

Les boucles et itérations

Table des matières

I. Contexte	3
II. Qu'est-ce qu'une boucle ?	3
III. Exercice : Appliquez la notion	5
IV. Les boucles while / do...while	5
V. Exercice : Appliquez la notion	9
VI. Les boucles for	9
VII. Exercice : Appliquez la notion	11
VIII. Compter le nombre de lettres dans un mot	12
IX. Exercice : Appliquez la notion	13
X. Compter le nombre d'occurrences d'une lettre dans un mot	14
XI. Exercice : Appliquez la notion	15
XII. Les boucles imbriquées	15
XIII. Exercice : Appliquez la notion	17
XIV. Essentiel	17
XV. Auto-évaluation	18
A. Exercice final	18
B. Exercice : Défi.....	20
Solutions des exercices	20

I. Contexte

Durée : 1 h

Environnement de travail : Repl.it

Prérequis : Connaître les bases de JavaScript, connaître les tableaux en JavaScript

Contexte

L'objectif de ce cours sera de se focaliser sur les boucles JavaScript, apprendre ce qu'est une boucle, voir les différents types de boucle et détailler leur utilisation.

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

Objectif

- Apprendre ce qu'est une boucle et quelle est son utilité

Mise en situation

Au cours de nos développements, nous pouvons être amenés à devoir effectuer plusieurs fois des actions strictement identiques, que ce soit pour parcourir une liste de données (ex : afficher une liste de produits) ou pour exécuter plusieurs fois un même bloc d'instructions à l'aide d'un compteur (ex : afficher une liste de nombres de 0 à 100 sans devoir tous les écrire manuellement).

Méthode À quoi ça sert ?

Une boucle va donc exécuter le même bloc d'instructions un nombre de fois défini.

Par exemple, nous allons pouvoir parcourir notre liste de valeurs afin d'exécuter le bloc d'instructions autant de fois que d'éléments présents dans la liste.

Imaginons une liste de notes (sur 20) "0, 5, 19, 13, 2, 16" pour laquelle nous aimerions savoir quelles sont les notes supérieures à la moyenne (10) afin d'afficher un message spécifique. Nous allons pouvoir parcourir cette liste, et pour chaque valeur, exécuter la même instruction. Dans cet exemple, nous allons vérifier si la valeur est supérieur à 10, et, afficher en conséquence un message.

Rappel Pseudo-code

Le pseudo-code permet de décrire de manière simple un algorithme sans s'appuyer sur un langage de programmation en particulier.

Méthode Boucle POUR en pseudo-code

La boucle **POUR** en pseudo-code utilise quatre valeurs numériques : `itérateur`, `valeur_initiale`, `valeur_de_sortie` et `incrément`.

Le bloc d'instructions sera exécuté **n** fois, où **n** correspond au nombre d'itérations pour que `itérateur`, partant de `valeur_initiale`, arrive à `valeur_de_sortie` en se voyant additionné `incrément` à chaque itération.

L'utilisation de cette boucle nous oblige à connaître à l'avance le nombre d'itérations que nous souhaitons effectuer (la `valeur_de_sortie`) : cette valeur peut correspondre au nombre d'éléments qui composent la liste.

```
1 Pour <itérateur> ALLANT_DE <valeur_initiale> A <valeur_de_sortie> AVEC_UN_PAS_DE <incrément>
2   <BLOC_INSTRUCTIONS>
3 FIN_POUR
```

Exemple

```
1 POUR i ALLANT_DE 0 À 100 {PAR_PAS_DE 1} FAIRE
2   SI i SUPÉRIEUR À 50 ALORS
3     AFFICHE("La valeur de i est supérieure à 50")
4   SINON
5     AFFICHE("La valeur de i est inférieure ou égale à 50")
6   FIN_SI
7 FIN_POUR
```

Méthode Boucle TANT QUE en pseudo-code

Avec la boucle **TANT QUE**, un bloc d'instructions est exécuté tant qu'une condition donnée est vérifiée.

```
1 TANT_QUE <CONDITION> FAIRE
2   <BLOC_INSTRUCTIONS>
3 FIN_TANT_QUE
```

Exemple

Dans l'exemple ci-dessous, nous avons une variable `total` initialisée avec la valeur zéro.

La boucle s'exécutera tant que `total` est inférieur à 100. Attention, il est nécessaire d'incrémenter la valeur de `total` dans le bloc d'instructions, sans quoi la condition de sortie de boucle ne sera jamais atteinte et va bloquer l'exécution du script : c'est ce qu'on appelle une boucle infinie.

`total` va donc valoir $0 + 1$ au premier passage, puis $1 + 1$ | $2 + 1$ | $3 + 1$ | ..., jusqu'à $99 + 1$.

`total` n'est donc plus inférieur à 100, donc la boucle s'arrête, car la condition n'est plus vérifiée.

```
1 total = 0
2 TANT_QUE total INFÉRIEUR À 100 FAIRE
3   total = total + 1
4 FIN_TANT_QUE
```

Complément Les boucles et les tableaux

Les boucles permettent de parcourir des éléments tels que des listes ou des tableaux.

Syntaxe À retenir

Une boucle va exécuter un nombre de fois défini le même bloc d'instructions.

Les boucles permettent notamment de parcourir des listes de valeurs afin de pouvoir les manipuler ou réaliser des actions sur celles-ci.

Remarque

Remarque : les vidéos présentent des boucles en PHP, la méthode est la même pour JS.

