiche un message dans une fenêtre contextuelle (popup) au navigateur. Cette fenêtre est modale, ce qui signifie qu'elle bloque l'interaction avec la page jusqu'à ce que l'utilisateur clique sur "OK".

Exemples :

```
alert("Bienvenue sur mon site !");
// Affiche une fenêtre avec le message "Bienvenue sur mon site!"
```

Points importants :

- ❖ Principalement utilisée pour des notifications ou des messages d'information simples.
- ❖ Elle peut être intrusive et gênante si elle est utilisée fréquemment.
- ❖ Ne permet pas de personnaliser la fenêtre.

2. document.write

La méthode document.write écrit directement dans le document HTML. Elle peut être utilisée pour ajouter du contenu pendant le chargement de la page.

Exemples :

```
document.write("Bonjour, monde !");
// Affiche "Bonjour, monde !" directement dans la page.
```

Points importants :

- ❖ Si utilisée après le chargement initial de la page, elle remplace tout le contenu existant de la page.
- ❖ Rarement utilisée aujourd'hui, sauf pour des démonstrations ou des scripts très simples.

IV. Déclaration et lecture des variables :**1. let**

let est un mot-clé qui permet de déclarer une **variable à portée de bloc**.

- ❖ **Portée** : bloc {}
- ❖ **Réassignable** : oui, on peut changer sa valeur
- ❖ **Non redéclarable** dans le même bloc
- ❖ **undefined**: Signifie qu'une variable n'a pas été initialisée
- ❖ **null** : Valeur vide.

Exemple d'utilisation :

```
let age = 25;
alert(age); // 25
age = 30; // Modification autorisée
alert(age); // 30
```

2. const (pour les constantes)

En JavaScript, **const** est un mot-clé qui permet de déclarer une **variable constante**, c'est-à-dire **une variable dont la valeur ne peut pas être réassignée** après sa déclaration.

- ❖ **Portée** : bloc {} (comme let)
- ❖ **Réassignation** : interdite
- ❖ **Mutabilité** : les objets et tableaux déclarés avec const peuvent être modifiés, mais pas réassignés.

Exemple d'utilisation :

```
const PI = 3.14;
alert(PI); // 3.14
// PI = 3.14159; // Erreur : une constante ne peut pas être modifiée
```

3. Lecture des variables avec prompt

La méthode prompt permet d'afficher une boîte de dialogue qui demande à l'utilisateur de saisir une valeur. Cette méthode est utilisée pour lire des entrées utilisateur directement depuis le navigateur.

Syntaxe de prompt

```
let variable = prompt(message, valeurParDefaut);
```

- ❖ **message** : Le message affiché dans la boîte de dialogue pour demander une saisie à l'utilisateur.
- ❖ **valeurParDefaut (facultatif)** : Une valeur initiale affichée dans le champ de saisie. Si elle n'est pas définie, le champ sera vide par défaut.

V. typage dans JavaScript

1. Number (Nombres)

En JavaScript, les nombres sont représentés par le type number, qu'il s'agisse d'entiers ou de nombres à virgule flottante.

Exemples de déclaration :

```
let entier = 42;           // Entier
let decimal = 3.14;        // Nombre décimal
let negatif = -10;         // Négatif
let nan = NaN;             // "Not a Number"
```

a. Opérations arithmétiques :

Opérateur	Description	Exemple	Résultat
+	Addition	5 + 3	8
-	Soustraction	5 - 3	2
*	Multiplication	5 * 3	15
/	Division	15 / 3	5
%	Modulo (reste de la division)	5 % 3	2
++	Incrémantation (ajoute 1)	let x = 5; x++;	x = 6
--	Décrémentation (retire 1)	let x = 5; x--;	x = 4

b. Objet Math :

Méthode	Description	Exemple	Résultat
Math.abs(x)	Valeur absolue de x	Math.abs(-5)	5
Math.sqrt(x)	Racine carrée de x	Math.sqrt(16)	4
Math.round(x)	Arrondi à l'entier le plus proche	Math.round(4.6)	5
Math.trunc(x)	Partie entière de x (supprime les décimales)	Math.trunc(4.9)	4
Math.random()	Renvoie un nombre aléatoire compris entre 0 (inclus) et 1 (exclus)	Math.random()*5	nombre entre 0 et 5 (exclus)

c. Conversion vers un nombre :

Méthode	Description	Exemple	Résultat
Number(value)	Convertit une valeur en nombre (entier ou flottant).	Number("42")	42
		Number("42.5")	42.5
		Number("abc")	NaN
parseInt(string)	Convertit une chaîne en entier (base 10 par défaut).	parseInt("42")	42
		parseInt("42.9")	42
parseInt(string, radix)	Convertit une chaîne en entier selon la base spécifiée (radix).	parseInt("1010", 2)	10
		parseInt("FF", 16)	255
		parseInt("77", 8)	63
parseFloat(string)	Convertit une chaîne en nombre flottant .	parseFloat("42.5")	42.5
		parseFloat("abc 42.5")	NaN

2. String (Chaînes de caractères)

Les chaînes de caractères sont des séquences de caractères, délimitées par des guillemets simples ('), doubles ("") ou des backticks (` `). Elles sont immuables : toute modification génère une nouvelle chaîne.

