

# Les boucles

# Table des matières

<b>I. Contexte</b>	<b>3</b>
<b>II. La boucle for</b>	<b>3</b>
A. Console log .....	3
B. Première utilisation de la boucle for .....	5
C. Utiliser for/in .....	7
D. Utiliser for/of .....	8
E. Exercice .....	8
<b>III. La boucle while</b>	<b>9</b>
A. La boucle while.....	9
B. Exercice .....	13
<b>IV. Essentiel</b>	<b>14</b>
<b>V. Auto-évaluation</b>	<b>15</b>
A. Exercice .....	15
B. Test.....	15
<b>Solutions des exercices</b>	<b>16</b>

## I. Contexte

Comme vous le savez, en programmation, il existe des mécanismes fondamentaux comme les structures de contrôle. Celles-ci permettent d'adapter le comportement d'un programme à des interactions en contrôlant le résultat de conditions. Cependant, les structures **if/else** et **switch** ne sont pas les seuls mécanismes fondamentaux permettant d'adapter le comportement d'un programme à des interactions.

Les systèmes de boucles sont plus qu'essentiels. Une boucle permet de répéter des instructions un nombre de fois donné, ou en fonction de conditions. En JavaScript, la boucle **for** et la boucle **while** sont des systèmes de boucles très connus. Nous verrons différentes manières d'utiliser ces deux systèmes de boucles. Pour cela, nous utiliserons des exemples de codes. Les boucles en programmation informatique sont des structures de contrôle qui permettent d'exécuter une série d'instructions plusieurs fois de suite, en fonction d'une condition spécifique. Elles sont utilisées pour automatiser des tâches répétitives et pour traiter des données en itérant à travers des collections d'objets.

Ce cours sera composé par ailleurs d'exercices pratiques que vous pourrez réaliser depuis l'espace Replit. Prenez bien le temps de vous familiariser avec les notions de boucles qui sont vraiment importantes en programmation, à l'aide de ces exercices, mais aussi des vidéos explicatives. Vous pourrez constater que la syntaxe des boucles est à nouveau assez proche d'autres langages de programmation comme Java ou PHP. Vous familiariser dès à présent avec ces notions vous permettra donc de maîtriser ces concepts essentiels en développement web et plus généralement, en scripting.