ComplémentStructure de contrôle Boucles sur Wikipédia¹Les boucles dans le code (MDN Web Docs)²

III. Exercice : Appliquez la notion

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

**Question 1**

[solution n°1 p.21]

Pour cet exercice, nous voulons afficher, de deux en deux, la valeur d'une variable "i", en commençant de 0.

Écrivez en pseudo-code une boucle "POUR" qui exécute l'instruction `AFFICHE (i)` quatre fois avec 1, pas de 2.

Question 2

[solution n°2 p.21]

Dans cet exercice, nous avons une variable `totale` qui est initialisée à zéro et que nous allons incrémenter de 1 tant qu'elle est inférieure à 20.

Écrivez en pseudo-code cette boucle "TANT QUE".

IV. Les boucles while / do...while

Objectifs

- La boucle while
- La boucle do...while

Mise en situation

Nous allons voir la boucle `while`, qui est l'implémentation JavaScript de la boucle **TANT QUE**.

Dans certains de nos algorithmes, nous allons devoir réaliser la même opération un certain nombre de fois. Pour cela, nous allons utiliser des boucles. Elles vont nous permettre par exemple de réaliser une action pour chaque lettre d'un mot ou pour chaque élément d'une liste.

En tant qu'humain, si nous voulons compter jusqu'à un certain chiffre ou nombre, nous allons commencer de façon croissante.

Méthode while

`While` va permettre de créer une boucle qui va s'exécuter tant que la condition renseignée entre parenthèses est vérifiée.

Cette condition est évaluée **avant** d'exécuter le bloc d'instructions contenu dans la boucle.

```
1 //Initialisation de la variable n à 0
2 let n = 0
3
4 //TANT QUE n est inférieur à 5, les instructions de la boucle sont exécutées.
5 while (n < 5) {
```

1 https://fr.wikipedia.org/wiki/Structure_de_contr%C3%B4le#Boucles

2 https://developer.mozilla.org/fr/docs/Apprendre/JavaScript/Building_blocks/Looping_code

3 <https://repl.it/>

```

6 // Affiche un message dans la console avec la valeur de n
7 console.log(n + ' : est inférieur à 5')
8
9 // Incrémente n de 1 (équivalent de n = n + 1)
10 n++
11
12 // Affiche un message avec la nouvelle valeur de n
13 console.log('n après incrément : ' + n)
14 }

```

Dans l'exemple ci-dessus, nous créons une variable `n` qui vaut 0.

Ensuite, nous créons une boucle `while` avec la condition `(n < 5)`, qui signifie que nous rentrons dans la boucle tant que `n` est inférieur à 5.

On remarque un `n++`. Le “++” est l'équivalent de `+1`. Nous aurions pu également écrire `n = n + 1`, cependant la formulation ++ est bien plus pratique.

Augmenter une valeur de 1 est une incrémentation. Dans notre exemple, on a incrémenté notre compteur.

Si `n` est inférieur à 5, les instructions suivantes sont exécutées :

- Affichage d'un message dans la console de type "<valeur de `n`> est inférieur à 5" (ex : 0 est inférieur à 5).
- Ensuite, nous incrémentons `n` de un, donc `n` vaut `n + 1` (ex : 0 + 1 au premier passage, puis 1 + 1, 2 + 1, ...).
- Affichage d'un message avec la nouvelle valeur de `n` (ex : `n après incrément : 1`).

À la fin du premier passage, `n` vaut donc 1, puis 2 à la fin du deuxième passage, et ainsi de suite jusqu'à ce qu'il soit égal à 5 et que la condition `(n < 5)` ne soit plus vérifiée, **ce qui entraîne l'arrêt de la boucle.**

La console affiche donc :

```

0 : est inférieur à 5
n après incrément : 1
1 : est inférieur à 5
n après incrément : 2
2 : est inférieur à 5
n après incrément : 3
3 : est inférieur à 5
n après incrément : 4
4 : est inférieur à 5
n après incrément : 5

```

Méthode do...while

Cette instruction va permettre de créer une boucle qui va exécuter un bloc d'instructions jusqu'à ce qu'une condition donnée soit vérifiée.

La différence avec `while` est que, cette fois, la condition est analysée **après** l'exécution du bloc d'instructions.

```

1 // Initialisation de la variable n
2 let n = 0
3
4 do {
5 // Affiche un message dans la console avec la valeur de n
6 console.log('n : ' + n)
7

```

```
8 // Incrémente n de 1 (n = n + 1)
9 n++
10 } while (n < 3)
```

L'exemple ci-dessus va afficher dans la console :

```
n : 0
n : 1
n : 2
```

Exemple Différence entre while et do...while

Afin de bien distinguer la différence entre les deux solutions, voici le code JavaScript avec les deux boucles :

```
1 let x = 3
2 do {
3   console.log('x : ' + x)
4
5   x++
6 } while (x < 3)
7
8 let y = 3
9 while (y < 3) {
10   console.log('y : ' + y)
11
12   y++
13 }
```

Dans l'exemple ci-dessus, la console va afficher uniquement :

- x : 3

En effet, même si `x` est initialisé à 3, le bloc d'instructions du `do` est exécuté avant de vérifier la condition (`x < 3`), donc le message de la console est affiché.

Dans le cas suivant, où `y` est initialisé à 3, la condition est testée tout de suite et, puisqu'elle n'est pas vérifiée, le message de la console n'est pas affiché.

Attention Boucle infinie

Il faut être vigilant lors de l'utilisation de `while` / `do...while`, car il est possible de créer une boucle infinie, c'est-à-dire une boucle dont la condition n'est jamais invalidée.

