

**Secteur Tertiaire Informatique**  
**Filière « Etude et développement »**

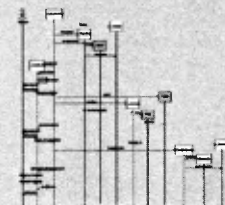
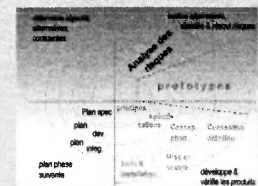
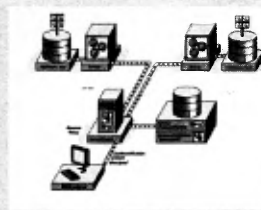
**Séquence « Développer des pages Web en lien  
avec une base de données »**

**Initiation au langage PHP**

**Apprentissage**

**Mise en pratique**

**Evaluation**



Version	Date	Auteur(s)	Action(s)
1.0	xx/03/16	B. Hézard	Création du document

## TABLE DES MATIERES

Table des matières .....	3
1. Le PHP, un langage pourquoi faire ? .....	7
2. Principes de base du langage PHP .....	7
2.1 Le PHP, un langage interprété côté serveur Web .....	7
2.2 La forme d'un script PHP .....	7
2.3 Le PHP, un langage complet et évolutif .....	8
3. Les variables en PHP .....	9
3.1 Généralités .....	9
3.2 Manipulation des variables de type string .....	9
3.3 Les tableaux .....	12
3.4 Les variables superglobales .....	15
4. Les opérateurs en PHP .....	16
4.1 Assignment .....	16
4.2 Concaténation .....	16
4.3 Opérateurs de comparaison .....	17
4.4 Opérateurs arithmétiques et d'incrémentation .....	17
4.5 Opérateurs logiques .....	17
5. Générer le code HTML avec PHP .....	17
5.1 L'instruction echo et le formatage des messages .....	17
5.2 Bufferisation .....	19
5.3 Redirections avec header() .....	20
6. Les structures de contrôle .....	21
6.1 Tests .....	21
6.2 Boucles .....	22
7. La mise au point des scripts .....	23
7.1 Fonctions de trace .....	23
7.2 La gestion des erreurs .....	24
8. Les procédures et fonctions en PHP .....	25
9. Portée des variables .....	27

10.	Ecriture modulaire du code PHP .....	28
11.	Propager des données de page Web en page Web.....	29
12.	Gérer les sessions en PHP .....	33
12.1	Principe .....	33
12.2	Mise en œuvre en PHP .....	33
13.	Encore et toujours plus en PHP .....	34

## Objectifs

Ce document se veut une présentation et un aide-mémoire sur le langage PHP ; il complète sans la remplacer la documentation de référence.

## Pré requis

Pour pouvoir bénéficier des apports livrés dans ce document, il est nécessaire d'être familier avec les notions d'algorithmie et les bases de la programmation :

- Notion de variable et de type de donnée ;
- Notion de tableau de variables ;
- Structures de contrôle d'un programme (tests et boucles) ;
- Opérateurs ;
- Modularisation à l'aide de procédures et fonctions ;
- Commentaires et documentation du code ;

De plus, une bonne connaissance des langages de base du Web (HTML/CSS/JavaScript) est nécessaire.

## Outils de développement

Tout éditeur ou IDE dédié au développement Web (NotePad++, CodeLobster, Brackets...).

## Méthodologie

Ce document se veut un guide de découverte du langage PHP et un aide-mémoire sur le langage.

Il ne remplace pas la documentation de référence de PHP en ligne ; il la complète en précisant les rôles et utilités des principales instructions et fonctions du langage et en organisant la découverte par thèmes du langage PHP.

De nombreux liens renvoient vers la documentation en ligne en français très complète et bien maintenue à jour.

## Mode d'emploi

Symboles utilisés :



Renvoie à des supports de cours, des livres ou à la documentation en ligne constructeur.



Propose des exercices ou des mises en situation pratiques.



Point important qui mérite d'être souligné !


## **Ressources**

Le site officiel sur le langage PHP : <http://php.net>

Et plus particulièrement la partie consacrée au manuel de référence : <http://php.net/manual/fr>

## **Lectures conseillées**

# 1. LE PHP, UN LANGAGE POURQUOI FAIRE ?


 PHP est un langage de programmation récent destiné à l'origine spécifiquement au développement de pages Web dynamiques, ces fameuses pages Web dont le contenu varie en fonction du contexte, comme les différentes pages d'un catalogue ou encore les pages d'une messagerie électronique.

En ce sens, PHP a la même finalité que les technologies JSP, ASP.Net ou Ruby On Rails par exemple : il s'agit de générer du code personnalisé à envoyer au navigateur client de l'utilisateur. Autant les technologies sont standardisées côté client Web avec HTML, CSS et JavaScript, autant, pour les pages Web dynamiques, les technologies sont variables en fonction des logiciels utilisés côté serveur Web.

Le langage PHP a beaucoup évolué depuis sa création en 1994 et ses fonctionnalités vont aujourd'hui au-delà de sa vocation originelle ; c'est aujourd'hui un langage complet utilisé principalement pour générer des pages Web dynamiques, mais pas seulement car il permet aussi de traiter des images, d'envoyer des emails, de traiter des fichiers...

## 2. PRINCIPES DE BASE DU LANGAGE PHP

### 2.1 LE PHP, UN LANGAGE INTERPRETE COTE SERVEUR WEB

 Le langage PHP est de type 'interprété' ; il n'y a donc pas d'étape de compilation pour le développeur mais toute erreur de syntaxe ne se révélera que lors de l'exécution ; il est donc très important de bien tester et mettre au point les scripts PHP avant leur publication en ligne !

Au contraire du langage HTML par exemple, lui aussi de type interprété, le code PHP n'est jamais envoyé au navigateur client : il est exécuté par le serveur Web de manière à générer ou personnaliser la page HTML et c'est bien le résultat de cette exécution, soit du code HTML/CSS/JavaScript pur qui est adressé au browser.

Comme le but est de générer ou personnaliser du code HTML, un script PHP contiendra un mélange (parfois très imbriqué) d'instructions HTML/CSS/JavaScript et PHP. Il est donc nécessaire de bien organiser et structurer le code ; des instructions PHP et quelques bonnes pratiques peuvent y aider.

### 2.2 LA FORME D'UN SCRIPT PHP

A la base, un script PHP est un simple fichier de texte identifié par son extension (.php, en général). Dès qu'un client Web demande ce type de document au serveur Web, ce dernier lit le contenu, transmet sans modification le code HTML/CSS/JavaScript mais exécute les instructions PHP et adresse au client le résultat de ce traitement.

 Voici déjà quelques règles de base :

- Une instruction PHP est reconnue par la balise spéciale `< ?php` `??>` qui peut occuper plusieurs lignes.
- Toute instruction PHP doit se terminer par un point-virgule.
- Il est important de différencier MAJUSCULES et minuscules (`$MaVariable` est différent de `$maVariable`).

- Les commentaires spécifiques PHP s'écrivent à l'intérieur des balises `< ?php ?>` et sont délimités de manière classique par les caractères `//` sur une ligne ou les séquences `/* */` sur plusieurs lignes.