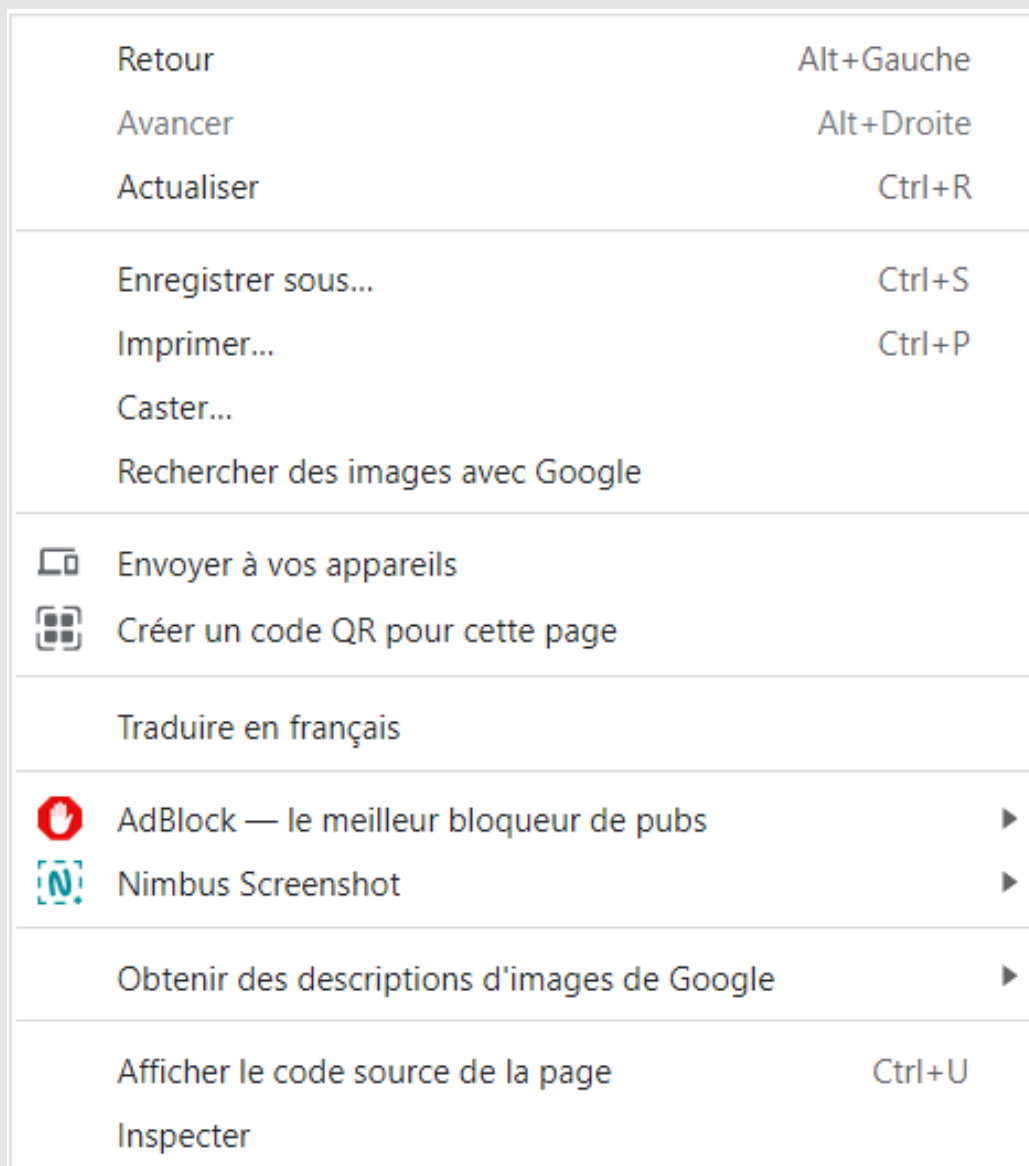
## II. La boucle for

### A. Console log

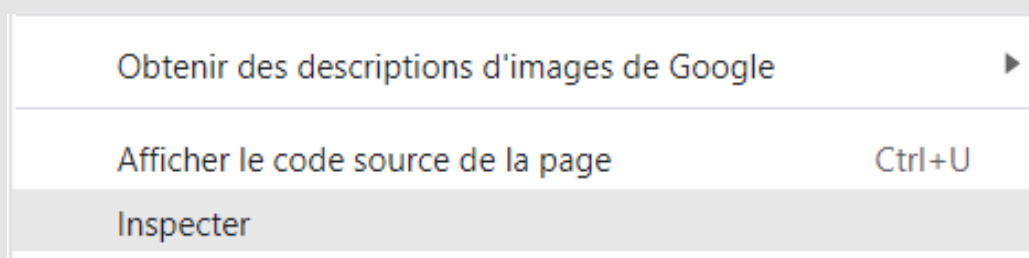
<b>Méthode</b>	<b>Atteindre le console log</b>
----------------	---------------------------------

Dans le cadre de ce cours, vous serez amené à utiliser nos exemples pour comprendre de quoi il retourne et avoir un retour visuel sur le code. Pour ce faire, vous devez :

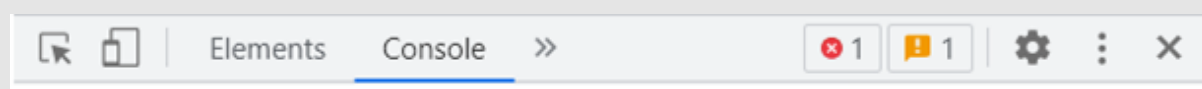
- Effectuer un clic droit sur la page



- Sélectionner l'option « *Inspecter* »



- Vous rendre dans « *Console* »



Cela fait, vous aurez un aperçu des informations supposées se trouver dans `console.log()`.

## B. Première utilisation de la boucle for

### Définition La boucle for

La boucle **for** est une boucle vraiment importante en programmation. Pour faire simple, disons qu'on peut l'utiliser dans tous les cas où, juste avant l'exécution de la boucle, le nombre d'itérations (de tours de boucle) est fixé. Autrement dit, le nombre d'itérations ne changera pas ou ne sera pas déterminé lors de l'exécution des itérations de la boucle.

### Méthode Première façon d'utiliser la boucle for

La première manière d'utiliser la boucle **for** se fait avec cette syntaxe :

```
1 for (/*initialisation*/; /*condition*/; /*incrementation*/) {  
2   //instructions;  
3 }
```

Dans les paramètres du **for**, nous allons spécifier :

- **Initialisation** : nous allons initialiser une variable qui servira de compteur à notre boucle.
- **Condition** : nous allons définir la condition qui déterminera le nombre d'itérations.
- **Incrémentation** : nous allons définir l'incrément de notre compteur.

### Exemple

Pour bien comprendre, voyons un exemple simple :

```
1 for (let i = 1; i <= 10; i++) {  
2   console.log (i);  
3 }
```

Dans cet exemple, dans les paramètres du **for**, nous initialisons le compteur **i** en le définissant sur le nombre 1. Nous indiquons ensuite que la boucle devra se répéter tant que **i** est inférieure ou égale à 10. Enfin, nous incrémentons à chaque tour de boucle la variable **i**. L'instruction à répéter est l'affichage dans la console de la variable **i**. On peut voir que la boucle fonctionne ainsi :

1. Le programme vérifie si **i** est inférieure ou égale à 10. Or **i** est égale à 1. L'expression **i <= 10** renvoie donc **true**. Une première itération de la boucle est lancée et la console affiche **i**, donc 1.
2. Le programme vérifie si **i** est inférieure ou égale à 10. Or, **i** a été incrémentée et est maintenant égale à 2. L'expression **i <= 10** renvoie donc **true**. Une deuxième itération de la boucle est lancée et la console affiche **i**, donc 2.
3. Etc.

Finalement, lorsque 10 tours de boucles ont été effectués, la variable **i** est incrémentée et vaut **11**. L'expression **i <= 10** renvoie donc **false**. La boucle s'arrête alors.

