php

2


```
 * @var boolean
 */
define('PSI_INTERNAL_XML', false);

if (version_compare("5.2", PHP_V
    die("PHP 5.2 or greater is r

}
if (!extension_loaded("pcre"))
    die("phpSysInfo requires th
        properly.");

}

require_once APP_ROOT.'/includes/autoloader.inc.php';

// Load configuration
require_once APP_ROOT.'/config.php';

if (!defined('PSI_CONFIG_FILE') || !defined('PSI_DEBUG')) {
    $tpl = new Template("/templates/html/error_config.html");
    echo $tpl->fetch();
    die();
                                    javascript
                                    strtolower(
```

Messmer Lucie

Dans ce cours :

- Conditions
- Boucles
- Inclusions
- Fonctions
- Variables et portées

Exercice PHP2

Conditions et Boucles

Structure de contrôle

IF, la condition

If, ou si en anglais, est l'instruction conditionnelle la plus utilisée.
Elle permet d'éxecuter une instruction si une expression booléenne est vraie.

Syntaxe :

```
if (ma condition){
 instructions à appliquer ;
}
if (booléen) {
      instruction 1; instruction 2; instruction 3;
}
```

L'instruction IF ELSE

Permet d'ajouter un bloc d'instructions à exécuter dans le cas où la condition booléene est fausse.

Syntaxe :

```
if (ma condition) {
        instruction 1; instruction 2;
} else {
        instruction à appliquer si ma condition n'est pas respectée;
}
```

IF et ELSEIF

Cette variante permet d'ajouter une nouvelle condition à tester si la premiere n'est pas vrai. Le ELSE de fin executant les instructions si aucune des conditions n'est vrai. Il est possible de créer ainsi un nombre infini de tests, cependant la structure deviendra vite lourde.

Syntaxe :

```
if (boolean1)    {
 instruction si ma condition 1 est vraie;
} elseif (boolean2)    {
 instruction si ma condition 1 n'est pas vrai mais ma condition 2 l'est :
} else {
 instruction dans le cas où aucune des deux conditions si dessus est vraie;
}
```

exemple :

```
//IF
if ($a > $b){
 echo '$a est supérieur a $b';
}
//ELSE IF
if ($a > $b){
 echo '$a est supérieur a $b';
} elseif ($a = $b) {
 echo '$a est égal a $b';
} else {
 echo '$a est inférieur à $b' ;
}
```

L'operateur ternaire ?

L'operateur ternaire ? permet une écriture condensée d'un test.
On passe la condition en premier, suivi de l'instruction dans le cas où la condition est vrai, suivi de l'instruction si la condition est fausse.
Pour des tests simples on gagne en place mais pas forcement en lisibilite.

Syntaxe :

(condition) ? instruction si vrai : instruction si faux ;

exemple :

```
// Est-ce que $a = $b ? si oui, afficher '$a est egal a $b' et si non '$a est different de $b'

($a == $b)    ? print('$a est égal a $b')    : print('$a est différent de $b');

// Variante

echo ($a == $b)    ?    '$a est egal a $b'    :    '$a est different de $b';
```

La comparaison avec SWITCH

Permet de faire plusieurs tests de valeurs sur le contenu d'**une même variable**.

Syntaxe :

```
 switch (expression) {

        case valeur1 :
                liste instructions;
                break;
        case valeur2 :
                liste instructions;
                break;
        default :
                liste instructions;
}
```

case definit les points d'entrée, les valeurs possibles de l'expression.

break definit les points de sortie.

default contient le bloc d'instruction à exécuter si aucun des tests de comparaison précédent n'a été concluant.

```
switch ($a){
        case "oui":
               echo "il est d'accord";
               break;
        case "non":
               echo "il n'est pas d'accord";
               break;
    }
```

Les boucles

La boucle FOR

La boucle **for** permet de réaliser une instruction de façon répetée (avec un nombre d'itérations connu).

- une condition initiale est faite en début de boucle
- une vérification de poursuite de boucle est faite avant de commencer une nouvelle répétition
- une instruction finale est faite en fin de chaque passage dans la boucle

Syntaxe :

```
for (condition initiale; condition; instruction final) {
      liste instructions;
}
```

exemple :

```
for ($i = 0; $i < 10; $i++)    {
      echo $i; // va afficher a la suite les chiffres de 0 à 9
}
```

La boucle WHILE

L'instruction while exécute une liste d'instructions tant qu'une condition est vraie.

Syntaxe :

```
while(condition) {
        liste instructions;
}
```

La boucle DO WHILE

Fonctionne de manière similaire à la boucle while, à la seule différence que même si la condition n'est pas vraie, la première itération sera effectuée.

Syntaxe :

```
do {
      expression à exécuter ;
} while (conditions à respecter);
```

exemple :

```
$i = 0;
while ($i < 6){
  echo $i++; // va afficher a la suite les chiffres de 0 a 5
}
```

L'instruction FOREACH

C'est une boucle FOR dediée à parcourir un tableau. Elle exécute un code pour chaque ligne d'un tableau. Syntaxe :

```
foreach(tableau as valeur) {
      liste d'instructions;
}
```

exemple :

```
$fruits = ['orange', 'banane', 'clémentine', 'pomme']; foreach
($fruits as $fruit) {
      echo $fruit . " a été ajouté à la liste de courses. </br>";
}
// va afficher :   orange a été ajouté à la liste de courses.
                   banane a été ajouté à la liste de courses.
                   clémentine a été ajouté à la liste de courses.
                   pomme a été ajouté à la liste de courses.
```

Variante de FOREACH qui permet également d'extraire les clefs.