Comme la condition est toujours vérifiée, on passera éternellement dans le bloc d'instructions et il faudra alors arriver à forcer manuellement l'arrêt, en fermant l'onglet du navigateur ou en utilisant la fonctionnalité d'arrêt sur certains navigateurs, par exemple.

```
1 // ATTENTION - boucle infinie à ne pas reproduire
2 // Exemple 1
3 let n = 0
4 while (n < 5) {
5   console.log('Boucle infinie')
6 }
7
8 // ATTENTION - boucle infinie à ne pas reproduire
9 // Exemple 2
10 let isValid = true
11 while (isValid === true) {
12   console.log('Boucle infinie')
```

```
13 }
```

Méthode Instructions break et continue

Il est néanmoins possible de terminer une boucle quand une condition est remplie à l'aide de l'instruction `break`. Cette instruction permet d'arrêter l'exécution d'une boucle. Dans l'exemple ci-dessous, la condition pour que la boucle `while` se termine est que `n` arrive à 10.

Cependant, la condition `if` à l'intérieur stipule que la boucle doit se terminer quand `n` arrive à 5 : la boucle s'arrêtera donc une fois la condition remplie, soit quand `n` atteint 5.

```
1 <code>
2 let n = 0;
3
4 while (n <= 10) {
5   if (n == 5) {
6     break;
7   }
8   n++
9   console.log(n)
10 }
11 </code>
```

L'instruction continue, elle, ignore le reste de la boucle et revient à l'itération suivante. Dans l'exemple ci-dessous, quand `n` aura la valeur 3, la boucle continuera sans s'appliquer à cette condition.

```
1 <code>
2 let n = 0;
3
4 while (n < 5) {
5   n++
6   if (n === 3) {
7     continue;
8   }
9   console.log(n)
10 }
11 </code>
```

Syntaxe À retenir

Les **boucles** permettent de répéter des instructions un certain nombre de fois. On peut les utiliser pour réaliser une action pour chaque lettre d'un mot, par exemple. Lorsque l'on parcourt un mot à l'aide d'une boucle, une variable contenant chaque lettre est créée.

Incrémenter une variable de 1 signifie l'augmenter de 1.

`while` : va permettre de créer une boucle qui va s'exécuter tant que la condition renseignée entre parenthèses est vérifiée.

`do...while` : la différence avec `while` est que la condition est analysée **après** l'exécution du bloc d'instructions.

Attention à ne pas créer de boucle infinie.

Complément`while1``do...while2`

V. Exercice : Appliquez la notion

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



Question

[solution n°3 p.21]

Dans cet exercice, nous avons une variable `total` initialisée avec la valeur zéro.

Nous souhaitons faire une boucle `while` qui permet d'incrémenter, et d'afficher ensuite la valeur `total`, tant que celle-ci est inférieure à 10.

Juste au-dessous de la boucle `while`, nous voulons exécuter une boucle `do...while` avec le même bloc d'instructions.

Que pouvons-nous constater dans la console et pourquoi ?

```
1 let total = 0
```

Indice :

Pour afficher le total dans la console, il faut écrire : `console.log(total)`.

VI. Les boucles for

Objectifs

- La boucle `for`
- La boucle `for...of`

Mise en situation

Nous allons voir comment faire une boucle **POUR** en JavaScript avec les différentes utilisations de `for`.

Méthode **for**

L'instruction `for` va nous permettre d'exécuter en boucle (le nombre de fois défini jusqu'à la valeur de sortie) un même bloc d'instructions.

- Le premier paramètre de l'instruction `for` est la valeur d'initialisation (comme un compteur).
- Le second est une condition évaluée avant chaque passage dans la boucle, si la condition est vérifiée, alors l'instruction est exécutée.
- Le troisième paramètre est une expression évaluée à la fin de chaque itération (passage dans la boucle).

1 <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/while>

2 <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/do...while>

3 <https://repl.it/>

```
1 // Affiche la valeur de i dans la console
2 for (let i = 0; i < 6; i++) {
3   console.log(i)
4 }
```

Explications :

Voici les détails de l'instruction `for` : Pour "`i`" = 0 (`let i = 0`), tant que "`i`" reste inférieur à 6 (`i < 6`), on incrémente "`i`" (`i++`) .

- **Incrémenter "`i`" signifie que l'on augmente sa valeur** : "`i`" vaudra donc "`i + 1`" (`i++`) à la fin de chaque passage dans la boucle, soit 1 (=0+1), 2 (=1+1), 3, 4, 5 et enfin 6 (=5+1).

Au premier passage dans la boucle, "`i`" vaut 0, ce qui équivaut à exécuter l'instruction présente dans la boucle :

- `console.log(0)`

Puis ainsi de suite, tant que "`i`" est inférieur à 6 :

- `console.log(1)`
- `console.log(2)`
- `console.log(3)`
- `console.log(4)`
- `console.log(5)`

Ensuite, "`i`" vaut 6 et n'est donc plus inférieur à 6, ce qui fait que la boucle se termine.

Méthode `for...of`

Cette instruction crée une boucle permettant d'exécuter un bloc d'instructions sur chaque élément d'une liste. Cela peut être un tableau, une chaîne de caractères (qui peut être considérée comme un tableau de caractères), etc.

```
1 let text = 'Lion'
2
3 for (let value of text) {
4   console.log(value)
5 }
```

L'exemple ci-dessus va afficher dans la console :

```
L
i
o
n
```

En effet, chaque lettre est considérée comme une valeur de la variable `text` qui vaut 'Lion' et est donc parcourue et affichée indépendamment dans la console.

```
console.log(L)
console.log(i)
...
```

Remarque `for...of` dans un tableau

Ce type de boucle permet également de parcourir un tableau de chaînes de caractères. Les tableaux ou « arrays », présents dans plusieurs langages de programmation y compris Javascript.

