Cours : thème 2 - Question 3 (web et php)

Question 3 (web) : La résolution de tous les problèmes de gestion est-elle automatisable ?

Notions abordées :

- Comprendre l'architecture client-serveur : fonctionnement général d'un site internet ou d'une application web ;
- Créer et mettre en forme des pages web : la création de pages web en HTML et leur mise en forme grâce à des feuilles de styles rédigées en langage CSS ;
- Programmer en PHP : l'utilisation de variables, de tableaux, de structures conditionnelles, de structures itératives, de fonctions, l'affichage de résultats, etc. ;
- Interagir avec l'utilisateur en PHP : l'utilisation de formulaires, des paramètres GET et POST et des paramètres de SESSION ;
- Interagir avec une base de données en PHP : l'utilisation de fonctions permettant de se connecter à une base de données, d'envoyer des requêtes SQL et d'en récupérer le résultat.

1. Préliminaire

1.1. Introduction

Le présent cours constitue une seconde approche de la question 3 du programme et constitue ainsi une seconde approche de la programmation et des problématiques liées à l'automatisation des processus de gestion. En effet, au travers d'un précédent cours, il était question d'utiliser et de programmer sous Excel. Ainsi, nous avons tout à la fois pu constater que :

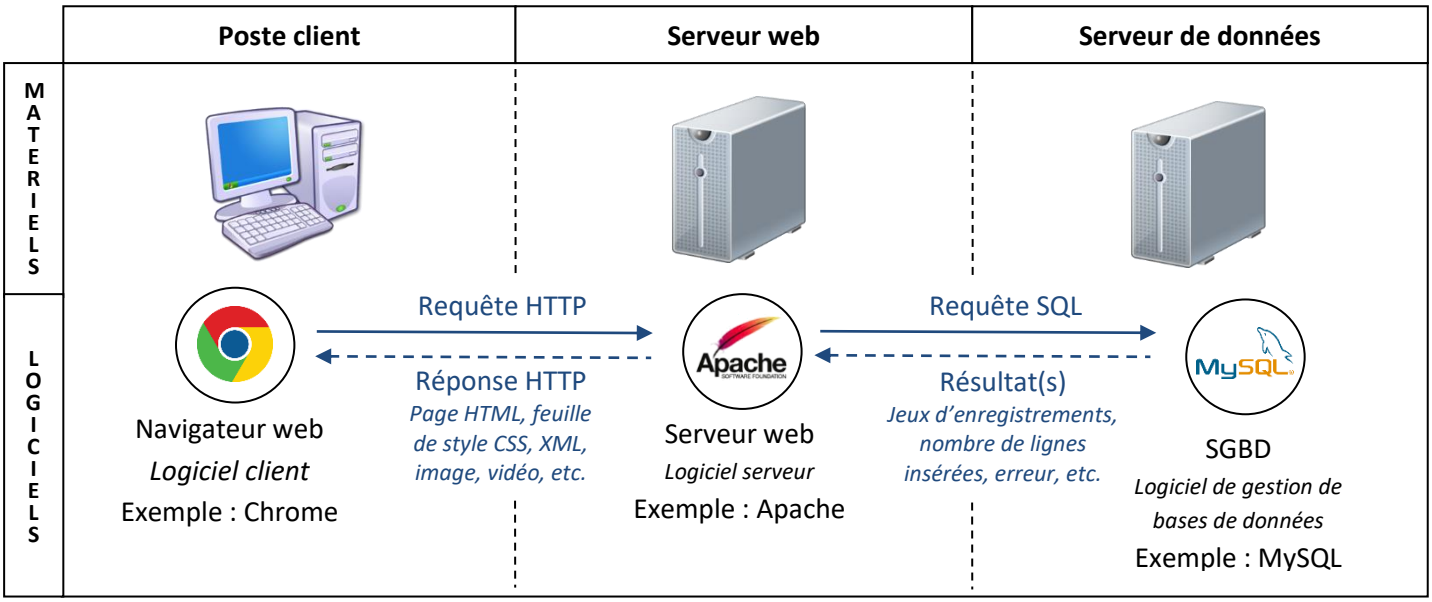
- Concevoir et programmer un outil Excel permet d'automatiser un processus et ainsi : de gagner du temps et de l'argent, de travailler plus efficacement ou encore de standardiser la façon de travailler et de fournir un travail de meilleure qualité ;
- Utiliser un outil Excel permet difficilement de partager de l'information entre plusieurs utilisateurs ce qui est problématique dès lors qu'il est nécessaire d'effectuer un travail collaboratif.

