COURS 02 – COMPLÉMENTS : PRÉSENTATION DU CODE ET ANALYSE DES ERREURS

ANALYSE DES PRINCIPALES ERREURS

Expliquer et corriger éventuellement les erreurs suivantes. Proposer une démarche pour résoudre les erreurs:

1 – Accès fichier

```
sio@ThinkPad:~/Bureau/php$ php dep.php
Could not open input file: dep.php
```

```
sio@ThinkPad:~/Bureau/php$ php test.php
PHP Fatal error:  Call to undefined function readlin() in /home/sio/Bureau/php/test.php
on line 2
```

2 – Vérification Code

```
$age = readline("entrer votre age : ");
```

```
if ($age=26 )
  echo ("vous avez 26 ans \n");
```

```
if ($age<26 ||  >60)
  echo ("tarif reduit \n");
else
  echo ("plein tarif \n");
```

```
if ($age<26 ||  $age>60);
  echo ("tarif reduit \n");
else
  echo ("plein tarif \n");
```

```
if ($age<26 && $age >60)
  echo ("tarif reduit \n");
else
  echo ("plein tarif \n");
```

```
if ($age<26 ||  >60)
  echo( "vous avez moins de 26 ans et plus de 60 ans \n");
  echo ("tarif reduit \n");
else
  echo ("plein tarif \n");
```

```
if ($age<18)
   echo ("tarif reduit \n");
else ($age>18)
  echo ("plein tarif \n");
```

Travail à faire : Pour chaque test vérifier si la syntaxe et/ou la logique sont correctes. Proposer une solution dans le cas contraire.

Faut-il mettre un ET? Faut-il mettre un OU ?

Dans une condition composée employant à la fois des opérateurs ET et des opérateurs OU, la présence de parenthèses possède une influence sur le résultat, tout comme dans le cas d'une expression numérique comportant des multiplications et des additions.

Toute structure de test requérant une condition composée faisant intervenir l'opérateur ET peut être exprimée de manière équivalente avec un opérateur OU, et réciproquement.

Exemple 1: un tarif réduit est accordé aux personnes de moins de 25 ans et de plus de 65 ans.
On aurait tendance à vouloir traduire cette condition de cette manière :

if ($age < 25 AND $age > 65)

En écrivant ce code, vous demandez si la variable $age peut être à la fois inférieure à 25 ET supérieure à 65, cette condition sera toujours fausse aucun nombre ne peut remplir cette condition.
Dans cet exemple, il faut mettre un OU : **if ($age < 25 OR $age > 65)** ainsi la condition sera VRAIE si l'âge est inférieur à 25 **OU** supérieur à 65.
Quand on utilise un **ET**, il faut que chaque condition soit remplie pour que l'expression soit VRAIE alors qu'avec un **OU**, il suffit qu'une seule des conditions soient remplies.

Exemple 2 : Concevoir un programme qui affiche le signe du produit de deux nombres (positif ou négatif) saisis et ceci sans calculer leur produit.
On peut traduire cela de la manière suivante, le produit sera positif si les deux nombres sont positifs OU alors si les deux nombres sont négatifs

if (($nb1 > 0 AND $nb2 > 0) OR ($nb1 < 0 AND $nb2 < 0))

condition 1 *OU* *condition 2*

switch : extrait du manuel PHP

L'instruction *switch* équivaut à une série d'instructions *if*. En de nombreuses occasions, vous aurez besoin de comparer la même variable (ou expression) avec un grand nombre de valeurs différentes, et d'exécuter différentes parties de code suivant la valeur à laquelle elle est égale. C'est exactement à cela que sert l'instruction *switch*.

Les deux exemples suivants sont deux manières différentes d'écrire la même chose, l'une en utilisant une séries de *if*, et l'autre en utilisant l'instruction *switch* :

```
if($i == 0){
    echo "i égal 0";
}
elseif($i == 1) {
    echo "i égal 1";
}
elseif ($i == 2) {
    echo "i égal 2";
}
else {
      echo 'i n'est pas égal à O ou 1 ou 2';
}
```

```
switch ($i) {
      case 0:
         echo "i égal 0";
         break;
      case 1:
         echo "i égal 1";
         break;
      case 2:
         echo "i égal 2";
         break;
      default :
         echo 'i n'est pas égal à O ou 1 ou 2' ;
         break;
}
```

Les différents 'case' testent la valeur, et exécutent le code contenu entre le 'case' en question et le 'break'.