Remarque Les autres boucles

Il existe d'autres façons de faire une boucle, comme certaines méthodes de l'objet `Array()` (ex : `forEach()`), qui est l'objet permettant de créer des tableaux ou encore `for...in` qui sera traitée dans le cours sur les objets Javascript.

Syntaxe À retenir

`for` : permet d'exécuter en boucle (le nombre de fois défini) un même bloc d'instructions.

`for...of` : crée une boucle permettant d'exécuter un bloc d'instructions sur chaque valeur d'un objet.

Complément

`for`¹

`for...of`²

Le protocole itérable³

VII. Exercice : Appliquez la notion

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

**Question 1**

[solution n°4 p.21]

Écrivez une boucle `for` en JavaScript qui initialise `i` à 0 ; tant que sa valeur est inférieure à 6 `i` s'incrémente de 1 en 1.

Question 2

[solution n°5 p.21]

Écrivez une boucle avec `for...of` qui donnera le résultat suivant dans votre environnement de travail :

J
a
v
a
s
c
r
i
p
t

Question 3

[solution n°6 p.22]

Quelle instruction permet de mettre fin à une boucle en cours ?

1 <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/for>

2 <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/for...of>

3 https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Les_protocoles_iteration#it%C3%A9rable

4 <https://repl.it/>

VIII. Compter le nombre de lettres dans un mot

Objectifs

- Compter le nombre de lettres dans un mot
- Voir un exemple de boucle

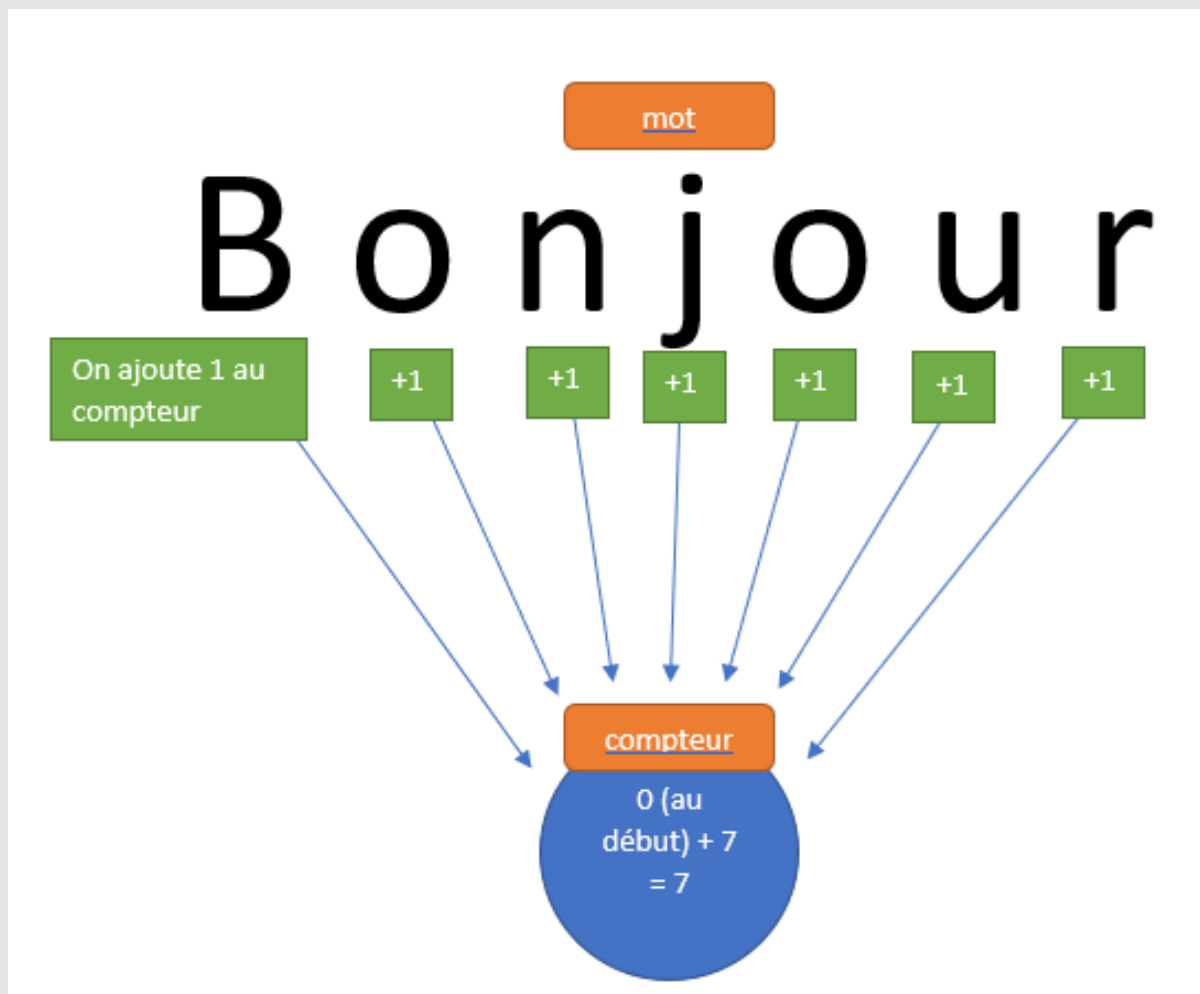
Mise en situation

Dans certains de nos algorithmes, nous allons devoir réaliser la même opération un certain nombre de fois. Pour cela, nous allons utiliser des **boucles**. Elles vont nous permettre par exemple de réaliser une action pour chaque lettre d'un mot ou pour chaque élément d'une liste.

