

Les boucles

Table des matières

I. Contexte	3
II. Qu'est-ce qu'une boucle ?	3
III. Exercice : Appliquez la notion	5
IV. Les boucles while et do while	5
V. Exercice : Appliquez la notion	9
VI. Les boucles for	11
VII. Exercice : Appliquez la notion	14
VIII. Les boucles foreach	14
IX. Exercice : Appliquez la notion	17
X. Auto-évaluation	19
A. Exercice final	19
B. Exercice : Défi	22
Solutions des exercices	22

I. Contexte

Durée : 1 h

Environnement de travail : Repl.it

Pré-requis : Les tableaux

Contexte

Quelle que soit l'application que nous voulons créer, nous serons amenés à implémenter des opérations qui se répètent : afficher chaque élément d'une liste, effectuer un même calcul sur plusieurs valeurs, etc. Nous pourrions bien entendu écrire autant de lignes de code que d'opérations à effectuer, mais nos scripts deviendraient difficiles à maintenir et illisibles.

II. Qu'est-ce qu'une boucle ?

Objectifs

- Définir une boucle
- Repérer quand utiliser une boucle

Mise en situation

Il existe des structures permettant de répéter une opération plusieurs fois : les boucles.

Définition Boucle

Une boucle est une structure permettant de répéter une ou plusieurs instructions jusqu'à ce qu'une condition pré-définie soit remplie ou qu'un résultat voulu soit obtenu. Elles permettent d'éviter la réécriture d'instructions et facilitent leur réutilisation.

`for`, `foreach`, `while`, `do while` sont les quatre types de boucles utilisés en programmation.

Définition Itération

Une **itération** est une répétition de la boucle.

Exemple Calcul simple sans boucle

Imaginons un monde sans les boucles. Que ferions-nous pour effectuer la somme de tous les nombres entre 1 et 100 ?

```
1 <?php
2
3 // Calcul ligne par ligne.
4 $addition = 1;
5 $addition = $addition + 2;
6 $addition = $addition + 3;
7 $addition = $addition + 4;
8 $addition = $addition + 5;
9 $addition = $addition + 6;
10 $addition = $addition + 7;
```

```
11 $addition = $addition + 8;
12 $addition = $addition + 9;
13 //... A répéter jusqu'à 100
14 echo $addition;
```

Pour obtenir le résultat, nous avons écrit l'instruction une première fois puis copier/coller ensuite les autres lignes en modifiant le nombre. Pour répondre à la question, nous avons donc copié/collé le même code 100 fois.

Exemple Calcul simple avec boucle

Il est possible d'écrire les 100 calculs avec seulement quatre lignes de code.

```
1 <?php
2
3 // Calcul grâce à une boucle.
4 for($i = 1; $i <= 100; $i++) {
5
6     $addition = $addition + $i;
7 }
8 echo $addition;
```

Exemple Table des 2

Imaginons maintenant que nous ayons un tableau stockant les nombres de 2 à 1000 et que nous souhaitons multiplier par 2 chacun de ces nombres.

Sans boucle, nous écririons 1000 fois la même instruction.

```
1 <?php
2 $nombres = [1,2,3,4,5,6]; // Pour l'exemple, nous nous arrêtons à 6
3
4 // Multiplication ligne par ligne.
5 $nombres[0] *= 2;
6 $nombres[1] *= 2;
7 $nombres[2] *= 2;
8 $nombres[3] *= 2;
9 $nombres[4] *= 2;
10 $nombres[5] *= 2;
11
12 var_dump($nombres);
13
```

Avec les boucles, nous pouvons factoriser notre code.

```
1 <?php
2 $nombres = [1,2,3,4,5,6];
3
4 // Multiplication grâce à une boucle.
5 foreach($nombres as $index => $nombre) {
6     $nombres[$index] *= 2;
7 }
8
9 var_dump($nombres);
10
```

Syntaxe **À retenir**

Si nous sommes amenés à réécrire plusieurs fois la même ligne de code pour le même traitement, alors c'est que nous devons utiliser une boucle.

III. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

**Question**

[solution n°1 p.23]

Nous allons manipuler un script qui pourrait être utilisé par des écoles pour générer simplement des tables de multiplication pour les élèves.

```
1 <?php
2
3 $a = 0;
4 $b = 20;
5 $c = 3;
6
7 for ($i = $a; $i <= $b; $i++) {
8     echo $c.' x '.$i.' = ' . ($c * $i) . '<br>';
9 }
```

Le comportement de ce script est déterminé par les trois variables présentes au début de ce script, mais elles ont été très mal nommées.

- Exécutez ce script dans Repl.it en modifiant les valeurs de ces trois variables afin de voir comment elles influent sur la boucle.
- Déterminez la fonction de chaque variable et proposez de meilleurs noms.

IV. Les boucles while et do while

Objectifs

- Savoir écrire une boucle `while`
- Savoir écrire une boucle `do while`

Mise en situation

Les premières boucles que nous allons voir sont `while` (« tant que »), et sa variante `do ... while` (« faire ... tant que »). Ce sont en fait des répétitions de la structure de contrôle `if`.

