HTML

Unité de base du HTML est la balise, sa structure est :

```
<nom de la balise>Contenu</nom de la balise>
```

Exemple: `h1, h5, p, div, body, style, script`

Ce genre de balises ont besoin d'être fermés (il suffit de réécrire le même nom de la balise précédé par un `/`) pour que le navigateur sache où le contenu se termine, ou, comment structurer l'arbre qui représente le HTML, appelé le DOM (Document Object Model).

Exemple: `<div><h1>Bonjour</h1></div>`

Il y a des cas où la balise n'a pas besoin d'être fermée, c'est le cas des balises qui n'ont pas besoin de contenus, ou elles n'ont pas besoin d'inclure des balises comme enfants, ces balises sont appelé des balises orphelines.

```
<nom de la balise />
```

Exemple: `img, meta, link, input`

Chaque balise peut avoir d'autres paramètres pour lui changer son comportement par défaut, ou, parceque ce paramètre est obligatoire (pour le cas des balises orphelines), ces paramètres sont appelé des Attributs, et ils sont du genre `clé=valeur`, quand la valeur n'est pas présente, c'est que la valeur est booléenne initialisée à `true`

```
<nom de la balise clé1="valeur 1" clé2=valeur2 clé3 />
```

Dans ce cas par exemple, `clé1` à des guillemets (`""`) parceque la valeur contient des espaces, `clé3` peut être écrite sous la forme `clé3=true`

Exemple:

```
<meta charset=utf8 />
<img src="./image.jpeg">
<input type=checkbox checked>
```

Structure d'une page web :

Elle se compose de deux parties: la tête (`head`), et le corps (`body`).

Le contenu de la page doit être mis dans le body, quant aux informations relatives à la page, elles doivent être mises dans le head.

```
<head>
Information sur la page
</head>
<body>
Contenu de la page que l'utilisateur va voir
</body>
```

Hello world HTML.

Créez un nouveau fichier texte, renommez le en premier.html et ouvrez le avec le navigateur web de votre choix (Chrome, Firefox).

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Hello</title>
</head>
<body>
<!—Mettez ici votre contenu-->
    <h1>Hello World !</h1>
</body>
</html>
```

Petite explication de ce qui s'est passé :

Première ligne est le `doctype`, cette ligne est faite pour dire au navigateur comment il va parser (parcourir) le contenu de la page, en HTML 5 il a été simplifié, parceque dans le passé, fallait écrire une longue ligne, exemple:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Vient après la ligne du `head`, là, on vient de donner l'encodage des caractères que le document va contenir (ici, `utf-8` veut dire `Unicode`), en d'autres termes; dire au document qu'on va insérer des caractères qui sont en dehors de la page de la table `ASCII` (français, arabe…)

Après, vient le contenu de la page qui est dans `body`, et on a commencé le contenu avec un commentaire, ce commentaire sera dans le code source de la page, mais le navigateur ne l'affichera pas à l'utilisateur.

On a terminé avec un texte de premier niveau (**Heading 1**) (ce concept se trouve dans les éditeurs de texte comme Word).

Quelques balises:

On va énumérer quelques balises, mais pas toutes, vu qu'il en existe plusieurs.

Copiez ces balises et mettez la entre le <body> et </body>

Texte de premier niveau (appelé souvent Titre)

```
<h1> Bonjour tout le monde </h1>
```

Paragraphe:

```
<p> mettez ici un long paragraphe … </p>
```

Image:

```
<img src="./img/image.jpeg">
```

Notez que la source contient une syntaxe du chemin un peu bizarre, et non ce qu'on connais des systèmes d'exploitations (sous windows **c:/chemin/vers/source** ou **/home/utilisateur/chemin** sous mac et linux), on aborde alors les concepts des chemins absolus et les chemins relatifs.

Chemin absolu: ici, on donne le chemin exacte de la source, par exemple **C:/utilisateurs/bureau/image/paysage.jpeg** mais le problème qui se pose avec cette syntaxe, est la non portabilité du projet, que ce soit d'une machine à une autre (tout le monde ne s'a pas cette structure de dossier sur sa machine), ou d'un système d'exploitation à une autre (sous linux, il n'y a pas de **C:/**), du coup, on aura un chemin qui sera faux. Voici pourquoi, il faut toujours penser aux chemins relatifs.

Chemin relatif : ici, on donne le chemin relativement au fichier qui l'appelle, en utilisant une syntaxe spéciale (**./** pour dire dossier actuel par rapport au fichier qui s'execute, **../** pour dire dossier parent par rapport au fichier qui s'exécute).

Dans notre cas de l'exemple de l'image, c'est le fichier **HTML** qui appelle l'image, donc le chemin relatif, est relatif au chemin du fichier HTML.

Les liens :

La balise qui sert à représenter le lien est la balise **<a>** en donnant le chemin vers la ressource dans l'attribut **href** et en mettant ce que l'utilisateur va voir entre les deux balises.

Exemple:

```
<a href="http://www.google.fr">http://www.facebook.com</a>
```

Ou un autre exemple:

```
<a href="http://www.google.fr"><img src="./img/image.jpeg"></a>
```

Dans cet exemple, notre image sera cliquable.

Les listes :

On a deux manières pour représenter une liste, avec des numéros, ou avec des puces (appelés respectivement liste ordonnées et listes non ordonnées), elles ont la même syntaxe pour les éléments, ce qui diffère, est juste la déclaration de la liste elle-même :

Liste ordonnée:

```
<ol>
<li> Element A </li>
<li> Element B </li>
<li> Element C </li>
</ol>
```

Liste non ordonné, il suffit de remplacer le **ol** par un **ul**, mais garder la même syntaxe des éléments de la liste.

```
<ul>
<li> Element A </li>
<li> Element B </li>
<li> Element C </li>
</ul>
```

Les tableaux:

Comme dans les listes, on a besoin de "boucler" pour avoir le résultat, sauf que là, on va utiliser une sorte de boucle imbriquée, on aura besoin de 3 balises essentielles, celle de la table, celle de la ligne, et celle de la colonne, et la logique de ce tableau consiste à boucler dans les lignes, et boucler dans chaque ligne pour créer une colonne.

```
<table>
 <tr>
  <td> A </td>
  <td> B </td>
 </tr>
 <tr>
  <td> C </td>
  <td> D </td>
 </tr>
</table>
```

Notez que pour dire à l'utilisateur comment lire son tableau (horizontalement ou verticalement), il est conseillé de remplacer la balise **`td`** par **`th`** pour rendre le contenu de la cellule en gras.

Notez que le navigateur ajoute une autre balise qui est **`<tbody>`** (en faisant inspecter l'élément sur Chrome) pour dire que c'est le contenu du tableau, vous pouvez même rajouter une balise **`<thead>`** qui va regrouper les titres (les **`th`**) du tableau.

Les CSS

On a vu le HTML, mais le HTML ne fait que structurer, il ne faut un moyen comment rajouter des couleurs et organiser ce contenu dans notre page, pour cela, il faut basculer vers une autre syntaxe appelée CSS, pour cela, il faut choisir l'une des 3 méthodes, choisissez une méthode selon votre besoin de "mettre des règles en facteur".

Méthode 1:

Cette méthode est appelée en anglais : inline CSS, qui veut dire, mettre les règles dans la balise elle-même, ceci est fait en utilisant l'attribut style.

Sa syntaxe est:

```
<balise style="regle1: valeur; regle2: valeur">Contenu</balise>
```

Exemple:

```
<h1 style="text-align : center; color: red"> Je suis centre <h1>
```

Méthode 2:

Maintenant, et si on avait 10 balises h1, répéter la même démarche serait mauvais, la solution consiste alors de mettre les éléments redondant en facteur, comme on fait en math, et c'est là, qu'on va utiliser la seconde méthode pour déclarer notre CSS, en utilisant la balise style, ceci se fait avec la syntaxe suivant :

```
<style>
element
{
regle1: valeur;
regle2: valeur
}
</style>
```

Notez que la dernière valeur n'a pas eu besoin du séparateur point-virgule, tout simplement parceque juste après, on trouve une accolade.

Exemple:

```
<h1 style="text-align: center; color:red">Je suis centré <h1>
<h1 style="text-align: center; color:red">centré aussi! <h1>
```

Ici, le texte change, mais la balise est la même, et le CSS est le même. Donc il suffit de refaire :

```
<style>
h1
{
text-align : center;
color : red
}
</style>
<body>
<h1> Je suis centré <h1>
<h1> Je suis centré aussi ! <h1>
</body>
```

Vous pouvez mettre la balise **`style`** dans le **`head`** ou dans le **`body`**, mais il est conseillé vivement de la mettre dans le **`head`**.

Méthode 3 :

On a vu la méthode 2 qui nous fait gagner du temps, mais cette méthode fonctionne quand on a une seule page, et un site web est souvent mis dans plusieurs pages, donc on aura besoin d'une 3ème méthode pour ça, dans cette méthode, on va prendre le contenu de la balise **`<style>`** et le mettre dans un nouveau fichier qui aura une extension **`.css`** et on l'appelle du fichier HTML avec la syntaxe :

```
<link rel="stylesheet" href="./monstyle.css">
```

Et on contenu du fichier monstyle.css aura la syntaxe :

```
element
{
regle1: valeur;
regle2: valeur
}
```

(Notez que cette fois, on n'a pas eu besoin de mettre la balise style vu que la balise appartient au HTML)

Les classes et les id :

On a vu que pour mettre un CSS avec la méthode 2 et 3, il faut préciser quelle balise cibler, mais parfois (et souvent) on voudrait varier les CSS des éléments qui ont la même balise (par exemple, on a 10 h1, et on voudrait mettre 5 en rouge, et 5 en bleu), là, il faut les distinguer, voici pourquoi le concept des classes existe.

Et parfois, on voudrait attribuer un CSS sans tenir en compte la nature de la balise, ainsi, on peut regrouper par exemple, des titres et des paragraphes pour avoir la même police !

Pour les utiliser, il suffit de les déclarer comme attribut dans la balise ciblée **`class="maClasse"`** et **`id="monId"`**, et les déclarer dans le CSS avec la syntaxe **`.maClasse`** et **`#monId`** (rajouter le point pour le nom de la classe, et un dièse (hash) pour le id)

Exemple:

```
<style>
.bleu
{
color : blue
}
.rouge
{
Color : red
}
</style>
<body>
<h1 class=bleu> Je suis en bleu <h1>
<h1 class=bleu> Je suis aussi en bleu ! <h1>
<p class=bleu> Je veux du bleu i ! <p>
<h1 class=rouge> Je suis en rouge ! <h1>
<h5 class=rouge> Je suis aussi en rouge ! <h5>
</body>
```

```
<style>
.bleu {
color : blue
}
#rouge {
color : red
}
h1 {
color : yellow
}
</style>
<body>
<h1 class=bleu  id=rouge> Je suis en bleu <h1>
```

Dans cet exemple, le texte sera en rouge, parceque la valeur de l'id est la plus prioritaire, suivie de celle de la classe, puis, celle de la balise.

Notez qu'on ne peut pas avoir le même id pour deux éléments, il faut qu'il soit unique.

Les pseudo-classes :

Parfois, on aura besoin de changer le comportement de la balise lors d'un événement, par exemple, quand on passe la souris dessus, pour cela, il suffit de déclarer l'événement après le caractère "**`:`**"

```
h1 {
color : red
}
h1: hover{
color: blue
}
```

Ici, le **`h1`** devient bleue à chaque fois on passe la souris dessus.

Pseudo-elements:

Avec ça, on pourra alors donner du style à une partie de l'élément, ou même en rajouter du contenu avec du CSS, la syntaxe est la même que celle du pseudo-classe, mais cette fois, en utilisant "**::**" (deux :)

```
p {
color: red
}
p::first-letter {
font-size : 2em
}
```

Ici, le paragraphe sera en rouge, mais la première lettre aura une taille de 2 fois que celle de la police actuelle.