Voyons un exemple grâce à un algorithme permettant de compter le nombre de lettres d'un mot.

Méthode Compter des lettres

En tant qu'humain, si nous voulons compter le nombre de lettres qu'il y a dans un mot, nous allons commencer par le début du mot et compter chaque lettre une par une.



En informatique, c'est exactement la même chose : manipuler un mot entier est quelque chose de complexe pour un ordinateur, donc nous allons préférer décomposer le traitement lettre par lettre.

Voici le code JavaScript permettant de compter le nombre de lettres d'un mot (ici, "Bonjour") :

```
1 let word = "Bonjour";
2 let i = 0;
3
4 /*
5  * On crée une boucle pour parcourir le mot contenu dans word
6  * word[i] renvoie la lettre du mot word à la position indiquée par "i", par exemple: 0 = "B"
7  * s'il n'y a plus de lettres pour la valeur de "i", on sort de la boucle.
8  */
9 while (word[i]) {
10   i = i + 1;
11 }
12
13 // Une fois sorti de la boucle, i contiendra le nombre de lettres. On l'affiche avec
14 console.log(i);
```

On remarque la déclaration d'une variable `i` qui va nous servir à parcourir le mot lettre par lettre : c'est une variable dite **itératrice**.

Augmenter une valeur de 1 est une **incréméntation**. Dans notre exemple, on a incrémenté notre compteur.

Exemple

Voici la trace de l'algorithme ci-dessus :

Etape	word	i	lettre courante - word[i]
let word = "Bonjour";	"Bonjour"		
let i = 0;	"Bonjour"	0	"B"
while (word[i])	"Bonjour"		
i = i + 1;	"Bonjour"	1	"o"
i = i + 1;	"Bonjour"	2	"j"
i = i + 1;	"Bonjour"	3	"u"
i = i + 1;	"Bonjour"	4	"r"
i = i + 1;	"Bonjour"	5	"o"
i = i + 1;	"Bonjour"	6	" "
i = i + 1;	"Bonjour"	7	
console.log(i);	"Bonjour"		

Syntaxe

À retenir

- Les **boucles** permettent de répéter des instructions un certain nombre de fois. On peut les utiliser pour réaliser une action pour chaque lettre d'un mot, par exemple. Lorsque l'on parcourt un mot à l'aide d'une boucle, une variable contenant chaque lettre est créée.
- Incrémenter** une variable de 1 signifie l'augmenter de 1.

IX. Exercice : Appliquez la notion

Question

[solution n°7 p.22]

Vous disposez des instructions suivantes, et on cherche à connaître le nombre de lettres du mot "JavaScript".

Écrivez la trace correspondante.

```
1 let word = "JavaScript";
2 let i = 0;
3 while(word[i]){
4   i = i + 1;
5 }
6
```

X. Compter le nombre d'occurrences d'une lettre dans un mot

Objectifs

- Compter le nombre d'occurrences d'une lettre dans un mot
- Voir un exemple de condition

Mise en situation

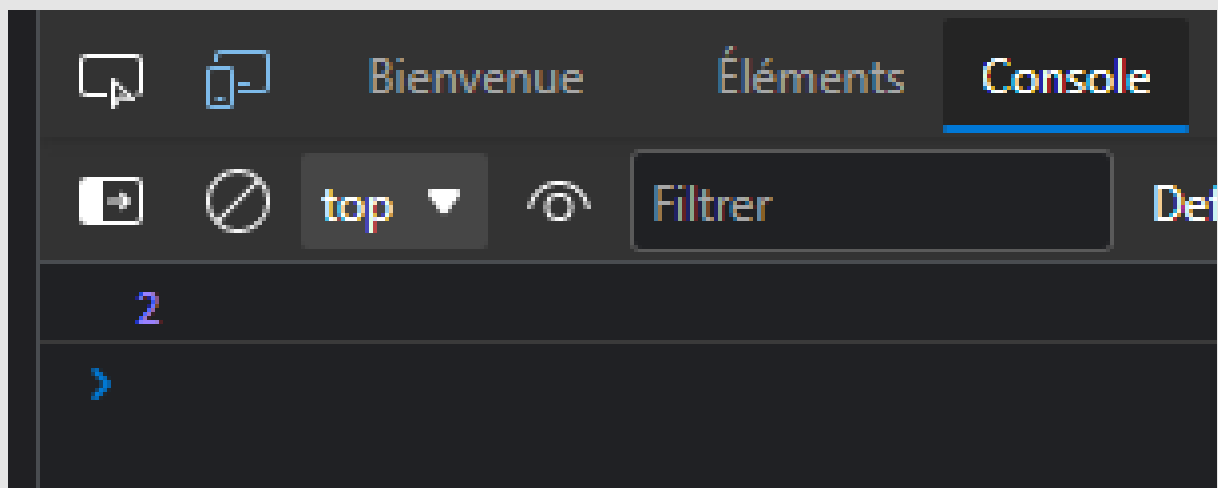
Nous allons voir comment compter le nombre de fois qu'une lettre apparaît dans mot.