Définition

`while` et `do while` sont des boucles exécutant des instructions sur la base d'une condition, tant que celle-ci est vraie.

1 <https://repl.it/>

Syntaxe while

while s'appelle en mettant la condition entre parenthèses et le code à exécuter entre accolades.

```
1 <?php
2 while (condition) {
3     // code à exécuter
4 }
5
```

Exemple

```
1 <?php
2
3     $i = 0;
4
5     while ($i < 10) {
6         $i = $i + 1;
7     }
8     echo $i;
9     ?
```

Dans cet exemple nous commençons par créer notre variable qui s'appelle \$i avec comme valeur 0, la boucle while va permettre de vérifier que **TANT QUE** la valeur de \$i est strictement inférieure à 10, nous allons incrémenter \$i.

Syntaxe do while

Cette boucle a une écriture un peu particulière : nous écrirons d'abord le mot-clé do, puis le code à exécuter entre des accolades, et enfin l'instruction while avec la condition.

```
1 <?php
2 do {
3     // code à exécuter
4 } while (condition)
5 ?>
```

Attention while vs do while

La différence entre ces deux boucles réside dans le fait que while n'exécutera jamais son code si la condition est fausse, alors que do while l'exécutera au moins une fois.

Exemple

```
1 <?php
2
3 $condition = false;
4
5 echo 'Qui va s\'afficher ? ';
6
7 while ($condition) {
8     echo 'boucle while';
9 }
10
11 do {
12     echo 'boucle do while';
13 } while ($condition)
14
```

15 ?>

La différence avec le **while**, c'est que le **do while** va d'abord s'exécuter pour ensuite vérifier la condition, c'est pour ça que dans cet exemple c'est uniquement le do while qui va s'afficher, car on veut vérifier que \$condition retourne **vrai** alors qu'il retourne **faux**.

Qui va s'afficher ? boucle do while

Exemple Affichage des nombres pairs

```
1 <?php
2
3 $compteur = 2;
4
5 while ($compteur <= 1000) {
6 for ($compteur = 2; $compteur <= 1000; $compteur++) {
7 if($compteur % 2 === 0){
8 echo $compteur. ' est pair <br>';
9 } else {
10 echo $compteur. ' est impair <br>';
11 }
12 }
13 }
14 }
15
16 echo 'Terminé !';
```

Ici, nous déclarons la variable \$compteur avec une valeur initiale à 2. Tant que cette valeur est inférieure ou égale à 1000, nous vérifions la parité de la valeur actuelle du compteur, puis nous incrémentons (donc ajoutons 1) à ce compteur. Puis la boucle recommence : on vérifie si 3 est inférieur à 1000 et, comme c'est le cas, on vérifie la parité et on incrémente le compteur, qui passe à 4. Il est toujours inférieur à 1000, donc on vérifie la parité, etc.

Une fois arrivés à 1000, on va faire un dernier tour de boucle (1000 étant **inférieur ou égal** à 1000) : on vérifie la parité de 1000 et on incrémente le compteur, qui passe à 1001. Cette fois, 1001 n'est plus inférieur ou égal à 1000, donc on sort de la boucle (sans vérifier la parité de 1001, la condition de la boucle étant fausse), et on affiche "Terminé !".

Attention

Dans l'exemple précédent, nous avons des bornes connues : nous voulions déterminer la parité des nombres de 2 à 1000. Cependant, la boucle while permet surtout de gérer d'autres conditions que de simples compteurs (on préférera pour cela les boucles for).

Exemple

Jouons à un jeu avec notre ordinateur : nous allons lui donner un nombre entre 0 et 100 qu'il va devoir deviner en faisant des propositions aléatoires. Ici, la condition d'arrêt de la boucle est donc le fait que l'ordinateur a trouvé le bon nombre.

Nous allons utiliser la fonction rand(), qui génère chiffre aléatoirement. Dans l'exemple ci-dessous le chiffre sera compris entre 0 et 100.

```
1 <?php
2
3 $number = 30;
4 $numberOfTries = 0;
5
6 do {
```

```

7  $guess = rand(0, 100); //L'ordinateur choisit un nombre aléatoire
8  echo 'Je tente ' . $guess . ' !<br>';
9  $numberOfTries++;
10 } while ($guess !== $number); //On boucle tant que la réponse n'a pas été trouvée
11
12 echo 'Trouvé au bout de ' . $numberOfTries . ' essais !';

```

```

Je tente 32 !
Je tente 74 !
Je tente 21 !
Je tente 25 !
Je tente 91 !
Je tente 27 !
Je tente 83 !
Je tente 21 !
Je tente 30 !
Trouvé au bout de 144 essais !

```

Remarque

Notons l'utilisation de la boucle `do while` ici : étant donné que l'ordinateur doit faire une proposition pour qu'elle soit juste, on a forcément besoin d'entrer dans la boucle une première fois.