Ainsi le code ci-dessous est correctement formé (mais le code PHP ne sert strictement à rien !) :

```
<html>
  <head></head>
  <body>
    <h1>Coucou</h1>
    <?php // du php pour rien ! ?>
  </body>
</html>
```

### 2.3 LE PHP, UN LANGAGE COMPLET ET EVOLUTIF

Le langage PHP fait partie du mouvement 'Open Source' ; il est donc gratuit et de nombreux développeurs de la planète travaillent à son amélioration. Il en résulte une profusion d'instructions, plus précisément de fonctions intégrées et d'extensions, et de nombreux synonymes, ce qui peut décontenancer le développeur. Les versions du langage se sont enchaînées à grande vitesse pendant 10 ans pour se stabiliser avec la version 5 en 2004 qui a intégré l'orientation Objet. La version 6 est morte dans l'œuf et la version 7, simple toilettage de la V5, est disponible depuis début 2016. Au-delà du 'noyau de base', le développeur peut acquérir/activer de nombreuses extensions spécialisées selon les besoins.


Puisque PHP est aujourd'hui un vrai langage de programmation, il dispose de tout ce qui est nécessaire pour le développeur : variables, opérateurs, structures de contrôles, fonctions... Ce document a pour vocation de donner des points de repère et des pistes de recherches sans pour autant se substituer à l'excellent manuel de référence, bien maintenu à jour et traduit en français, disponible à tout moment sur : <http://php.net/manual/fr>





## 3. LES VARIABLES EN PHP

### 3.1 GENERALITES


 Toute variable utilisable en langage PHP doit être identifiée par un nom commençant par le signe \$ afin de les différencier du vocabulaire du langage.

Les variables PHP n'ont pas besoin d'être déclarées à l'avance et leur type dépend de leur contenu courant.


La déclaration initiale des variables n'est donc pas nécessaire et peut se faire au fil de l'eau. Le développeur n'a pas besoin de se préoccuper des problèmes de libération de mémoire car l'interpréteur détruit systématiquement toute variable utilisée en fin d'exécution de script. PHP permet tout de même de supprimer intentionnellement une variable à l'aide de la fonction intégrée `unset()`.

Les types de données disponibles en PHP sont :

- Chaînes de caractères : `string` ;
- Nombre entier : `int` ;
- Nombre décimal : `float` ou `double` ;
- Logique : `bool` qui accepte les valeurs `true` et `false` ;
- Valeur nulle : `NULL` ou `null` ;
- Au-delà, les types `array` et `object` permettent de gérer des tableaux de variables et des structures orientées objet. Enfin, le type `resource` mentionné parfois dans la documentation correspond à des ressources extérieures comme la référence vers une connexion à une base de données ou à un fichier ouvert sur disque.

 Pour tout savoir sur les types de données : <http://php.net/manual/fr/language.types.intro.php>

Pour gérer des données calendaires comme des dates ou des heures, PHP propose la fonction `getDate()` qui récupère sous forme de tableau de variables la date et heure courante sur le serveur, et une structure objet standard `DateTime` qu'il suffit d'instancier pour en utiliser les propriétés et méthodes ainsi que la fonction intégrée `getdate()` retournant date et heure du jour (du serveur) sous forme de 'tableau associatif'.

 Pour tout savoir sur les dates et heures : <http://php.net/manual/fr/class.datetime.php>  
et <http://php.net/manual/fr/function.getdate.php>

### 3.2 MANIPULATION DES VARIABLES DE TYPE STRING

Toute application de gestion doit manipuler des textes pour restituer des informations à l'utilisateur ; ceci est encore plus vrai pour les applications Web en PHP puisqu'il s'agit de générer des pages Web d'informations constituées essentiellement de textes.

Le nombre de fonctions intégrées disponibles pour manipuler les chaînes de caractères en PHP peut donner le vertige ! Il n'est pas question ici de toutes les citer ou expliciter mais de repérer les plus usuelles et de savoir où rechercher en cas de besoin.



Pour manipuler les textes, une seule adresse : <http://php.net/manual/fr/ref.strings.php>



Parcourez cette documentation de référence et complétez le tableau ci-dessous en définissant succinctement le rôle de ces principales fonctions de manipulation de texte.

<i>Fonction PHP</i>	<i>Rôle à compléter</i>
<code>trim()</code>	
<code>ltrim()</code>	
<code>rtrim()</code>	
<code>strlen()</code>	
<code>strtoupper()</code>	
<code>ucfirst()</code>	
<code>substr()</code>	
<code>strpos()</code>	
<code>sprintf()</code>	
<code>explode()</code>	

Eléments de solution :

<i>Fonction PHP</i>	<i>Rôle</i>
<code>trim()</code>	Supprime les espaces inutiles en début et fin de chaîne
<code>ltrim()</code>	Supprime les espaces inutiles en début de chaîne
<code>rtrim()</code>	Supprime les espaces inutiles en fin de chaîne
<code>strlen()</code>	Retourne le nombre de caractères d'une chaîne
<code>strtoupper()</code>	Transforme une chaîne de caractères en MAJUSCULES
<code>ucfirst()</code>	Transforme uniquement le 1 <sup>er</sup> caractère d'une chaîne en MAJUSCULE
<code>substr()</code>	Extrait une partie d'une chaîne de caractères
<code>strpos()</code>	Détermine si une chaîne est incluse dans une autre et retourne la position de sa première occurrence
<code>sprintf()</code>	Retourne une chaîne de caractères paramétrée formatée
<code>explode()</code>	Permet de découper une chaîne de caractères selon un caractère séparateur

Afin de coller aux besoins spécifiques du développement Web, PHP expose des fonctions particulières applicables aux chaînes de caractères ; en voici les principales :

<i>Fonction PHP</i>	<i>Rôle</i>
<code>htmlspecialchars()</code>	Convertit les caractères spéciaux comme les lettres accentuées ou les signes < et > en 'entités' HTML (&acute; ; &gt;...)
<code>htmlentities()</code>	Même principe mais étendu à plus de cas de figure (tester au cas par cas pour choisir entre ces deux fonctions)
<code>html_entity_decode()</code>	Convertit à l'inverse les entités HTML en caractères ordinaires
<code>urlencode()</code>	Convertit certains caractères de manière à respecter la syntaxe des URL (ex : espace devient %20)



Les fonctions `htmlspecialchars()` et `htmlentities()` sont essentielles pour protéger les applications contre certaines tentatives d'intrusion et autres actions malveillantes à travers la saisie de données dans les formulaires HTML. Une bonne règle à respecter : ne jamais faire confiance à ce qui est saisi par un utilisateur dans un formulaire, et filtrer les données reçues à travers les 'moulinettes' `htmlspecialchars()` ou `htmlentities()`.

### 3.3 LES TABLEAUX

Le langage PHP permet bien entendu de stocker des variables multiples dans des structures de type '*tableau de variables*' qui peuvent être à une ou plusieurs dimensions.

Les tableaux sont omniprésents dans la programmation en langage PHP : variables d'environnement, données saisies dans les formulaires, récupération d'informations depuis une base de données...

Les différents items d'un tableau peuvent être de type différent.

De plus, un tableau de variables PHP n'a pas de longueur prédéfinie et peut à loisir être agrandi ou réduit (tout comme pour les collections en C# ou Java).

On peut déclarer un nouveau tableau PHP grâce à l'instruction `array(...)` dans laquelle on définit éventuellement le contenu initial entre les parenthèses.

On accède à un item d'un tableau simple par son indice.