```
h1::after {
content : "hello"
}
```

Dans cet exemple, à chaque fois qu'on trouve du contenu dans **`h1`**, un texte **`hello`** sera rajouté à la fin.

Remarque par rapport aux chemins relatifs et absolus :

Si on utilise un chemin dans un CSS, il faut utiliser le chemin relatif au chemin où se trouve la règle du CSS, exemple:

```
<style>
div {
background-image : url("../img/plan.jpeg")
}
<style>
```

Ici, on chercher l'image par rapport au dossier où se trouve le fichier HTML,

Par contre; si on avait cette règle dans un fichier CSS à part, ça sera différent, parceque cette fois on sort du dossier là où se trouve le fichier CSS et non là où se trouve le fichier HTML

Javascript

On termine ce petit document avec le coté programmation de la page HTML, vu que le langage HTML n'est pas un langage de programmation, on doit utiliser un langage à part pour gérer le coté programmation, ce langage est javascript (Ecmascript)

Ici, contrairement au CSS, on utiliser deux méthodes pour insérer du code javascript, on va utiliser la méthode 2 et la méthode 3:

Inclus dans la même page :

```
<script>
Votre programme ici
</script>
```

Appelé dans un fichier externe, l'extension est .js

```
<script src="./monscript.js"></script>
```

Notez aussi que contrairement au CSS, on utilise la même balise.

Pour l'emplacement du script, là, il faut toujours mettre son javascript à la fin du document et non dans le **head**, sinon vous pouvez avoir soit un ralentissement de chargement (code bloquant), soit un code qui ne marche pas (dépendant des éléments du DOM qui n'ont pas encore été dessinés).

Quand vous travaillez avec javascript, sachez que les données que vous allez récupérer sont du type texte, donc il faut les convertir vers le format que vous voulez, trois méthodes existent :

`Number("123")` aura **`123`** comme résultat (entier).

`Number("123.56")` aura **`123.56`** comme résultat (réel).

Mais, **`Number("123.56a")`** aura une erreur parceque le nombre contient une lettre, voici pourquoi, il existe les deux méthodes :

`parseInt("123")` aura **`123`** et même si on met **`parseInt("123abc")`** on aura **`123`**

`parseFloat("123.56")` aura **`123.56`** et même **`parseFloat("123.56a")`** on aura **`123.56`** , comme règle, **`parseInt`** sort dès qu'elle trouve une valeur qui n'est pas numérique, **`parseFloat`** sort dès qu'elle trouve une valeur qui n'est pas numérique, ou un second point (parcequ'on ne peut pas avoir un nombre du genre **`123.56.54`**)

Accéder à un élément du DOM:

Soit le fragment de code suivant :