La boucle `while` peut aussi nous permettre de parcourir les tableaux. Nous mettrons en place un compteur partant de 0 et allant jusqu'à la taille du tableau (calculé par la fonction `count()`).

Attention Boucles infinies

Une boucle infinie est une boucle qui ne cessera jamais d'itérer, car sa condition n'est jamais remplie.

Pour éviter les boucles infinies avec `while` et `do while`, il faut **s'assurer que la condition évolue au cours de l'exécution**.

Exemple

```

1 <?php
2
3     $compteur = 1;
4
5     while ($compteur <= 10) {
6         echo $compteur . ' <br>';
7     }
8

```


La boucle ci-dessus affichera la valeur "1" indéfiniment.

Syntaxe À retenir

```
1 while (condition) {  
2   // code à exécuter.  
3   // Si utilisation d'un compteur, penser à l'incrémenter.  
4 }  
  
1 do {  
2   // code à exécuter.  
3   // Si utilisation d'un compteur, penser à l'incrémenter.  
4 } while(condition)
```

Complément

while¹

do while²

La boucle **while** peut aussi nous permettre de parcourir les tableaux.

V. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



Question

[solution n°2 p.23]

Le "jeu du bâtonnet" est un jeu simple dans lequel deux joueurs ont à leur disposition 15 bâtonnets de bois alignés sur une table. Chacun leur tour, les joueurs peuvent retirer soit un, soit deux, soit trois bâtonnets. Le joueur qui retire le dernier bâtonnet perd la partie.

Afin de simuler des parties de bâtonnets pour essayer d'élaborer une stratégie, le but de cet exercice est de réaliser un script permettant de générer une partie de bâtonnets avec des tirages aléatoires.

1 <https://www.php.net/manual/fr/control-structures.while.php>

2 <https://www.php.net/manual/fr/control-structures.do.while.php>

3 <https://repl.it/>

Voici deux exemples de résultat du script :

Joueur 1 retire 3 bâtons. Il en reste 12
 Joueur 2 retire 2 bâtons. Il en reste 10
 Joueur 1 retire 1 bâtons. Il en reste 9
 Joueur 2 retire 1 bâtons. Il en reste 8
 Joueur 1 retire 1 bâtons. Il en reste 7
 Joueur 2 retire 3 bâtons. Il en reste 4
 Joueur 1 retire 1 bâtons. Il en reste 3
 Joueur 2 retire 2 bâtons. Il en reste 1
 Joueur 1 retire 1 bâtons. Il en reste 0

Joueur 1 retire 3 bâtons. Il en reste 12
 Joueur 2 retire 1 bâtons. Il en reste 11
 Joueur 1 retire 3 bâtons. Il en reste 8
 Joueur 2 retire 1 bâtons. Il en reste 7
 Joueur 1 retire 1 bâtons. Il en reste 6
 Joueur 2 retire 1 bâtons. Il en reste 5
 Joueur 1 retire 2 bâtons. Il en reste 3
 Joueur 2 retire 2 bâtons. Il en reste 1
 Joueur 1 retire 2 bâtons. Il en reste -1

On propose la base algorithmique ci-après : complétez-là avec `while` pour répondre au problème posé. Puis faites la même chose avec `do while`.

```
1 <?php
2
3 $tries = 0;
4 $sticksNumber = 15;
5
6 // Contenu de la boucle
7
8 $removedSticks = rand(1, 3);
9 $playerNumber = ($tries % 2) + 1;
10 $sticksNumber -= $removedSticks;
```

```
11 echo 'Joueur '.$playerNumber.' retire '.$removedSticks.' bâtons. Il en reste  
12 '.$sticksNumber.'<br>';  
    $tries++;
```

Indice :

La fonction PHP `rand($min, $max)` permet de gérer un nombre aléatoire compris entre le premier paramètre et le deuxième. Par exemple, si je souhaite obtenir un nombre aléatoire entre 5 et 10, je dois appeler la fonction `rand(5, 10)`.

Indice :

Nous allons retirer des bâtonnets *tant que* son nombre est supérieur à 0.

Indice :

Le numéro du joueur peut être déterminé selon le numéro du tour de boucle : s'il est pair, c'est le joueur 1 ; s'il est impair, c'est le joueur 2.

VI. Les boucles for

Objectif

- Savoir écrire une boucle `for`

Mise en situation

Nous allons voir comment écrire une boucle `for` (*pour*) et quel est l'intérêt de cette dernière par rapport à une boucle `while`.

Définition

La boucle `for` est une boucle spécialisée dans les compteurs. Elle se différencie du `while` par le fait que sa signature intègre directement le compteur d'itération : la variable `$compteur`, sur laquelle on faisait un `$compteur++` plus loin dans la boucle, est remplacée par une structure intégrée au `for`.

Syntaxe

Pour faire une boucle `for`, nous indiquerons le mot-clé `for` suivi de trois instructions permettant de définir le cycle de vie de notre compteur :

- La valeur de départ du compteur, par exemple `$compteur=0`.
- La condition de sortie basée sur la valeur du compteur, comme `$compteur !== 10` ou `$compteur <= 10`.
- Le pas d'incréméntation du compteur, par exemple `$compteur++`.

