Initiation aux langages du Web :

PHP

Table des matières

1. Introduction

Le PHP ou ***Hypertext Preprocessor*** est un langage de script qui permet de produire des pages Web de façon dynamique. Il peut également fonctionner de façon locale en mode lignes de commandes. Depuis la version 5, PHP est devenu un langage disposant de fonctionnalités objets très poussées.

Le gros avantage de PHP est également sa capacité à interagir avec des bases de données.

Ci-dessous un exemple de script qui permet d'afficher du texte :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
	<title>PHP basic</title>
	<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
	<meta http-equiv="Content-Language" content="fr" />
</head>
	<body>
		<?php
			$message = "PHP dit bonjour !»;
			echo $message;
		?>
	</body>
</html>
```

Exemple 1: PHP basique

On remarque que :

- les balises « <?php » et « ?> » encadrent le code PHP ;
- les variables commencent par « $ » ;
- chaque ligne se termine par « ; ».

2. Syntaxe

a) Les balises

Il est possible d'insérer du code PHP de trois façons différentes :

- avec les tags canoniques « <?php » et « ?> » ;
- avec les tags SGML « <? » et « ?> » ;
- avec les tags HTML « <script language="php"> » et « </script> ».

La choix le plus communément utilisé est le premier.

b) Les commentaires

Il est possible d'insérer des commentaires dans votre code PHP des façons suivantes :

```
<?php
# Ceci est un commentaire
# Deuxième ligne de commentaire
// Ceci est également un commentaire;
/*
Il est possible de faire des
commentaires de plusieurs
lignes
*/
?>
```

Exemple 2: Commentaires en PHP

c) Insertion de texte

Il est possible d'insérer du texte grâce aux fonctions « echo » et « print » de la manière suivante :

```
<?php
# Utilisation de echo
echo "Bonjour";
# Utilisation de echo
print("Bonjour");
# Il est possible d'afficher un texte multiligne en utilisant des bloc...
print <<<END
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Sed non risus.
Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor.
END;
# ... ou les ""
print "Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Sed non risus.
Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor.";
?>
```

Exemple 3: Insertion de texte

3. Inclusion

L'inclusion permet d'insérer le contenu d'un fichier PHP avant qu'il soit exécuté par le serveur. Il existe deux fonctions pour insérer du code PHP : « include » et « require ».

a) Include

La fonction « include » prend tous les textes contenus dans le script PHP et le copie dans le fichier qui l'utilise. Si le script contient une erreur, « include » génère un warning mais ne bloque pas l'exécution du script.

```
<?php
include("menu.php");
echo "Test"; // Même si menu.php contient une erreur le texte s'affichera
?>
```

Exemple 4: Utilisation de « include »

b) Require

La fonction « require » agit un peu comme la fonction « include » à la différence que si le script inclus contient une erreur, « require » génère une erreur fatale (***fatal error***) et bloque l'exécution du script.

```
<?php
require("menu.php");
echo "Test"; // Si menu.php contient une erreur le texte ne s'affichera pas
?>
```

Exemple 5: Utilisation de « require »

4. Les Objets

a) Types primitifs

La caractéristique la plus importante d'un langage de programmation est certainement la faculté de manipuler des variables. PHP permet de manipuler les styles suivants :

- les nombres (eg. 1, 12,50, ...) ;
- les chaînes de caractères (eg. "ceci est une chaîne") ;
- les booléens (eg. vrai ou faux) ;
- les objets (eg. tableaux, ressources,...).

Il existe également un type de données trivial : NULL.

Les variables sont précédées du cratère « $ » comme montré ci-dessous :

```
<?php
```

```
$pet;
$num_legs, $tail;
?>
```

Exemple 6: Déclaration de variables

Le fait de stocker une valeur dans une variable s'appelle **l'initialisation**.