Exemple de déclaration :

```
let chaine = "Bonjour, JavaScript!";
```

Opération sur les chaines :

Méthode / Propriété	Description	Exemple	Résultat
+	Opérateur de concaténation de chaînes	"Hello " + "World"	"Hello World"
ch.length	Retourne la longueur de la chaîne ch	"Hello".length	5
ch.indexOf(ch1 [, p])	Position de la 1ère occurrence de ch1 dans ch, recherche à partir de p (optionnel)	"Hello".indexOf("l")	2
ch.lastIndexOf(ch1 [, p])	Position de la dernière occurrence de ch1 dans ch, recherche à partir de p (optionnel)	"Hello".lastIndexOf("l")	3

Méthode / Propriété	Description	Exemple	Résultat
ch.substring(d, f)	Retourne une sous-chaîne de ch de la position d à f (non incluse)	"Hello".substring(1,4)	"ell"
ch.replace(ch1, ch2)	Remplace la 1ère occurrence de ch1 par ch2 dans ch	"Hello".replace("l","L")	"HeLlo"
ch.toLowerCase()	Convertit tous les caractères de ch en minuscules	"HELLO".toLowerCase()	"hello"
ch.toUpperCase()	Convertit tous les caractères de ch en majuscules	"hello".toUpperCase()	"HELLO"
ch.trim()	Supprime les espaces au début et à la fin de ch	" Hello ".trim()	"Hello"
String.fromCharCode(nu m, ..., numN)	Crée une chaîne à partir des codes ASCII passés en paramètre	String.fromCharCode(72 ,101,108,108,111)	"Hello"

3. Les booléens :

Un booléen est un type de donnée en JavaScript qui ne peut avoir que deux valeurs :

- ❖ true (vrai)
- ❖ false (faux)

Les booléens sont souvent le résultat d'une opération de comparaison ou logique. `false`

a. Opérateurs de comparaison

Les opérateurs de comparaison permettent de comparer des valeurs. Ils renvoient toujours une valeur booléenne (true ou false).

Opérateur	Description	Exemple	Résultat
<code>==</code>	Égalité (valeur seulement)	<code>5 == "5"</code>	true
<code>!=</code>	Différent (valeur seulement)	<code>5 != "5"</code>	false
<code>></code>	Supérieur	<code>10 > 5</code>	true
<code><</code>	Inférieur	<code>5 < 10</code>	true
<code>>=</code>	Supérieur ou égal	<code>10 >= 10</code>	true
<code><=</code>	Inférieur ou égal	<code>5 <= 10</code>	true

b. Opérateurs logiques :

Opérateur	Nom	Description
&&	ET logique	Renvoie true si toutes les conditions sont vraies
	OU logique	Renvoie true si au moins une condition est vraie
!	NON logique (négation)	Renverse la valeur : true devient false et vice-versa

4. Objet Date :

L'objet **Date** en JavaScript permet de **travailler avec les dates et les heures**. On peut créer un objet date, récupérer ses composants (jour, mois, année) et afficher la date sous différentes formes.

Méthode / Syntaxe	Description	Exemple	Résultat
<code>new Date()</code>	Crée un objet Date représentant la date et l'heure courante	<code>let d = new Date();</code>	d contient la date et l'heure système
<code>new Date(ch)</code>	Crée un objet Date à partir d'une chaîne de caractères représentant une date	<code>let d= new Date("2025-08-19")</code>	"Tue Aug 19 2025 00:00:00 GMT+0100"
<code>d.getDate()</code>	Retourne le jour du mois (1 à 31)	<code>d.getDate()</code>	19
<code>d.getMonth()</code>	Retourne le numéro du mois (0 = janvier, 11 = décembre)	<code>d.getMonth()</code>	7
<code>d.getFullYear()</code>	Retourne l' année sur 4 chiffres	<code>d.getFullYear()</code>	2025
<code>d.toString()</code>	Retourne la date complète sous forme de chaîne	<code>d.toString()</code>	"Mon Aug 19 2025 09:00:00 GMT+0100"

5. Array (Tableaux)

En JavaScript, un tableau (ou Array) est une collection ordonnée de valeurs. Ces valeurs peuvent être de n'importe quel type (numériques, chaînes de caractères, objets, autres tableaux, etc.).