Aussi, afin de s'affranchir des limites d'Excel, notre attention se porte désormais sur la programmation web. De fait, l'on constatera, mais vous le constatez déjà tous les jours, que le web permet de partager de l'information en réseau et ainsi de faire interagir entre eux de multiples acteurs ce qui a en outre vocation à faciliter le travail collaboratif.

1.2. Architecture web

Concevoir une application web, c'est créer un logiciel basé sur les technologies du web : le protocole HTTP, les langages HTML **et** CSS, le langage de programmation PHP, etc. Un site internet, un site e-commerce, un réseau social tel que Facebook ou LinkedIn, un intranet ou encore un extranet, etc. Tous ces logiciels constituent des applications web ayant recours à un fonctionnement particulier.

Ils utilisent tous en effet une architecture dite architecture client-serveur dont on résume le fonctionnement au travers du schéma fourni ci-après.



Expliquons plus en détail le fonctionnement d'une application web :

1 Tout commence par le navigateur ! Lorsqu'on clique sur un lien pour accéder à un site internet par exemple, lorsqu'on saisit une URL (exemple : https://www.facebook.com) dans la barre d'adresse ou lorsqu'on valide un formulaire en ligne, une requête HTTP est envoyée. C'est l'URL qui permet de connaître le destinataire de la requête. En effet, l'URL contient le nom de domaine (exemple : facebook.com). Et un nom de domaine correspond à un serveur. Ce sont les serveurs DNS qui permettront, à partir du nom de domaine, de connaître l'adresse IP du serveur web destinataire. Créée par le navigateur, la requête HTTP est alors acheminée jusqu'au serveur web. Voilà une chose de faite…

2 Maintenant que le serveur web a reçu la requête HTTP, il va la traiter. Par exemple, si on lui demande une image, il va typiquement retourner une réponse HTTP contenant l'image en question. Si on demande une page web, par exemple une page écrite en PHP, les choses vont s'avérer un peu plus complexe. Le PHP, c'est du code, un programme. Et le programmeur se charge de dire au serveur web ce qu'il doit faire. Si un formulaire a été soumis, à savoir validé et transmis, le programme aura entre autres accès aux saisies de l'utilisateur.

Le cas échéant, le programme pourra se connecter à un serveur de base de données (SGBD) puis à une base de données. Il pourra ensuite émettre des requêtes SQL et utiliser les résultats retournés par le SGBD.

Par ailleurs, c'est l'interpréteur PHP qui va se charger d'interpréter le code PHP du programmeur, c'est-à-dire qu'il va exécuter les instructions que le développeur à écrites. En d'autres termes, l'interpréteur va faire ce que le programmeur lui a dit. Le programme va finalement produire un résultat, celui que le développeur a souhaité produire.

Finalement, le serveur web construit une réponse HTTP contenant le résultat et cette réponse est retournée au poste client, plus exactement au navigateur.

3 Pour finir le travail, le navigateur va interpréter le résultat qu'il a reçu de la part du serveur web. Il va par exemple afficher l'image qu'il a reçue ou encore la page web qu'il a reçue. C'est par exemple le navigateur qui va se charger d'appliquer à du HTML les mises en forme que le programmeur a décrites au moyen de CSS.

