

# Les boucles

# Table des matières

<b>I. Contexte</b>	<b>3</b>
<b>II. Les bases des boucles</b>	<b>3</b>
A. Les boucles “for” .....	3
B. Les boucles “while” .....	4
C. Les boucles “do... while” .....	4
D. L’instruction “break” .....	4
E. L’instruction “continue” .....	6
F. Exercice : Quiz .....	7
<b>III. Essentiel</b>	<b>8</b>
<b>IV. Auto-évaluation</b>	<b>9</b>
A. Exercice .....	9
B. Test .....	9
<b>Solutions des exercices</b>	<b>11</b>

## I. Contexte

**Durée : 1 h**

**Environnement de travail : replit et Visual Studio Code avec XAMPP**

### Contexte

Un aspect essentiel de la programmation est l'automatisation, c'est-à-dire le fait de pouvoir effectuer des traitements de façon automatique. L'automatisation est d'autant plus utile que certaines tâches doivent être accomplies de façon répétitive. Cette notion de répétition est traduite en langage informatique par des boucles, c'est-à-dire un ensemble d'instructions qui est répété plusieurs fois.

Un exemple d'illustration : un changement du taux de TVA nécessite de modifier tous les prix des produits vendus dans un magasin ; une boucle est pratique pour calculer le nouveau prix pour chaque produit.

Au début de ce cours, nous vous présenterons les opérateurs “++” et “--” qui sont très utilisés dans les boucles. Ensuite, vous serez initié au principe des itérations avec des tableaux, ce qui permet d'utiliser des boucles “foreach”. Vous découvrirez les bases générales des boucles et leur mise en pratique avec “for” et “while”. Enfin, nous vous exposerons les instructions “break” et “continue” qui s'utilisent dans les boucles.

## II. Les bases des boucles

### A. Les boucles “for”

Certains développeurs vous diraient que les boucles “for” sont les plus compliquées, mais elles sont simples si vous avez bien compris les 3 phases présentées juste avant : initialisation, condition, actualisation. En effet, pour implémenter une telle boucle en PHP, il convient d'écrire le mot-clé “for” suivi, entre parenthèses, des 3 phases, puis le code qui sera itéré.

### Exemple

Voyons un exemple :

```
1 <?php
2 for ($k = 1 ; $k < 10 ; $k++) { # les 3 phases sont séparées par le caractère “;”
3     echo $k;
4 }
5 # affiche 123456789
6 ?>
```

Dans cet exemple :

- La phase d'initialisation est donnée par “\$k=1” : on définit une variable \$k à la valeur 1.
- La phase de condition est donnée par “\$k<10” : la boucle est réitérée tant que la condition est vraie, dans ce cas tant que la variable \$k est strictement inférieure à 10.
- La phase d'actualisation est donnée par “\$k++” : à chaque itération, la variable \$k est incrémentée de 1.
- Entre accolades, le code constituant la boucle est exécuté à chaque itération, dans cet exemple l'affichage de la valeur de la variable \$k.

## B. Les boucles “while”

Les boucles “while” peuvent parfois être plus intuitives que les boucles “for”, et peuvent paraître plus simples à mettre en œuvre. La phase d’initialisation doit être codée avant la boucle. Le mot-clé “while” est suivi, entre parenthèses, de la phase de test conditionnel. Enfin, la phase d’actualisation doit être codée dans le corps de la boucle. Si vous oubliez cette phase d’actualisation dans une boucle while, cela amène la boucle à... boucler indéfiniment ! Il faut donc bien respecter les 3 phases.

### Exemple

Voyons maintenant l’implémentation avec le même exemple que le précédent :

```
1 <?php
2 $d = 1 ; # phase d'initialisation
3 while ($d < 10){ # phase de test conditionnel
4     echo $d ;
5     $d++ ; # phase d'actualisation
6 }
7 # affiche 123456789
8 ?>
```

Dans cet exemple, on se contente d’afficher la variable \$d, mais dans vos programmes à venir, le corps de la boucle pourra comporter beaucoup plus de lignes.