Ces instructions doivent être entre parenthèses et séparées par des points-virgules, sachant qu'une boucle `for` peut s'incrémenter, mais aussi se décrémenter.

```
1 <?php  
2 // Incrément.  
3 for (min; max; pas++) {  
4     // Code à exécuter.  
5 }  
6  
7 // Décrément.  
8 for (max; min; pas--) {  
9     // Code à exécuter.  
10 }
```

Exemple Cas pratique

```
1 <code>
2 <?php
3 for($i = 1; $i <= 10; $i++){
4 echo $i . '<br>';
5 }
6 <code>
```

Dans cet exemple, on va simplement afficher la valeur de \$i jusqu'à ce qu'elle vaut 10.

Reprenons l'exemple des nombres pairs, mais cette fois avec une boucle `for`.

```
1 <?php
2
3 // Pour les valeurs allant de 2 à 1000.
4 for ($compteur = 2; $compteur <= 1000; $compteur++) {
5 if($compteur % 2 == 0){
6 echo $compteur. ' est pair <br>';
7 } else {
8 echo $compteur. ' est impair <br>';
9 }
```

Complément

Même si c'est plus rare, l'incréméntation peut se faire de 2 en 2, 3 en 3, etc.

```
1 // Incréméntation de 2 en 2.
2 for ($compteur = 2; $compteur <= 1000; $compteur += 2) {}
3
4 // Incréméntation de 3.
5 for ($compteur = 2; $compteur <= 1000; $compteur += 3) {}
```

Attention for avec les tableaux

On peut utiliser des boucles `for` pour parcourir un tableau mais ce n'est pas vraiment la meilleure façon de le faire, nous verrons donc plus tard ce qu'il faut utiliser dans ce cas là.

Méthode for vs while

La différence entre `for` et `while` est donc que `for` est plus adapté aux situations qui nécessitent un compteur : les risques de boucles infinies sont réduits, étant donné que la gestion du compteur se fait directement dans la déclaration de la boucle, et non pas dans le corps.

`while` restera utile dans les cas où la condition de sortie de la boucle est autre, par exemple parcourir ligne par ligne un fichier dont on ne connaît pas la taille à l'avance.

Attention

La boucle `for` peut aussi parcourir les tableaux. Attention toutefois à l'utilisation de la fonction `count()`, retournant la taille d'un tableau. En effet, la condition du compteur est ré-évaluée à chaque tour de boucle, donc si la fonction `count()` s'y trouve, elle est appelée à chaque itération.

Exemple

```
1 <?php
2
3 $nombres = [1,2,3,4,5];
4
5 for ($compteur = 0; $compteur < count($nombres); $compteur++) { // Ici, on recalcule la taille
    du tableau à chaque itération !
6     echo $nombres[$compteur];
7 }
```

Cela aura pour effet de ralentir l'exécution de notre programme.

Il est préférable d'écrire :

```
1 <?php
2
3 $nombres = [1,2,3,4,5];
4
5 $max = count($nombres); //On calcule la taille une seule fois et on la stocke dans une
    variable
6
7 for ($compteur = 0; $compteur < $max; $compteur++) { // On utilise la variable
8     echo $nombres[$compteur];
9 }
10 ?>
```

Attention **Gare aux boucles infinies**

Attention là aussi aux boucles infinies. Bien que le compteur soit déclaré dans la signature du `for` pour limiter les risques, il faut s'assurer que ce dernier évolue dans le bon sens.

C'est-à-dire, pour une boucle allant de 1 à 10, que l'incréméntation soit positive, et pour une boucle descendant de 10 à 1, que l'incréméntation soit négative.

Exemple

```
1 <?php
2 // La boucle doit s'arrêter quand $compteur = 9
3 // Or le compteur part de 0 et décroît : -1, -2 ,etc. Nous avons donc une boucle infinie !
4 for($compteur = 0; $compteur < 10; $compteur--) {
5     echo $compteur;
6 }
```

```
0
-1
-2
-3
-4
-5
-6
-7
-8
-9
-10
11
```

Syntaxe **À retenir**

```
1 for (min; max; pas) {
2     // Code à exécuter.
3 }
```

Complément

for¹

VII. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



Question

[solution n°3 p.24]

Comme nous l'avons vu, la boucle `for` permet d'afficher beaucoup d'informations en utilisant le minimum de lignes de code.

L'affichage de fin devra ressembler à ceci :

```
1 1
2 22
3 333
4 4444
5 55555
6 666666
7 7777777
8 88888888
9 999999999
```

VIII. Les boucles `foreach`

Objectifs

- Savoir écrire une boucle `foreach`

Mise en situation

Nous pouvons itérer sur les tableaux grâce aux boucles `while` et `for`. À l'usage, ces dernières ne sont pas véritablement adaptées au parcours des tableaux, surtout les tableaux associatifs, dont les clefs ne peuvent pas être calculées en incrémentant une valeur. Nous allons apprendre une nouvelle boucle, `foreach`, qui a été spécialement conçue pour cela.