2. HTML et CSS

2.1. Introduction

Au travers de cette partie, l'on tâche de présenter deux éléments constitutifs d'une pages web : le code HTML d'une part, lequel permet de décrire la structure d'une page web, et le code **CSS** d'autre part, lequel permet de mettre en forme le code HTML.
Cependant, ce cours n'est pas un cours de HTML et de CSS mais plutôt un cours de programmation et de PHP. Aussi, il ne s'agit ici que d'apporter les notions essentielles à la compréhension de ces deux langages. Libre à vous d'aller plus loin.

2.2. Structure : le HTML

HTML	Le HTML est un langage permettant de décrire le contenu d'une page web et les ressources dont elle a besoin pour s'afficher correctement dans un navigateur. Les contenus et ressources : textes, images, feuilles de styles CSS, vidéo, son, etc.
Balise	Le HTML est un langage à balises, à savoir que les zones d'une page HTML sont délimitées par des éléments appelés marqueurs ou encore balises : • Une balise ouvrante (exemple :) marque le début d'une zone ; • Une balise fermante (exemple :) marque la fin d'une zone ; • Si on a ouvert une balise, l'on doit la fermer ; • Une balise peut être à la fois ouvrante et fermante (exemple :) ; • Chaque balise a une signification et/ou un comportement qui lui est propre.
Attribut	Les balises HTML peuvent avoir des attributs. Ces derniers permettent en quelque sorte d'apporter une précision supplémentaire. Un attribut a un nom et une valeur. Les valeurs que peuvent prendre un attribut sont parfois prédéfinies, auquel cas il convient d'utiliser l'une des valeurs prédéfinies. Exemple : <input name="unChamp" type="text" /> Commentaire : La balise input permet de définir un champ de formulaire. On a utilisé ses attributs name et type qui ont respectivement pour valeur unChamp et text.

Une page HTML « bien formée » contient *a minima* la code figurant ci-dessous.

Structure minimale d'une page web

```
<!DOCTYPE html>
<html>
    <!-- en-têtes de la page HTML -->
    <head>
        <title>Titre de ma page web</title>
    </head>
    <!-- contenus de la page HTML -->
    <body>
        Le contenu de ma page web.
    </body>
</html>
```

DOCTYPE : permet de préciser au navigateur la version HTML utilisée, en l'occurrence HTML5.
HTML : permet de préciser le début et la fin de la page ;
HEAD : marque le début et la fin des en-têtes de la page. Les en-têtes permettent de préciser le titre de la page, de définir des métadonnées ou encore d'importer des ressources telles que feuilles de styles CSS ou script JavaScript ;
BODY : marque le début et la fin du corps de la page web, à savoir du contenu visible de la page web : textes, images, vidéos, etc.

Exemple de code HTML : dans le contenu d'une page web, on peut par exemple définir un formulaire.

```
<form>
    <label>Nom :</label> <input name="nom" type="text" placeholder="[Votre nom]" />
    <br/>
    <label>Né le :</label> <input name="anniv" type="date" />
    <br/>
    <label>Newsletter ?</label> <input name="news" type="checkbox" />
    <br/>
    <label>Statut :</label>
    <input name="status" type="radio" value="étudiant" /> Etudiant
    <input name="status" type="radio" value="professeur" /> Professeur
</form>
```

Au exemple de code HTML : on peut encore placer un tableau, constitué de lignes.

```
<table>
    <thead><tr><th>Nom</th><th>Né le</th><th>News</th><th>Statut</th></tr></thead>
    <tbody>
        <tr><td>Gregorio</td><td>11/11/1982</td><td>Oui</td><td>Professeur</td></tr>
        <tr><td>Doroté</td><td>29/02/1978</td><td>Oui</td><td>Professeur</td></tr>
        <tr><td>Camille</td><td>08/05/2000</td><td>Oui</td><td>Eleve</td></tr>
        <tr><td>Gertrude</td><td>08/05/2001</td><td>Oui</td><td>Eleve</td></tr>
    </tbody>
</table>
```