On peut constater qu'avant chaque tour de boucle, la condition est testée et le résultat détermine si une nouvelle itération aura lieu ou non.

Avant de voir un autre exemple un peu plus complexe, introduisons rapidement le concept de tableau.

### Définition Tableau

Pour faire simple, en JavaScript, un tableau est une liste d'éléments ordonnés stockés dans une variable. On utilise les crochets pour stocker des valeurs dans un tableau.

### Exemple

Par exemple :

```
1 let tableau = ["valeur 1", "valeur 2"];
```

Chaque élément du tableau est accessible via son index, sachant que le premier élément du tableau à pour index 0, le second 1, le troisième 2, etc.

### Exemple

On peut récupérer un élément de ce tableau comme ceci :

```
1 tableau[1]; // = "valeur 2"
```

### Méthode

Maintenant prenons un autre exemple de boucle **for** interagissant avec un tableau :

```
1 const nombre = 10;
2
3 let table = [];
4
5 for (let i = 1; i <= 10; i++) {
6   let resultat = nombre * i;
7   table.push(resultat);
8 }
```

Ici, l'objectif de notre boucle est de dresser à l'intérieur d'un tableau et plus particulièrement dans notre cas faire la table de multiplication d'un nombre. Nous commençons par définir la variable **nombre**, puis nous déclarons un tableau (avec les crochets) que nous appelons **table**.

Celui-ci contiendra tous les nombres de la table de **nombre**. Dans le **for**, on ne change rien, car on souhaite que la première valeur de **i** soit 1 et qu'il y ait 10 itérations. À chaque tour de boucle, nous déclarons et définissons une variable **resultat** qui stockera le produit de **nombre** par **i** (qui vaudra 1 puis 2, puis 3, etc., jusqu'à 10).

On ajoute ensuite à chaque itération le résultat dans notre tableau **table**. On peut voir que ce script fonctionne bien, d'ailleurs, si on fait un **console.log** de la 4<sup>e</sup> valeur de **table** (donc avec l'index 3), on obtient :

```
1 const nombre = 10;
2
3 let table = [];
4
5 for (let i = 1; i <= 10; i++) {
6   let resultat = nombre * i;
7   table.push(resultat);
8 }
9
10 console.log(table[3]); // = 40 (10 x 4)
```

Bien évidemment, nous pourrions utiliser les boucles avec des paramètres et des instructions complètement différentes, en fonction des nécessités. Maintenant, voyons une autre manière d'utiliser la boucle **for**.

## C. Utiliser for/in

### Méthode

Le système **for/in** va nous permettre d'utiliser la boucle **for** à travers les propriétés d'un objet. La syntaxe est la suivante :

```
1 for (/*variable*/ in /*objet*/) {  
2   //instructions;  
3 }
```

Le concept d'objet en programmation ne vous est peut-être pas familier et c'est normal. Pour l'instant, sans rentrer dans les détails, dites-vous qu'un objet est une entité ; un ensemble de données qui contient des propriétés qui le caractérisent. Prenons un exemple :

### Exemple

```
1 let animation = new Animation();  
2  
3 for (let i in animation) {  
4   console.log(i);  
5 }
```

Ici, nous créons un objet que l'on appelle **animation** (c'est le nom de la variable) et il est de type **Animation**. Cet objet a de nombreuses propriétés, et, avec la boucle **for**, on peut parcourir chacune de ces propriétés pour en afficher une par une leur nom (et non leur valeur). Dans les paramètres, nous déclarons la variable **i** qui représentera à chaque tour de boucle une propriété dans l'objet **animation**. On peut voir que la console affiche le nom de toutes les propriétés de l'objet. Toutefois, ce concept reste peut-être un peu flou. Voyons ce que cela donne si on parcourt un tableau avec cette méthode :

### Exemple

```
1 let tableau = ["Apple", "HP", "Acer"];  
2  
3 for (let i in tableau) {  
4   console.log(i);  
5 }
```

On peut voir que la console affiche 0, 1, 2. Pourquoi ? Tout simplement parce que mon tableau a 3 propriétés correspondant aux index des éléments qui le composent. En ce sens, cela pourrait être intéressant d'utiliser ce système pour afficher la valeur des éléments de notre tableau comme ceci :

```
1 let tableau = ["Apple", "HP", "Acer"];  
2  
3 for (let i in tableau) {  
4   console.log(tableau[i]);  
5 }
```

Effectivement, cela fonctionne, la console affiche chaque valeur du tableau. Cependant, il n'est pas préconisé de procéder ainsi pour récupérer les valeurs d'un tableau. Pourquoi ? Tout bonnement parce que le **in** permet de parcourir les propriétés du tableau. Si l'on ajoute des propriétés à notre tableau (qui ne sont pas nécessairement des éléments du tableau), alors la boucle **for** les parcourra aussi, ce qui posera un problème (puisque l'on aura les éléments du tableau et les autres propriétés). En ce sens, quand on veut parcourir un tableau, il est important d'utiliser le système **for/of**.