```
<?php
$pet;
$num_legs, $tail;

$pet = "dog";
$num_legs = 4;
$tail = true;

?>
```

Exemple 7: Initialisation de variables

La création et l'initialisation peuvent se faire en même temps.

b) Nommage

Il n'est pas possible d'attribuer n'importe quel nom à une variable et, de manière générale, il faut respecter les règles suivantes :

- les variables ne peuvent contenir les caractères : +, -, %, (,), &, ... ;
- toujours faire commencer le nom par une lettre ou « _ » ;
- PHP est sensible à la casse.

c) Portée ou périmètre

La portée ou périmètre peut être assimilée à la « région » du programme où la variable est **définie** et **utilisable**. Il existe deux périmètres :

- **global**, cela signifie que la variable est définie partout dans le code PHP, même dans des fichiers différents;
- **local**, cela signifie que la variable est définie dans une portion de code restreinte, généralement une fonction.

Il existe plusieurs types de variables globales :

- **statique**, cela signifie que la variable ne perdra pas sa valeur, même après exécution d'une fonction ;
- **constante**, cela signifie que la variable peut être appelée sans mettre de « $ ».

Dans le corps d'une fonction, les variables locales prennent l'ascendance sur les variables globales :

```
<?php
define("VALEUR_E", 3);
$a = 2; // Variable globale
$b = 1; // Variable globale
static $c = 0;
function add() {
        $a = 5; // Variable locale
        global $b; // On récupère la valeur de « d »
        global $c;
        $c = $a + $b; // c = 6
}
add();
$d = $c + VALEUR_E;  // d = 9
?>
```

Exemple 8: Notion de périmètre ou portée

Le langage PHP contient d'office quelques variables statiques :

Nom	Description
__DIR__	Contient le chemin canonique menant jusqu'au fichier
__LINE__	Contient le numéro de ligne du fichier courant
__FILE__	Contient le nom du fichier courant
__FUNCTION__	Contient le nom de la fonction en cours d'exécution
__CLASS__	Contient le nom de la classe en cours d'utilisation
__METHOD__	Contient le nom de la méthode en cours d'exécution

d) Les tableaux

Il existe trois types de tableaux en PHP :

- **numériques**, qui ne contiennent qu'une colonne accessible via un index numérique ;
- **associatifs**, qui ne contiennent qu'une colonne indexée avec des chaînes de caractères ;
- **multidimensionnels**, qui contiennent plusieurs tableaux accessibles à travers leurs index respectifs.

Tableaux numériques

Ces tableaux permettent de stocker des nombres, chaînes de caractères et n'importe quel objet. Leurs index sont représentés par des nombres et ils commencent par l'index 0 (zéro).

```
// Création avec l'objet
$chiens = array("caniche", "cocker", "bulldog");
// Création par indexes
$chiens[0] = "caniche";
$chiens[1] = "cocker";
$chiens[2] = "bulldog";
```

Exemple 9: Création d'un tableau numérique

Tableaux associatifs

Les tableaux associatifs sont très proches des tableaux numériques en terme de fonctionnalités mais ils diffèrent au niveau des index. Les index utilisés sont des chaînes de caractères et cela permet de renforcer le lien clé / valeur.

```
// Création avec l'objet
$poids = array("caniche"=>4, "cocker"=>9, "bulldog"=>12);
// Création par indexes
$poids["caniche"] = 4;
$poids["cocker"] = 9;
$poids["bulldog"] = 12;
```

Exemple 10: Création d'un tableau associatif

Tableaux multidimensionnels

Avec les tableaux multidimensionnels, on garde l'association clé / valeur des tableaux associatifs sauf que, cette fois-ci, la valeur peut être un tableau et la valeur du sous-tableau peut également être un tableau et ainsi de suite.

```
$chiens = array(
                "gipsy" => array
                (
                "race" => "caniche",
                "poids" => 4,
                ),
                "poufy" => array
                (
                "race" => "cocker",
                "poids" => 9,
                ),
                "flash" => array
                (
                "race" => "bulldog",
                "poids" => 12,
                ),
);
```

Exemple 11: Création d'un tableau multidimensionnel

Taille d'un tableau

La fonction « count » permet de connaître le nombre d'éléments d'un tableau :

```
$poids = array("caniche", "cocker", "bulldog");
$nb_element = count($chiens); // nb_element = 3
```

Exemple 12: Calcul de la taille d'un tableau

e) Les chaînes de caractères

Création

Les chaînes de caractères permettent de stocker une liste de caractères dans une seule variable.