## C. Les boucles “do... while”

Les boucles “do...while” sont très similaires aux boucles “while”. La différence vient de l’emplacement du test conditionnel. Tandis que la boucle “while” effectue le test conditionnel au début, c’est-à-dire avant d’exécuter le corps de la boucle, la boucle “do...while” effectue le test conditionnel à la fin, c’est-à-dire après avoir exécuté le corps de la boucle. Ainsi, une boucle “while” peut tout à fait ne jamais être exécutée - dans le cas où le test conditionnel est faux dès le départ - alors qu’une boucle “do...while” est toujours exécutée au moins une fois.

### Méthode

Voyons l’usage du “do...while” avec le même exemple, afin de vous permettre de bien comparer les différentes syntaxes :

```
1 <?php
2 $s = 1 ; # phase d'initialisation
3 do {
4     echo $s++ ; # phase d'actualisation combinée à l'affichage
5 } while ($s < 10) ; # phase de test conditionnel
6 # affiche 123456789
7 ?>
```

## D. L’instruction “break”

L’instruction “break” peut être utilisée dans toutes les boucles : foreach, for, while, do...while. Cette commande permet de sortir instantanément de la boucle où elle est placée. C’est un peu une sorte d’astuce de développeur pour terminer rapidement une boucle sans achever l’itération courante.

La commande “break” peut aussi être suivie d’un nombre entier, lorsque plusieurs boucles sont imbriquées, pour indiquer le nombre de boucles à terminer.

**Exemple**

Voyons maintenant quelques exemples sur l'usage de l'instruction "break". Pour le premier exemple, imaginons que nous souhaitons afficher tous les éléments d'un tableau, en considérant que tous ces éléments sont des nombres entiers. Nous allons utiliser l'instruction "break" pour avorter la boucle dans le cas où un élément ne correspond pas au type attendu.

```
1 <?php
2 $tab = [1,2,3,4.5,5,6] ; # initialise un tableau de nombres entiers
3 # regardez le 4.5 parmi les autres nombres entiers, il s'agit d'une erreur volontaire
4 foreach($tab as $v){ # on parcourt le tableau
5     # chaque élément, à chaque itération, est mémorisé dans $v
6     if (!is_int($v)){ # s'il ne s'agit pas d'un nombre entier
7         break; # alors on avorte la boucle
8     }
9     echo $v ; # on affiche l'élément courant qui est un nombre entier
10 }
11 # affiche uniquement "123" puisque la boucle se termine quand on traite l'élément 4.5
12 ?>
```

Nous allons maintenant voir un second exemple, découpé en 3 morceaux. Exécutez d'abord le code suivant afin de voir ce qu'il affiche. Remarquez en particulier la dernière ligne "10-10=0".

```
1 <?php
2 for($a=1 ; $a<=10 ; $a++){ # on prévoit 10 itérations à l'aide de la variable $a
3     $d = 10 ; # on initialise une variable $d
4     do {
5         echo $d, '-', $a, ' = ', $d-$a, "\n" ; # on affiche la soustraction de $d par $a
6         $d-- ; # on décrémente $d
7     } while($d>$a) ; # on répète cette boucle tant que $d est supérieur à $a
8 echo "-----\n" ; # on affiche une ligne de séparation
9 }
10 ?>
```

Maintenant que vous avez compris le fonctionnement de ces deux boucles imbriquées, nous rajoutons un test pour s'assurer que \$a est toujours strictement inférieur à \$d, c'est-à-dire que la soustraction \$d-\$a est toujours positive. Comme vous l'avez vu, la dernière ligne affichée indique un résultat à 0, ce qui va donc déclencher un "break". Dans l'exemple suivant, le "break" va quitter la boucle "while", mais la boucle "for" va continuer, et deux lignes de séparation vont être affichées.

```
1 <?php
2 for($a=1 ; $a<=10 ; $a++){ # on prévoit 10 itérations à l'aide de la variable $a
3     $d = 10 ; # on initialise une variable $d
4     do {
5         if($a>=$d){
6             break 1; # on sort de la boucle while quand $a et $d valent 10
7             # on continue donc la boucle for et affiche la ligne de séparation
8         }
9         echo $d, '-', $a, ' = ', $d-$a, "\n" ; # on affiche la soustraction de $d par $a
10        $d-- ; # on décrémente $d
11    } while($d>$a) ; # on répète cette boucle tant que $d est supérieur à $a
12    echo "-----\n" ; # on affiche une ligne de séparation
13 }
14 ?>
```