## D. Utiliser for/of

### Méthode

La boucle **for/of** permet de parcourir un objet itérable et de récupérer ses valeurs. C'est le cas par exemple des tableaux. Si l'on reprend notre exemple précédent et que l'on cherche à afficher dans la console chaque élément du tableau, on peut faire simplement :

```
1 let tableau = ["Apple", "HP", "Acer"];
2
3 for (let i of tableau) {
4   console.log(i);
5 }
```

Cela fonctionne, car à chaque tour de boucle, **i** prend la valeur d'un élément de tableau, et la boucle parcourt tous les éléments du tableau. Avec la boucle **for/of**, on peut donc rapidement récupérer les valeurs d'un tableau.

### Méthode

Modifions-la pour que notre programme permette de concaténer dans une chaîne de caractères les valeurs des différents éléments du tableau :

```
1 let tableau = ["Apple", "HP", "Acer"];
2
3 let marques = "";
4
5 for (let i of tableau) {
6   marques += i + ", ";
7 }
8
9 console.log (marques);
```

On peut voir que la console affiche : Apple, HP, Acer,

À chaque tour de boucle, le programme ajoute à la fin de la chaîne de caractères **marques** la valeur de **i** qui contient une valeur d'un élément du tableau. Tous les éléments du tableau sont parcourus.

## E. Exercice

### Question 1

[solution n°1 p.17]

Insérer la bonne condition pour que la chaîne de caractères **nombres** contienne les nombres entiers de 120 (inclus) à 130 (inclus).

```
1 let nombres = "";
2
3 nb = 120;
4
5 for (/*condition*/) {
6   nombres += nb + " ";
7   nb ++;
8 }
9
10 console.log (nombres);
```



**Question 2**

[solution n°2 p.17]

Écrire une boucle pour que la variable **totale** contienne l'addition de tous les prix présents dans le tableau **prix** :

```
1 let total = 0;
2
3 let prix = [27, 36, 89, 18, 25];
4
5 //boucle
6
7 console.log (total);
```

**Question 3**

[solution n°3 p.17]

Écrire une boucle permettant de stocker dans une chaîne de caractères (**multiples**) tous les 20 premiers multiples de la variable **nombre**. Dans la chaîne, ils seront séparés par un espace.

```
1 let nombre = 8;
2
3 let multiples = "";
4
5 //boucle
6
7 console.log (multiples);
```

**Question 4**

[solution n°4 p.18]

Écrire une boucle permettant de dire pour chaque item du tableau **marque** si la marque est valide. Les marques acceptées sont : « Apple », « HP », « Dell » et « Microsoft ». Si la marque est valide, la console devra afficher « La marque est valide », sinon « La marque n'est pas valide ».

```
1 let marques = ["Apple", "Acer", "HP", "Packard-Bell"];
2
3 //boucle
```

**Question 5**

[solution n°5 p.18]

Dans ce script, on crée un objet **ordinateur**. Sans aller dans les détails, on comprend juste que cet ordinateur va avoir deux propriétés : **ram** et **stockage**. Écrire une boucle qui permet d'afficher dans la console pour chaque propriété, le nom de la propriété et sa valeur. Par exemple : "ram = 256".

```
1 let ordinateur = new Object();
2
3 ordinateur.ram = 256;
4
5 ordinateur.stockage = 512;
6
7 //boucle
```

### III. La boucle while

#### A. La boucle while

**Définition**

À la différence de la boucle **for**, nous utiliserons en général la boucle **while** quand, juste avant l'exécution de la première itération de la boucle, le nombre d'itérations n'est pas encore fixé. Cette règle n'est toutefois pas absolue. En réalité, dans de nombreux cas où l'on utilise une boucle **for**, il est également possible d'utiliser une boucle **while**. Voyons comment utiliser la boucle **while**.

## Méthode Utilisation de la boucle while

La syntaxe de la boucle **while** est :

```
1 while (/*condition*/) {
2   //instructions;
3 }
```

On indique simplement en paramètre du **while** la condition pour qu'une itération se produise. Le programme commence par vérifier si la condition renvoie **true** et si c'est le cas, elle exécute les instructions de la boucle. Puis, le programme vérifie à nouveau la condition et si elle renvoie **true**, alors, les instructions sont à nouveau exécutées, ainsi de suite.

On voit que le système de **while** est assez simple à comprendre. Prenons un exemple simple :

```
1 let nombre = 1;
2
3 while (nombre <= 10) {
4   console.log ("Le nombre est : " + nombre);
5   nombre += 2;
6   nombre -=0.5;
7 }
8
9 console.log (nombre)
```

Ici, nous définissons une variable **nombre** sur la valeur de 1. Puis, dans le **while**, nous passons en paramètre l'expression conditionnelle **nombre <= 10**. Tant que **nombre** sera inférieure ou égale à 10, alors la boucle continuera de s'exécuter. On peut voir que dans les instructions, on commence par afficher le nombre dans une chaîne concaténée, puis on lui ajoute 2 et on lui retire 0,5. Cet ensemble d'instructions sera exécuté à chaque itération. Toutefois, en quel sens peut-on dire que le nombre d'itérations n'est pas fixé avant la première itération ?