Méthode Compter des occurrences

Pour compter le nombre d'occurrences d'une lettre dans un mot, le principe est le même que pour calculer la longueur d'un mot : nous allons parcourir le mot lettre par lettre.

Or, cette fois, plutôt que d'incrémenter notre compteur à chaque lettre, nous allons d'abord vérifier si la lettre en question correspond à celle que l'on cherche. Pour cela, nous allons utiliser une **condition**.

Par exemple, nous allons compter le nombre de "o" dans le mot "Bonjour". Donc, pour chaque lettre, si la lettre est égale à "o", alors on va incrémenter notre compteur.



Voici le code en JavaScript :

```

1  let word = "Bonjour";
2  let cpt = 0;
3  for (let letter of word) {
4      if (letter == "o") { // Si la lettre actuelle est égale à "o", on réalise l'opération
entre les accolades, sinon on ignore.
5          cpt = cpt + 1; // On incrémente notre compteur.
6      }
7  }
8
9  // Une fois sorti de la boucle, nombre compteur contiendra le nombre de "o". On l'affiche
avec console.log :
10 console.log(cpt);

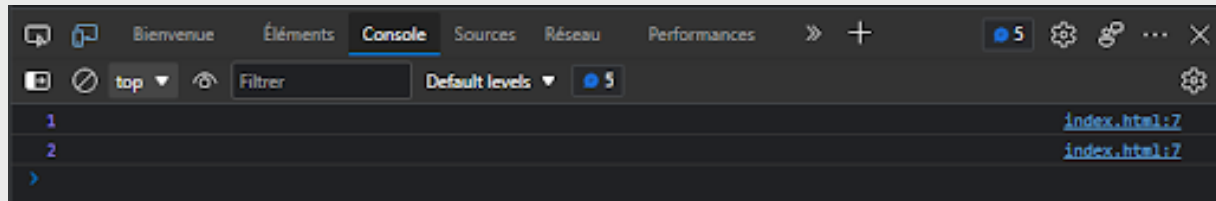
```

On déclare dans le for une variable letter, qui va contenir à chaque passage une lettre du mot word.

Pour faire une condition, on utilise le mot-clé if. Dans cette condition, on compare deux valeurs grâce au symbole ==.

Exemple

Voici la trace de l'algorithme ci-dessus :

**Syntaxe** **À retenir**

Les conditions permettent de ne réaliser des instructions que dans certains cas.

Complément

Les conditions (Wikipédia)¹

XI. Exercice : Appliquez la notion

Question

[solution n°8 p.22]

Vous disposez des instructions suivantes : on cherche à connaître le nombre de fois où la lettre "é" apparaît dans le prénom "Jérémy".

Écrivez la trace correspondante.²

```
1   let firstname = "Jérémy";
2   let i = 0;
3   for (let letter of firstname) {
4     if (letter == "é") {
5       i = i + 1;
6     }
7   }
8   console.log(i);
```

XII. Les boucles imbriquées

Objectif

- Apprendre ce que sont les boucles imbriquées, comment et quand les utiliser

Mise en situation

Les boucles imbriquées sont des boucles contenant elles-mêmes d'autres boucles. Par exemple, pour parcourir en entier les cases d'un jeu d'échecs ligne à ligne (8 lignes contenant chacune 8 cases), on utilisera une première boucle pour parcourir les lignes, et dans celle-ci une boucle parcourant les cases de la ligne en cours.

Méthode **Boucles imbriquées for/for**

Reprenons notre exemple de parcours d'un jeu d'échecs ligne à ligne : nous pouvons le réaliser en utilisant une imbrication de boucles `for`.

1 [https://fr.wikipedia.org/wiki/Instruction_conditionnelle_\(programmation\)](https://fr.wikipedia.org/wiki/Instruction_conditionnelle_(programmation))

2 <https://repl.it/>

Exemple

```
1 let squareNumber = 1
2
3 for (let x = 1; x < 9; x++) {
4   for (let y = 1; y < 9; y++) {
5     console.log('x:' + x + '|y:' + y + ' [' + squareNumber + ']')
6     squareNumber++
7   }
8 }
```

Dans l'exemple ci-dessus, la première boucle parcourt les lignes (x) et la deuxième parcourt chaque case (y) **de la ligne en cours**.

Une fois que les cases de la ligne ont été parcourues, on passe à la ligne suivante et on parcourt ses cases, et ainsi de suite.

La console va donc afficher :

```
x:1|y:1 [1]
x:1|y:2 [2]
...
x:1|y:8 [8]
x:2|y:1 [9]
x:2|y:2 [10]
...
x:8|y:8 [64]
```

Attention

Attention à bien nommer vos variables d'itération de manières différentes : dans l'exemple précédent, la première boucle utilise **x** et la seconde **y**. Utiliser la même variable pourrait provoquer des comportements inattendus.

Remarque

L'imbrication **for/for** est aussi très utilisée dans les tableaux à plusieurs dimensions (tableau contenant d'autres tableaux) afin de pouvoir tous les parcourir.

Méthode while/while

while/while fonctionne de la même façon.

Exemple

```
1 let squareNumber = 1
2 let x = 1
3
4 while (x < 9) {
5   let y = 1
6   while (y < 9) {
7     console.log('x:' + x + '|y:' + y + ' [' + squareNumber + ']')
8     squareNumber++
9     y++
10  }
11  x++
```



```
12 }
```

Méthode Mélange for/while

Il est également possible de mélanger les boucles en fonction des besoins rencontrés.