Enfin, avec le code suivant, nous avons remplacé le "break 1" par "break 2", qui a pour effet de terminer 2 niveaux de boucles, c'est-à-dire de mettre fin à la boucle "for". Ainsi, contrairement à la version précédente, une seule ligne de séparation est affichée.

```

1 <?php
2 for($a=1 ; $a<=10 ; $a++){ # on prévoit 10 itérations à l'aide de la variable $a
3     $d = 10 ; # on initialise une variable $d
4     do {
5         if($a>=$d){
6             break 2; # on sort de la boucle for quand $a et $d valent 10
7             # on n'affiche plus la ligne de séparation
8         }
9         echo $d, '-', $a, ' = ', $d-$a, "\n" ; # on affiche la soustraction de $d par $a
10        $d-- ; # on décrémente $d
11    } while($d>$a) ; # on répète cette boucle tant que $d est supérieur à $a
12    echo "-----\n" ; # on affiche une ligne de séparation
13 }
14 ?>

```

## E. L'instruction "continue"

L'instruction "continue" ressemble légèrement à l'instruction "break" dans la mesure où l'itération en cours est éludée, c'est-à-dire que les instructions suivantes à l'intérieur de la boucle ne sont pas exécutées. Cependant il y a une grande différence, car l'instruction "break" fait sortir complètement de la boucle, alors que l'instruction "continue" ne fait sauter que la fin de l'itération, amenant à l'itération suivante dans la même boucle.

Lorsque plusieurs boucles sont imbriquées, comme pour l'instruction "break", il est possible d'indiquer une valeur à l'instruction "continue", afin de définir quelle boucle il faut éluder.

### Méthode

Afin de bien illustrer le comportement de l'instruction "continue", nous allons regarder un exemple en 3 étapes, comme nous l'avons fait pour l'instruction "break".

**Étape 1 :** Exécutez le code suivant, inspiré de l'exemple précédent, qui affiche le résultat de la soustraction de deux variables. Tandis que la variable \$a va de 1 à 10, la variable \$d va de 10 à 1, mais \$d restera toujours supérieure à \$a.

```

1 <?php
2 for($a=1 ; $a<=10 ; $a++){ # on prévoit 10 itérations à l'aide de la variable $a
3     for($d=10 ; $d>$a ; $d--){ # on boucle avec une variable $d
4         # on répète cette boucle tant que $d est supérieur à $a
5         echo $d, '-', $a, ' = ', $d-$a, "\n" ; # on affiche la soustraction de $d par $a
6     }
7     echo "-----\n" ; # on affiche une ligne de séparation
8 }
9 ?>

```

**Étape 2 :** Une fois que vous avez bien compris la logique de l'affiche du code précédent, vous allez pouvoir exécuter le code suivant, dans lequel nous avons placé une instruction "continue 1;". Cela a pour effet de terminer l'itération de la boucle intérieure, sans impacter la boucle extérieure. Vous pouvez alors constater que toutes les lignes de séparations sont toujours bien affichées -car elles sont codées dans la boucle extérieure - mais qu'il n'y a plus les lignes des soustractions pour toutes les valeurs de \$d inférieures à 5 - qui sont codées dans la boucle intérieure.

```

1 <?php
2 for($a=1 ; $a<=10 ; $a++){ # on prévoit 10 itérations à l'aide de la variable $a
3     for($d=10 ; $d>$a ; $d--){ # on boucle avec une variable $d
4         # on répète cette boucle tant que $d est supérieur à $a
5         if($d<5){
6             continue 1; # on passe à l'itération suivante de la boucle intérieure quand $d est
7             inférieur à 5
8         }
9         echo $d, '-', $a, ' = ', $d-$a, "\n" ; # on affiche la soustraction de $d par $a
10    }
11    echo "-----\n" ; # on affiche une ligne de séparation
12 }
13 ?>