Si on regarde la boucle, on peut voir que, hors instructions (c'est-à-dire hors du corps de la boucle), le nombre d'itérations est inconnu. Ce sont, dans cet exemple, les instructions de la boucle qui permettent de déterminer le nombre d'itérations en fonction de la condition (car on modifie la variable **nombre** dans les instructions du **while**, tandis que la condition porte sur cette même variable **nombre**). Dans ce cas, il est donc judicieux d'utiliser la boucle **while**. À la différence, quand nous utilisons la boucle **for**, le nombre d'itérations est fixé dans les paramètres de la boucle (sauf si on ajoute des instructions de type **break** ou **continue** que nous aborderons). Pour notre **while**, la console affichera donc :

- Le nombre est : 1
- Le nombre est : 2.5
- Le nombre est : 4
- Le nombre est : 5.5
- Le nombre est : 7
- Le nombre est : 8.5
- Le nombre est : 10
- 11.5

On peut voir qu'après la 7<sup>e</sup> itération, le programme vérifie la valeur de **nombre** qui est maintenant égale à 11.5. La variable **nombre** étant supérieure à 10, le programme sort de la boucle.

## Utilisation de la boucle do/while

Comme vous avez pu le voir, lorsque l'on utilise un **while**, le programme commence par vérifier la condition puis, si l'expression conditionnelle renvoie **true**, il lance une première itération de la boucle, sinon, aucune itération n'est lancée.

**Exemple**

Prenons l'exemple suivant :

```
1 let cv = 100;
2
3 while (cv > 100 && cv <= 110) {
4   console.log ("La voiture a " + cv + " cv");
5   cv ++
6 }
```

On peut voir dans ce script que la condition du **while** est que **cv** soit strictement supérieure à 100 et inférieure ou égale à 110. Concrètement, la boucle ne va jamais se lancer, car **cv** est égale à 100.

Le programme vérifie l'expression **cv > 100 && cv <= 110** et, étant donné qu'elle renvoie **false**, alors aucune itération de la boucle n'est opérée. Il nous faudrait donc un système qui lance une première itération de la boucle avant de contrôler l'expression conditionnelle. C'est ce que permet le **do/while**.

**Exemple Do/while**

```
1 do {
2   //instructions;
3 } while (/*condition*/);
```

Si on modifie notre exemple :

```
1 let cv = 100;
2
3 do {
4   console.log ("La voiture a " + cv + " cv");
5   cv ++;
6 } while (cv > 100 && cv <= 110);
```

Cela fonctionne et la console affiche :

- La voiture a 100 cv
- La voiture a 101 cv
- La voiture a 102 cv
- La voiture a 103 cv
- La voiture a 104 cv
- La voiture a 105 cv
- La voiture a 106 cv
- La voiture a 107 cv
- La voiture a 108 cv
- La voiture a 109 cv
- La voiture a 110 cv

Le **do/while** permet donc d'opérer une première itération de la boucle avant de vérifier la condition.

**Éviter les boucles infinies**

Abordons maintenant une erreur à éviter, les boucles infinies.

Un problème qui peut avoir lieu est celui des boucles infinies : des boucles qui ne s'arrêtent jamais et qui font planter le programme. Une boucle infinie a lieu quand l'expression de la condition d'un **while** ne cesse jamais de renvoyer **true**, c'est-à-dire qu'il n'y a aucune instruction qui va rendre cette condition fausse.

### Attention

Nous allons prendre un exemple mais sachez qu'il est déconseillé de tester la manipulation, car cela risque de faire planter votre navigateur.

### Exemple

```
1 let a = 120;
2
3 while (typeof(a) == "number" && (a >= 100 && a <= 140)) {
4   console.log(a);
5 }
```

Dans cet exemple, la condition du **while** spécifie que le type de la variable **a** doit être un nombre, et que **a** doit être comprise entre 100 (inclus) et 140 (inclus). Étant donné que la variable **a** est égale à 120, cette condition renvoie **true**. Toutefois, le problème est qu'aucune instruction ne vient dans le **while** changer la valeur de **a**. La condition sera alors toujours vérifiée (elle renverra toujours **true**), et la boucle ne s'arrêtera pas tant que le programme n'est pas stoppé. En revanche, si l'on ajoute une incrémentation de la variable dans la boucle, la boucle ne sera plus infinie et le code fonctionnera correctement.

### Exemple

```
1 let a = 120;
2
3 while (typeof(a) == "number" && (a >= 100 && a <= 140)) {
4   console.log(a);
5   a ++;
6 }
```

## Utilisation de break et continue

Le mot clé **break** permet de sortir d'une boucle, tandis que le mot clé **continue** permet de reprendre l'exécution d'une boucle, de passer directement à l'instruction suivante.