a. Créer un tableau vide :

```
let tableauVide = [];
```

b. Créer un tableau avec des valeurs initiales :

```
let fruits = ["pomme", "banane", "cerise"];
```

c. accéder à un élément du tableau :

```
let fruits = ["pomme", "banane", "cerise"];
fruits[2] = "orange";
alert(fruits[2]) // orange
```

d. taille d'un tableau (length)

```
let fruits = ["pomme", "banane", "cerise"];
alert(fruits.length) // 3
```

VI. Les structures de contrôle conditionnelles :

Les structures conditionnelles permettent d'exécuter des instructions spécifiques en fonction de la validité d'une ou plusieurs conditions.

Structure	Description	Exemple simple
if	Exécute un bloc de code si la condition est vraie	if(x > 0) { alert("Positif"); }
if...else	Exécute un bloc si vrai, un autre bloc si faux	if(x > 0) { alert("Positif"); } else { alert("Non positif"); }
if...else if...else	Permet tester plusieurs conditions successives	if(x > 0){ alert("Positif"); } else if(x < 0){ alert("Négatif"); } else{ alert("Zéro"); }
switch	Compare une expression à plusieurs valeurs possibles	<pre>switch(month) { case 9:case 10 :case 11: case 12 : alert("1er trimestre"); break; case 1:case 2:case 3: alert("2ème trimestre"); break; case 4:case 5:case 6: alert("3ème trimestre"); break; default: alert("Mois invalide"); }</pre>

VII. Structures itératives

Les **structures itératives** permettent d'exécuter un bloc de code **plusieurs fois**, tant qu'une condition est vraie ou selon un compteur.

Structure	Description	Exemple simple
for	Exécute un bloc de code un nombre déterminé de fois	<code>for (let i=1; i<=3; i++) { alert(i); }</code>
while	Exécute un bloc tant que la condition est vraie	<code>let i=1; while(i<=3) { alert(i); i++; }</code>
do...while	Exécute un bloc au moins une fois , puis répète tant que la condition est vraie	<code>let i=1; do{ alert(i); i++; } while(i<=3);</code>

VIII. Les fonctions

Une **fonction** en JavaScript est un **bloc de code réutilisable** qui peut être appelé plusieurs fois. Elle peut recevoir des **paramètres** et retourner une **valeur**.

Syntaxe d'une fonction :

```
function nomDeLaFonction(param1, param2, ...) {
    // Instructions à exécuter
    return valeur; // (optionnel)
}
```

Exemple simple :

```
function addition(a, b) {
    return a + b;
}
alert(addition(4, 6)); // Résultat : 10
```

IX. Manipulations sur les éléments HTML

1. Ciblage des éléments :

Pour manipuler un élément HTML dans une page, on doit d'abord le **cibler**. Il existe plusieurs façons de le faire, les plus simples étant via **id** et **name**.

a. Par identifiant :

Le ciblage par **id** se fait avec `document.getElementById("id")`. Cette méthode retourne un seul élément car l'attribut id doit être unique dans la page. On peut ensuite accéder ou modifier son contenu.

Exemple :

```
<h1 id="titre">Bienvenue</h1>
<h1 id="titre2"> Contact </h1>
<script>
    let t = document.getElementById("titre");
    alert(t.innerHTML); // Affiche : Bienvenue
</script>
```

b. Par nom :

Le ciblage par **name** se fait avec `document.getElementsByName("nom")`. Cette méthode retourne un tableau car plusieurs balises peuvent partager le même attribut name. Pour accéder à un élément précis, il faut utiliser un indice comme `0,1,...,tableau.length-1`.

```
<input type="radio" name="genre" value="M">Malle
<input type="radio" name="genre" value="F">Femelle

<script>
  let champ = document.getElementsByName("email") [1];
  alert(champ.value); // Affiche : F
</script>
```

2. Manipulation de contenu d'un élément innerHTML :

La propriété **innerHTML** permet de récupérer ou de modifier le contenu HTML d'un élément ciblé. On l'utilise après avoir sélectionné l'élément (par id, name, etc.).