#### Exemples de tableaux simples :

```
// déclaration et initialisation de tableaux
$array1 = array("foo", "bar", "hello", "world");
$array2 = array("foo", "bar", 12, true);
// déclaration et initialisation d'un tableau
$tableau[0] = "1er valeur";
// extension progressive de la taille du tableau en connaissant les rangs
$tableau[1] = "2eme valeur";
$tableau[2] = "3eme valeur";
// extension de la taille du tableau en stockant à la suite
$tableau[] = "4eme valeur"; // sera stocké à l'indice 3
```

Les items de ces tableaux sont simplement indicés, le premier élément portant le numéro zéro. Pour accéder à un item, il suffit de connaître son rang (`$array1[1]` vaut "bar") ou de parcourir le tableau à l'aide d'une boucle `foreach` (voir plus loin).

Au-delà de ces tableaux de base, PHP permet au développeur de créer des '*tableaux associatifs*' dans lesquels on stocke deux données pour chaque item : la valeur et une clé textuelle donnant du sens à l'information et permettant de la retrouver plus facilement (principe des 'dictionnaires' C# ou Java). Le lien entre clé et valeur associée est exprimé par l'opérateur `=>`.

#### Exemples de tableaux associatifs :

```
$array = array("foo" => "bar", "bar" => "foo" ,100 =>-100, -100=> 100);
$facture["Janvier"] = 500;
$facture["Février"] = 620;
```

Dans ce dernier exemple, il est plus facile et explicite de récupérer la facture de janvier par l'instruction :

```
$a_payer = $facture["Janvier"];
plutôt que par l'instruction :
$a_payer = $facture[0];
```

Initiation au langage PHP

Comme pour les chaînes de caractères, PHP expose de très nombreuses fonctions concernant les tableaux de variables.



Pour manipuler les tableaux, une seule adresse : <http://php.net/manual/fr/language.types.array.php>



Parcourez cette documentation de référence et complétez le tableau ci-dessous en définissant succinctement le rôle de ces principales fonctions de manipulation de tableaux.

<i>Fonction PHP</i>	<i>Rôle à compléter</i>
<code>key_exists()</code>	
<code>in_array()</code>	
<code>is_array()</code>	
<code>array_search()</code>	
<code>array_keys()</code>	
<code>array_values()</code>	
<code>count()</code>	
<code>sort()</code>	
<code>ksort()</code>	
<code>asort()</code>	
<code>each()</code>	
<code>print_r()</code>	

Eléments de solution :

<i>Fonction PHP</i>	<i>Rôle à compléter</i>
<code>key_exists()</code>	Vérifie si une clé existe dans un tableau associatif
<code>in_array()</code>	Vérifie si une valeur existe dans un tableau
<code>array_search()</code>	Recherche la clé associée à une valeur dans un tableau associatif
<code>array_keys()</code>	Retourne un tableau contenant tous les indices d'un tableau simple ou toutes les clés d'un tableau associatif
<code>array_values()</code>	Retourne un tableau contenant toutes les valeurs d'un tableau
<code>is_array()</code>	Teste si la variable est de type tableau (indiqué ou associatif)
<code>count()</code>	Retourne le nombre de valeurs présentes dans un tableau
<code>sort()</code>	Effectue un tri des valeurs présentes dans un tableau
<code>ksort()</code>	Effectue un tri, selon l'ordre des clés, des valeurs présentes dans un tableau
<code>asort()</code>	Effectue un tri des valeurs présentes dans un tableau associatif en conservant les relations clé => valeur
<code>each()</code>	Retourne chaque paire clé/valeur d'un tableau
<code>print_r()</code>	Instruction idéale pour restituer en clair le contenu d'un tableau (à des fins de mise au point du code PHP)

En fait, pour les tableaux indicés, l'indice numérique est considéré comme la clé d'accès à la valeur correspondante ; PHP ne différencie pas réellement la nature des tableaux indicés et associatifs. Le développeur doit être au clair avec la structure du tableau qu'il manipule car certaines fonctions sont plus adaptées à un type ou l'autre des tableaux PHP.

### 3.4 LES VARIABLES SUPERGLOBALES

L'environnement d'exécution du langage PHP met à disposition du développeur des variables dite '*superglobales*' accessibles et utilisables à tout moment dans tout script PHP.

Ces variables, dont le nom est toujours en MAJUSCULES précédé du signe `_`, sont exposées sous forme de tableaux associatifs qui regorgent d'informations précieuses pour le développeur. En voici les principales.

Variable superglobale PHP	Rôle et contenu
<code>\$_SERVER</code>	Informations diverses sur le serveur, le client et le protocole de communication entre eux (nom du serveur, adresses IP...)
<code>\$_SESSION</code>	Stocke les variables de session qui permettent de 'suivre à la trace' un utilisateur de page en page (personne connecte, panier d'achat...)
<code>\$_COOKIE</code>	Stocke des variables qui permettent de suivre à la trace un utilisateur lors de ses différentes visites (enregistrement d'informations de connexion...); les données d'un cookie sont transmises au serveur automatiquement
<code>\$_GET</code>	Stocke tous les paramètres passés à un script en méthode GET HTTP, c'est-à-dire dans l'URL demandée au serveur Web (comme pour des formulaires utilisant l'attribut HTML <code>method='GET'</code> )
<code>\$_POST</code>	Stocke tous les paramètres passés à un script en méthode POST HTTP, c'est-à-dire dans le corps du message HTTP adressé au serveur Web (comme pour des formulaires utilisant l'attribut HTML <code>method='POST'</code> )
<code>\$_REQUEST</code>	Stocke tous les paramètres passés à un script en méthodes GET et POST HTTP
<code>\$_FILES</code>	Stocke les références des fichiers externes joints dans un message HTTP (comme les pièces jointes par les balises HTML <code>&lt;input type='file'&gt;</code> )



Pour tout savoir sur les variables *superglobales* PHP, une seule adresse : <http://php.net/manual/fr/language.variables.superglobals.php>



Et maintenant, comment s'en servir ?

On utilisera les variables de session pour tester qu'un utilisateur s'est bien connecté et qu'il a les droits suffisants pour exécuter sa demande.

On utilisera les variables de session et/ou les cookies pour constituer un panier d'achat.

On récupérera les données saisies dans les formulaires HTML à l'aide des variables `$_GET`, `$_POST` et `$_REQUEST` et les éventuels documents joints à l'aide de `$_FILES`.

Des chapitres sont consacrés à ces problématiques plus loin dans ce document.



## 4. LES OPERATEURS EN PHP

### 4.1 ASSIGNATION

Tout programme a besoin de stocker et modifier des données dans des variables.  
Pour ce faire, PHP propose l'opérateur classique d'affectation (ou assignation) `=`.  
Le type de donnée placé dans la variable détermine le type courant de la variable.

#### Exemples :


```
$message = "Coucou"; // variable type string
$compteur = 1; // variable type int
$compteur = '1' ; // la variable devient de type string
$compteur = 12.0; // la variable devient de type float
```

PHP offre aussi les opérateurs de progression classiques en C#/Java : `+=`, `-=`

```
$compteur += 1; // variable augmentée de 1
$apayer -= $remise; // variable diminuée de la valeur d'une autre
```

### 4.2 CONCATENATION


Pour manipuler les chaînes de caractères, le développeur a souvent besoin de reconstituer un message par concaténation.