```

```

8     }
9     echo $d, '-', $a, ' = ', $d-$a, "\n" ; # on affiche la soustraction de $d par $a
10  }
11  echo "-----\n" ; # on affiche une ligne de séparation
12 }
13 ?>

```

**Étape 3 :** Maintenant que vous avez testé l'instruction “continue 1;”, nous allons tester le même code avec l'instruction “continue 2;”. Dans ce cas précis, quand la variable \$d est inférieure à 5, la boucle extérieure passe directement à l'itération suivante. Ainsi le code qui affiche la ligne de séparation n'est pas exécuté. Vous pouvez vérifier qu'en-dessous de la valeur 5 pour \$d, dans l'affichage à l'écran, aucune ligne de séparation n'est affichée.

```

1 <?php
2 for($a=1 ; $a<=10 ; $a++){ # on prévoit 10 itérations à l'aide de la variable $a
3     for($d=10 ; $d>$a ; $d--){ # on boucle avec une variable $d
4         # on répète cette boucle tant que $d est supérieur à $a
5         if($d<5){
6             continue 2; # on passe à l'itération suivante de la boucle extérieure quand $d est
7             inférieure à 5
8             # on n'affiche donc plus la ligne de séparation dans ce cas
9         }
10        echo $d, '-', $a, ' = ', $d-$a, "\n" ; # on affiche la soustraction de $d par $a
11    }
12    echo "-----\n" ; # on affiche une ligne de séparation
13 }
14 ?>

```

## F. Exercice : Quiz

[solution n°1 p.13]

### Question 1

Le code “for (\$i=1;\$i>10;\$i++) {echo \$i;}” affiche :

- ☐ 123456789
- ☐ 12345678910
- ☐ Rien

### Question 2

Combien de fois la boucle « while » suivante est exécutée ?

```

1 <?php
2 $init=10;
3 while($init<20){
4     $init+=2;
5     echo $init;
6 }
7 ?>
8 Fin du listing informatique

```

- ☐ 5 fois
- ☐ 10 fois
- ☐ 20 fois

### Question 3

Le programme suivant affichera « *Boucle terminée.* » à la fin de son exécution.

```
1 <?php
2 $var=10;
3 do{
4     echo $var++;
5 }while($var>0);
6 echo "Boucle terminée.";
7 ?>
```

- ☐ Vrai
- ☐ Faux

#### Question 4

Qu'affiche le programme suivant ?

```
1 <?php
2 $t = [1,2,3,4,5,6];
3 foreach($t as $v){
4     if($v==4){
5         break;
6     }
7     echo $v;
8 }
9 ?>
```

- ☐ 123
- ☐ 12356
- ☐ 123456

#### Question 5

Qu'affiche le programme suivant ?

```
1 <?php
2 $t = [1,2,3,4,5,6];
3 foreach($t as $v){
4     if($v==4){
5         continue;
6     }
7     echo $v;
8 }
9 ?>
```

- ☐ 123
- ☐ 12356
- ☐ 123456

### III. Essentiel

Vous avez découvert les opérateurs “++” et “--” qui s'utilisent en préfixe et suffixe d'un nom de variable, afin d'effectuer des incrémentations et respectivement des décrémentations. Vous avez été initié aux tableaux en tant qu'objet itérable et vous les avez manipulés dans des boucles “foreach”. Vous avez été sensibilisé aux expressions booléennes ainsi qu'aux 3 phases des boucles qui sont l'initialisation, les tests conditionnels et l'actualisation. Vous avez pu voir la mise en pratique de ces notions dans des boucles “for” et “while”. Enfin, les instructions “break”



et “continue” vous ont été présentées avec des exemples concrets, afin d'en comprendre le fonctionnement. Vous avez donc acquis toutes les bases sur les boucles en PHP, et vous pouvez maintenant poursuivre votre apprentissage avec des cas pratiques.