Syntaxe

Soit le tableau associatif suivant :

```
1 <?php
2 $user = [
3     'username' => 'John',
4     'email' => 'john@email.com',
5     'age' => 30
6 ];
```

1 <https://www.php.net/manual/fr/control-structures.for.php>

2 <https://repl.it/>

Si je veux parcourir les informations de ce tableau avec un `while` ou un `for`, c'est impossible. Cependant, je peux utiliser le **foreach** : il s'écrit avec le mot-clé `foreach` suivi de la variable contenant mon tableau (ici, `$user`), le mot-clé `as`, puis une variable qui va recevoir les données. Cette variable est déclarée directement dans l'expression du `foreach`.

```
1 <?php
2 foreach($user as $information) { // Il n'y a pas besoin de déclarer la variable "information"
3     echo $information.'<br>'; // Va afficher "John" au premier tour de boucle, "john@email.com"
    au deuxième et 30 au troisième.
4 }
```

Il est également possible de récupérer les clés du tableau lors d'un `foreach` : pour cela, il suffit de remplacer la variable recevant les données par la notation `$cle => $valeur`. Ceci signifie que, pour chaque itération, la clé ou indice de l'élément courant du tableau est associé à `$cle` et l'élément courant à `$valeur`. Ces deux variables peuvent avoir n'importe quel nom.

```
1 <?php
2 foreach($user as $userField => $userInformation) {
3     echo $userField.' : '.$userInformation.'<br>';
4 }
```

Boucle par référence

Pour modifier une valeur du tableau, nous pouvons précéder "valeur" par un `&`. Au lieu de copier la valeur courante, cela assignera une référence vers celle-ci, que nous pourrions modifier.

Sans référence, nous aurions modifié la valeur comme ceci : `$tableau['cle'] = $valeur + 2`. Grâce à la référence, nous pouvons écrire : `$valeur = $valeur+2`.

Syntaxe

Grâce au symbole `&`, nous assignons une référence à valeur.

```
1 <?php
2
3 foreach($user as &$information) {
4     // Affecter une valeur a $information ici écrasera la valeur du tableau
5 }
6
7 foreach($user as $information) {
8     // Affecter une valeur a $information ici n'aura aucune incidence sur la valeur du tableau
9 }
```

Exemple

Nous pouvons maintenant modifier les valeurs du tableau simplement. Imaginons que nous voulions doubler toutes les valeurs d'un tableau de nombres :

```
1 <?php
2
3 $numbers = [4, 8, 15, 16, 23, 42];
4
5 foreach($numbers as &$value) {
6     $value *= 2;
7 }
8
9 var_dump($numbers);
```

Sans la référence, nos valeurs n'auraient pas été modifiées, il aurait fallu passer par la clef du tableau.

```
1 <?php
2
3 $numbers = [4, 8, 15, 16, 23, 42];
4
5 foreach($numbers as $index => $value) {
6     $numbers[$index] *= 2;
7 }
8
9 var_dump($numbers);
```

Attention

Lorsque nous faisons une boucle `for`, la variable contenant les valeurs successives n'est **pas détruite à la fin de la boucle**. Ainsi, même si ce n'est pas recommandé, il est possible de l'utiliser après être sorti de la boucle :

```
1 <?php
2
3 $numbers = [4, 8, 15, 16, 23, 42];
4
5 foreach($numbers as $value) {
6 }
7
8 var_dump($value); // Affiche 42 !
```

En utilisant la référence `&`, si nous modifions cette variable, nous modifions également le dernier élément du tableau.

```
1 <?php
2
3 $numbers = [4, 8, 15, 16, 23, 42];
4
5 foreach($numbers as &$amp;value) {
6 }
7
8 $value = 26;
9 echo $numbers[5]; // Affiche 26 !
```

Il faut être très vigilant lorsque l'on utilise des variables d'une boucle hors de leur contexte de boucle, et encore plus lorsque l'on utilise des références ! Pour éviter ces problèmes, il est possible de détruire la référence grâce à la fonction `unset()`.

```
1 <?php
2
3 $numbers = [4, 8, 15, 16, 23, 42];
4
5 foreach($numbers as &$amp;value) {
6 }
7 unset($value); // Nous détruisons la référence. La variable n'existe plus.
8 $value = 26; // Nous créons une variable en lui donnant le même nom
9 echo $numbers[5]; // Affiche 42
```

Syntaxe À retenir

```
1 <?php
2 foreach(tableau as valeur) {
3     // Code à exécuter.
4 }
5 ?>
```



```
1 <?php
2 foreach(tableau as cle => valeur) {
3     // Code à exécuter.
4 }
5 ?>
```

& nous permettra d'assigner une référence vers l'élément courant. Pour éviter les effets de bords, nous devons la détruire après utilisation, avec la méthode `unset`.