 **Attention ! en PHP, l'opérateur de concaténation est le point (.) et non le plus (+) comme bien souvent !**

#### Exemple :

```
$message = "Bonjour" ;
$destinataire = 'tous ! ' ;
$message = $message . ' à ' . $destinataire ;
```

Pour indiquer une valeur de chaîne de caractères, on peut aussi bien utiliser les délimiteurs apostrophe (') que guillemets ("). Mais en PHP, le travail de l'interpréteur ne sera pas le même !

 **Toute chaîne de caractères délimitée par des apostrophes est traitée tel que, sans transformation.**

**Toute chaîne de caractères délimitée par des guillemets est scannée par l'interpréteur ; toute variable présente est remplacée par sa valeur et PHP effectue une concaténation implicite.**

Ainsi dans l'exemple précédent, la dernière instruction pourrait aussi bien s'écrire :

```
$message = "$message à $destinataire" ;
```

Cette syntaxe particulière au langage PHP peut paraître déroutante mais elle économise de nombreuses et périlleuses concaténations explicites.



### 4.3 OPERATEURS DE COMPARAISON

Pour comparer des variables entre elles ou avec un contenu particulier, PHP propose des opérateurs classiques comme `==` `>` `>=` `<` `<=` ou encore `!=`.

### 4.4 OPERATEURS ARITHMETIQUES ET D'INCREMENTATION

De même, on retrouve en PHP les opérateurs arithmétiques classiques comme `+` `-` `*` `/`.

Et pour incrémenter ou décrémenter une variable numérique, PHP supporte les opérateurs 'plus plus' (`++`) et 'moins moins' (`--`) comme en langage C# ou Java.

Ce mécanisme est étendu à la concaténation des chaînes de caractères avec l'opérateur point égal (`.`). Ainsi, l'exemple précédent peut encore s'écrire :

```
$message .= ' à ' . $destinataire ;
```

### 4.5 OPERATEURS LOGIQUES

Les opérateurs logiques permettent de conjuguer des conditions. PHP propose les classiques 'et' (`and` ou `&&`), 'ou' (`or` ou `||`) et 'non' (`!`).

Bref, rien de bien différent en PHP par rapport aux autres langages de programmation.



Pour les opérateurs en PHP, une seule adresse : <http://php.net/manual/fr/language.operators.php>

## 5. GENERER LE CODE HTML AVEC PHP

### 5.1 L'INSTRUCTION ECHO ET LE FORMATAGE DES MESSAGES

Le but du langage PHP est bien de générer du code HTML/CSS/JavaScript personnalisé à retourner au navigateur client qui a demandé la page Web dynamique.

PHP offre deux moyens de réaliser cette génération de code :

- Comme de nombreuses portions de code HTML/CSS/JavaScript ne varient pas dans une page Web, il suffit de placer ces portions de code en dehors des balises spécifiques `< ?php` `?>` pour que l'interpréteur les transmette en l'état au demandeur ;
- Pour le code HTML/CSS/JavaScript variable, PHP propose l'instruction `echo` pour insérer en lieu et place la valeur d'une variable ou d'une expression PHP.

### Exemple de codes aboutissant au même résultat :

```
<?php $Visites++;
```

```
$str= "<h3>Le serveur renvoie un cookie de valeur : <br />$Visites</h3>";  
echo $str; ?>.
```

Exemple de code 'tout PHP'  
(a priori, plus efficace)

```
<?php $Visites++; ?>
```

```
<h3>Le serveur renvoie un cookie de valeur : <br />
```

```
<?php echo $Visites; ?></h3>
```

Exemple d'alternance code PHP et HTML  
(code 'spaghetti')

La particularité du langage PHP appliqué aux pages dynamiques Web est qu'il s'agit bien de générer du code personnalisé HTML/CSS/JavaScript qui sera à son tour interprété par le navigateur de l'utilisateur. Le développeur doit bien définir le code souhaité et en particulier bien identifier toutes les balises HTML nécessaires (comme ici, les balises de mise en forme <h3> ou <br />). Une bonne pratique consiste à maquetter le résultat en écrivant tout d'abord un code HTML/CSS/JavaScript de base, à bien tester le rendu, puis à reprendre ce code pour y insérer les instructions PHP nécessaires.

Pour formater la présentation des variables et des messages, PHP offre deux fonctions intégrées efficaces, `printf()` et `sprintf()`. Ces fonctions aux multiples variantes (attention à la syntaxe !) permettent essentiellement de :

- Préparer la structure d'un message à générer en identifiant la place et le type des variables à y insérer et réaliser les concaténations nécessaires ;
- Formater des variables numériques ou date ;

### Exemple de message préparé :

```
$num = 5;
```

```
$location = 'bananier';
```

```
$format = 'Il y a %d singes dans le %s';
```

```
echo sprintf($format, $num, $location);
```

Variables à insérer

Structure du message

Génération du message final par  
remplacement des symboles %d et %s par  
le contenu formaté des variables

### Exemple de formatage de variables :

```
echo sprintf("%'.9d\n", 123);
```

```
echo sprintf("%'.09d\n", 123);
```

Résultat :

.....123

000000123



Pour tout savoir sur ces fonctions de formatage, une seule adresse :  
<http://php.net/manual/fr/function.sprintf.php>

## 5.2 BUFFERISATION

Pour optimiser les échanges entre client Web et serveur Web et éviter des blocages sur des affichages partiels dans le navigateur, PHP propose un mécanisme de mise en mémoire tampon du flot HTML/CSS/JavaScript généré jusqu'à l'achèvement des traitements concernant la page en cours. Ceci est d'autant plus utile que la page est longue ou lourde et qu'elle fait appel à des services distants (bases de données, Web services...) par toujours prompts à répondre.


A la base, le développeur débute une phase de bufferisation par l'appel de la fonction `ob_start()` ; il alimente le buffer à l'aide des instructions PHP ordinaires de génération de code à adresser au client ; tout ce code se cumule dans la tampon sans que rien ne soit transmis au client. Au final, pour déclencher l'envoi, le développeur récupère le contenu cumulé du buffer grâce à la fonction `ob_get_contents()` et vide le buffer en appelant `ob_end_clean()`.

Comme toujours en PHP, il existe de nombreuses variantes (même pour cette modeste fonctionnalité !) ; ainsi `ob_get_clean()` est un l'équivalent des deux fonctions successives `ob_get_contents()` et `ob_end_clean()`.

Les fonctions de gestion de la bufferisation ne changent rien à la logique des traitements et peuvent bien entendu être ajoutées en fin de développement, après mise au point des scripts.

### Exemple :

```
<?php
/* procedure Menu */
function afficheMenu()
{
    ob_start(); // debute bufferisation
    ?>
    <div id='sub_gestion' class='sub_menu'>
        <span><a href='clients'>clients</a></span>
        <span><a href='natures'>nature client</a></span>
        <span><a href='secteur_activite'>secteur d'activité</a></span>
        <span><a href='types'>type</a></span>
        <span><a href='contacts'>contacts</a></span>
        <span><a href='../deconnexion' >déconnexion|</a></span>
    </div>
    <?php
    $output = ob_get_contents(); // recupere buffer en variable
    ob_end_clean(); // vide buffer et fin bufferisation
    echo $output; // envoi direct au client
}
?>
```

 Pour tout savoir sur les fonctions de bufferisation, une seule adresse : <http://php.net/manual/fr/ref.outcontrol.php>

Initiation au langage PHP

Afpa © 2016 – Section Tertiaire Informatique – Filière « Etude et développement »

### 5.3 REDIRECTIONS AVEC HEADER()

Enfin, dans ce chapitre consacré aux réponses exprimées par le serveur Web, il est nécessaire de citer le mécanisme de redirection de page.