```
$chien = "caniche";
```

Exemple 13: Création d'une chaîne de caractères

Longueur

De la même façon que pour les tableaux, il est possible de connaître la longueur d'une chaîne de caractères mais, cette fois-ci, en utilisant la fonction « strlen ».

```
$chien = "caniche";
$length = strlen($chien); // length = 7
```

Exemple 14: Calcul de la longueur d'un chaîne de caractères

Concaténation

Le caractère « . » sert à concaténer deux chaînes de caractères entres elles :

```
$chien = "caniche"." nain";
```

Exemple 15: Concaténation de deux chaînes de caractères

Séquences d'échappement

Il existe des chaînes de caractères qui permettent de reproduire des séquences d'échappement comme un saut à la ligne ou encore une tabulation. Ci-dessous un tableau résumant ces chaînes et leurs significations :

Séquences	Description
\n	La séquence est remplacée par le caractère ***newline*** *(saut de ligne)*
\r	La séquence est remplacée par le caractère ***carriage-return*** *(retour à la ligne)*
\t	La séquence est remplacée par le caractère tabulation
\$	La séquence est remplacée par le caractère « $ »
\"	La séquence est remplacée par le caractère « " »
\\	La séquence est remplacée par le caractère « \ »

Ces séquences ne sont remplacées que lorsque la chaîne de caractères est affichée avec les « " » et non avec les « ' » :

```
$chiens =  "caniches";
print('les $chiens font yiyi'); // affichera :  les $chiens font yiyi
print("les $chiens font yiyi"); // affichera :  les caniches font yiyi
```

Exemple 16: Séquences d'échappement

Position d'un mot

La fonction « strpos » permet de connaître la position d'un mot dans une chaîne.

```
$chien =  "J'aime les caniches";
$pos_caniche = strpos($chien, "caniches"); // $pos_caniche = 11
```

Exemple 17: Calcul de la position d'un mot

f) Les fonctions

Une fonction est une partie de code réutilisable partout dans la page et qui permet de ne pas dupliquer du code. Un exemple de fonction, que nous avons vu précédemment, est « count ».

Définition

Le mot clé « function » permet de créer une fonction. Ci-dessous le schéma de création général :

```
function nom_de_la_fonction(paramètre1,...,paramètreX){
        instructions
}
```

Exemple 18: Schéma de création d'une fonction

Appel

Pour appeler une fonction il suffit d'utiliser son nom suivi des « () » qui contiendront la liste des paramètres :

```
// Création de la fonction
function bonjour($prenom){
        print("Hello ".$prenom);
}
// Appel
bonjour("Magali"); // affichera : Hello Magali
```

Exemple 19: Appel d'une méthode

Il est possible d'utiliser « & » de passer les arguments par référence, c'est à dire, qu'aucune copie mémoire de la variable ne sera faite mais un pointeur vers celle-ci sera utilisé.

```
// Création de la fonction
function addition($num1, $num2, $result){
	$result = $num1 + $num2;
}
// Appel
$result = 0;
addition(2, 2, &$result); // la méthode modifie $result
echo $result // result = 4;
```

Exemple 20: Passage d'un argument par référence

Dans l'exemple précédent, la fonction ne retourne rien et on l'appelle donc une méthode.

Pour retourner une valeur, il faut utiliser le mot-clé « return ».

Valeur de retour

Ci-dessous, la même fonction « bonjour » mais qui, cette fois-ci, retourne une chaîne de caractères :

```
function bonjour($prenom){
	return "Hello ".$prenom;
}
$message = bonjour("Magali");
print($message);
```

Exemple 21: Appel d'une fonction

Arguments par défaut

Il est possible de spécifier des valeurs « par défaut » pour les arguments d'une fonction. Cela permet de ne pas alourdir le code ainsi que de simplifier l'usage d'une fonction.

```
function bonjour($prenom, $mot="Hello "){
	return $mot.$prenom;
}
$message = bonjour("Magali"); // Ne bloquera pas l'exécution
print($message); // affichera: Hello Magali
```

Exemple 22: Fonction avec valeur par défaut

Appel dynamique

Il est possible d'assigner le nom d'une fonction à une chaîne de caractères et d'utiliser « () » pour appeler cette fonction.