La propriété **innerHTML** permet de récupérer ou de modifier le contenu HTML d'un élément ciblé. On l'utilise après avoir sélectionné l'élément (par id, name, etc.).

```
<p id="paragraphe">Texte original</p>
<button onclick="changer()">Changer le texte</button>
<script>
  function changer() {
    let p = document.getElementById("paragraphe");
    p.innerHTML = "Texte modifié avec JavaScript";
  }
</script>
```

Texte original

Changer le texte



Texte modifié avec JavaScript

Changer le texte

si l'utilisateur clic sur le bouton

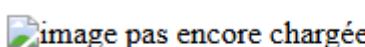
3. Modification des attributs :

En JavaScript, on peut directement modifier la valeur d'un attribut d'un élément HTML en utilisant la syntaxe :

```
element.attribut = "valeur";
```

Exemple d'utilisation :

```
<img id="photo" src="" alt="image pas encore chargée">
<button onclick="changer()">Changer l'image</button>
<script>
  function changer() {
    let img = document.getElementById("photo");
    img.src = "image2.jpg"; // changement d'attribut src
  }
</script>
```

 image pas encore chargée

**4. Modification du style**

En JavaScript, on peut accéder directement au style d'un élément HTML et changer ses propriétés CSS avec la syntaxe :

```
element.style.propriété = "valeur";
```

Remarque : en JavaScript, les propriétés CSS qui contiennent un tiret (background-color, font-size) doivent être écrites en **camelCase** :

- ✓ backgroundColor
- ✓ fontSize

Exemple d'utilisation :

```
<p id="texte">Bonjour tout le monde !</p>
<button onclick="changerCouleur()">Changer couleur</button>
<script>
  function changerCouleur() {
    let p = document.getElementById("texte");
    p.style.color = "red";           // couleur du texte
    p.style.fontSize = "20px";       // taille de la police
    p.style.backgroundColor = "yellow"; // couleur de fond
  }
</script>
```

Bonjour tout le monde !

Bonjour tout le monde !

5. Méthodes principales pour contrôler une vidéo/audio

En JavaScript, on peut cibler un élément <video> ou <audio> et utiliser ses méthodes intégrées :

Syntaxe	Description
element.play()	Lance la lecture du média.
element.pause()	Met en pause la lecture du média.

Exemple d'utilisation :

```
<video id="maVideo" width="320" height="240" controls>
  <source src="exemple.mp4" type="video/mp4">
  Votre navigateur ne supporte pas la vidéo.
</video>
<br>
<button onclick="lire()">Lire</button>
<button onclick="pause()">Pause</button>
<script>
  let video = document.getElementById("maVideo");
  function lire() {
    video.play(); // jouer la vidéo
  }
  function pause() {
    video.pause(); // mettre en pause
  }
</script>
```



Chapitre V PHP



I. Introduction

PHP (Hypertext Preprocessor) est un langage de programmation côté serveur, spécialement conçu pour le développement web. PHP est souvent utilisé en conjonction avec des technologies comme HTML, CSS, JavaScript, et des bases de données, formant un socle solide pour la majorité des applications web modernes.

Les principales utilisations de PHP incluent :

- ❖ **Création de sites dynamiques** : Génération de contenu en fonction des préférences des utilisateurs.
- ❖ **Gestion des formulaires** : Collecte, validation et traitement des données saisies par les utilisateurs.
- ❖ **Interaction avec des bases de données** : Gestion des données stockées, comme les utilisateurs, les produits, ou les articles.

II. Fonctionnement d'un site web dynamique en PHP



PHP est un langage de programmation côté serveur, ce qui en fait un outil idéal pour créer des sites web dynamiques. Voici les étapes qui expliquent comment un site web dynamique fonctionne avec PHP :

1. Requête de l'utilisateur :

L'utilisateur envoie une requête HTTP (par exemple, en saisissant une URL ou en soumettant un formulaire). Cette requête est envoyée au serveur web (Apache dans notre cas).

2. Traitement par le serveur web :

- Le serveur web identifie le fichier PHP correspondant à la requête (par exemple, index.php).

- Si le fichier demandé contient du code PHP, le serveur web transmet ce fichier à l'interpréteur PHP pour traitement.

3. Exécution du script PHP :

- Le script PHP peut :
 - ✓ Récupérer des données depuis une base de données (MySQL dans notre cas).
 - ✓ Traiter les données envoyées par l'utilisateur (via un formulaire, une requête,...).
 - ✓ Effectuer des calculs ou des traitements sur les données.
- PHP génère ensuite un fichier HTML contenant du contenu dynamique basé sur les résultats des traitements.

4. Réponse envoyée au navigateur :

- Une fois le script PHP exécuté, le serveur web renvoie le résultat (un fichier HTML avec le contenu généré dynamiquement) au navigateur de l'utilisateur.
- Le navigateur interprète ce fichier HTML et affiche le contenu.

III. Environnement de travail



XAMPP Lite est une version légère de XAMPP, un environnement de développement local qui regroupe tous les outils nécessaires pour exécuter des scripts PHP et interagir avec des bases de données MySQL sur un serveur local. Il est composé principalement de :

- ❖ **Apache** : Serveur web qui exécute les fichiers PHP.
- ❖ **PHP** : Langage de programmation côté serveur pour générer des pages dynamiques.
- ❖ **MySQL** : Système de gestion de bases de données relationnelles.
- ❖ **PhpMyAdmin** : Interface graphique pour gérer les bases de données MySQL.