Les pages Web sont véhiculées selon le "protocole HTTP" entre logiciel "client Web" (= browser) et logiciel "serveur Web" (Apache, IIS...)

- Le client adresse une "requête HTTP" contenant principalement l'URL de la page demandée (et paramètres éventuels après le signe ?) ainsi que des informations de service (adresse IP du demandeur, durée de vie de la page, autorisation de mise en cache par les routeurs...) ; cela constitue un message de type "entête HTTP" ;
- Le serveur Web retourne une "réponse HTTP" : page HTML dans un "corps de message HTTP", précédée d'un "entête HTTP" contenant des infos de service (dont le code erreur 200 si tout va bien ou les familiers 403, 404, 500...) ;

Tout cela est pris en compte de manière transparente par les logiciels client Web et serveur Web, et tout va bien tant que l'on reste sur ces mécanismes standards, demande de page, service de page.

Le développeur identifie souvent des situations différentes qui doivent enchaîner sur une même page. C'est le cas typique de la gestion de certaines erreurs comme l'absence d'identification d'un utilisateur demandant une page privée d'un site ; en effet, en Web, toute page publiée est un point d'entrée potentiel dans le site et il est impossible de forcer l'utilisateur à commencer sa visite par la page de login (Ah ! le bon temps du Minitel ! Ah ! le bonheur des 'applis' mobiles !). Par contre il reste aisé de le rediriger vers cette page quand on détecte l'absence d'identification depuis une page quelconque.



Pour rediriger un utilisateur vers une autre URL que le script en cours d'exécution, PHP propose la fonction intégrée `header('location: ');`.

Cette fonction modifie la réponse du serveur mais ne met pas fin à l'exécution du script ; en général elle doit donc être suivie d'une fonction `exit();` ou `die();` pour mettre fin au script en cours d'exécution.

Cette fonction `header()` admet de nombreux paramètres ; le paramètre `location:` permet la redirection vers une autre URL.

Au-delà, `header()` sert à paramétrer l'entête de réponse HTTP concernant la transaction en cours entre client Web et serveur Web.

Ainsi `header('Content-Type:application/pdf');` déclare le type de document qui sera retourné au client (en l'occurrence, du pdf et non du HTML).

Pour forcer uniquement le 'code d'erreur HTTP', PHP propose la fonction intégrée `http_response_code();` cet exemple permet de forcer un indicateur de réponse 'Not Found' : `http_response_code(404);`.



Pour tout savoir sur la fonction `header()`, une seule adresse : <http://php.net/manual/fr/function.header.php>

## 6. LES STRUCTURES DE CONTROLE

Le langage PHP est directement inspiré des langages modernes comme C# ou Java en ce qui concerne les structures de contrôle du déroulement des programmes, y compris le système de 'blocs' délimités par les accolades ouvrantes et fermantes ({}).

En bref, si on sait écrire un test ou une boucle en Java ou C#, on sait les écrire en PHP !

### 6.1 TESTS

Le langage PHP dispose donc des classiques structures :

```
if( condition ) { actions si vrai } ;
```

```
if( condition ) { actions si vrai } else { actions si faux } ;
```

Les bonnes pratiques conduisent à écrire la condition sur la première ligne puis à effectuer des sauts de ligne pour chaque instruction, des indentations par blocs, et à écrire systématiquement les accolades même si une seule instruction est nécessaire dans un bloc (une seule aujourd'hui mais... et demain ?).

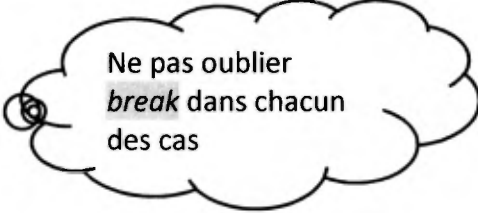
Exemple de test :

```
if (strtoupper($usrn timer)) != "ADMIN")
{
    echo "ERREUR login !";
}
else
{
    echo "Au menu...";
} ;
```


PHP offre de plus la variante `if( ) { } elseif { }elseif { }` qui permet d'écrire plus simplement une série de tests exclusifs les uns des autres.

Enfin, quand l'action à réaliser ne dépend que de la valeur d'une variable, on peut encore utiliser la structure

```
switch( nomvariable ) {
    case valeur1 : action pour valeur1 ; break ;
    case valeur1 : action pour valeur1 ; break ;
    ...
} ;
```



Ne pas oublier  
*break* dans chacun  
des cas

 Pour les tests PHP, une seule adresse : <http://php.net/manual/fr/language.control-structures.php>

## 6.2 BOUCLES

Pas de surprise non plus pour exprimer des boucles : PHP reprend encore les syntaxes C#/Java.

Pour recommencer des opérations tant qu'une condition est vérifiée, on utilisera :

```
while ( condition ) { actions à recommencer } ;
```

Ce type de boucle sera exécuté de zéro à n fois.

*Attention que pour entrer au moins une fois dans la boucle, la condition devra être initialisée avant cette structure.*

Pour faire et recommencer éventuellement des opérations tant qu'une condition est vérifiée, on utilisera :

```
do { actions à recommencer. } while ( condition ) ;
```

Pour répéter des opérations un certain nombre déterminé de fois, PHP propose la structure

```
for ( situation initiale ; condition d'arrêt ; progression )  
    { actions à recommencer } ;
```

Enfin pour parcourir tous les items d'un tableau, le développeur pourra utiliser :

```
foreach ( nomtableau as nomvariable ) { actions à recommencer } ;
```

et s'il s'agit d'un tableau associatif, on peut récupérer directement la paire clé/valeur :

```
foreach ( nomtableau as cle => valeur ) { actions à recommencer } ;
```

### Exemples de boucles équivalentes :

```
$i = 1; // initialisation  
while ($i <= 10) { // condition en début de boucle  
    echo $i ; // actions à recommencer  
    $i++; // progression  
} ;
```

```
$i = 1; // initialisation  
do {  
    echo $i ; // actions à recommencer  
    $i++; // progression  
} while ($i <= 10); // condition en fin de boucle
```

```
for ($i = 1 ; $i <= 10 ; $i++) { // définition des paramètres de boucle  
    echo $i ; // actions à recommencer  
};
```



Pour les boucles en PHP, la même adresse : <http://php.net/manual/fr/language.control-structures.php>



## 7. LA MISE AU POINT DES SCRIPTS

### 7.1 FONCTIONS DE TRACE

En cas d'erreur lors de l'exécution d'un script, l'interpréteur PHP interrompt son travail et génère un magnifique tableau orange de messages d'erreur adressé au navigateur. Si cela est 'normal' au cours de la mise au point d'un programme PHP, c'est inacceptable dans une application ou un site mis en production !

Pour aider à la compréhension de l'erreur et à la mise au point des programmes, un développeur utilise habituellement un débogueur... qui fait encore cruellement défaut dans la panoplie d'outils PHP ! En conséquence, il reste très difficile d'effectuer en PHP des pauses pour observer l'état des variables, d'autant plus que le script est en général exécuté sur un serveur distinct et distant du poste de développement.