```
function bonjour($prenom, $mot="Hello "){
	return $mot.$prenom;
}
$func = "bonjour";
$func("Magali");
```

Exemple 23: Appel dynamique

g) Les objets

On peut imaginer notre monde fait de différents objets comme des voitures, des plantes, des animaux et ainsi de suite. De la même manière, la programmation orientée objet a pour concept de définir ou implémenter un logiciel avec des objets.

Concepts

Il existe plusieurs concepts qu'il est nécessaire de maîtriser avant d'aller plus loin.

- Classe : c'est un type de donnée défini par le programmeur et qui contient des fonctions et attributs locaux. Il est important de voir la classe comme un moule qui va permettre de créer plusieurs objets de la même « forme » que l'on appellera **instance**.
- Objet : c'est une **instance** individuelle d'une structure de données définies par une classe. Chaque instance dispose des même attributs mais qui ont chacun leurs valeurs.
- Attributs : les attributs sont définis à l'intérieur d'une classe et leurs valeurs sont positionnées à l'intérieur d'une instance et peuvent être utilisées par les fonctions membres.
- Fonctions : les fonctions membres sont définies à l'intérieur d'une classe et permettent généralement de manipuler les instances.
- Héritage : l'héritage est la propagation des attributs et fonctions membres d'une classe parent vers une classe fille. La classe parent regroupe des propriétés communes et les classes filles gèrent les spécificités.
- Classe parent : c'est une classe dont hérite une autre classe. Elle est également appelée classe de base ou ***super*** classe.
- Classe fille : c'est une classe qui hérite d'un autre classe. Elle est également appelée sous-classe ou classe dérivée.
- Polymorphisme : c'est la capacité d'une fonction à prendre des paramètres de natures différentes. Le nom de la fonction reste le même mais le travail effectué diffère selon l'objet d'entrée.
- Surcharge : (sorte de polymorphisme) qui permet de redéfinir le contenu d'une fonction au niveau des classes filles.
- Abstraction : correspond à une représentation des données qui est cachée (abstraite).

- Encapsulation : concept qui voit les attributs et fonctions membres comme formant un tout qui est l'objet.
- Constructeur : fonction spéciale qui est appelée de manière implicite pour la création d'un objet.
- Destructeur : fonction spéciale qui est appelée de manière implicite lorsqu'un objet passe en dehors du périmètre d'exécution du code.

Définition d'une classe

La définition d'un classe se fait grâce au mot « class » de la manière suivante :

```
<?php
class className{
        var $var1;
        var $VAR2 = "constante";

        function funcname ($arg1, $arg2) {
                [...]
        }
        [..]
}
?>
```

Exemple 24: Définition d'une classe

On peut remarquer :

- le mot-clé « class » qui est suivi du nom de la classe ;
- les accolades ouvrantes et fermantes qui permettent d'encapsuler la suite d'attributs et fonctions membres ;
- les attributs qui commencent par le mot clé « var » suivi du conventionnel « $ » et qui peuvent avoir une valeur (constante) ;

1. les fonctions membres sont définies de la même façon que les fonctions traditionnelles mais ne peuvent être utilisées qu'au sein de la classe.

Ci-dessous un exemple d'implémentation d'une classe chien :

```
<?php
class Chien{
	/* Constantes */
	public static $SMALL_BARK = "yiyi";
	public static $BIG_BARK = "WAOUF";

	/* Attributs */
	private $nom;
	private $poids;

	/* Constructeur */
	function __construct($nom, $poids){
		$this->nom = $nom;
		$this->poids = $poids;
	}