## IV. Auto-évaluation

### A. Exercice

#### Calculs de moyennes

Dans une école, 1 000 étudiants sont évalués 8 fois dans l'année. Pour poursuivre leur formation, ils doivent obtenir une moyenne d'au moins 10 sur 20. Dans ce cas, ils obtiennent le statut « *reçu* ». Dans ce cas pratique, nous considérons le tableau de notes défini aléatoirement dans la variable “*notes*” indiquée ci-après. Il s'agit d'un tableau de 1 000 éléments (chaque élément correspond à un étudiant), où chaque élément est lui-même un tableau de 8 nombres entiers compris entre 0 et 20 (chaque élément correspond à une évaluation).

```
1 <?php
2 $notes = [] ;
3 for ($e=0 ; $e<1000 ; $e++){
4     $notes[$e] = [] ;
5     for ($n=0 ; $n<8 ; $n++){
6         $notes[$e][$n]=random_int(0,20);
7     }
8 }
9 ?>
```

Le but est de connaître le nombre d'étudiants « *reçus* » à partir de la liste de notes fournie.

#### Question 1

[solution n°2 p.14]

Vous devez rédiger un paragraphe qui explique votre raisonnement pour calculer le nombre d'étudiants « *reçus* ».

#### Question 2

[solution n°3 p.15]

Implémentez votre raisonnement pour que votre programme PHP vous indique le nombre de « *reçus* ». Exécutez plusieurs fois votre code pour vous assurer du bon résultat (à chaque exécution les notes seront renouvelées).

### B. Test

#### Exercice 1 : Quiz

[solution n°4 p.15]

##### Question 1

Quel est le dernier affichage du programme suivant ?

```
1 <?php
2 $citation=['l','e','s',' ','b','o','u','c','l','e','s',' ','e','n',' ','P','H','P'];
3 $compteur=0;
4 foreach($citation as $lettre){
5     echo $lettre;
6     if($lettre=="e"){
7         $compteur++;
8     }
9 }
10 echo "\n";
11 echo $compteur;
12 ?>
```

- ☐ 0
- ☐ 3
- ☐ 18

## Question 2

Que va afficher le programme suivant ?

```
1 <?php
2 $s = 0;
3 for($i=12;$i<20;$i+=2){
4     $s += $i;
5 }
6 echo $s;
7 ?>
```

- ☐ 50
- ☐ 60
- ☐ 70

## Question 3

Combien de lignes sont affichées par le programme suivant ?

```
1 <?php
2 $t = 40;
3 while($t>0){
4     echo $t, "\n";
5     $t-=5;
6 }
7 ?>
```

- ☐ 8
- ☐ 40
- ☐ Aucune

## Question 4

Combien de lignes sont affichées par le programme suivant ?

```
1 <?php
2 $t = 40;
3 while($t>0){
4     for($i=($t-10);$i<$t;$i+=2){
5         echo $i, "\n";
6     }
7     $t-=5;
8 }
9 ?>
```

- ☐ 8
- ☐ 40
- ☐ Aucune

## Question 5

Quel est le dernier affichage du programme suivant ?

```
1 <?php
2 $n = [3,2,0,1,2,-2,2,3,-1,0,2,0];
3 $s = 0;
4 foreach($n as $c=>$v){
5     if($v<0){
6         echo "valeur négative détectée en position {$c}\n";
7         continue;
8     }
9     $s += $v ;
10 }
11 echo $s;
12 ?>
```

- ☐ 0
- ☐ 12
- ☐ 15

## Solutions des exercices




## Exercice p. 7 Solution n°1

## Question 1

Le code “for (\$i=1;\$i>10;\$i++) {echo \$i;}” affiche :

- ☐ 123456789
- ☐ 12345678910
- ☒ Rien


 La phase du test conditionnelle “\$i>10” indique que cette boucle s’exécute tant que \$i est supérieur à 10. Or \$i est initialisé à 1, ce qui est inférieur à 10. Donc la boucle n’est jamais exécutée, et n’affiche rien.

## Question 2

Combien de fois la boucle « while » suivante est exécutée ?

```
1 <?php
2 $init=10;
3 while($init<20){
4     $init+=2;
5     echo $init;
6 }
7 ?>
8 Fin du listing informatique
```

- ☒ 5 fois
- ☐ 10 fois
- ☐ 20 fois


 À chaque itération, la commande “echo” affiche la valeur de la variable \$init : 12, 14, 16, 18, 20. La boucle est donc exécutée 5 fois.