PHP propose malgré tout deux fonctions très utiles lors de la phase de mise au point. Elles ont toutes deux vocation à inclure dans le flux adressé au navigateur client des informations techniques sur l'état des variables à l'instant t :



`var_dump( nomvariable )` ; restitue en clair le contenu d'une variable et fournit des informations très claires et très complètes sur les tableaux et les objets.

`print_r( nomtableau )` ; est plus spécialement adapté à l'exploration du contenu de tableaux indicés ou associatifs.

*Attention : l'absence d'erreur d'exécution côté serveur Web ne certifie pas un code côté client correct ; le développeur doit systématiquement contrôler le code HTML/CSS/JavaScript reçu par le navigateur en usant des outils de débogage des browsers.*

#### Exemples :

The diagram illustrates the output of PHP functions used for debugging. It shows a sequence of outputs with callouts explaining what each line represents:

- Coucou**: A simple string output.
- 12345678910**: A numeric output.
- int 11**: An integer output.
- object(DateTime)[1]**: The start of an object dump.
- public 'date' => string '2016-03-18 07:22:55' (length=19)**: A public property of the DateTime object.
- public 'timezone\_type' => int 3**: Another public property of the DateTime object.
- public 'timezone' => string 'Europe/Paris' (length=12)**: A third public property of the DateTime object.
- array (size=4)**: The start of an array dump.
- 0 => string 'foo' (length=3)**: The first element of the array.
- 1 => string 'bar' (length=3)**: The second element of the array.
- 2 => int 12**: The third element of the array.
- 3 => boolean true**: The fourth element of the array.
- Array ( [0] => foo [1] => bar [2] => 12 [3] => 1 )**: The output of print\_r() for the same array.

Annotations (callouts) point to specific parts of the output:

- "Résultat d'une boucle ci-dessus" points to "Coucou".
- "var\_dump() sur variable \$i en fin de boucle" points to "int 11".
- "var\_dump() sur variable DateTime" points to the DateTime object dump.
- "var\_dump() sur tableau associatif" points to the array dump.
- "print\_r() sur même tableau associatif" points to the print\_r output.

## 7.2 LA GESTION DES ERREURS

Comme pour toute application, la gestion des erreurs d'exécution doit être soignée en PHP. Heureusement, depuis PHP5, le développeur dispose des bons outils !

Pour protéger une séquence d'instructions potentiellement faillible (utilisation d'une connexion réseau, lecture ou écriture sur disque, appel d'un service distant...), le développeur pourra encadrer la séquence dans un bloc `try{ };` ; en cas d'erreur d'exécution, l'interpréteur sautera directement dans le bloc `catch( ){ }` correspondant sans provoquer de 'levée d'erreur' PHP. En option, on peut prévoir un bloc `finally { }` qui sera exécuté dans tous les cas de figure.

```
try { actions potentiellement dangereuses }
catch ( ) { actions en correction }
finally { actions dans tous les cas }
```



Une bonne pratique dans ce type de gestion des erreurs d'exécution consiste à ajouter ces structures `try/catch` au dernier moment, après mise au point du script PHP (afin d'éviter un masquage des erreurs dues à une mauvaise programmation).

Si le développeur veut intentionnellement lever une erreur PHP dans certains cas identifiés, il peut utiliser l'instruction `throw`.

Le bloc `catch` reçoit en paramètre un objet de type `Exception` ; l'instruction `throw` génère un objet `Exception`.

La classe `Exception` expose principalement les méthodes `getMessage()` et `getCode()` qui permettent au développeur d'en savoir un peu plus sur la nature de l'erreur rencontrée et de restituer ces informations à l'utilisateur de manière un peu plus habillée.



Pour tout savoir sur la gestion des erreurs en PHP, une seule adresse : <http://php.net/manual/fr/language.exceptions.php>

Pour interrompre volontairement l'exécution du script en cours sans pour autant lever une erreur PHP, le développeur peut utiliser l'une ou l'autre des fonctions synonymes `exit()` et `die()` qui peuvent accepter en paramètre un message final à adresser au navigateur.

### Exemple :

```
if( /* c'est vraiment la cata ! */ )
{exit('<h3>Erreur fatale !</h3>'); }
```

Ajoute le message au flux HTML en cours et interrompt le traitement du script sans lever d'erreur



## 8. LES PROCEDURES ET FONCTIONS EN PHP

En programmation, les '*procédures*' et les '*fonctions*' permettent de modulariser le code en isolant des portions de code pour réutilisation. Les '*sous-programmes*' procédures et fonctions sont donc constitués d'instructions qui ne seront exécutées que lors de leur appel par le programme principal.

Une procédure appelée exécute ses instructions et redonne la main au programme principal.

Une fonction appelée exécute ses instructions et retourne une valeur au programme principal.

Les procédures et fonctions permettent une meilleure lisibilité du code source, un gain de productivité ainsi qu'une maintenance facilitée.

En développement Web, les besoins de réutilisation de portions de code sont omniprésents puisque chaque page d'un site fait l'objet d'un script et que de nombreuses parties de la page comme les entêtes ou les menus se répètent de page en page. Le développeur PHP se doit donc de bien modulariser son code ; PHP lui permet de définir ses propres fonctions et d'isoler des portions de code qui seront fusionnées par l'interpréteur au moment de l'exécution (voir plus loin).

Le langage PHP ne différencie pas les procédures des fonctions dans leur déclaration : elles sont toutes définies par un bloc nommé et déclaré comme `function`. Toutefois, une fonction devra contenir une ou plusieurs instructions `return` de manière à retourner la valeur résultante au programme appelant.

En PHP, l'appel d'une `function` se limite à l'invocation de son nom en lieu et place où l'on souhaite faire exécuter le traitement correspondant.

Comme le PHP est un langage faiblement typé, la '*signature*' d'une `function` se limite à déclarer le nom et énumérer entre parenthèses les éventuels paramètres nécessaires. Même en l'absence de paramètres, les parenthèses sont obligatoires. En conséquence, il n'est pas facile de savoir comment utiliser une fonction PHP dont on n'est pas l'auteur et une bonne documentation s'impose (comme sur le site de référence [php.net](http://php.net)) !

### Exemples :

```
$texte=ucfirst("hello world...");
```

Appel d'une fonction intégrée acceptant un paramètre

```
// fonction de contrôle du login
// retourne true si OK, false sinon
function ctrlLogin()
{
    global $usrn timer;
    if (!(isset($usrn timer)))
    {
        return false;
    }
    else
    {
        return true;
    }
};
```

Exemple de définition de fonction sans paramètre

```
// contrôle du login
if (!(ctrlLogin()))
{
    echo "ERREUR login !"; // erreur de login
}
else // login OK
{
    echo "Au menu...";
}
```

Exemple d'utilisation de la fonction dans le programme principal

Pour une bonne réutilisation de ces fonctions, il est nécessaire de les isoler dans des fichiers de script séparés ; un chapitre suivant explique comment faire appel à ces portions de code externes.

## 9. PORTEE DES VARIABLES

La portée des variables doit être assez soignée en PHP.

Toute variable déclarée à l'intérieur d'un bloc de code (une fonction, une boucle) n'existe que le temps de l'exécution de cette structure. De même pour les paramètres reçus par une fonction. On parle de variable de *portée locale*.

Toute variable déclarée en dehors d'un bloc de code est de *portée globale* pour le contexte d'exécution, soit, en général, pour la page Web en cours de génération.