Exemple

```
1 let squareNumber = 1
2 let x = 1
3
4 while (x < 9) {
5   for (let y = 1; y < 9; y++) {
6     console.log('x:' + x + '|y:' + y + ' [' + squareNumber + ']')
7     squareNumber++
8   }
9   x++
10 }
```

Syntaxe À retenir

Les boucles imbriquées sont des boucles contenant elles-mêmes d'autres boucles.

XIII. Exercice : Appliquez la notion

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



Question

[solution n°9 p.23]

En utilisant des boucles imbriquées, vous reproduirez l'affichage suivant dans la console du navigateur.

```
1 0
2 0 1
3 0 1 2
4 0 1 2 3 4
5 0 1 2 3 4 5
6 0 1 2 3 4 5 6
7 0 1 2 3 4 5 6 7
8 0 1 2 3 4 5 6 7 8
9 0 1 2 3 4 5 6 7 8 9
```

Indice :

Il faut construire une chaîne de caractères qui correspond à une ligne : quand tous les éléments de la ligne sont présents, on affiche cette ligne et on réinitialise la chaîne de caractères pour afficher la ligne suivante.

XIV. Essentiel

1 <https://repl.it/>

XV. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°10 p.23]

Exercice

Citez les types de boucles qui existent en JavaScript.

- ☐ `meanwhile()`
- ☐ `for...of()`
- ☐ `for()`
- ☐ `loop()`
- ☐ `while()`
- ☐ `for...in()`
- ☐ `for...while()`

Exercice

Que se passe-t-il si je rends la condition de sortie d'une boucle "TANT QUE" inatteignable ?

- ☐ Le script va tourner sans cesse, c'est une boucle infinie.
- ☐ Le script va afficher une erreur et se stopper immédiatement sans passer dans la boucle.
- ☐ Le script va exécuter la boucle 255 fois et s'arrêter, c'est la valeur par défaut.

Exercice

Pour incrémenter la valeur d'une variable numérique "i", je peux utiliser la syntaxe :

- ☐ `increment(i, 1)`
- ☐ `i++`
- ☐ `i = ++1`
- ☐ `i = i+1`

Exercice

La différence entre `while` et `do...while` est liée...

- ☐ Au moment où la condition est testée.
- ☐ Au fait qu'une boucle `do...while` ne peut pas servir à parcourir un tableau JavaScript.
- ☐ Il n'y a pas de différence entre ces boucles.

Exercice

Quelle sera la valeur affichée par l'instruction `console.log()` de la dernière ligne ?

```
1 let increment = 0;
2 let i = 0;
3
4 do {
5   increment = increment + 2;
```

```
6 i++;  
7 } while (i <= 3)  
8  
9 console.log(increment);
```

- ☐ 2
- ☐ 4
- ☐ 6
- ☐ 8

Exercice

Quelle syntaxe est correcte ?

- ☐ for (let i = 0, i < 4, i++)
- ☐ for (i++ ; i = 0 ; i < 4)
- ☐ for (let i = 0 ; i < 4 ; i++)
- ☐ for (i < 4)

Exercice

La boucle `for...in` est le plus souvent utilisée sur un objet JavaScript.

- ☐ Vrai
- ☐ Faux

Exercice

La boucle `for...of` est capable de fonctionner sur :

- ☐ Une chaîne de caractères
- ☐ Un tableau
- ☐ Un objet
- ☐ Un entier

Exercice

La boucle `for...of` est apparue avec JavaScript...

- ☐ ES5
- ☐ ES6
- ☐ Service pack 2

B. Exercice : Défi

Nous allons créer un outil pour réviser nos tables de multiplication.

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



Question

[solution n°11 p.25]

Avec l'aide de boucles imbriquées, vous devrez afficher en console les tables de multiplication de 1 à 9 de la manière suivante :

```
1 === table de 1 ===
2 1 x 1 = 1
3 2 x 1 = 2
4 3 x 1 = 3
5 4 x 1 = 4
6 5 x 1 = 5
7 6 x 1 = 6
8 7 x 1 = 7
9 8 x 1 = 8
10 9 x 1 = 9
11 === table de 2 ===
12 1 x 2 = 2
13 2 x 2 = 4
14 3 x 2 = 6
15 ...
```

Indice :

Vous pouvez utiliser plusieurs types de boucles pour parvenir à ce résultat, si vous le souhaitez.

Attention, en utilisant une boucle imbriquée, de bien nommer vos variables d'itération de manières différentes pour chaque boucle ! Par convention, elles sont souvent nommées **i** (pour *iterator*), puis **y**, **z**...

Solutions des exercices

1 <https://repl.it/>

p. 5 Solution n°1

```

1 POUR i ALLANT_DE 0 À 8 {PAR_PAS_DE 2} FAIRE
2   AFFICHE(i)
3 FIN_POUR

```

p. 5 Solution n°2

```

1 total = 0
2 TANT_QUE total INFÉRIEUR À 20 FAIRE
3   total = total + 1
4 FIN_TANT_QUE

```

p. 9 Solution n°3

```

1 let total = 0
2
3 while (total < 10) {
4   total++
5   console.log(total)
6 }
7
8 do {
9   total++
10  console.log(total)
11 } while (total < 10)

```

Nous constatons que la console affiche les valeurs de 1 à 11.

En effet, la boucle `while` va incrémenter et afficher la valeur de `total` jusqu'à 10, puisque nous l'incrémentons avant d'afficher sa valeur.