## Question 3

Le programme suivant affichera « Boucle terminée. » à la fin de son exécution.

```
1 <?php
2 $var=10;
3 do{
4     echo $var++;
5 }while($var>0);
6 echo "Boucle terminée.";
7 ?>
```

- ☐ Vrai
- ☒ Faux

 Le problème provient du test conditionnel qui sera indéfiniment vrai. En effet, la variable « var » est initialisée à la valeur 10 puis incrémentée de 1 à chaque itération, donc elle sera toujours positive. Ce programme ne termine jamais. Par conséquent, la dernière ligne « Boucle terminée. » n'est jamais affichée.

## Question 4

Qu’affiche le programme suivant ?

```
1 <?php
2 $t = [1,2,3,4,5,6];
3 foreach($t as $v){
4     if($v==4){
5         break;
6     }
7     echo $v;
8 }
9 ?>
```

- ☒ 123
- ☐ 12356
- ☐ 123456

**Q** La boucle “foreach” parcourt le tableau \$t et affiche chaque élément avec l’instruction “echo \$v”. Cependant lorsque l’élément vaut 4, l’instruction “break” est exécutée, ce qui a pour effet de terminer la boucle immédiatement. Par conséquent, l’instruction “echo \$v” ne s’exécute que pour les 3 premiers éléments du tableau, affichant donc 123.

### Question 5

Qu’affiche le programme suivant ?

```
1 <?php
2 $t = [1,2,3,4,5,6];
3 foreach($t as $v){
4     if($v==4){
5         continue;
6     }
7     echo $v;
8 }
9 ?>
```

- ☐ 123
- ☒ 12356
- ☐ 123456

**Q** La boucle “foreach” parcourt le tableau \$t et affiche chaque élément avec l’instruction “echo \$v”. Cependant lorsque l’élément vaut 4, l’instruction “continue” est exécutée, ce qui a pour effet de terminer l’itération immédiatement, et de poursuivre la boucle à l’itération suivante. Par conséquent, l’instruction “echo \$v” s’exécute pour les 3 premiers éléments du tableau, ainsi que pour les 2 derniers, affichant donc 12356.

### p. 9 Solution n°2

Au départ de cette problématique, nous avons un tableau, c'est-à-dire un objet itérable. Nous pouvons donc utiliser une boucle « foreach » pour itérer sur ce tableau d'étudiants. Afin de calculer la moyenne de chaque étudiant, nous pouvons utiliser une seconde boucle « foreach » en itérant sur chaque tableau de notes (pour un étudiant donné). Nous utiliserons une variable 'moyenne' pour additionner les 8 notes de l'étudiant puis en diviser le résultat par le nombre de notes, c'est-à-dire par 8. Une fois que la moyenne est calculée, il faudra la comparer à 10, afin de savoir si cet étudiant est reçu. Dans ce cas, nous devons mémoriser que cet étudiant est reçu (pour le comptabiliser à la fin). Pour cela, nous utiliserons une variable 'recu' qui sera initialisée à 0. Puis, lorsqu'une moyenne sera supérieure ou égale à 10, nous incrémenterons "recu" de 1. Enfin, après avoir traité tous les étudiants de la liste, nous afficherons le résultat de la variable "recu".

## p.9 Solution n°3

Voici un programme qui répond au sujet :

```
1 <?php
2 $recu = 0 ; # initialisation d'une variable pour le nombre de reçus
3 foreach ($notes as $e){ # pour chaque étudiant
4     $moyenne = 0 ; # initialisation d'une variable pour chaque étudiant
5     foreach ($e as $n){ # pour chaque note
6         $moyenne += $n ; # on ajoute la note à cette variable
7     }
8     # on est maintenant sortie de la boucle des notes pour un étudiant
9     $moyenne = $moyenne / 8 ; # on divise la somme des notes par le nombre de notes
10    if($moyenne>=10){ # si la moyenne est au moins de 10
11        $recu++ ; # alors cet étudiant est reçu
12    }
13 }
14 echo $recu ; # à la sortie de la boucle des étudiants, on affiche le nombre de reçus
15 ?>
```