### Exemple :

```
< ?php
$a = 1; /* portée globale pour le script en cours */
function Test() {
    $a = 2; /* même nom de variable mais portée locale à la fonction */
    echo $a; /* affichera le contenu de la variable locale */
} ;
Test(); /* valeur affichée ? Compléter : _____ */
?>
```



Particularité du langage PHP : si une variable globale doit être réutilisée dans une fonction sans lui avoir été passée en paramètre, il faut la re-déclarer dans la fonction avec le mot-clé **global**.

### Exemple :

```
<?php
$a = 1;
$b = 2;
function Sum() {
    /* redéclarer les var. globales pour pouvoir les utiliser */
    global $a, $b;
    $b = $a + $b;
};
Sum();
echo $b; /* Valeur de $b ? Compléter : _____ */
?>
```

## 10. ECRITURE MODULAIRE DU CODE PHP

*Pour une bonne réutilisation des fonctions définies par le développeur, il est nécessaire de les isoler dans des fichiers de script séparés. Mais comme le langage PHP est de type interprété, l'interpréteur doit disposer de l'ensemble du code source nécessaire au traitement d'une page avant de commencer le travail d'interprétation ; il est donc nécessaire de fusionner tous les fichiers de scripts au moment de l'exécution de chaque page dynamique. C'est l'objet de quatre fonctions PHP intégrées : `include()`, `include_once()`, `require()` et `require_once()` que le développeur PHP doit manier avec précautions car ces fusions de fichiers sont consommatrices de ressources serveur Web et peuvent provoquer elles aussi des erreurs d'exécution.*

Ces quatre fonctions demandent en paramètre le chemin d'accès et le nom du fichier source à fusionner. La fusion se fait en lieu et place de l'instruction de fusion ; il faut donc veiller à placer ces fonctions de fusion à des endroits corrects.

La fonction `include()` recherche sur disque le script à fusionner ; en cas de succès, le code est fusionné et il remplace la fonction `include()` ; en cas d'échec, rien ne se passe et la fonction `include()` est ignorée.

La fonction `require()` est basée sur le même principe sauf que si le fichier source demandé reste introuvable, une erreur est levée.

Les variantes `include_once()` et `require_once()` permettent de s'assurer que le même fichier source ne sera fusionné qu'une seule fois pour la page en cours. En effet, après fusion, si l'interpréteur rencontre plusieurs fois la même définition de fonction, il lève une erreur. Mais au bout du compte, si le développeur organise bien ses portions de code, il ne devrait jamais recourir à ces deux variantes !

### Exemple de modularisation pour la gestion de login précédente.

```
// fonction de contrôle du login
// retourne true si OK, false sinon
function ctrlLogin($usrn timer)
{
    if (!(isset($usrn timer)))
    {
        return false;
    }
    else
    {
        return true;
    }
};
```

Fonction(s) stockée(s) dans un fichier source séparé login.inc

Pour une meilleure écriture, sans variables globale, la fonction reçoit maintenant un paramètre

Fusion du module de script externe  
login.inc situé dans le même  
dossier que le script principal

Fichier de script principal

```
require("login.inc") ;
```

```
...
```

```
$nom = ...
```

```
// contrôle du login
```

```
if (!(ctrlLogin($nom)))
```

```
{
```

```
    echo "ERREUR login !"; // erreur de login
```

```
}
```

```
else // login OK
```

```
{
```

```
    echo "Au menu...";
```

```
}
```

Pour une meilleure écriture, sans  
variables globale, la valeur d'une  
variable locale est passée à la fonction

Il est de bonne pratique de bien concevoir les différents modules de code source est d'utiliser systématiquement la fonction `require()` pour les fusionner avec le script principal. Le développeur veillera aussi à ne fusionner que les modules nécessaires afin de limiter les consommations de ressources du serveur Web.

## 11. PROPAGER DES DONNEES DE PAGE WEB EN PAGE WEB

Le dialogue entre client Web et serveur Web est géré par le protocole HTTP qui présente le principal inconvénient d'être un protocole *'sans connexion persistante'*, autrement dit *'sitôt servi, sitôt oublié'* ! Les différentes pages servies aux utilisateurs sont comme des phrases de conversations totalement déconnectées entre elles. Dans ces conditions il est difficile de propager des données entre les différentes pages visitées, ce qui pose un problème sérieux pour afficher systématiquement le nom de la personne qui s'est connectée en début de conversation ou pour suivre la progression d'un panier d'achat !

Pour résoudre ces limitations, le développeur dispose de trois techniques, celle utilisée par les formulaires HTML et les mécanismes de cookies et de variables de session.

Le protocole HTTP admet deux méthodes de transmission des données du client vers le serveur :

- en méthode GET, les données sont accolées à l'URL de la page demandée et seul l'entête de message http est véhiculé sur le réseau ; le volume des données reste limité par la taille maximale de cet entête HTTP ;
- en méthode POST, les données sont transmises dans le corps du message HTTP et les deux parties du message sont véhiculées sur le réseau ; le volume des données n'est pas limité ;

Le langage HTML reprend ces principes dans l'attribut `method` de la définition des formulaires. L'utilisateur peut identifier la méthode en observant la barre d'adresse du navigateur :



## Méthode POST



## Méthode GET

Le nom des variables transmises correspond en standard à l'attribut HTML `name` des différentes zones de saisie constituant le formulaire, que ces zones aient été renseignées ou non côté client. Ainsi dans l'exemple ci-dessus, le formulaire HTML était constitué de zones de saisie `nom`, `prenom`, `age` et `b1` et seule la zone `b1` a son attribut `value` renseigné.

✶ L'environnement PHP met à disposition du développeur toutes les données reçues par le serveur dans des tableaux associatifs *superglobaux* :

- `$_GET` contient toutes les données passées en méthode HTTP GET ;
- `$_POST` contient toutes les données passées en méthode HTTP POST ;
- `$_REQUEST` contient toutes les données passées en méthodes HTTP GET et POST ;

Les données reçues par le serveur sont disponibles dans ces tableaux selon une valeur de clé qui correspond au nom de la variable. Ainsi dans l'exemple ci-dessus, en méthode POST, le développeur dispose des variables `$_POST["nom"]`, `$_POST["prenom"]`, `$_POST["age"]` et `$_POST["b1"]` mais le tableau `$_GET` reste vide ; en méthode GET, il dispose des variables `$_GET["nom"]`, `$_GET["prenom"]`, `$_GET["age"]` et `$_GET["b1"]` mais le tableau `$_POST` reste vide.

Le développeur doit donc savoir quelle méthode a été utilisée par un formulaire HTML pour pouvoir en exploiter les données saisies. Au besoin, le développeur peut toujours exploiter le tableau `$_REQUEST` car il est alimenté par PHP dans tous les cas.