```
<form name=monForm>
<input type=password name=pass1 id=p1 />
<input type=password name=pass2 id=p2/>
<input type=text />
<button>Verifier</button>
</form>
```

Si on voulait avoir la valeur saisie dans le premier champ du mot de passe, on doit utiliser deux étapes :

1. Extraire le nœud (la balise)
2. Extraire la valeur de l'attribut qui nous intéresse.

On applique ce concept sur l'exemple précédent;

1. Pour extraire le nœud, on peut utiliser plusieurs méthodes, mais ces méthodes commencent toujours par l'objet **`document`**, après, on utilise soit un parcours par niveau, soit en cherchant un élément selon un attribut:
    a. Le parcours par niveau, consiste à utiliser les valeurs de l'attribut **`name`**, niveau par niveau, donc la solution à notre problème serait **`document.monForm.pass1`** parcequ'on a le **`body`**, et le niveau de l'arbre qui vient après est le formulaire qui a un **`name=monForm`** puis vient le niveau là où il y a le input qui a le **`name=pass1`**
    b. En cherchant un élément selon son attribut, et ceci se fait par plusieurs manières, mais celle qu'on verra ici serait de chercher un élément avec le nom de la balise, ou avec le nom de son id, et la syntaxe, consiste à faire **`document.getElement`**
        i. Avec le nom de sa balise : vu que dans un document, on peut avoir de 0 à n balise, donc le résultat final serait un tableau, et la syntaxe pour cela est **`document.getElementsByTagName`** notez que **`Element`** est **`Elements`** (pluriel) à cause des cardinalités, et dans notre exemple, on veut chercher la balise **`input`** (balise est appelé aussi HTML tag) :

            ```
            document.getElementsByTagName("input")
            ```

            On aura alors : **`[input, input]`** et puisqu'on veut extraire le premier, on récupère l'indice 0 du tableau :

            ```
            document.getElementsByTagName("input")[0]
            ```

ii. Avec son id : un id est unique, donc on peut soit avoir 0 soit 1, or, le résultat final, ne sera pas dans un tableau, donc, on n'aura pas besoin d'utiliser les indices pour extraire le nœud, la syntaxe est **`document.getElementById`** et notez qu'ici, **`Element`** est au singulier, donc on aura à la fin : **`document.getElementById("p1")`** et le résultat sera directement **`<input type=password name=pass1 id=p1/>`**

2. Maintenant, un fois qu'on a extrait le nœud, on a besoin d'extraire la valeur qui nous intéresse de ce nœud, et vu que javascript est un langage qui n'a pas de type balise, on doit utiliser un type qui peut nous aider pour ça, et ce type-là, est le type objet; si on avait par exemple **`a={clé1: valeur1, clé2: valeur2}`** pour avoir **`valeur1`**, il suffit de faire **`a.clé1`** ou **`a["clé1"]`**
   La clé dans une balise serait l'attribut, et sa valeur est la valeur de cet attribut, par exemple, la balise
   `<input type=password name=pass1 id=p1/>`
   peut être représenté dans un objet sous la forme

```
{"type": "password", "name": "pass1", "id": p1, "value": "", 
className=""}
```

   Notez que tous les attributs qui n'ont pas été déclarés, auront la valeur par défaut chaine vide si la valeur est de type texte (et non un booléen), et l'attribut **`className`** est celui de la classe, vu que le nom **`class`** est un nom réservé.
   Donc, pour récupérer la valeur du mot de passe saisie, on va utiliser directement **`document.getElementById("p1").value`**
   Notez que les valeurs retournées, sont toutes de type **`str`** (chaine de caractère), il faut penser alors à les convertir (en entier par exemple en utilisant **`parseInt`**)

Les événements.

Quand un programme est écrit dans javascript, il sera exécuté une seule fois (lors du chargement de la page), ou, on doit invoquer le code dans la console à chaque fois, donc, on doit trouver un moyen pour "déclencher" ce code à chaque fois qu'un "événement" se produit.

La méthode consiste à attacher à un nœud (une balise) un "écouteur d'événement" (event listener en anglais), et attacher à cet écouteur d'événement la fonction qu'il doit déclencher.
Ces écouteurs ou "sentinelles" peuvent être écrites dans une balise, ou dans le code javascript, on verra pour l'instant uniquement comment l'inclure dans une balise.
Un event listener, est un attribut qui commence toujours par un **`on`** suivi du nom de l'événement qui égale à la fonction voulue, on verra 4 event listeners :
`onclick` : cet événement se déclenche à chaque fois qu'un utilisateur clique sur la zone couverte page la balise.
`onblur` : cet événement se déclenche à chaque fois qu'un utilisateur quitte l'élément, il est souvent utilisé avec la balise **`input`**.
`onsubmit` : cet événement se déclenche à chaque fois qu'un utilisateur envoie un formulaire, donc ça marche uniquement avec la balise **`form`**, si on voulait utiliser le **`onclick`**, on le mettait dans le bouton qui déclenche l'envoie (**`button`** ou **`input type=submit`**).
`onchange` : cet événement se déclenche à chaque fois qu'un utilisateur change le contenu, ceci peut être utilisé avec la balise **`select`** ou avec un contenu visuel (**`input type=range`**, ou **`input type=number`**).
Maintenant, une fois qu'on a précisé quel event listener utiliser, il faut déclarer la fonction à utiliser, avec la syntaxe :

```
function maFonction()
{
Votre code ici
}
```

Et on aura alors la forme finale de la balise :