2.3. Mise en forme : le CSS

CSS et Feuille de style	Le **CSS** est un langage permettant de décrire la mise en forme de pages web. Il utilise la notion de sélecteur. Un sélecteur permet de préciser le contenu de la page web auquel un ensemble de règles de mise en forme vont s'appliquer. Un fichier contenant du CSS est appelé une feuille de styles.
Importer une Feuille de style	Il existe 3 manières d'intégrer de la mise en forme CSS à une page web. Nous ne retiendrons qu'une méthode, la plus « propre des 3 », celle recommandée. Le marqueur link est une balise que l'on place typiquement entre les balises <head> et </head>. Ce marqueur permet entre autres d'importer une feuille de style CSS à partir de son chemin d'accès (URL, URI, etc). On procède comme suit : <link rel="stylesheet" href="/chemin/vers/fichier.css" />. Les règles de mises en forme définies dans la feuille de styles seront dès lors appliquées au contenu de la page web.

Quelques exemples de mises en forme vaudront mieux que de longs discours. Donnons-nous deux fichiers, côte-à-côte au sein d'un même répertoire : « une_page.html » et « une_page.css ».

On part du fichier « une_page.html » suivant. Initialement, aucune règle de mise en forme n'est définie dans le fichier « une_page.css ».

Code HTML de « une_page.html »	Aperçu de « une_page.html » (Navigateur)

```html
<!DOCTYPE html>
<html>
 <head>
      <link rel="stylesheet" href="./une_page.css" />
 </head>
 <body class="laClasse">
      <div id="unId" class="uneClasse">
           <div class="texte1">Vive le web !</div>
           <div class="texte2">C'est trop bien...</div>
      </div>
 </body>
</html>
```

Vive le web !
C'est trop bien...

A présent, on décide d'ajoute un brin de code CSS au fichier une_page.css et l'on obtient un résultat visuellement plus intéressant.

Code CSS de « une_page.css »	Aperçu de « une_page.php » (Navigateur)

```css
html{
    height: 100%;
}
body{
    font-family: Arial;
    background: url(http://www.babelio.com/users/QUIZ_Le-quiz-special-geek-_7288.jpeg) no-repeat center center / contain;
}
.uneClasse{
    margin: auto; width: 200px; border: solid 3px #000000;
    padding: 32px; border-radius: 50%;
}
```

Finalement, si l'on ajoute encore un peu de code CSS, l'on obtient un résultat plutôt sympathique.

Code CSS de « une_page.css »	Aperçu de « une_page.php » (Navigateur)

```css
.texte1, .texte2{
    text-align: center;
}
.texte1{
    font-weight: bold;
    margin-bottom: 6px;
}
.uneClasse:hover{
    cursor: pointer;
    background: #ffcc00;
}
```

3. Programmation PHP

3.1. Utiliser le langage PHP

PHP	Contrairement au HTML et au CSS, le PHP est bel et bien un langage de programmation. Il permet d'utiliser les notions que nous avons déjà étudiées en VBA : les conditions (if), les boucles (while, for, foreach), les variables, les tableaux, les fonctions et procédures, etc.
Ajout du PHP	Pour créer une page web PHP, il faut tout d'abord créer un fichier « .php » au sein duquel vous pouvez librement insérer du code HTML. Il convient, pour ajouter du PHP dans ce fameux fichier, d'utiliser les marqueurs <?php et ?> qui annoncent respectivement le début et la fin d'un « bout de code » PHP. Et c'est entre ces marqueurs que vous pouvez placer votre code PHP.