IV. Les base d'un script PHP

Un script PHP est composé de différents éléments structurés pour exécuter des instructions côté serveur. Voici les principaux composants d'un script PHP :

1. Déclaration du code PHP :

Le code PHP doit être placé entre les balises de délimitation suivantes :

```
<?php
//Code PHP ici
echo 'bonjour' ;
?>
```

La balise d'ouverture `<?php` indique au serveur que tout ce qui suit doit être interprété comme du PHP. La balise de fermeture `?>` est utilisée pour terminer le script.

Remarques :

- ❖ Le `//` est utilisé pour faire des commentaires. Les commentaires permettent de documenter le code.
- ❖ Les instructions PHP terminent généralement par un point-virgule ;
- ❖ `echo` Affiche une ou plusieurs expressions séparées par des points

2. Variables :

Les variables en PHP commencent par le symbole `$` et servent à stocker des données :

```
<?php
$nom = "Mohamed"; // Variable qui contient une chaîne de caractères
$age = 25; // Variable qui contient un entier
?>
```

3. Typage et fonctions prédéfinies :

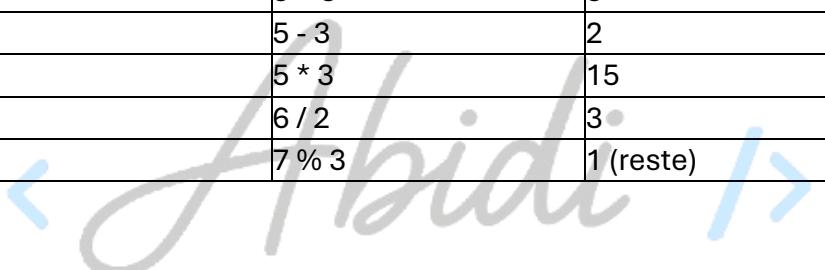
a. Les nombres

En PHP, les nombres sont des types de données utilisés pour effectuer des calculs mathématiques. On distingue deux principaux types de nombres :

1. `int` (entiers)
2. `float` (nombres à virgule flottante)

Opérations arithmétiques de base :

Opération	Opérateur	Exemple	Résultat
Addition	<code>+</code>	<code>5 + 3</code>	8
Soustraction	<code>-</code>	<code>5 - 3</code>	2
Multiplication	<code>*</code>	<code>5 * 3</code>	15
Division	<code>/</code>	<code>6 / 2</code>	3
Modulo	<code>%</code>	<code>7 % 3</code>	1 (reste)



Fonctions mathématiques :

Fonction	Description	Syntaxe	Exemple	Résultat
abs()	Retourne la valeur absolue d'un nombre.	abs(nombre)	abs(-15)	15
sqrt()	Calcule la racine carrée d'un nombre.	sqrt(nombre)	sqrt(16)	4
rand()	Génère un nombre entier aléatoire entre deux bornes.	rand(min, max)	rand(1, 100)	Un nombre entre 1 et 100
round()	Arrondit un nombre à l'entier le plus proche ou à une précision définie.	round(nombre, précision)	round(3.14159, 2)	3.14

b. Les chaînes de caractères en PHP

En PHP, les chaînes de caractères (ou **strings**) sont des séquences de caractères. Vous pouvez effectuer plusieurs opérations sur les chaînes de caractères.

La concaténation de chaînes permet de combiner plusieurs chaînes de caractères en une seule. En PHP, on utilise l'opérateur **"."** pour cela.

Exemple de concaténation :

```
<?php
$prenom = "Amine";
$nom = "Tounsi";
$nom_complet = $prenom . " " . $nom; echo $nom_complet;
// Affiche : Amine Tounsi
?>
```

Fonctions prédéfinies pour les chaînes de caractères

Fonction	Description	Exemple	Résultat
chr()	Retourne le caractère correspondant à un code ASCII.	chr (65)	A
ord()	Retourne le code ASCII du premier caractère.	ord ("A")	65
strlen()	Retourne la longueur d'une chaîne.	strlen ("Hello")	5
substr()	Retourne une sous-chaîne d'une chaîne.	substr ("Hello", 1, 3)	ell
strpos()	Retourne la position de la première occurrence d'une sous-chaîne.	strpos ("Hello", "e")	1

Fonction	Description	Exemple	Résultat
strcmp()	Compare deux chaînes.	strcmp ("apple", "banana")	-1
str_replace()	Remplace une sous-chaîne par une autre.	str_replace ("World", "PHP", "Hello, World!")	Hello, PHP!
strtolower()	Convertit une chaîne en minuscules.	strtolower ("HELLO")	hello
strtoupper()	Convertit une chaîne en majuscules.	strtoupper ("hello")	HELLO
trim()	Supprime les espaces au début et à la fin d'une chaîne.	trim (" Hello ")	Hello

c. Les booléens :

En PHP, les **booléens** sont utilisés pour représenter deux valeurs possibles : true (vrai) et false (faux). Ils sont principalement utilisés dans les conditions et les boucles.