Complément

`foreach`¹

référence²

`unset()`³

IX. Exercice : Appliquez la notion

Pour cet exercice, nous allons créer un script permettant de visualiser les informations d'un tableau de nombre de ventes d'une boutique de prêt-à-porter.

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



Question 1

[solution n°4 p.24]

Une boutique de prêt-à-porter extrait le résultat de ses ventes annuelles de jeans sous cette forme :

```
1 <?php
2
3 $sales = [
4     'janvier' => 64,
5     'fevrier' => 45,
6     'mars' => 32,
7     'avril' => 45,
8     'mai' => 35,
9     'juin' => 47,
10    'juillet' => 68,
11    'aout' => 76,
12    'septembre' => 43,
13    'octobre' => 23,
14    'novembre' => 83,
15    'decembre' => 15,
16 ];
```

Réalisez une interface permettant d'afficher ces données dans une liste à puces, suivie du total.

1 <https://www.php.net/manual/fr/control-structures.foreach.php>

2 <https://www.php.net/manual/fr/language.references.php>

3 <https://www.php.net/manual/fr/function.unset.php>

4 <https://repl.it/>

Liste des ventes annuelles

- janvier : 64
- fevrier : 45
- mars : 32
- avril : 45
- mai : 35
- juin : 47
- juillet : 68
- aout : 76
- septembre : 43
- octobre : 23
- novembre : 83
- decembre : 15

Total : 576

Indice :

Pour calculer le total, il serait possible d'utiliser la fonction `array_sum`, mais cela nous obligerait à faire deux fois le tour de notre tableau : une première fois pour l'affichage, puis une deuxième fois par la fonction `array_sum`. Pour des raisons de performances, il est toujours préférable de ne faire qu'un minimum de boucles, donc il va falloir faire la somme à la main !

Question 2

[solution n°5 p.24]

Modifiez votre script pour indiquer, pour chaque mois, si c'est une hausse (symbolisée par un +) ou une baisse (symbolisée par un -) de ventes par rapport au mois dernier. Le premier mois sera forcément une hausse puisqu'il n'existe pas de "mois précédent".

Liste des ventes annuelles

- janvier : 64+
- fevrier : 45-
- mars : 32-
- avril : 45+
- mai : 35-
- juin : 47+
- juillet : 68+
- aout : 76+
- septembre : 43-
- octobre : 23-
- novembre : 83+
- decembre : 15-

Total : 576

Indice :

Il va falloir stocker la valeur du dernier mois dans une variable pour pouvoir la comparer à la valeur du mois en cours.

X. Auto-évaluation**A. Exercice final****Exercice 1**

[solution n°6 p.25]

Exercice

Combien existe-t-il de boucles différentes ?

- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5
- ☐ 6

Exercice

Cette boucle est une boucle infinie.

```
1 <?php
2
3 $counter = 0;
4
5 while ($counter < 10) {
6     if ($counter % 2 === 0) {
7         $counter++;
8     } else {
9         $counter--;
10    }
11 }
```

- ☐ Vrai
- ☐ Faux

Exercice

Cette boucle est une boucle infinie.

```
1 <?php
2
3 for ($counter = 0; $counter < 10; $counter *= -2) {
4     echo $counter.'<br>';
5 }
```

- ☐ Vrai
- ☐ Faux

Exercice

Cette boucle est une boucle infinie.

```
1 <?php
2
3 $doubles = 1;
4 $isFinished = false;
5 do {
6     $doubles *= 2;
7     if ($doubles > 20000) {
8         $isFinished = true;
9     }
10 } while (!$isFinished);
11
```

- ☐ Vrai
- ☐ Faux

Exercice

Combien de fois est-ce que le mot "ping" va être écrit ?

```
1 <?php
2
3 for ($counter = 0; $counter <= 10; $counter++) {
4     echo 'ping';
5 }
```

- ☐ 9
- ☐ 10
- ☐ 11
- ☐ 12
- ☐ 13

Exercice

Combien de fois est-ce que le mot "ping" va être écrit ?

```
1 <?php
2
3 $counter = 1;
4 do {
5     if ($counter * 2 > 25) {
6         $counter = $counter * 3;
7     } else {
8         $counter = $counter * 2;
9     }
10    echo 'ping';
11} while ($counter > 50);
```

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

Exercice

Combien de fois est-ce que le mot "ping" va être écrit ?

```
1 <?php
2 for ($i = 0; $i < 10; $i++) {
3     for ($j = 0; $j < 5; $j++) {
4         echo 'ping';
5     }
6 }
7
```

- ☐ 10
- ☐ 15
- ☐ 50
- ☐ 66
- ☐ 150

Exercice

Combien de fois est-ce que le mot "ping" va être écrit ?

```
1 <?php
2 $numbers = [1, 2, 3, 4, 5];
3
4 foreach ($numbers as $number) {
5     for ($i = 0; $i < $number; $i++) {
6         echo 'ping';
7     }
8 }
```

- ☐ 9
- ☐ 10
- ☐ 15
- ☐ 16
- ☐ 20

B. Exercice : Défi

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



Question

[solution n°7 p.28]

Pour réviser les tables de multiplication, le professeur de mathématiques voudrait que chaque élève trouve une façon originale de les afficher. Vous avez donc eu l'idée de les afficher sur votre ordinateur, avec l'utilisation de php. Grâce à une boucle for.