	/* Fonctions membres */
	function setNom($nom){
		$this->nom = $nom;
	}
	function getNom(){
		return $this->nom;
	}
	function setPoids($poids){
		$this->poids = $poids;
	}
	function getPoids(){
		return $this->poids;
	}
	function aboiement(){
		if($this->poids < 5){
			retun self::SMALL_BARK;
		}else {
			return self::BIG_BARK;
		}
	}
}
```

Exemple 25: Classe Chien

On peut remarquer que :

- les attributs constants (***static***) sont en majuscule et déclarés en début de classe ;
- les attributs non constants sont déclarés après les constantes ;
- les mots-clés « public » et « private » définissent un périmètre ;
- le constructeur est placé directement après la déclaration des variables ;
- il existe des fonctions qui permettent d'affecter une valeur à un attribut (setter) et d'autres qui permettent de consulter ces attributs (getter) ;
- les getters et setters sont positionnés directement en dessous du constructeur ;
- le mot-clé « $this » permet de faire référence aux attributs de la classe lorsqu'il y a ambiguïté ;

- les autres fonctions sont déclarées en fin de classe.
- le mot-clé « static » permet d'accéder à des attributs sans nécessiter une instance de la classe (***self::***) ;

Instancier une classe

Une fois la classe définie, il est possible de créer autant d'instances que nécessaires :

```
<?php
$caniche = new Chien("caniche", 4);
$cocker = new Chien("cocker", 9);
$bulldog = new Chien("bulldog", 12);
?>
```

Exemple 26: Instanciation d'une classe

Appel d'un attribut ou d'une fonction membre

Pour appeler une fonction membre il suffit d'utiliser « -> »

```
<?php
$caniche = new Chien("caniche", 4);
echo "Les chien qui pèsent $caniche->getPoids() aboie de la sorte : "
echo $caniche->aboiement ;
?>
```

Exemple 27: Appel d'attribut et fonction membre

On peut remarquer qu'il n'est pas possible d'accéder directement à l'attribut poids car il est déclaré de manière privé et c'est pourquoi il faut utiliser le ***setter*** approprié.

5. Les opérateurs

Pour les exemples suivants, nous allons utiliser deux variables a et b qui valent respectivement 6 et 2.

a) Arithmétiques

Ci-dessous un tableau résumant les opérateurs arithmétiques utilisés en PHP :

Opérateur	Description	Exemple
+	addition	a + b = 8
-	soustraction	a – b = 4
*	multiplication	a * b = 12
/	division	a / b = 3
%	modulo	a % b = 0
++	incrément	a++ = 7
--	décrément	a-- = 5

b) Comparaisons

Ci-dessous un tableau résumant les opérateurs de comparaison utilisés en PHP, les résultats sont booléens (vrai / faux) :

Opérateur	Description	Exemple
==	égalité	a == b = faux
!=	différent	a != b = vrai
>	supérieur	a > b = vrai
<	inférieur	a < b = faux
>=	supérieur ou égal	a >= b = vrai
<=	inférieur ou égal	a <= b = faux

c) Logiques

Ci-dessous un tableau résumant les opérateurs de comparaison utilisés en PHP, les résultats sont booléens (vrai / faux) :

Opérateur	Description	Exemple
and	ET logique	(a && b) = vrai
or	OU logique	(a || b) = vrai

&&	ET logique	(a && b) = vrai
||	OU logique	(a || b) = vrai
!	Négation logique	!(a && b) = faux

d) Affectations

Ci-dessous un tableau résumant les opérateurs d'affectation utilisés en PHP, nous utiliserons une troisième variable c :

Opérateur	Description	Exemple
=	affectation	c = (a + b) équivaut à c = 8
+=	addition ET affectation	c += a équivaut à c = c + a
-=	soustraction ET affectation	c -= a équivaut à c = c - a
*=	multiplication ET affectation	c *= a équivaut à c = c * a
/=	division ET affectation	c /= a équivaut à c = c / a
%=	modulo ET affectation	c %= a équivaut à c = c % a

e) Conditionnel

L'opérateur conditionnel « ? » évalue dans un premier temps la condition donnée de manière booléenne puis, en fonction du résultat, choisit entre deux instructions :

- ((a > b) ? 10 : 20) équivaut à 10 ;
- ((a < b) ? 10 : 20) équivaut à 20.

6. Boucles conditionnelles et itératives

a) If

La boucle « if » permet de faire des choix entre deux possibilités ou plus.

If

C'est la forme la plus simple de créer une instruction conditionnelle :

```
if (condition){
        Instructions à exécuter si la condition est vérifiée
}
```

Exemple 28: Boucle if

If / else

Cette boucle permet un contrôle plus fin en autorisant l'exécution d'une instruction si la condition n'est pas vérifiée :