Les opérateurs de comparaison

Opérateur	Description	Exemple	Résultat
==	Vérifie si deux valeurs sont égales (sans tenir compte du type).	5 == "5"	true
<>	Vérifie si deux valeurs ne sont pas égales.	5 <> 3	true
>	Vérifie si la première valeur est plus grande que la deuxième.	5 > 3	true
<	Vérifie si la première valeur est plus petite que la deuxième.	5 < 3	false
>=	Vérifie si la première valeur est plus grande ou égale à la deuxième.	5 >= 5	true
<=	Vérifie si la première valeur est plus petite ou égale à la deuxième.	3 <= 5	true

Les opérateurs logiques

Opérateur	Description	Exemple	Résultat
&&	ET logique : Retourne true si les deux expressions sont vraies.	(5 > 3) && (8 < 10)	true
 	OU logique : Retourne true si l'un de deux expression est vraies.	5 == 2 4 > 3	true
!	NON logique : Inverse la valeur booléenne d'une expression.	!(5 > 3)	false

d. Fonctions des date/heure :

En PHP, les fonctions de **date/heure** sont utilisées pour manipuler et formater les dates et heures.
Voici un aperçu des principales fonctions pour travailler avec la date et l'heure :

Fonction	Description	Exemple	Résultat
checkdate(mois, jour, année)	Vérifie si une date est valide. Retourne true si oui, sinon false .	checkdate(2, 29, 2023)	false (2023 n'est pas bissextile)
date(format [, timestamp])	Retourne une date formatée selon le format indiqué.	date("d/m/Y")	19/08/2025(varie selon la date système)
time()	Retourne le timestamp Unix courant (nombre de secondes depuis 01/01/1970).	time()	1766208904 (varie selon la date système)
strtotime(string [, timestamp])	Convertit une chaîne en timestamp Unix .	strtotime("2025-08-19")	1755571200

e. Les tableaux :

En PHP, les **tableaux** sont des structures de données permettant de stocker plusieurs valeurs sous une seule variable. Il existe deux types principaux de tableaux : les **tableaux indexés** et les **tableaux associatifs**.

Exemple 1 tableau indexé :

```
<?php
$fruits = array ("Pomme", "Banane", "Orange");
echo $fruits[0]; // Pomme
echo $fruits[2]; // Orange
?>
```

Exemple 2 tableau associatif :

```
<?php
$personne = array (
    "nom" => "Ali",
    "age" => 25,
    "ville" => "Tunis"
);
// Accès aux éléments
echo $personne["nom"]; // Ali
echo $personne["ville"]; // Tunis
?>
```

Exemple 3 fonction count :

La fonction count() retourne le nombre d'éléments d'un tableau.

```
<?php
$fruits = ["Pomme", "Banane", "Orange"];
echo count($fruits); // 3
?>
```

4. Structures de contrôles conditionnelles :

Les structures de contrôle conditionnelles permettent d'exécuter un bloc de code uniquement si une condition est vraie, ou de choisir entre plusieurs blocs selon différentes conditions.

Structure	Description	Exemple
if	Exécute un bloc de code si la condition est vraie	<code>if (\$x > 0) { echo "Positif"; }</code>
if...else	Exécute un bloc si la condition est vraie, sinon un autre bloc	<code>if (\$x > 0) { echo "Positif"; } else { echo "Non positif"; }</code>
if...elseif...else	Permet de tester plusieurs conditions successives	<code>if (\$x > 0) { echo "Positif"; } elseif (\$x < 0) { echo "Négatif"; } else { echo "Zéro"; }</code>
switch	Compare une expression à plusieurs valeurs possibles	<code>switch (\$mois) { case 1: echo "Janvier"; break; case 2: echo "Février"; break; default: echo "Autre mois"; }</code>

5. Structures itératives :

Les structures itératives permettent **d'exécuter un bloc de code plusieurs fois**, soit un nombre déterminé de fois, soit tant qu'une **condition est vraie**.