✶ Pour une bonne organisation de la communication entre client et serveur Web, le développeur doit prendre garde que :

- Seules les zones de formulaire nommées par l'attribut `name` sont transmises par le navigateur ; il est en général inutile de nommer les boutons de commande ;
- Si plusieurs zones de formulaire portent le même `name` pour des informations différentes (cas de listes), il est prudent de les nommer en HTML avec des crochets (`name="matricule[]"`) de manière à ce que PHP les récupère bien dans des tableaux `array` ;
- Dans le cas des boutons radio qui ne supportent qu'une seule et même information pouvant prendre plusieurs valeurs, une seule variable est passée au serveur Web et sa valeur correspond au `value` du bouton choisi par l'utilisateur ;
- Dans le cas des cases à cocher, seules les cases réellement cochées sont transmises au serveur ;



Le développeur doit donc fréquemment tester si une information a bien été transmise depuis le navigateur client. PHP propose pour cela :

- La fonction `isset()` qui teste l'existence d'une variable ;
- La fonction `empty()` qui teste le contenu d'une variable ;
- La fonction `count()` qui permet de connaître le nombre de poste d'un tableau array ;

#### Exemple d'utilisation :

```
if(isset($_GET["matricule"])) // au moins 1 matricule reçu dans l'URL
{
    if(count($_GET["matricule"]) >1)
    { // plusieurs matricules recus ==> boucle
        foreach ($_GET["matricule"] as $unMatricule) {...}
    }
    else // un seul matricule reçu
    {
        if (empty($_GET["matricule"])) { // reçu mais non renseigne }
        else { // reçu et renseigne }
    }
}
```

**NB :** Les fonctions `isset()` et `empty()` ne sont bien entendu pas réservées à l'exploitation des formulaires.

**NB :** Il n'est pas inutile de rappeler que toute donnée saisie dans un formulaire est potentiellement dangereuse ('faille XSS' ou risque 'd'injection SQL') ; elle doit donc être systématiquement contrôlée ou transformées grâce aux fonctions intégrées comme `htmlentities()` citées plus haut.

Au-delà du dialogue avec l'utilisateur en utilisant les formulaires, le mécanisme de communication d'informations par la méthode GET entre pages Web peut être utilisé par le développeur pour propager intentionnellement des données de page en page, et pour paramétrer le fonctionnement de la page dynamique suivante sans pour autant mettre en œuvre le mécanisme (gourmant) de gestion des variables de session (voir chapitre suivant).

En effet, il est très facile de reproduire par programmation une URL paramétrée :

- Par PHP :

```
<?php
$param = "?nom=" . $lenom;
?>

...
<a href= "pagesuite.php"<?php echo $param ;?>
...

```

- Et même en utilisant JavaScript puisque PHP a pour but de personnaliser le code HTML/CSS/JavaScript à livrer au client :

```
<script >
...
param = "?nom=<?php echo $lenom; ?>";
...
window.location.href = "pagesuite.php" + param ;
</script>

```

Dans le premier cas, le script PHP prépare une URL paramétrée que l'utilisateur pourra invoquer à l'aide d'un lien hypertexte. Dans le deuxième cas, le lien vers cette page suivante pourra être demandé par JavaScript en fonction d'événements détectés par JavaScript. Dans les deux cas, le script `pagesuite.php` disposera d'une variable `$_GET["nom"]` dont le contenu aura été déterminé par une page dynamique PHP précédemment exécutée.



## 12. GERER LES SESSIONS EN PHP

### 12.1 PRINCIPE

Les mécanismes de gestion des sessions sont des extensions offertes par les logiciels serveur Web de manière à compenser la '*mémoire courte*' du protocole HTTP.

Comme chaque page dynamique est servie indépendamment des autres, le plus haut niveau de portée des variables reste le niveau du script en cours d'exécution. Les variables de session permettent d'enregistrer des données et de les préserver pour les réutiliser au fil des pages visitées par un même utilisateur.

Ces variables de session sont très utiles pour mémoriser des données de connexion de l'utilisateur ou pour enregistrer temporairement des paniers d'achat par exemple.

Quel que soit le logiciel serveur Web, une session :

- Est identifiée automatiquement (identifiant attribué par le serveur) ;
- A une durée de vie limitée en cas d'inactivité de l'utilisateur ;
- Est liée à une session de travail sur un navigateur (autre navigateur = autres session ; fermer et rouvrir le même navigateur provoque un changement de session) ;
- Est consommatrice de ressources en particulier lors de la montée en charge (chaque utilisateur simultané doit disposer d'un espace mémoire spécifique pour stocker ses variables de session) ;

### 12.2 MISE EN ŒUVRE EN PHP

Le tableau associatif *superglobal* `$_SESSION` permet de gérer des variables dont la durée de vie est celle de la session de l'utilisateur ; ce tableau permet donc de suivre l'utilisateur au fil des pages qu'il visite.



Comme ce mécanisme est gourmand en ressources serveur, il n'est pas activé par défaut ; chaque page qui souhaite **utiliser** ce service devra tout d'abord **l'activer** en insérant la fonction `session_start()` ; en première ligne du script principal. La fonction `session_destroy()` quant à elle permet de supprimer toutes les données de session (cas d'abandon du panier ou de déconnexion volontaires de l'utilisateur).

Dès lors, le développeur peut créer, interroger, modifier la valeur ou encore détruire chacune des variables de session dont il a besoin à travers ce tableau `$_SESSION`.

Exemples :

```
session_start() ; // activation sessions
$_SESSION['nom'] = htmlentities($_POST['nom']) ; // mémo nom user
... ..
```

```
session_start() ; // activation sessions
if ( !isset($_SESSION['nom'])) { // la var de session n'existe pas
    header('location:login.php ; // ==> redirection
    exit() ; // et fin
}
```



Pour tout savoir sur les variables de session en PHP, une seule adresse : <http://php.net/manual/fr/ref.session.php>

## 13. ENCORE ET TOUJOURS PLUS EN PHP

Explorer le monde du PHP donne un idée de ce que peut être explorer l'univers : flirter avec la notion d'infini ! Avec PHP, point de limites et des armées de développeurs font évoluer le langage en permanence, principalement en développant des extensions que l'on peut activer ou non sur le serveur Web en fonction des besoins.

En conséquence, le développeur recherchera ou activera la bibliothèque de fonctions, voire le framework, correspondant à ses besoins spécifiques avant de concevoir et créer ses propres bibliothèques. Bien souvent, un temps d'exploration non négligeable sera nécessaire pour comprendre, assimiler et savoir choisir parmi les très nombreuses fonctions proposées.

 Parmi les fonctionnalités les plus courantes, on peut citer :

- La simplissime fonction intégrée `mail()` pour générer et envoyer des emails en PHP (<http://php.net/manual/fr/function.mail.php>) ;
- L'utilisation intégrée des '*expressions régulières*' pour réaliser des tests (et bien plus encore !) comme dans les langages C#/Java/JavaScript (<http://php.net/manual/fr/ref.pcre.php>) ;
- La manipulation de fichiers stockés sur le serveur Web (<http://php.net/manual/fr/ref.filesystem.php>) ;
- La manipulation d'images (<http://php.net/manual/fr/ref.image.php> et extension ImageMagick <http://php.net/manual/fr/book.imagick.php>) ;
- La manipulation de documents pdf (<http://php.net/manual/fr/ref.pdf.php>) ;
- Et bien d'autres encore...

Avec la dimension '*orientée objet*' intégrée depuis la version 5, le langage PHP est réellement aujourd'hui un vrai langage de programmation pour le Web... et pour le reste !

## **CREDITS**

### **ŒUVRE COLLECTIVE DE L'AFPA**

**Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services**

#### **Equipe de conception (IF, formateur, mediatiseur)**

B. Hézard - Formateur

Ch. Perrachon – Ingénieure de formation

**Date de mise à jour : jj/03/16**

### **Reproduction interdite**

Article L 122-4 du code de la propriété intellectuelle.

« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droits ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction par un art ou un procédé quelconque. »

Initiation au langage PHP

Afpa © 2016 – Section Tertiaire Informatique – Filière « Etude et développement »