Indice :

Vous allez devoir utiliser une variable intermédiaire pour chaque table qui sera instancié de 1 à 10.

Chaque table sera affichée les unes après les autres.

Solutions des exercices

1 <https://repl.it/>

p.5 Solution n°1

- \$a pourrait s'appeler \$start, par exemple, car cette variable permet de définir le premier nombre par lequel nous devons commencer à multiplier.
- \$b pourrait s'appeler \$end, car cette variable permet de définir jusqu'à quel nombre nous devons multiplier.
- \$c pourrait s'appeler \$numberToMultiply, puisqu'elle définit le nombre à multiplier, donc de quel nombre nous souhaitons avoir la table de multiplication.

Rappel

Bien entendu, ces noms sont des exemples. N'importe quel nom aurait été valide, du moment qu'il respecte les bonnes pratiques de code, à savoir :

- Un nom valide en PHP (qui commence par une lettre ou un *underscore*),
- Écrit en **camelCase**, c'est-à-dire avec la première lettre en minuscule, sans espace, et une majuscule à chaque nouveau mot,
- Que le nom soit en anglais,
- Que le nom décrive la valeur contenue dans la variable.

p.9 Solution n°2

```

1 <?php
2
3 $tries = 0;
4 $sticksNumber = 15;
5
6 while ($sticksNumber > 0) {
7     $removedSticks = rand(1, 3);
8     $playerNumber = ($tries % 2) + 1;
9     $sticksNumber -= $removedSticks;
10    echo 'Joueur '.$playerNumber.' retire '.$removedSticks.' bâtons. Il en reste
        '.$sticksNumber.'<br>';
11    $tries++;
12 } ;

```

```

1 <?php
2
3 $tries = 0;
4 $sticksNumber = 15;
5
6 do {
7     $removedSticks = rand(1, 3);
8     $playerNumber = ($tries % 2) + 1;
9     $sticksNumber -= $removedSticks;
10    echo 'Joueur '.$playerNumber.' retire '.$removedSticks.' bâtons. Il en reste
        '.$sticksNumber.'<br>';
11    $tries++;
12 } while ($sticksNumber > 0);

```

p. 14 Solution n°3

```
1 <?php
2     for($nombre = 1;$nombre <= 9;$nombre++) {
3         for($repetier = 1;$repetier <= $nombre;$repetier++){
4             echo $nombre;
5         }
6         echo ' <br />';
7     }
8     ?>
```

p. 17 Solution n°4

```
1 <html>
2 <head></head>
3 <body>
4 <h1>Liste des ventes annuelles</h1>
5 <ul>
6 <?php
7
8 $sales = [
9     'janvier' => 64,
10    'fevrier' => 45,
11    'mars' => 32,
12    'avril' => 45,
13    'mai' => 35,
14    'juin' => 47,
15    'juillet' => 68,
16    'aout' => 76,
17    'septembre' => 43,
18    'octobre' => 23,
19    'novembre' => 83,
20    'decembre' => 15,
21 ];
22
23 $total = 0;
24 foreach ($sales as $month => $quantity) {
25     echo '<li>' . $month . ' : ' . $quantity . '</li>' ;
26     $total += $quantity;
27 }
28 ?>
29 </ul>
30 Total :
31 <?php
32     echo $total;
33 ?>
34 </body>
35 </html>
```

p. 19 Solution n°5


```
1 <html>
2 <head></head>
3 <body>
4 <h1>Liste des ventes annuelles</h1>
5 <ul>
6 <?php
7
8 $sales = [
9     'janvier' => 64,
10    'fevrier' => 45,
11    'mars' => 32,
12    'avril' => 45,
13    'mai' => 35,
14    'juin' => 47,
15    'juillet' => 68,
16    'aout' => 76,
17    'septembre' => 43,
18    'octobre' => 23,
19    'novembre' => 83,
20    'decembre' => 15,
21 ];
22
23 $total = 0;
24 $lastMonthQuantity = 0;
25 foreach ($sales as $month => $quantity) {
26     echo '<li>'.$month.' : '.$quantity;
27     if($lastMonthQuantity > $quantity){
28         echo '-';
29     } else {
30         echo '+';
31     }
32     echo '</li>';
33
34     $lastMonthQuantity = $quantity;
35     $total += $quantity;
36 }
37 ?>
38 </ul>
39 Total :
40 <?php
41     echo $total;
42 ?>
43 </body>
44 <html>
```


Exercice p. 19 Solution n°6

Exercice

Combien existe-t-il de boucles différentes ?

- ☐ 2
- ☐ 3
- ☒ 4
- ☐ 5

☐ 6

 Les 4 types de boucles existantes sont `while`, `do while`, `for` et `foreach`.