Ecrire un contenu en PHP	Outre le fait que vous puissiez directement écrire du contenu en HTML, PHP permet d'écrire dans la page (du texte, des nombres, le contenu d'une variable, etc.) :
	`<?php`
	`  // déclaration d'une variable contenant une chaîne de caractères`
	`  $uneVariable = "Un texte dans une variable" ;`
	`  echo "Ma variable contient : " . $uneVariable ;`
	`?>`
	N.B. : le caractère « . » permet de concaténer des chaînes de caractères.

Inclure un autre contenu PHP	Pour diverses raisons, vous pouvez avoir besoin d'inclure dans une page PHP le contenu d'un autre fichier PHP :
	`<?php`
	`  // inclusion du fichier « fichier_inclus.php » qui se trouve dans le même dossier`
	`  include __DIR__ . "/fichier_inclus.php" ;`
	`?>`
	N.B. : __DIR__ permet de récupérer le chemin vers le dossier courant.

Les variables	Comme dans tout langage de programmation, PHP permet d'utiliser des variables, c'est-à-dire de créer des petits espaces mémoire ayant un nom, d'y stocker des valeurs et de faire des calculs avec celles-ci :
	`<?php`
	`  // Déclaration de variables (en PHP, préfixées par un $)`
	`  $a = 5.50 ;`
	`  $b = 6 ;`
	`  $c = $a * $b ;`
	`  echo $a . " x " .  $b . " = " . $c ;`
	`?>`

Les tableaux	Les tableaux sont des variables un peu particulières, lesquelles permettent de stocker plusieurs valeurs :
	`<?php`
	`  // Déclaration d'un tableau contenant 4 éléments (4 valeurs)`
	`  $unTableau = array(18, "2ème valeur", "valeur 3", 3.141592) ;`

```php
// Affichage des 1er et 4ème éléments du tableau
echo $unTableau[0]  . "<br/>"; // « [0] » permet d'utiliser le 0+1 = 1ère élément
echo $unTableau[3]  . "<br/>"; // « [3] » permet d'utiliser le 3+1 = 4ème élément
// Modification puis affichage du 4ème élément du tableau
$unTableau[3] = 2.71828 ;
echo $unTableau[3] . "<br/>";
// Ajout d'un 5ème élément à la fin du tableau, puis affichage
$unTableau[] = 3.141592 ; // « [] » permet d'ajouter un élément à la fin du tab.
echo $unTableau[4] ;
// Récupèration du nombre d'éléments du tableau, puis affichage
$unNombre = count($unTableau) ;
echo "Le tableau ''unTableau'' compte " . $unNombre . " élément(s). " ;
?>
```

Remarque : l'un des avantages (et inconvénients) de PHP, c'est qu'il n'est pas nécessaire de préciser le type de la variable. PHP est faiblement typé. Et une variable peut accepter n'importe quel type de valeur : nombre, chaîne, tableau, etc. En PHP, c'est le contenu de la variable qui en détermine le type.

3.2. Interagir avec les utilisateurs en PHP

Les paramètres HTTP en PHP

On crée souvent des pages web afin que les internautes puissent interagir avec le programme (exemple : formulaires). Pour cela entre autres, le protocole HTTP introduit la notion de paramètres. Cela signifie qu'il est possible pour le client (le navigateur) de transmettre des données au serveur (le serveur web) et ainsi à notre programme PHP. Parmi ces données pouvant être transmises, nous retiendrons en outre les paramètres GET et POST.

Les paramètres GET

Les paramètres GET sont transmis directement dans l'URL, exemple :

- https://localhost/tests/thirdpage.php?unMessage=Hello World !&unNom=José
- Ici, les paramètres sont « unMessage » et « unNom ». Ils sont séparés par un caractère « & » (prononcé « et commercial ») ;
- Ils ont respectivement pour valeur « Hello World ! » et « José ».