```
$age = 20 ;
if ($age > 18){
	print("Vous êtes apte à conduire !");
}else{
	print("Patienter encore un peu avant de conduire...");
}
```

Exemple 29: Boucle if / else

If / else if

C'est la forme la plus évoluée de boucle pour créer des expressions conditionnelles. Elle offre une très grande souplesse en permettant la gestion de choix multiples :

```
$age = 20 ;
if ($age > 16){
	print("Vous êtes apte à conduire un scooter !");
}else if($age > 18){
	print("Vous êtes apte à conduire une voiture !");
}else{
	print("Patienter encore un peu avant de conduire...");
}
```

Exemple 30: Boucle if / else if

b) For

La boucle « for » permet de répéter une instruction. Elle se décompose en trois parties :

```
for (initialisation; condition / test; instruction itérative){
	Instructions
}
```

Exemple 31: Schéma d'une boucle « for »

- L'initialisation permet de mettre en place une variable qui fait office de counter (eg. « var i = 0 ») ;
- la condition ou test s'il échoue permet d'arrêter l'itération (eg. I < 5) ;
- l'instruction itérative permet d'incrémenter ou décrémenter le compteur (eg. i++) ;

Ci-dessous un exemple de boucle « for » :

```
$chiens = ["caniche", "cocker", "bulldog"];
$count = count($chiens);
for($i=0; i < $count; i++){
	$chien = chiens[$i];
	echo($chien);
}
```

Exemple 32: Boucle « for »

c) Foreach

La boucle « foreach » permet d'itérer sur les valeurs d'un tableau. Si on reprend l'exemple précédent, la boucle « foreach » permet d'itérer sur les valeurs :

```
$chiens = ["caniche", "cocker", "bulldog"];
foreach($chien as chiens){
        print($chien);
}
```

Exemple 33: Boucle « foreach »

d) While

La boucle « while » est le moyen le plus simple d'itérer. Elle suit le schéma suivant :

```
while (condition){
    Instructions
}
```

Exemple 34: Schéma de la boucle « while »

Le but d'une telle boucle est de répéter le bloc d'instructions tant que la condition est vérifiée :

```
$chiens = ["caniche", "cocker", "bulldog"];
$i = 0;
$count = count($chiens);
while($i < $count){
        $chien = $chiens[$i];
        print($chien);
        $i++ ;
}
```

Exemple 35: Boucle « while »

e) Do...while

La boucle « do...while » permet de placer la vérification de la condition en fin de boucle, ce qui permet d'exécuter au moins une fois les instructions :

```
do{
    Instructions
}while (condition) ;
```

Exemple 36: Schéma de la boucle « do...while »

Veuillez noter le « ; » en fin de boucle.

7. Contrôle des boucles

PHP fournit les outils nécessaires pour contrôler les itérations car il se peut qu'il y ait une situation où il soit nécessaire de quitter une boucle sans avoir atteint la fin de l'itération.

a) Break

Le mot-clé break permet de quitter une boucle avant d'avoir atteint la fin de celle-ci. Cela placera le curseur d'exécution du code à la fin de la boucle, c'est à dire après l'accolade fermante « } ».

```
$chiens = ["caniche", "cocker", "bulldog"];
foreach($chiens as $chien) {
        $chien = $chiens[$i];
        if($chien == "cocker"){
                break;
        }
        echo "$chien";
}
```

Exemple 37: Utilisation de « break »

b) Continue

Le mot clé « continue » permet de « sauter » une itération en passant directement à la suivante :