```
<balise onclick="maFonction()" > </balise>
```

Ce qui veut dire, si on clique sur l'espace couvert par la balise, on déclenchera le code qui se trouve dans la function **`maFonction`**

```
console.log("bon courage !")
```

soit un formulaire suivant :

```
<form>
  <input type="text">
  <input type="text">
  <button>Login</button>
</form>
```

pour que ce formulaire soit visible par le serveur, il faut l'attribut name

```
<form action="">
  <input type="text" name="nom">
  <input type="text" name="mdp">
  <button>Login</button>
</form>
```

et il faut rajouter après le fichier à exécuter une fois que formulaire atteint le serveur, avec l'attribut action , si action sera omit, ça sera le même fichier qui sera exécuté.

```
<form action="fichier.php">
  <input type="text" name="nom">
  <input type="text" name="mdp">
  <button>Login</button>
</form>
```

Une fois ce formulaire envoyé (cliqué sur Login), voici la requête dans le protocole HTTP:

```
http://site.to/fichier.php?nom=abcd&mdp=1234
```

Une foi ce formulaire par défaut, est envoyé avec la méthode (appelé Verbe dans HTTP) GET, donc ce formulaire est en réalité :

```
<form action="fichier.php" method="GET">
  <input type="text" name="nom">
  <input type="text" name="mdp">
  <button>Login</button>
</form>
```

si on voulait cacher des arguments et ne pas être visible dans le lien, on utilise le verbe POST

```
<form action="fichier.php" method="POST">
  <input type="text" name="nom">
  <input type="text" name="mdp">
  <button>Login</button>
</form>
```

et on aura

```
http://site.to/fichier.php
```

Maintenant dans fichier.php il suffit de faire déjà la structure du php :

```
<?php
// votre code ici
?>
```

Exemple 1 :

```
<?php
echo "bonjour tout le monde";
?>
```

Exemple 2 :

```
<?php
$NOM = "Abdelouahab";
echo $NOM;
// on aura Abdelouahab affiché dans la page HTML
?>
```

Exemple Il faut sauvegarder le fichier en format *.php et non *.html sinon les balises php et tout le code php sera affiché comme texte et non évalué, par contre l'extension *.php évalue aussi le code HTML
Maintenant revenant vers notre formulaire, si on veut récupérer les valeurs envoyées par le client, la solution est de découper le requête HTTP; si on a le formulaire qui a été envoyé avec la méthode GET, alors on dit à PHP d'aller regarder le header de la requête et non son body (à ne pas mélanger avec head et body de html, mais ici on parle de requête réseau), or, si on a la lien suivant `http://site.to/fichier.php?nom=abcd&mdp=1234`

la racine de notre application web est / donc on prends `fichier.php?nom=abcd&mdp=1234` et on découpe ça :

`fichier.php :` le fichier à exécuter par le serveur web (Apache dans notre cas)

`? :` le délimiteur pour commencer à fournir les arguments

`nom=abcd :` le premier argument, notez que le seule attribute visible est name et que le serveur ne regarde pas la nature du input si c'est du texte ou password ou radio ....etc

`& :` le délimiteur pour concaténer les arguments envoyés au serveur

`mdp=1234 :` deuxième argument

donc comme on sait que les arguments sont dans le header, et que la méthode est GET, il suffit de faire dans fichier.php :

```
<?php
$PSEUDO = $GET_["nom"]
$PASSWORD = $GET_["mdp"]
?>
```

donc on utilise ici des Array (plusieurs valeurs envoyés en même temps), et on donne la valeur qu'on a mis dans name dans notre HTML

Maintenant si notre formulaire était

```
<form action="fichier.php" method="POST">
  <input type="text" name="nom">
  <input type="text" name="mdp">
  <button>Login</button>
</form>
```

Il suffit de faire :

```
<?php
$PSEUDO = $POST_["nom"]
$PASSWORD = $POST_["mdp"]
?>
```

Gestion de la base de données à partir de PHP

L'astuce est facile, envoyer une chaine de caractère de PHP à notre SGBD et le laisser se débrouiller ;)

ce qui veut dire, notre requête SQL on l'écris comme on a l'habitude, et pour ne pas éveiller des erreurs dans PHP de type syntax error, comme SELECT par exemple n'est pas un mot réservé à PHP, on fait "SELECT" ^_^

```
$IP = "L'ADRESSE IP DU SERVEUR DE BASE DE DONNÉES, ON MET DES FOIS LOCALHOST OU 127.0.0.1 POUR DIRE QUE LE SERVEUR DE BASE DE DONNÉES EST DANS LA MÊME MACHINE."