PHP permet de récupérer ces paramètres grâce à la variable globale $_GET qui est ce qu'on appelle un tableau associatif car il permet de récupérer une donnée à partir d'un nom, appelé clef :

```php
<?php echo "Mon paramètre ''unMessage'' vaut : " . $_GET["unMessage"] ; ?>
```

Les paramètres POST

Les paramètres POST ne sont pas transmis via l'URL. Ils sont particulièrement utilisés afin de récupérer le contenu des formulaires. On peut récupérer ces paramètres cette fois-ci grâce à la variable globale $_POST.

```html
<form action="/tests/fourthpage.php" method="POST">
  <!-- Affichage de deux champs -->
  Adresse mail : <input name="email" type="text" /> <br/>
  Mot de passe : <input name="mdp" type="password" /> <br/>
  <!-- Bouton pour soumettre le formulaire -->
  <input type="submit" value="Se connecter" />
</form>
```

```
<br/>
<?php
  // Teste si les paramètres ''email'' et ''mdp'' existent
  echo "Paramètres POST reçu : <br/>" ;
  if(isset($_POST["email"], $_POST["mdp"])){
    // Affiche des deux paramètres transmis par l'utilisateur
    echo "- valeur du paramètre ''email'' : " . $_POST["email"] . "<br/>" ;
    echo "- valeur du paramètre ''mdp'' : " . $_POST["mdp"] ;
  }else{
    echo "encore rien..." ;
  }
?>
```

Les paramètres de session

Si les paramètres GET et POST n'existent que le temps d'une requête, les paramètres de session peuvent être utilisés toute la durée de vie d'une session. Ce qu'il faut retenir, c'est que :

- Ces paramètres peuvent être utilisés grâce à la variable globale $_SESSION et supprimés grâce à la fonction unset (exemple : unset($_SESSION["email"])) ;
- Ils permettent de conserver des données propres à l'utilisateur pendant plusieurs requêtes, en principe au moins tant que l'utilisateur ne ferme pas le navigateur.

Exemple :

```
<?php
  // A placer avant tout code HTML pour pouvoir utiliser les sessions.
  session_start() ;
  // Mise en session d'un paramètre ''email'' puis affichage
  $_SESSION["email"] = "quelquechose@gmail.com" ;
  echo $_SESSION["email"] ;
?>
```

3.3. Se connecter à une base de données et la manipuler en PHP

Se connecter à une base en PHP

Afin de pouvoir exécuter une requête SQL (SELECT, INSERT, UPDATE, DELETE, etc.), il faut d'abord se connecter au serveur de bases de données. Pour ce faire, PHP met à votre disposition la fonction suivante :

- mysqli_connect($host, $user, $password)
- *Paramètres :* $host
  - $host : l'adresse (URL) permettant d'accéder au serveur de base de données ;
  - $user : le nom de l'utilisateur ;
  - $password : le mot de passe de l'utilisateur.
- *Valeur de retour :* une connexion à la base de données.

Exemple :

```
<?php
  $connexion = mysqli_connect("localhost", "root", "") ;
?>
```

Sélectionner une base en PHP

Une fois la connexion au SGBD récupérée, pour manipuler une base de données, il faut encore sélectionner la bonne base :

```php
<?php
  mysqli_select_db($connexion, "db_gest_wissen") ;
?>
```

Exécuter une requête en PHP

Nous pouvons désormais exécuter des requêtes ainsi qu'en récupérer le résultat. Pour ce faire, PHP met en outre à votre disposition la fonction suivante :

- mysqli_query($connection , $query)
- *Paramètres :* $host
  - $query : la requête SQL sous forme de chaîne de caractères ;
  - $connection : la connexion récupérée à l'aide de la fonction mysqli_connect(…).
- *Valeur de retour :*
  - s'il s'agit d'une requête SELECT, la fonction retourne le résultat de la requête ou false si la requête échoue du fait d'une erreur. Il peut être exploité entre autres grâce aux fonction mysqli_fetch_assoc($result) et mysqli_num_rows($result) ;
  - s'il s'agit d'une requête INSERT, UPDATE ou encore DELETE, la fonction retourne true ou false selon si la requête réussit ou échoue. On peut ensuite utiliser la fonction mysqli_affected_rows() pour connaître le nombre de lignes affectées par la requête mais aussi mysqli_insert_id() pour connaître le dernier identifiant auto-incrémenté en cas de requête INSERT.