Syntaxe :

```
foreach(tableau as clef => valeur) { liste instructions;}
```

exemple :

```
$fruits = ['orange', 'banane', 'clémentine', 'pomme'];

foreach ($fruits as $clef => $valeur) {
      echo 'Clef '.$clef.' => '.$valeur.'\; <br>';
}
```

```
// va afficher :
Clef 0 => orange;
Clef 1 => banane;
Clef 2 => clémentine;
Clef 3 => pomme;
```

L'instruction CONTINUE

L'instruction CONTINUE permet de mettre fin au traitement de la boucle courante et de passer au début de l'itération suivante.

exemple :

Dans cet exemple on gère le cas de la division par zéro pour eviter une erreur.

```
$x=1;
while ($x <= 10){
  if ($x == 7){
    echo 'Division par zero !';
    $x++;
    continue;
  }
  $result = 1/($x-7);
  echo $result;
  $x++;
}
```

L'instruction BREAK

Elle permet de mettre fin au traitement d'une boucle et de passer à la suite d'un script.

exemple :

Il s'agit du meme exemple que precedent mais cette fois-ci, dès que l'on rencontre le cas qui va engendrer l'erreur, on sort de la boucle.

```
$x=1;
while ($x <= 10){
    if ($x == 7){
       echo 'Fin du traitement car Division par zéro';
       break;
     }
     $result = 1/($x-7);
     echo $result;
     $x++;
 }
```

L'instruction EXIT

L'instruction EXIT provoque la fin du script où que l'on soit dans le code. Le serveur n'envoie plus d'information au navigateur.

Les inclusions de code

PHP permet d'inclure des fichiers de code et donc de partager un même code par plusieurs scripts.
Ceci est utile entre autres pour :

- la declaration de constantes
- la declaration de fonctions
- la declaration de classes...

Deux types d'inclusions sont possibles :
include() fait une inclusion permettant une erreur sans interrompre le script.
require() fait une inclusion générant une erreur fatale en cas de problème de chargement.

Afin d'éviter les inclusions multiples, il existe :
include_once() fait une inclusion permettant une erreur sans interrompre le script.
require_once() fait une inclusion générant une erreur fatale en cas de problème de chargement.
Dans le cas où la ressource n'a pas encore été chargée, ces fonctions se comportent comme include() et require().
Dans le cas contraire elles ne rechargeront pas le contenu de la ressource.

Les fonctions

Une fonction est un bloc de code auquel est attribué un nom suivi d'arguments.
Le passage de paramètres entre le programme principal et la fonction se fait par les arguments positionnés lors de l'appel.
Les arguments sont des noms de variables placés entre **()** (parenthèses) et séparés par des **,** (virgules). Tout type de variable peut être passé en argument : nombres, chaînes, tableau, objets...
En PHP, les fonctions n'ont pas besoin d'être définies avant d'être utilisées. Elles sont appelées grâce au mot-clef **function** et prennent généralement des arguments, mais ils ne sont pas nécessaires.

Syntaxe de declaration :

```
function ma_fonction ($arg_1, $arg_2, ..., $arg_n){
	// Corps de ma fonction...
	return $valeur_de_retour ;
}
```

exemple :

```
function bonjour($prenom) {
	echo 'Bonjour '.$prenom;
}
bonjour('Aramis'); // va afficher : Bonjour Aramis
```

La fonction retournera eventuellement une valeur par le mot cle return. (Si l'on souhaite retourner plusieurs valeurs il est tout a fait possible de retourner un tableau les contenant.)

Attention : return met également fin à la fonction.

exemple :

```
function au_carre($val) {
   return $val*$val;
}
$resultat = au_carre(2); // va affecter le resultat de la fonction à $resultat
```

Variables et portée

Variables globales

Selon les langages, les variables declarées dans une fonction sont continues dans la fonction ou visibles dans l'ensemble du programme. Le comportement par défaut en PHP est de ne donner qu'une visibilité locale des variables. Ce comportement est modifiable par la déclaration global des variables.

Remarque : *En programmation il est sain d'eviter de declarer des variables globales car elles nuisent a la portabilité et permettent des collisions de nommage avec le script appelant.*

exemple :

```
$a = 1;
function test_portee($val) {
    global $a;
    $a = $a+$val;
}
test_portee(4);
echo $a; // va afficher : 5
$a = 1;
function test_portee($val) {
    $a = $a+$val;
}
test_portee(4);
echo $a; // va afficher : 1 (et une alerte de variable non existante...)
```

Variables par références

Il est possible de passer une variable par référence à une fonction, de manière à ce que celle-ci puisse la modifier.
Cela permet notamment d'éviter l'utilisation de variable globale ou d'une éventuelle valeur de retour.
Il suffit de faire précéder la variable de & (esperluette, aussi appellee "et" commercial) lors du passage d'argument a la fonction.

exemple :

```
function par_ref(&$val) {
    $val++;
}
$a = 5;
par_ref($a);
echo $a; // va afficher : 6
```

Valeurs d'arguments par defaut

Il est possible d'appeler une fonction sans indiquer tous ses arguments. Ceci est permis par l'utilisation de valeurs par défaut lors de la déclaration de la fonction. Il suffit d'attribuer une valeur avec le signe = dans la liste d'arguments.

- Les arguments optionnels devront se situer en fin de liste.
- Le valeurs par défaut devront être des constantes.

exemple :

```
function bonjour($qui='inconnu') { echo 'Bonjour '.$qui; }

bonjour(); // va afficher : Bonjour inconnu
bonjour('Porter'); // va afficher : Bonjour Porter
```

Récursivité

exemple :

```
function exponentielle($val, $puiss) {
        if ($puiss == 1)    {
                return $val;
        } else {
                return $val * exponentielle($val, $puiss-1);
        }
}
$v=5;
echo exponentielle($v, 3); // va afficher : 125
```