### Exemple

Prenons l'exemple où on insère une condition dans une boucle **while** par exemple :

```
1 let a = 110;
2
3 while (a >= 100 && a <= 150) {
4
5   console.log(a);
6   a += 10;
7
8   if (a == 130) {
9     continue;
10  }
11  else if (a == 140) {
12    break;
13  }
14
15  console.log("nv");
16 }
17
18 console.log("fini");
```

Essayons de comprendre comment fonctionne le programme. La console affiche :

- 110
- nv
- 120
- 130
- fini

La variable **a** a pour valeur 110. Le **while** vérifie la condition **a >= 100 && a <= 150** qui renvoie **true**. Une première itération de la boucle est donc lancée. La variable **a** est affichée dans la console (donc 110) et le programme lui ajoute 10 (**a** est maintenant égale à 120). Les deux conditions vérifient si **a** est égale à 130 et si **a** est égale à 140, mais ce n'est pas le cas. Le programme passe à l'instruction suivante **console.log("nv")** qui affiche donc « nv » dans la console.

Le **while** vérifie à nouveau la condition **a >= 100 && a <= 150**. Elle renvoie **true**, donc une nouvelle itération de la boucle est exécutée. La variable **a** est affichée dans la console (donc 120) et le programme lui ajoute 10 (**a** est maintenant égale à 130). La condition **if** vérifie si **a** est égale à 130. Comme c'est le cas, l'instruction du **if** est exécutée, donc **continue**. Cette instruction permet de ne pas exécuter la suite de l'itération et de passer à la prochaine itération de la boucle.

Le **while** vérifie à nouveau la condition **a >= 100 && a <= 150**. Elle renvoie **true**, donc une nouvelle itération de la boucle est exécutée. La variable **a** est affichée dans la console (donc 130) et le programme lui ajoute 10 (**a** est maintenant égale à 140). La condition **if** vérifie si **a** est égale à 130, mais ce n'est pas le cas, le **else if** est donc traité. Le programme vérifie si **a** est égale à 140, et c'est bien le cas, donc l'instruction du **else if** est exécutée, c'est-à-dire le **break**. Cette instruction fait s'arrêter la boucle et continue le script (après la boucle).

Le **console.log("fini")** est donc exécuté et la console affiche cette chaîne.

Voilà, c'est un exemple assez basique, mais on peut voir que les instructions **break** et **continue** vont permettre de rajouter du potentiel aux boucles. Bien évidemment, ces mots clés peuvent être utilisés avec la boucle **for**.

## B. Exercice

### Question 1

[solution n°6 p.18]

Écrire la bonne condition pour que le **while** affiche les multiples de **nombre** inférieur à 200.

```
1 let nombre = 28;
2 let compteur = 1;
3
4 while (/*condition*/) {
5   console.log(nombre * compteur);
6   compteur ++;
7 }
```

### Question 2

[solution n°7 p.19]

Transformer la boucle suivante pour qu'une itération soit opérée avant que le programme contrôle la condition.

```
1 let prix = 100;
2 let tabPrix = [];
3
4 while (prix > 100 && prix <= 110) {
5   tabPrix.push(prix);
6   prix ++;
7 }
8
9 console.log(tabPrix[2]);
```

### Question 3

[solution n°8 p.19]

Insérer le bon mot clé pour que, quand le prix est égal à 108, la boucle s'arrête.

```
1 let prix = 101;
2 let tabPrix = [];
3
4 while (prix > 100 && prix <= 110) {
5   tabPrix.push(prix);
6   if (prix == 108) {
7     /*mot cle*/;
8   }
9   prix ++;
10 }
11
12 console.log(tabPrix[7]);
```

### Question 4

[solution n°9 p.19]

Insérer une instruction pour que la boucle ne soit pas infinie. L'objectif du script est d'afficher les nombres entiers de 0 (inclus) à 19 (inclus).

```
1 let compteur = 0;
2
3 let nombres = "";
4
5 while (compteur < 20) {
6   nombres += compteur + " ";
7   /*code*/;
8 }
9
10 console.log (nombres);
```

### Question 5

[solution n°10 p.20]

Écrire une boucle **while** permettant de dire pour tous les nombres entiers inférieurs à **nombre** (en démarrant à 1) si ce sont des multiples ou non de 2. Pour chaque nombre, la console devra afficher « *multiple de 2* » si c'est un multiple de 2 et « *pas un multiple de 2* » si ce n'est pas le cas.

```
1 let nombre = 15;
2
3 //code
```

## IV. Essentiel

Dans ce cours, nous avons parlé des boucles en JavaScript. Les boucles permettent de réitérer des instructions, ce qui évite d'avoir à écrire toutes les itérations une par une. Il existe plusieurs boucles dans de nombreux langages de programmation, que l'on retrouve en JavaScript.

Tout d'abord, on retrouve la boucle **for** qui est, en règle générale, utilisée lorsque, juste avant la première itération de la boucle, le nombre d'itérations est fixé. On peut l'utiliser en spécifiant 3 paramètres : l'initialisation d'un compteur, la condition et l'incrément du compteur. On peut aussi l'utiliser avec le mot clé **in** permettant de parcourir les propriétés d'un objet et le mot clé **of** permettant de parcourir les valeurs d'un objet étable, par exemple les valeurs des éléments d'un tableau.