```
$chiens = ["caniche", "cocker", "bulldog"];
foreach($chiens as $chien) {
        $chien = $chiens[$i];
        if($chien == "cocker"){
                continue;
        }
        echo "$chien";
}
```

Exemple 38: Utilisation de « continue »

8. La session

Le protocole HTTP est utilisé entre le serveur et le navigateur pour communiquer.
L'inconvénient de HTTP est qu'il ne mémorise pas les états entre chaque page, on dit qu'il est « ***stateless*** ».
Les sessions permettent de pallier ce manque en mémorisant les variables dans un fichier temporaire.

a) Démarrer une session

La fonction « session_start » permet de démarrer une session.

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

Exemple 39: Démarrage d'un session

Le démarrage d'une sesison implique que :

- PHP crée une chaîne aléatoire de 32 caractères qui va permettre d'identifier de manière unique la session en cours ;
- Un cookie appelé ***PHPSESSID*** est automatiquement créé et envoyé au navigateur client pour stocker l'identifiant de session ;
- Un fichier est automatiquement créé sur le serveur et qui va permettre de stocker les données de la session.

b) Terminer une session

Une session se termine lorsque l'utilisateur ferme son navigateur, quitte le site ou plus généralement après une période d'inactivité de 30 minutes.
Pour fermer volontairement une session il faut utiliser la fonction « session_destroy() ».

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

Exemple 40: arrêt d'une session

c) *Le tableau de session*

Il existent un tableau, « $_SESSION », qui se crée et se détruit automatiquement avec la session et qui permet de stocker les informations de session .

```
if( isset( $_SESSION['var'] ) )
{
	$var = $_SESSION['var']; // On récupère la variable
}
else
{
	$_SESSION['var'] = $var; // On positionne la variable
}
```

Exemple 41: Utilisation du tableau associatif *$_SESSION*

9. Les formulaires

a) *La méthode GET*

La méthode ***GET*** envoie les informations utilisateur en les concaténant à la suite de l'URL. Les paramètres sont séparés par « ? » de l'URL de base et entre eux par « & » :

```
http://www.epsi.fr/index.php?name1=value1&name2=value2
```

Exemple 42: Concaténation avec ***GET***

La méthode ***GET*** :

- produit une URL très longue, incompréhensible et qui va apparaître dans la barre d'adresse du navigateur client ;
- autorise seulement 1024 caractères ;
- ne permet pas de faire transiter des mots de passe (car ils sont en clairs) ;
- PHP fournit le tableau associatif « **$_GET »** pour accéder aux informations envoyées avec la méthode ***GET***.

```
if( isset( $_GET['var'] ) )
{
	$var = $_GET['var']; // On récupère la variable
}
else
{
	$_GET['var'] = $var; // On positionne la variable
}
```

Exemple 43: Utilisation du tableau associatif ***$_GET***

b) *La méthode POST*

La méthode ***POST*** envoie les informations utilisateur en les encodant dans le ***header*** HTTP. Cela permet de :

- ne plus avoir la limite des 1024 caractères ;
- de faire transiter des données ***ASCII*** et binaires ;
- faire transiter de manière sécurisée des informations grâce au protocole HTTPS ;
- PHP fournit le tableau associatif « **$_POST »** pour accéder aux informations envoyées avec la méthode ***POST***.

```
if( isset( $_POST['var'] ) )
{
        $var = $_POST['var']; // On récupère la variable
}
else
{
        $_POST['var'] = $var; // On positionne la variable
}
```

Exemple 44: Utilisation du tableau associatif ***$_POST***

c) *Le tableau $_REQUEST*

Le tableau associatif « $_REQUEST » permet de récupérer le contenu des tableaux « $_GET », « $_POST » et également « $_COOKIE ».

```
<?php
 if(isset($_REQUEST["name"])) {
   echo "Bonjour ". $_REQUEST['name']. "<br />";
   exit();
 }
?>
<html>
        <body>
                <form action="<?php $_PHP_SELF ?>" method="POST">
                        Name: <input type="text" name="name" />
                        <input type="submit" />
                </form>
        </body>
</html>
```

Exemple 45: *Utilisation du tableau associatif $_REQUEST*

On peut noter l'utilisation de la constante « $_PHP_SELF » qui permet d'obtenir le nom du fichier courant.

10. Les cookies

a) Création

La fonction « setcookie » permet de créer un cookie grâce à maximum six paramètres :

```
setcookie(name, value, expire, path, domain, security);
```

Exemple 46: Création d'un cookie

- « Name » contient le nom du cookie ;
- « Expiry » contient la date à laquelle le cookie expire. Si laissé vide, le cookie expirera à la fin de la visite du site ;
- « Domain » contient le domaine du site ;
- « Path » contient le chemin de la page qui a créé le cookie. Si laissé vide, le cookie pourra être récupéré de n'importe quelle page ;
- « Security » : si cet attribut contient la valeur « 1 » alors le cookie pourra être récupéré uniquement depuis un serveur sécurisé ;
- « Value » permet de stocker une valeur.