L'instruction contenue dans la clause **default** est l'instruction à exécuter par défaut lorsque la variable $i ne prend aucune des valeurs définies dans les différents 'case'.
Attention avec un switch vous ne pouvez pas écrire **case ($i > 0)**, on teste uniquement la valeur de la variable.

PROPRETÉ DU CODE

Rechercher et corriger ses erreurs est une tâche difficile qui demande beaucoup de temps, souvent plus que l'écriture initiale du programme. Pour avoir un programme sans erreurs, le pire que l'on puisse faire est donc d'écrire son programme le plus vite possible, pour ensuite le tester et chercher les erreurs une par une. Comme on dit souvent, mieux vaut prévenir que guérir : il vaut mieux prendre son temps lorsque l'on écrit un programme et faire bien attention à tout ce que l'on écrit, plutôt que de l'écrire rapidement, pour corriger les erreurs lorsqu'elles apparaîtront.

Prendre son temps pour faire les choses bien est la base indispensable pour écrire des programmes avec peu d'erreurs et économiser le temps de leur recherche et correction. Cela ne suffit cependant pas et un certain nombre d'habitudes doivent être prises dans la manière dont vous programmez, pour réduire le nombre d'erreurs au minimum. Nous en verrons plusieurs tout au long de ce cours, en voici quelques unes :

- **Réfléchissez avant de coder.**

  Si vous vous précipitez sur le clavier dès que vous avez une idée de programme, vous avez toutes les chances de partir dans une mauvaise direction. Une fois que vous commencez à taper votre code, vous vous concentrez sur son écriture et perdez du recul sur l'ensemble de votre programme. Prenez plutôt une feuille de papier et un crayon et réfléchissez à la manière dont vous allez organiser votre programme.

- **Faites simple.**

  Ceci est probablement la règle la plus importante en programmation : faire le plus simple possible. **Ne soyez jamais fier d'avoir écrit des milliers de lignes de code**. Pour un même résultat, un programme court et simple a beaucoup plus de valeur qu'un long programme qui fait la même chose.

- **Utilisez des identifiants explicites.**

  Lorsque vous déclarez une variable, prenez le temps de rechercher l'identifiant le plus adapté à ce qu'elle va contenir. Une instruction comme "`while ($ligne < $nblignes)`" est bien plus facile à comprendre que si l'on avait écrit "`while ($toto < $titi)`". L'objectif est que votre programme se lise quasiment comme du français, et se comprenne sans effort. Si vous devez écrire une variable qui va servir de compteur de colonnes, n'appelez pas votre variable "compteur", qui est assez vague, mais plutôt "nbColonne".

- **Fixez-vous une convention d'écriture.**

  Le langage PHP vous laisse une grande liberté dans la manière dont vous présentez votre programme. Vous pouvez ajouter des espaces, tabulations, retours à la ligne n'importe où sauf au milieu d'un mot, sans que cela change son fonctionnement. Il est cependant important de prendre des habitudes et d'écrire ses programmes toujours de la même manière. Cela les rend plus faciles à relire et permet de voir les erreurs plus facilement.

- **Une instruction par ligne :** c'est une règle systématique qui facilite la lecture et réduit les risques d'erreurs. Une ligne contient une instruction et une seule. De même, les lignes contenant des accolades de début et de fin de bloc ne contiennent rien d'autre.

- **Indentation du code :** un programme est constitué de blocs imbriqués les uns dans les autres. Pour bien mettre en évidence cette structure, les instructions d'un bloc sont décalées d'un certain nombre de caractères vers la droite , par rapport à celles du bloc qui le contient. Ceci permet de voir au premier coup d'oeil où commence et se termine chaque bloc d'instructions :

```
$nombre = 0;
echo("Instruction entre accolades, donc indentée de 3 caractères\n");
  while ($nombre < 5)
  {
    echo("Instruction indentée de six caractères\n");
    if ($nombre < 2) {
      echo("Celle-ci l'est encore plus (9)\n");
    }
    else  {
      echo("On indente, qu'il y ait ou non des accolades.\n");
    }
    $nombre++;
  }
  echo("Après un bloc, revient au niveau d'avant le bloc.\n");
```

  De nombreux éditeurs de texte peuvent vous faciliter le travail de l'indentation. Vous pouvez par exemple utiliser la touche tabulation, pour ajouter un niveau d'indentation. Un conseil cependant : configurez votre éditeur pour qu'il ajoute des espaces et non un caractère tabulation. En effet, selon l'outil, une tabulation n'a pas toujours la même taille, et un code source contenant des tabulations peut changer d'aspect suivant l'éditeur, et vous pouvez finir par avoir une indentation irrégulière.