Nous avons ensuite la boucle **while** qui, quant à elle, est en général utilisée quand le nombre d'itérations dépend des instructions présentes dans le corps de la boucle. Elle est donc utilisée quand, juste avant la première itération, le nombre total d'itérations n'est pas fixé. Il faut simplement spécifier une condition en paramètre qui permettra d'exécuter les itérations de la boucle tant que cette condition renvoie **true**.

Le système **do/while** permet d'opérer une première itération de la boucle avant que la condition soit contrôlée.

Le mot clé **break** permet de stopper l'exécution d'une boucle et de passer aux instructions suivantes (après la boucle). Le mot clé **continue** permet d'interrompre une itération de la boucle à un endroit donné pour passer directement à l'itération suivante.

Enfin, un point auquel il faut constamment veiller est celui d'éviter absolument les boucles infinies qui font planter le programme.

## V. Auto-évaluation

### A. Exercice

Vous cherchez à réaliser un script permettant de dresser la liste de tous les diviseurs entiers d'une liste de nombre. Vous avez un script de départ :

```
1 const list = [24, 67, 18];
2
3 let resultat = "";
4
5 //code
6
7 console.log(resultat);
```

Un système de boucles vous permettra d'afficher dans la console tous les diviseurs de chaque nombre dans la liste **list**. Par exemple, dans le cas présent, la console affichera :

```
1 Diviseurs entiers de 24 : 1, 2, 3, 4, 6, 8, 12, 24,
2
3 Diviseurs entiers de 67 : 1, 67,
4
5 Diviseurs entiers de 18 : 1, 2, 3, 6, 9, 18,
```

#### Question 1

[solution n°11 p.20]

Écrire un script qui remplit ce rôle. Vous serez amenés à imbriquer une boucle dans une boucle. Même si le plus conventionnel serait d'utiliser 2 boucles **for**, utiliser ici une boucle **for** et une boucle **while**.

#### Question 2

[solution n°12 p.21]

Réaliser le même système en remplaçant la boucle **while** imbriquée par une boucle **for**.

### B. Test

#### Exercice 1 : Quiz

[solution n°13 p.21]

##### Question 1

Quelle est la boucle généralement utilisée lorsque le nombre d'itérations est fixé avant la première itération ?

- ☐ for
- ☐ while
- ☐ do/while

##### Question 2

Quelle est la boucle à privilégier pour parcourir toutes les valeurs d'un tableau ?

- ☐ for/of
- ☐ for/in
- ☐ while

##### Question 3

Quelle est la boucle permettant de parcourir toutes les propriétés d'un objet ?

- ☐ for/of
- ☐ for/in
- ☐ while

Question 4

Quelle instruction permet de stopper l'exécution d'une boucle ?

- ☐ case
- ☐ break
- ☐ continue

Question 5

Quelle boucle permet d'exécuter une première itération avant de contrôler la condition ?

- ☐ for
- ☐ do/while
- ☐ while

## Solutions des exercices



**p. 8 Solution n°1****Sortie attendue :** "120 121 122 123 124 125 126 127 128 129 130 "**Solution possible 1 :**

```
1 let nombres = "";
2
3 nb = 120;
4
5 for (let i = 0; i < 11; i++) {
6   nombres += nb + " ";
7   nb++;
8 }
9
10 console.log (nombres);
```

**Solution possible 2 :**

```
1 let nombres = "";
2
3 nb = 120;
4
5 for (let i = 0; i <= 10; i++) {
6   nombres += nb + " ";
7   nb++;
8 }
9
10 console.log (nombres);
```

**p. 9 Solution n°2****Sortie attendue :** 195**Solution possible :**

```
1 let total = 0;
2
3 let prix = [27, 36, 89, 18, 25];
4
5 for (let i of prix) {
6   total += i;
7 }
8
9 console.log (total);
```

**p. 9 Solution n°3****Sortie attendue :** "8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144 152 160"**Solution possible :**

```
1 let nombre = 8;
2
3 let multiples = "";
4
5 for (let i = 1; i <= 20; i++) {
6   multiples += (nombre * i) + " ";
7 }
```

```
7 }
8
9 console.log (multiples);
```

#### p. 9 Solution n°4

##### Sortie attendue :

```
1 La marque est valide
2 La marque n'est pas valide
3 La marque est valide
4 La marque n'est pas valide
```

##### Solution possible :

```
1 let marques = ["Apple", "Acer", "HP", "Packard-Bell"];
2
3 for (let marque of marques) {
4   if (marque == "Apple" || marque == "HP" || marque == "Dell" || marque == "Microsoft") {
5     console.log("La marque est valide");
6   }
7   else {
8     console.log("La marque n'est pas valide");
9   }
10 }
```

#### p. 9 Solution n°5

##### Sortie attendue :

```
1 ram = 256
2 stockage = 512
```

##### Solution possible :

```
1 let ordinateur = new Object();
2
3 ordinateur.ram = 256;
4
5 ordinateur.stockage = 512;
6
7 for (let i in ordinateur) {
8   console.log (i + " = " + ordinateur[i]);
9 }
```

#### p. 13 Solution n°6

##### Sortie attendue :

```
1 28
2 56
3 84
4 112
5 140
6 168
7 196
```

**Solution possible :**

```
1 let nombre = 28;
2 let compteur = 1;
3
4 while (nombre * compteur < 200) {
5   console.log(nombre * compteur);
6   compteur ++;
7 }
```

[cf.]