```
<?php
  setcookie("nom", "Magali", time()+3600, "/","", 0);
?>
```

Exemple 47: Création d'un cookie

b) Accès

PHP propose d'accéder aux cookies avec les tableaux associatifs « $_COOKIE » et « $HTTP_COOKIE_VARS ».

```
<?php
if(isset($_COOKIE["nom"])){
        echo "Bonjour " . $_COOKIE["nom"] . "<br/>";
}else
        echo "Bonjour bel(le) inconnu(e)..." . "<br/>";
}
?>
```

c) Effacement

Pour effacer un cookie, il suffit de le recréer avec une date dans le passé :

```
<?php
setcookie("nom", "", time()- 60, "/","", 0);
?>
```

Exemple 48: Effacement d'un cookie

11. Gestion des fenêtres

a) *Les redirections*

Les redirections se font en positionnant l'attribut « location » du ***header*** HTTP grâce à la fonction « header » :

```
header("Location:"http://www.epsi.fr");
exit(); // Après la redirection on stop l'exécution de la page
...
```

Exemple 49: Redirection

b) *Boîte de téléchargement*

Il est possible d'ouvrir une boîte de dialogue pour le téléchargement de fichier. On utilise pour cela toujours la fonction « header ». Ci-dessous un exemple qui permet d'afficher une boîte de dialogue pour un fichier PDF :

```
<?php
header('Content-type: application/pdf');
header('Content-Disposition:attachment;filename="mon_fichier.pdf"');

readfile('le_fichier.pdf');
?>
```

Exemple 50: Boîte de téléchargement

12. Best practices

Chaque entreprise possède ses propres standards de programmation, cependant il existe une base essentielle et commune à tous:**les bonnes pratiques** !
Voici quelques raisons d'utiliser de telles pratiques :

- La majeure partie du temps, vous êtes amenés à travailler en binômes et il est important que votre homologue puisse relire le code que vous produisez sans avoir à le déchiffrer ;
- La simplicité et la clarté qui émanent d'un code consistent permettent d'éviter les erreurs communes ;
- Si vous relisez votre code après un certain temps, ce code vous paraît simple ;
- Cela fait partie des standards de l'industrie de suivre les bonnes pratiques car elles garantissent une certaine qualité.

Il existe certaines lignes directrices qu'il est intéressant de suivre lorsque l'on code en PHP :

- **Indentation**

  Utilisez une indentation de quatre espaces et surtout pas de tabulation car elles ont un rendu différent selon les systèmes.

- **Longueur de ligne**

  Une longueur de ligne de 75 à 85 caractères permet une bonne lisibilité.

- **Structure de contrôle**

  Cela concerne toutes les boucles (if, for, while, ...). Les contrôles de structure doivent avoir un espace entre le mot de contrôle et la première parenthèse, pour les différencier des appels de fonctions. Il est fortement recommandé d'utiliser les accolades mais quand, techniquement, elles ne sont pas nécessaires.

```
if ((condition1) || (condition2)) {
   action1;
} elseif ((condition3) && (condition4)) {
   action2;
} else {
   action par défaut;
}
```

Exemple 51: Syntaxe d'une structure de contrôle

- **Appel de fonction**

  Les fonctions doivent être appelées avec aucun espace entre le nom de la fonction et la parenthèse ouvrante, un espace entre virgules et paramètres, et aucun espace entre le dernier paramètre et la dernière parenthèse.

```
$var = $func($arg1, $arg2, $arg3);
```

Exemple 52: Syntaxe d'un appel de fonction

- **Commentaires**

  L'utilisation des « */ /* » ainsi que les « // » est encouragé alors que l'utilisation des « # » est découragée.

- **Tag PHP**

  L'utilisation des « <?php ?> » est recommandée alors que l'utilisation de « <? ?> » est découragée. Le premier tag est le plus utilisé et le plus portable.

Il existe encore bien des points qui méritent d'être abordés mais les principaux sont exposés. L'objectif d'un standard est de vous permettre de produire un code de qualité et compréhensible par le plus grand nombre.

13. Index des exemples

Index des exemples