Exemple de requête en PHP

```php
<?php
  // On récupère une connexion au SGBD
  $connexion = mysqli_connect("localhost", "root", "") ;
  // On se place sur la base de données
  mysqli_select_db($connexion, "db_gest_wissen") ;
  // On insère un utilisateur
  $success = mysqli_query($connexion, "
    INSERT INTO Utilisateur (email, mot_de_passe)
    VALUES('matt.damon@outlook.fr', 'georgette')
  ") ;
  // Si l'utilisateur a bien été enregistré on affiche un petit message de succès
  if($success){
    echo "L'utilisateur matt.damon@outlook.fr a bien été enregistré <br/>" ;
  }
  // On récupère l'utilisateur au moyen d'une requête
  $lignes = mysqli_query($connexion, "
    SELECT *
    FROM Utilisateur
    WHERE email = 'matt.damon@outlook.fr'
  ") ;
  // Si la requête a échoué
  if(!$lignes){
    echo "La requête SQL est incorrecte. <br/>" ;
  }
  // On récupère le nombre de lignes trouvées
  $nb = mysqli_num_rows($lignes) ;
```

```
  // Si l'on a aucune ligne, c'est que l'utilisateur n'existe pas
  if($nb == 0){
    echo "L'utilisateur n'existe pas ! <br/>" ;
  }
  // On récupère la première ligne
  $ligne = mysqli_fetch_assoc($lignes) ;
  // On affiche les informations de la ligne
  echo "Utilisateur : " . $ligne["email"]  . "<br/>" ;
  echo "Mot de passe : " . $ligne["mot_de_passe"] . "<br/>" ;
?>
```

3.4. Afficher des données en provenance d'une base de données

Structure alternative

Une structure alternative ou structure conditionnelle permet d'exécuter une série d'instructions (=opérations) sous réserve qu'une condition soit vérifiée.

○ Exemple :

```
<?php
  // on récupère l'heure actuelle grâce à la function "date"
  $date = date("H:i:s");
  if($date < "12:00:00"){
    echo "bonjour !" ;
  }else if($date < "17:00:00"){
    echo "bon après-midi !";
  }else{
    echo "bonsoir ! ";
  }
?>
```

○ Principaux opérateurs de comparaison en PHP :
- $A == $B : opérateur « égal à », soit ici A égal à B ;
- $A != $B : opérateur « différent de », soit ici A différent de B ;
- !$A : opérateur « non », soit ici non A ;
- $A && $B : opérateur « et », soit ici A et B ;
- $A || $B : opérateur « ou », soit ici A ou B.

Fonction

Une fonction est un fragment de code réutilisable. Une fonction peut retourner un résultat. Lorsque la fonction ne retourne pas de résultat, on parle de procédure. En PHP, on déclare une fonction grâce au mot-clef function et l'on utilise le mot-clef return pour retourner un résultat. Par ailleurs, il possible de passer des informations à une fonction en utilisant ce qu'on appelle des paramètres.

