PHP – COURS 03 – Les boucles : Les structures répétitives

(inspiré du cours d'algorithmique en ligne : http://www.pise.info/algo/logique.htm)

Les boucles, c'est généralement le point douloureux de l'apprentissage de la programmation. C'est là que ça coince, car autant il est assez facile de comprendre comment fonctionnent les tests, autant il est souvent long d'acquérir les réflexes qui permettent de maîtriser les boucles pour traiter un problème donné.

1. A quoi cela sert-il ?

Prenons le cas d'une saisie au clavier (une lecture), où par exemple, le programme pose une question à laquelle l'utilisateur doit donner un âge supérieur à 18 . Mais tôt ou tard, l'utilisateur, facétieux ou maladroit, risque de taper autre chose que la réponse attendue. Dès lors, le programme peut planter. Alors, dans tout programme un tant soit peu sérieux, on met en place ce qu'on appelle un contrôle de saisie, afin de vérifier que les données entrées au clavier correspondent bien à celles attendues par le programme. On pourrait essayer avec un SI. Voyons voir ce que ça donne :

```
<?php

  $nbErreur = 0 ;
  $age = readline("entrer un age > 18 : ");

  if ($age < 18){
     $age = readline("entrer un age > 18 : ");
     $nbErreur = $nbErreur + 1 ;
   }

 if ($age < 18){
    $age = readline("entrer un age > 18 : ");
    $nbErreur = $nbErreur + 1 ;
 }

 if ($age < 18){
    $age = readline("entrer un age > 18 : ");
    $nbErreur = $nbErreur + 1 ;
 }

 echo "vous vous êtes trompé  $nbErreur fois \n";
?>
```

Ça marche, du moins tant que l'utilisateur a le bon goût de ne se tromper que trois fois, et d'entrer une valeur correcte ensuite. Si l'on veut également « bétonner » en cas de quatrième erreur, il faudrait rajouter un SI. Et ainsi de suite, on peut rajouter des centaines de SI.
La solution consistant à aligner des SI... en pagaille est donc une impasse. La seule issue est donc d'utiliser une structure répétitives, qui se présente ainsi :

TantQue la condition est « **VRAIE** »
...
on exécute les **INSTRUCTIONS**
...
FinTantQue

Le principe est simple : le programme arrive sur la ligne du TantQue. Il examine alors la condition. Si cette valeur est VRAI, le programme exécute les instructions qui suivent, jusqu'à ce qu'il rencontre la ligne FinTantQue. Il retourne ensuite sur la ligne du TantQue, procède au même examen, et ainsi de suite. On sort de cette structure lorsque la condition prend la valeur FAUX.

Traduction en PHP :

```
while ($age < 18){
  $age = readline("entrer un age > 18 : ");
   $nbErreur = $nbErreur + 1 ;
}
```

2. Boucler en comptant, ou compter en bouclant

Dans l'exemple précédent, vous avez remarqué qu'une boucle pouvait être utilisée pour augmenter la valeur d'une variable. Cette utilisation des boucles est très fréquente, elle permet par exemple de savoir combien de fois la boucle a été exécutée ou alors combien de fois on souhaite l'exécuter.

Par exemple, si on souhaite afficher sur chaque ligne les nombres de 1 à 100. On peut évidemment écrire le programme suivant :

```
<?php
  echo ( " 1 \n") ;
  echo ( " 2 \n") ;
  echo ( " 3 \n") ;
  .........
  echo ( " 100 \n") ;
```

Mais lorsque l'on connaît les boucles, il est possible de poser le problème ainsi :

nombre = 1
TantQue nombre <= 100
  écrire nombre
  nombre = nombre + 1
FinTantQue

Traduction en PHP :

```
   $nombre = 1 ;
   while ($nombre <= 100){
        echo ("$nombre \n");
       $nombre = $nombre + 1;
    }
```

L'opération qui consiste à ajouter 1 à la variable $nombre à chaque tour de boucle s'appelle l'incrémentation, il est possible d'utiliser une notation simplifié :

$nombre = $nombre + 1 → notation simplifié **$nombre++**

Une fois la valeur 1 affichée, la variable $nombre sera incrémentée de 1, et vaudra 2 ; la condition de boucle deviendra fausse lorsque la valeur de la variable sera supérieure à 100, et on n'exécutera plus le code correspondant à la boucle.

De la même manière, il est possible de décrémenter la valeur d'une variable à chaque tour de boucle :

$nombre = $nombre - 1 → notation simplifié **$nombre--**

Reprendre l'exemple précédent, mais on souhaite afficher les nombres en commençant par 100 et en terminant par 1 :

Il est également possible d'incrémenter ou de décrémenter la valeur de la variable d'un nombre supérieur ou inférieur à 1.
Trouver le code permettant :

1 - Afficher tous les nombres pairs de 0 à 100

2 – Afficher les 50 premiers multiples de 5
5 10 15 20 25 30 35.....250

3 – Quel est l'objectif de cette boucle ?

```
$reponse = "non";

while ($reponse != "oui" ){
  $reponse =  readline("voulez vous sortir de la boucle ( répondre par oui ou par non)  : ");

}
echo " on est sorti de la boucle \n";
```

4 – Que va afficher cette boucle

```
$a = 3;

while($a < 10) {
  echo ("$a  \n");
}
```

EXERCICE :

Exercice 1 : Compteur
Écrire un programme compteur.php qui permet de compter et d'afficher de 1 jusqu'à 20.

Exercice 2 : Compteur
Améliorer le programme pour faire saisir à l'utilisateur jusqu'à quel chiffre il souhaite que le programme compte.

Exercice 3 : Compteur, on recommence
Améliorer le programme faire en sorte que l'on puisse recommencer lorsque l'utilisateur répondra oui à la question voulez vous recommencer ?

Exercice 4 : pair ou impair
Ecrire un programme qui demande à l'utilisateur de saisir un chiffre, le programme devra indiquer si

ce chiffre est pair ou impair (utilisé le modulo %) puis afficher les dix nombres pairs ou impairs suivant. Exemple si l'utilisateur saisie 10, le programme affichera :
→ nombre pair → 10 – 12 - 14 – 16 28
Remarque: le modulo représente le reste entier de la division d'un chiffre, pour savoir si un nombre est pair, il faut que le modulo 2 de ce chiffre soit égal à 0
exemple : 4%2 == 0 le chiffre 4 est pair
7%2 == 1 le chiffre est impair

Exercice 5 : Plus difficile
Écrire un programme qui demande à l'utilisateur d'entrer un chiffre. Cette action devra être répétée tant que l'utilisateur répondra oui à la question voulez vous recommencer ? Le programme devra indiquer :
· le nombre de valeurs saisies
· le nombre valeurs paires
· le nombre valeurs impaires