Structure	Description	Exemple
for	Exécute un bloc un nombre déterminé de fois	<code>for (\$i=1; \$i<=5; \$i++) { echo \$i . "
"; }</code>
while	Exécute un bloc tant que la condition est vraie	<code>\$i=1; while (\$i<=5) { echo \$i . "
"; \$i++; }</code>
do...while	Exécute un bloc au moins une fois puis répète tant que la condition est vraie	<code>\$i=1; do { echo \$i . "
"; \$i++; } while (\$i<=5);</code>

6. Les fonctions :

Une **fonction** en PHP est un bloc de code réutilisable qui peut **prendre des paramètres et retourner une valeur**. Les fonctions permettent de structurer le code et d'éviter les répétitions.

Syntaxe d'une fonction :

```
function nomDeLaFonction(param1, param2, ...) {
    // Instructions à exécuter
    return valeur; // optionnel
}
```

Exemple :

```
<?php
function addition($a, $b) {
    return $a + $b;
}
echo addition(4, 6); // Affiche 10
?>
```

7. Opérateurs de transtypage :

En PHP, les **opérateurs de transtypage** permettent de **convertir explicitement une variable d'un type à un autre**, par exemple transformer une chaîne en entier ou un entier en booléen.

Opérateur	Description	Exemple	Résultat
(int)	Convertit une variable en entier	\$x = (int) "123";	123
(float)	Convertit une variable en réel (float)	\$x = (float) "12.3";	12.3
(string)	Convertit une variable en chaîne	\$x = (string) 123;	"123"
(bool)	Convertit une variable en booléen	\$x = (bool) 0;	false
(array)	Convertit une variable en tableau	\$x = (array) "texte";	["texte"]

8. Fonctions diverses :

a. die()

La fonction **die()** arrête immédiatement l'exécution du script et peut afficher un message d'erreur. Elle est souvent utilisée pour gérer les erreurs ou vérifier qu'une condition critique est remplie avant de continuer.

Exemple :

```
<?php
$nom = "";
if ($nom == "") {
    die ("Erreur : le nom est vide !");
} ?>
```

⇒ Dans cet exemple, si \$nom est vide, le script s'arrête et affiche le message d'erreur.

b. **isset()**

La fonction **isset()** permet de vérifier si une variable existe et n'est pas nulle. Elle retourne **true** si la variable existe et **false** sinon.

Exemple :

```
<?php
if(isset($age)) {
    echo "La variable age existe et vaut $age";
} else {
    echo "La variable age n'existe pas";
}
?>
```

c. **require()**

La fonction **require()** permet d'inclure un fichier PHP dans le script. Si le fichier n'existe pas ou ne peut pas être chargé, le script s'arrête avec une erreur fatale. C'est utile pour séparer le code en plusieurs fichiers (ex : fonctions, configuration, templates).

Exemple :

```
<?php
require ("config.php"); // inclut le fichier config.php
echo "Fichier inclus avec succès";
?>
```

V. Principe de transmission de données entre des pages web

Lorsqu'une application web nécessite de transmettre des informations d'une page à une autre, PHP utilise des mécanismes adaptés. Les données peuvent être transmises via des **méthodes HTTP** (comme GET ou POST), et PHP les rend accessibles via des **variables superglobales** comme `$_GET` et `$_POST`.

1. La méthode GET :

La méthode **GET** envoie les données via l'**URL**. Elle est visible dans la barre d'adresse et convient pour des informations non sensibles.

Exemple :

Page html :

```
<form action="traitement_get.php" method="get">
    Nom : <input type="text" name="nom">
    <input type="submit" value="Envoyer">
</form>
```

Si l'utilisateur saisit "Ali" dans le formulaire, l'URL devient :

```
traitement_get.php?nom=Ali
```

Page PHP :

```
<?php
// Récupérer la valeur de la variable nom
$nom = $_GET['nom'];
echo "Bonjour, $nom !";
?>
```

2. La méthode POST :

La méthode **POST** envoie **les données dans le corps de la requête**, ce qui est plus sûr pour les informations sensibles (mot de passe, email...).

Exemple :

Page html :

```
<form action="traitement_post.php" method="post">
    Nom : <input type="text" name="nom">
    <input type="submit" value="Envoyer">
</form>
```

Dans ce cas, la donnée **n'apparaît pas** dans l'URL.

Page PHP :

```
<?php
// Récupérer la valeur de la variable nom
$nom = $_POST['nom'];
echo "Bonjour, $nom !";
?>
```

VI. Fonctions PHP pour MySQL

L'extension **MySQLi** fournit un ensemble de fonctions permettant d'interagir avec une base de données MySQL dans une interface procédurale. Voici un aperçu des principales fonctions, leur rôle et des exemples d'utilisation.