En arrivant sur la boucle `do...while`, `total` vaut donc déjà 10, mais comme la condition de cette boucle est exécutée après les instructions du bloc `do`, alors on passe dans ce bloc ce qui incrémente `total` et l'affiche dans la console (avec la valeur 11).

C'est pourquoi la console affiche également la valeur 11. Ensuite, la condition est exécutée : comme `total` vaut 11, on ne repasse pas dans la boucle `do...while`.

p. 11 Solution n°4

```

1 for (let i = 0; i < 6; i++)
2 {
3   console.log( i );
4 }

```

p. 11 Solution n°5

```

1 let text = 'Javascript'
2 for (let value of text) {
3   console.log(value)
4 }

```

p. 11 Solution n°6

C'est l'instruction « break » qui permet de terminer une boucle en cours.

p. 13 Solution n°7

Étape	mot	compteur	lettre
let word = "JavaScript";	"JavaScript"		
let i = 0 ;	"JavaScript"	0	"J"
while(word[i])	"JavaScript"	0	
i = i;	"JavaScript"	0	"J"
i = i + 1;	"JavaScript"	1	"a"
i = i + 1;	"JavaScript"	2	"v"
i = i + 1;	"JavaScript"	3	"a"
i = i + 1;	"JavaScript"	4	"S"
i = i + 1;	"JavaScript"	5	"c"
i = i + 1;	"JavaScript"	6	"r"
i = i + 1;	"JavaScript"	7	"i"
i = i + 1;	"JavaScript"	8	"p"
i = i + 1;	"JavaScript"	9	"t"
i = i + 1;	"JavaScript"	10	

p. 15 Solution n°8

Étape	prénom	compteur	lettre	lettre == "é"
let firstname = "Jérémy";	Jérémy			
et i = 0;	Jérémy	0	"J"	
for (lettre == "é")	Jérémy	0		
if (lettre == "é")	Jérémy	0	"J"	FAUX

if (letter == "é")	Jérémy	1	"é"	VRAI
i = i + 1	Jérémy	1	"é"	VRAI
if (letter == "é")	Jérémy	1	"r"	FAUX
if (letter == "é")	Jérémy	2	"é"	VRAI
i = i + 1	Jérémy	2	"é"	VRAI
if (letter == "é")	Jérémy	2	"m"	FAUX
if (letter == "é")	Jérémy	2	"y"	FAUX

p. 17 Solution n°9

```

1 let row;
2 for (let i = 0; i <= 10; i++) {
3   row = '';
4   for (let j = 0; j < i; j++) {
5     row += j + ' ';
6   }
7   console.log(row);
8 }

```

Exercice p. 18 Solution n°10


Exercice

Citez les types de boucles qui existent en JavaScript.

- ☐ meanwhile()
- ☒ for...of()
- ☒ for()
- ☐ loop()
- ☒ while()
- ☒ for...in()
- ☐ for...while()

Exercice

Que se passe t-il si je rends la condition de sortie d'une boucle "TANT QUE" inatteignable ?

- ☒ Le script va tourner sans cesse, c'est une boucle infinie.
- ☐ Le script va afficher une erreur et se stopper immédiatement sans passer dans la boucle.
- ☐ Le script va exécuter la boucle 255 fois et s'arrêter, c'est la valeur par défaut.
-  Si on rend la condition de sortie d'une boucle TANT QUE inatteignable, le script tournera sans cesse, comme une boucle infinie.


Exercice

Pour incrémenter la valeur d'une variable numérique "i", je peux utiliser la syntaxe :

- ☐ `increment(i, 1)`
- ☒ `i++`
- ☐ `i = ++1`
- ☒ `i = i+1`

Exercice

La différence entre `while` et `do...while` est liée...

- ☒ Au moment où la condition est testée.
- ☐ Au fait qu'une boucle `do...while` ne peut pas servir à parcourir un tableau JavaScript.
- ☐ Il n'y a pas de différence entre ces boucles.
-  La différence entre `while` et `do...while` est liée au moment où la condition est testée.

Exercice

Quelle sera la valeur affichée par l'instruction `console.log()` de la dernière ligne ?

```
1 let increment = 0;
2 let i = 0;
3
4 do {
5   increment = increment + 2;
6   i++;
7 } while (i <= 3)
8
9 console.log(increment);
```

- ☐ 2
- ☐ 4
- ☐ 6
- ☒ 8


Exercice

Quelle syntaxe est correcte ?

- ☐ for (let i = 0, i < 4, i++)
- ☐ for (i++; i = 0; i < 4)
- ☒ for (let i = 0; i < 4; i++)
- ☐ for (i < 4)

Exercice

La boucle `for...in` est le plus souvent utilisée sur un objet JavaScript.

- ☒ Vrai
 - ☐ Faux
-  C'est vrai.


Exercice

La boucle `for...of` est capable de fonctionner sur :

- ☒ Une chaîne de caractères
- ☒ Un tableau
- ☐ Un objet
- ☐ Un entier

Exercice

La boucle `for...of` est apparue avec JavaScript...

- ☐ ES5
 - ☒ ES6
 - ☐ Service pack 2
-  La boucle `for...of` est apparue avec JavaScript ES6.

p. 20 Solution n°11

```
1 let i = 1;
2
3 while (i < 10) {
4   console.log(`=== table de ${i} ===`)
5   for(let y = 1; y < 10; y++) {
6     console.log(`${y} x ${i} = ${i*y}`);
7   }
8   i++;
9 }
```