Exercice

Cette boucle est une boucle infinie.

```
1 <?php
2
3 $counter = 0;
4
5 while ($counter < 10) {
6     if ($counter % 2 === 0) {
7         $counter++;
8     } else {
9         $counter--;
10    }
11 }
```

☒ Vrai

☐ Faux

 Cette boucle est une boucle infinie, car le compteur va avoir pour valeurs 1, puis 0, puis 1, puis 0 etc. Or, il doit atteindre 10 pour sortir de la boucle, valeur qu'il n'atteindra jamais.


Exercice

Cette boucle est une boucle infinie.

```
1 <?php
2
3 for ($counter = 0; $counter < 10; $counter *= -2) {
4     echo $counter.'<br>';
5 }
```

☒ Vrai

☐ Faux

 Cette boucle est une boucle infinie, car 0×-2 fera toujours 0. Le compteur ne va jamais augmenter.


Exercice

Cette boucle est une boucle infinie.

```
1 <?php
2
3 $doubles = 1;
4 $isFinished = false;
5 do {
6     $doubles *= 2;
7     if ($doubles > 20000) {
8         $isFinished = true;
9     }
10 } while (!$isFinished);
11
```

☐ Vrai

☒ Faux

 Cette boucle n'est pas infinie : en multipliant le nombre `$doubles` par 2 à chaque tour de boucle, il arrivera forcément à être supérieur à 20000, provoquant le passage de la variable `$isFinished` à `true`, sortant ainsi de la boucle.

Exercice

Combien de fois est-ce que le mot "ping" va être écrit ?

```
1 <?php
2
3 for ($counter = 0; $counter <= 10; $counter++) {
4     echo 'ping';
5 }
```


☐ 9

☐ 10

☒ 11

☐ 12

☐ 13

 Le compteur va compter de 0 à 10, donc on va faire 11 tours de boucle. Attention au symbole `<=` qui fait un tour de boucle de plus que `<` !

Exercice

Combien de fois est-ce que le mot "ping" va être écrit ?

```
1 <?php
2
3 $counter = 1;
4 do {
5     if ($counter * 2 > 25) {
6         $counter = $counter * 3;
7     } else {
8         $counter = $counter * 2;
9     }
10    echo 'ping';
11 } while ($counter > 50);
```


☒ 1

☐ 2

☐ 3

☐ 4

☐ 5

 La condition pour rester dans la boucle est que la valeur de `$counter` doit être **supérieure** à 50. Au premier tour de boucle, `$counter` vaut $1 * 2 = 2$, qui est inférieur à 50, donc on sort tout de suite de la boucle !

Exercice

Combien de fois est-ce que le mot "ping" va être écrit ?

```
1 <?php
2 for ($i = 0; $i < 10; $i++) {
3     for ($j = 0; $j < 5; $j++) {
4         echo 'ping';
5     }
6 }
7
```

- ☐ 10
- ☐ 15
- ☒ 50
- ☐ 66
- ☐ 150



La première boucle, comptée par `$i`, va s'exécuter 10 fois (on compte de 0 à 9, donc 10 tours de boucle).

La seconde boucle, comptée par `$j`, va compter 5 fois (de 0 à 4) et afficher "ping".

La première boucle va donc appeler 10 fois la deuxième boucle, qui va elle même afficher 5 fois "ping" par appel. Donc, au final, il y aura 10×5 "ping", donc 50 pings !

Exercice

Combien de fois est-ce que le mot "ping" va être écrit ?

```
1 <?php
2 $numbers = [1, 2, 3, 4, 5];
3
4 foreach ($numbers as $number) {
5     for ($i = 0; $i < $number; $i++) {
6         echo 'ping';
7     }
8 }
```

- ☐ 9
- ☐ 10
- ☒ 15
- ☐ 16
- ☐ 20



La variable `$number` va être égale successivement à 1, 2, 3, 4 et 5. À chaque tour de boucle, on va compter de 0 jusqu'à ce nombre diminué de 1 (puisque, dans le `for`, le compteur doit être **strictement inférieur** à la valeur pour pouvoir rester dans la boucle).

Au premier tour de boucle, on va donc afficher une seule fois "ping" (`$i` ayant pour valeur 0, et devant être strictement inférieur à `$number` qui est égal à 1). Au second tour, il sera affiché deux fois (pour les valeurs de `$i` égales à 0 puis 1), etc.

En réalité, dans cette boucle, les valeurs du tableau sont le nombre de fois que "ping" sera affiché. Pour avoir le total, il suffit donc d'additionner ces chiffres, ce qui donne 15.

p. 22 Solution n°7

```
1 <?php
2 for($table= 1;$table <= 10;$table++)
3 {
4     for($multiplicateur = 1;$multiplicateur <= 10;$multiplicateur++)
5     {
6         echo $table. 'x' . $multiplicateur. '=' . ($table * $multiplicateur). '<br>';
7     }
8     echo '<br>';
9 }
10 ?>
```