## Exercice p.9 Solution n°4

## Question 1

Quel est le dernier affichage du programme suivant ?

```
1 <?php
2 $citation=['l','e','s',' ','b','o','u','c','l','e','s',' ','e','n',' ','P','H','P'];
3 $compteur=0;
4 foreach($citation as $lettre){
5     echo $lettre;
6     if($lettre=="e"){
7         $compteur++;
8     }
9 }
10 echo "\n";
11 echo $compteur;
12 ?>
```

- ☐ 0
- ☒ 3
- ☐ 18



Avec la boucle “foreach”, nous itérons sur chaque élément du tableau \$citation. Chaque élément est une lettre. Lorsqu’un élément est un “e”, la variable \$compteur est incrémentée. À la fin du programme, la variable \$compteur qui est affichée correspond donc au nombre de lettres “e” présentes dans le tableau \$citation.

## Question 2

Que va afficher le programme suivant ?

```
1 <?php
2 $s = 0;
3 for($i=12;$i<20;$i+=2){
4     $s += $i;
5 }
6 echo $s;
7 ?>
```

- ☐ 50
- ☒ 60
- ☐ 70

La boucle «`for`» initialise la variable `$i` à 12, puis l'actualise à chaque itération en lui ajoutant 2. Cette boucle est exécutée tant que `$i` est inférieur à 20, ainsi la variable `$i` prend successivement les valeurs 12, 14, 16, 18. La variable `$s` est initialisée à la valeur 0, et est incrémentée à chaque itération avec la valeur de `$i`. À la fin, la variable `$s` correspond donc à la somme des 4 termes  $12 + 14 + 16 + 18$ , soit 60.

### Question 3

Combien de lignes sont affichées par le programme suivant ?

```
1 <?php
2 $t = 40;
3 while($t>0){
4     echo $t, "\n";
5     $t-=5;
6 }
7 ?>
```

- ☒ 8
- ☐ 40
- ☐ Aucune

La phase d'initialisation affecte la valeur 40 à la variable `$t`. Dans la boucle, la variable `$t` est décrémentée de 5 à chaque itération. La condition d'exécution de la boucle impose que la variable `$t` soit strictement positive. Ainsi, à chaque itération, la variable `$t` vaut successivement les valeurs : 40, 35, 30, 25, 20, 15, 10, 5. Ce programme affiche donc ces 8 valeurs.

### Question 4

Combien de lignes sont affichées par le programme suivant ?

```
1 <?php
2 $t = 40;
3 while($t>0){
4     for($i=($t-10); $i<$t; $i+=2){
5         echo $i, "\n";
6     }
7     $t-=5;
8 }
9 ?>
```

- ☐ 8
- ☒ 40
- ☐ Aucune

Comme pour la question précédente, la boucle «`while`» est exécutée 8 fois. L'affichage est exécuté dans la boucle «`for`» qui exécute 5 itérations. En effet, la variable `$i` va successivement prendre les valeurs «`$t-10`», «`$t-8`», «`$t-6`», «`$t-4`», «`$t-2`», soient 5 éléments. La commande «`echo`» est donc exécutée  $8 \times 5$ , soit 40 fois.

### Question 5

Quel est le dernier affichage du programme suivant ?



```
1 <?php
2 $n = [3,2,0,1,2,-2,2,3,-1,0,2,0];
3 $s = 0;
4 foreach($n as $c=>$v){
5     if($v<0){
6         echo "valeur négative détectée en position {$c}\n";
7         continue;
8     }
9     $s += $v ;
10 }
11 echo $s;
12 ?>
```

- ☐ 0
- ☐ 12
- ☒ 15

**Q** Une variable `$s` est initialisée à 0. La boucle “`foreach`” parcourt le tableau `$n`. Chaque élément du tableau, mémorisé dans la variable `$v`, est ajouté à `$s` avec la ligne “`$s += $v`”. La variable `$s` est donc la somme des termes du tableau. Cependant, l’addition des termes négatifs est sautée. En effet, lorsqu’un terme est négatif, l’instruction “`continue`” est exécutée, passant à l’itération suivante. L’affichage final correspond donc à la somme des valeurs positives du tableau,  $3 + 2 + 0 + 1 + 2 + 2 + 3 + 0 + 2 + 0$ , soit 15.