Exemple n°1 : fonction permettant de se connecter à la base « db_gest_wissen ».

```
<?php
  function se_connecter(){
    // récupération d'une connexion au SGBD
    $connexion = mysqli_connect("localhost", "root", "") ;
    // sélection de la base de données « db_gest_wissen »
```

```
    mysqli_select_db($connexion, "db_gest_wissen") ;
    // retourne la connexion à la base de données « db_gest_wissen »
    return $connexion ;
  }
?>
```

Exemple n°2 : fonction retournant un utilisateur à partir de son adresse email ou faux si l'utilisateur n'existe pas.

```
<?php
  function recup_utilisateur($email){
    // Connection à la base « db_gest_wissen » grâce à la fonction précédente
    $connexion = se_connecter() ;
    // Récupération de l'utilisateur dont l'email est fourni en paramètre
    $lignes = mysqli_query($connexion, "
      SELECT *
      FROM Utilisateur
      WHERE email = 'matt.damon@outlook.fr'
    ") ;
    // Récupération du nombre de lignes trouvées
    $nb = mysqli_num_rows($lignes) ;
    // Si l'on a aucune ligne, c'est que l'utilisateur n'existe pas, on retourne faux
    if($nb == 0){
      return false ;
    }
    // Récupération de la première ligne, c'est-à-dire de l'utilisateur
    $ligne = mysqli_fetch_assoc($lignes) ;
    // On retourne l'utilisateur
    return $ligne ;
  }
?>
```

Exemple n°3 : utilisation de la fonction recup_utilisateur.

```
<?php
  // Récupération de l'utilisateur dont l'adresse email est matt.damon@outlook.fr
  $utilisateur = recup_utilisateur("matt.damon@outlook.fr") ;
  // Affichage du prénom et du nom de l'utilisateur
  echo "Bonjour " . $utilisateur["prenom"] . " " . $utilisateur["nom"] ;
?>
```

Structures itératives

Une structure itérative permet d'itérer, c'est-à-dire de répéter une série d'instructions (=opérations) plusieurs fois. On parle plus simplement de boucle.

N.B. : afin de bien comprendre les exemples suivants, il convient préalablement de bien comprendre ce que fait la fonction mysqli_fetch_assoc. Cette fonction retourne la ligne courant et passe à la ligne suivante. Cette fonction retourne faux (false) quand il n'y a plus de lignes.

Exemple de boucle « POUR » : une boucle « POUR » est idéale pour répéter un traitement un nombre de fois connu à l'avance. Dans l'exemple qui suit, on affiche la liste des utilisateurs.

```
<?php
```

```php
    // Connection à la base « db_gest_wissen »
    $connexion = se_connecter() ;
    // Récupération de tous les utilisateurs
    $utilisateurs = mysqli_query($connexion, "SELECT * FROM Utilisateur") ;
    // Récupération du nombre d'utilisateurs trouvés
    $nb = mysqli_num_rows($utilisateurs) ;
    // Parcours des utilisateurs
    for($i=0 ; $i<$nb ; $i++){
      // Récupération de la ligne courante et passage à la ligne suivante
      $utilisateur = mysqli_fetch_assoc($utilisateurs) ;
      // Affichage du prénom et du nom de l'utilisateur puis saut de ligne
      echo $utilisateur["prenom"] . " " . $utilisateur["nom"] . "<br/>" ;
    }
?>
```

Exemple de boucle « TANT QUE » : une boucle « TANT QUE » est idéale pour répéter un traitement un nombre de fois inconnu à l'avance. Dans l'exemple qui suit, on affiche la liste des utilisateurs mais avec une boucle « TANT QUE » cette fois-ci.

```php
<?php
  // Connection à la base « db_gest_wissen »
  $connexion = se_connecter() ;
  // Récupération de tous les utilisateurs
  $utilisateurs = mysqli_query($connexion, "SELECT * FROM Utilisateur") ;
  // Récupération du premier utilisateur
  $utilisateur = mysqli_fetch_assoc($utilisateurs) ;
  // Parcours des utilisateurs tant qu'il y en a
  while($utilisateur){
    // Affichage du prénom et du nom de l'utilisateur puis saut de ligne
    echo $utilisateur["prenom"] . " " . $utilisateur["nom"] . "<br/>" ;
    // Récupération de l'utilisateur suivant
    $utilisateur = mysqli_fetch_assoc($utilisateurs) ;
  }
?>
```