**p. 13 Solution n°7****Sortie attendue :** 102**Solution possible :**

```
1 let prix = 100;
2 let tabPrix = [];
3
4 do {
5   tabPrix.push(prix);
6   prix ++;
7 } while (prix > 100 && prix <= 110);
8
9 console.log(tabPrix[2]);
```

[cf.]

**p. 14 Solution n°8****Sortie attendue :** undefined**Solution possible :**

```
1 let prix = 101;
2 let tabPrix = [];
3
4 while (prix > 100 && prix <= 110) {
5   tabPrix.push(prix);
6   if (prix == 108) {
7     break;
8   }
9   prix ++;
10 }
11
12 console.log(tabPrix[8]);
```

[cf.]

**p. 14 Solution n°9**

**Sortie attendue :** "0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19"

**Solution possible :**

```
1 let compteur = 0;
2
3 let nombres = "";
4
5 while (compteur < 20) {
6   nombres += compteur + " ";
7   compteur ++;
8 }
9
10 console.log (nombres);
```

[cf.]

#### p. 14 Solution n°10

**Sortie attendue :**

```
1 pas un multiple de 2
2 multiple de 2
3 pas un multiple de 2
4 multiples de 2
5 pas un multiple de 2
6 multiple de 2
7 pas un multiple de 2
8 multiples de 2
9 pas un multiple de 2
10 multiple de 2
11 pas un multiple de 2
12 multiples de 2
13 pas un multiple de 2
14 multiples de 2
```

**Solution possible :**

```
1 let nombre = 15;
2
3 compteur = 1;
4
5 while (compteur < nombre) {
6   if (compteur % 2 == 0) {
7     console.log("multiple de 2");
8   }
9   else {
10    console.log ("pas un multiple de 2");
11  }
12  compteur ++;
13 }
```

[cf.]

#### p. 15 Solution n°11

Voici un script qui fonctionne :

```
1 const list = [24, 67, 18];
2
3 let resultat = "";
4
5 for(nb of list) {
6   resultat += "Multiples de " + nb + " : ";
7   let compteur = 1;
8   while (compteur <= nb) {
9     if(nb % compteur == 0) {
10      resultat += compteur + ", ";
11    }
12    compteur++;
13  }
14  resultat += "\n \n";
15 }
16
17 console.log(resultat);
```

Dans cet exemple, on utilise une boucle **for/of** pour parcourir les nombres du tableau, et on y imbrique une boucle **while** permettant de tester chaque entier inférieur ou égal au nombre de références et vérifier si c'est un multiple ou non. C'est **compteur** qui prendra à chaque tour de la boucle **while** la valeur d'un nombre qui sera testé pour vérifier si c'est un multiple ou non.

#### p. 15 Solution n°12

Voici un script qui fonctionne :

```
1 const list = [24, 67, 18];
2
3 let resultat = "";
4
5 for(nb of list) {
6   resultat += "Multiples de " + nb + " : ";
7   for(let compteur = 1; compteur <= nb; compteur++) {
8     if(nb % compteur == 0) {
9       resultat += compteur + ", ";
10    }
11  }
12  resultat += "\n \n";
13 }
14
15 console.log(resultat);
```


Dans ce cas, on remplace la boucle **while** par une boucle **for**. Étant donné que le nombre d'itérations est déjà fixé hors de la boucle, il est plus cohérent d'utiliser une boucle **for**. D'ailleurs, le nombre de lignes de codes est moins important que dans la correction précédente.

#### Exercice p. 15 Solution n°13

### Question 1

Quelle est la boucle généralement utilisée lorsque le nombre d'itérations est fixé avant la première itération ?


- ☒ for
- ☐ while
- ☐ do/while

 La boucle **for** est utilisée en règle générale quand le nombre d'itérations est fixé avant l'exécution de la première itération. En théorie, le corps de la boucle (les instructions) ne va pas impacter sur le nombre d'itérations (hormis dans le cas où il y a un **break** ou un **continue**).

### Question 2

Quelle est la boucle à privilégier pour parcourir toutes les valeurs d'un tableau ?


- ☒ for/of
- ☐ for/in
- ☐ while

 La boucle **for/of** permet de parcourir les valeurs d'un objet étirable comme un tableau.

### Question 3

Quelle est la boucle permettant de parcourir toutes les propriétés d'un objet ?


- ☐ for/of
- ☒ for/in
- ☐ while

 La boucle **for/in** permet de parcourir les propriétés d'un objet.

### Question 4

Quelle instruction permet de stopper l'exécution d'une boucle ?

- ☐ case
- ☒ break
- ☐ continue

 L'instruction **break** permet de stopper l'exécution d'une boucle et de passer aux instructions suivantes (après la boucle).

### Question 5

Quelle boucle permet d'exécuter une première itération avant de contrôler la condition ?

- ☐ for
- ☒ do/while
- ☐ while

Q La boucle **do/while** permet d'exécuter une première itération de la boucle avant de contrôler la condition inscrite en paramètre du **while**.