$USER = "L'UILISATEUR QUI A LES PRÉVILÈGES DANS LA BASE DE DONNÉES, ON MET DES FOIS L'UTILISATEUR ROOT COMME IL EST L'ADMIN PAR DÉFAUT DE LA BASE DE DONNÉES"

$PASSWORD = "LE MOT DE PASSE DE L'ADMIN DE LA BASE DE DONNÉES, ICI LE MOT DE PASSE DE ROOT"

$DBNAME = "LE NOM DE LA BASE DE DONNÉES, ... ET NON CELUI DE LA TABLE ! "
```

On déclare maintenant une variable, et on utilise la fonction `mysqli_connect()` qui va prendre ces arguments.

```
$con = mysqli_connect($IP, $USER, $PASSWORD, $DBNAME);
```

On déclare le requête qu'on veut ici, et on la met dans "" pour faire croire à PHP que c'est une simple chaine de caractères

```
$tous = "SELECT * FROM USERS";
```

On exécute le tout dans la fonction `mysqli_query()` fonction qui prends comme premier argument les informations nécessaire pour la base de données, et comme second argument le requête à exécuter.

```
$requete = mysqli_query($con, $tous);
```

On donc, PHP envoie à MySql les données en texte, si on veut maintenant les récupérer (de MySQL à PHP) on utilise le type Array, et la fonction qui permet de parcourir ce tableau est `mysqli_fetch_array()`

```
while ($row = mysqli_fetch_array($requete)){
    echo $row["id"];
    echo $row["mdp"];
    echo $row["nom"];
}
```

On met une variable ici row pour juste dire la ligne de notre table, et on met les noms des champs de notre table comme indice à récupérer de notre array, donc notre exemple, notre table s'appelle USERS et elle contient les champs id, mdp, nom

Comme on parle chaine de caractères, si on voulait "forger" notre requête SQL, il suffit d'utiliser la concaténation pour créer une chaine de caractères à base de variable :

```
$inserer = "INSERT INTO USERS (id, pwd, name) VALUES (".$x.", ".$y.", ".$z.")";
```

Voici ce vas se passer :

```
"INSERT INTO USERS (id, pwd, name) VALUES ("

+

$x

+

","

+

"$y"

+

","

+

"$z"

+

")"
```

Gestion de la session à partir de PHP

On La session est géré par le "cookie" est le cookie est juste un fichier texte qui contient une clé et une valeur : A=B, C=D, E=F ....
donc il suffit de mettre les valeurs qu'on veut dans le cookie pour pouvoir les ré-utiliser ultérieurement

```
session_start(); // COOKIE  {CLÉ : VALEUR}
$_SESSION['username']=$row["nom"];//on créé un cookie qui contient une clé qu'on nomme nous même "username" et qui est la valeur retournée lors de notre requête vers la base de données.
```

Et pour rediriger un utilisateur, il suffit d'utiliser la fonction `header('location: destination');`
Par exemple :

```
header('location: accueil.php');
```

Voici comment on le contrôle à chaque fois s'il est connecté ou pas, si on trouve pas ce cookie qu'on a créé qui contient la clé "username" alors on l'envoie vers une autre page index.html, sinon, on le laisse continuer à exécuter le reste de la page.

```php
<?php

if(!isset($_SESSION))
{
    session_start();
    if(!isset($_SESSION['username']))
    {
        header('location: index.html');
    }
}
// le reste du code de la page
?>
```

Pour se déconnecter, en HTTP, il suffit de supprimer le cookie de la session, en PHP il suffit de faire :

```
session_start();
session_destroy();
header('Location: index.php');
```

Ainsi, on lui supprime le cookie et on le redirige vers une autre page.