Fonction	Description	Exemple
mysqli_connect(host, user, password, db)	Établit une connexion à la base de données MySQL. On peut utiliser or die() pour afficher une erreur si la connexion échoue.	\$conn = mysqli_connect("localhost", "root", "", "test") or die ("Connexion échouée");
mysqli_query(conn, requete)	Exécute une requête SQL sur la connexion donnée. On peut utiliser or die() pour afficher une erreur en cas d'échec.	\$res = mysqli_query(\$conn, "SELECT * FROM utilisateurs") or die(mysqli_error(\$conn));
mysqli_error(conn)	Retourne le message d'erreur de la dernière requête SQL.	echo mysqli_error(\$conn);
mysqli_fetch_array(res)	Récupère une ligne de résultat sous forme de tableau associatif et/ou indexé.	\$row = mysqli_fetch_array(\$res); echo \$row['nom'];
mysqli_fetch_row(res)	Récupère une ligne de résultat sous forme de tableau indexé.	\$row = mysqli_fetch_row(\$res); echo \$row[1];
mysqli_num_rows(res)	Retourne le nombre de lignes dans le résultat d'une requête SELECT.	echo mysqli_num_rows(\$res);
mysqli_affected_rows(conn)	Retourne le nombre de lignes affectées par la dernière requête INSERT, UPDATE ou DELETE.	echo mysqli_affected_rows(\$conn);
mysqli_close(conn)	Ferme la connexion à la base de données.	mysqli_close(\$conn);

Exemple 1 : insertion des données d'un formulaire dans un base de données

Soit le formulaire html suivant :

```
<body>
    <h2>Formulaire d'inscription</h2>
    <form action="insertion.php" method="post">
        Nom : <input type="text" name="nom" required><br><br>
        Email : <input type="email" name="email" required><br><br>
        <input type="submit" value="S'inscrire">
    </form>
</body>
```

Formulaire d'inscription

Nom :

Email :



Partie php : insérer les données dans une table "utilisateurs" d'une base de données "test".

```
<?php
// Connexion à la base de données
$conn = mysqli_connect("localhost", "root", "", "test") or
die("Connexion échouée");
// Récupération des données du formulaire
$nom = $_POST['nom'];
$email = $_POST['email'];
// Insertion dans la table
$sql = "INSERT INTO utilisateurs (nom, email) VALUES ('$nom', '$email')";
if(mysqli_query($conn, $sql)){
    echo "Inscription réussie !";
} else {
    echo "Erreur : " . mysqli_error($conn);
}
// Fermeture de la connexion
mysqli_close($conn);
?>
```

Explications :

1. Le formulaire HTML envoie les données via **POST** à insertion.php.
2. La connexion à la base se fait avec `mysqli_connect()` et `or die()` pour gérer l'erreur.
3. La requête SQL `INSERT` ajoute les données dans la table utilisateurs.
4. `mysqli_query()` exécute la requête et `mysqli_error()` permet d'afficher l'erreur si nécessaire.
5. Enfin, `mysqli_close()` ferme la connexion à la base.

Exemple 2 afficher tous les utilisateurs dans un tableau :

```
<?php
// Connexion à la base de données
$conn = mysqli_connect("localhost", "root", "", "test") or
die("Connexion échouée : ");
// Requête pour récupérer tous les utilisateurs
$sql = "SELECT * FROM utilisateurs";
$result = mysqli_query($conn, $sql) or die(mysqli_error($conn));
// Vérification s'il y a des résultats
if(mysqli_num_rows($result) > 0){
    echo "<table border='1' cellpadding='5'>";
    echo "<tr><th>ID</th><th>Nom</th><th>Email</th></tr>";
    // Parcours de chaque ligne avec mysqli_fetch_array
    while($row = mysqli_fetch_array($result)){
        echo "<tr>";
        echo "<td>" . $row['id'] . "</td>";
        echo "<td>" . $row['nom'] . "</td>";
        echo "<td>" . $row['email'] . "</td>";
    }
}
```

```

        echo "</tr>";
    }
    echo "</table>";
} else {
    echo "Aucun utilisateur trouvé.";
}

// Fermeture de la connexion
mysqli_close($conn);
?>

```

Explications

1. **Connexion à la base** : mysqli_connect() avec gestion d'erreur grâce à or die().
2. **Requête SELECT** : SELECT * FROM utilisateurs pour récupérer tous les utilisateurs.
3. **Exécution de la requête** : mysqli_query() retourne un résultat.
4. **Vérification des résultats** : mysqli_num_rows() permet de savoir si la table contient des données.
5. **Affichage dans un tableau HTML** :
 - o mysqli_fetch_array() récupère chaque ligne sous forme de **tableau associatif**.
 - o On parcourt toutes les lignes avec une boucle while pour remplir le tableau.
6. **Fermeture de la connexion** : mysqli_close() libère les ressources

