> Cette page a été traduite à partir de l'anglais par la communauté. Vous pouvez également contribuer en rejoignant la communauté francophone sur MDN Web Docs.

Boucles et itérations

Les boucles permettent de répéter des actions simplement et rapidement. Ce chapitre du guide JavaScript présente les différentes instructions qu'il est possible d'utiliser en JavaScript pour effectuer des itérations.

Les boucles permettent de répéter des actions simplement et rapidement. Une boucle peut être vue comme une version informatique de « copier N lignes » ou de « faire X fois quelque chose ». Par exemple, en JavaScript, on pourrait traduire « Faire 5 pas vers l'est » avec cette boucle :

JS

```
for (let pas = 0; pas < 5; pas++) {
  // Ceci sera exécuté 5 fois
  // À chaque éxécution, la variable "pas" augmentera de 1
  // Lorsque'elle sera arrivée à 5, le boucle se terminera.
  console.log("Faire " + pas + " pas vers l'est");
}
```

Il y a différents types de boucles mais elles se ressemblent toutes au sens où elles répètent une action un certain nombre de fois (ce nombre peut éventuellement être zéro). Les différents types de boucles permettent d'utiliser différentes façon de commencer et de terminer une boucle. Chaque type de boucle pourra être utilisé en fonction de la situation et du problème que l'on cherche à résoudre.

Voici les différentes boucles fournies par JavaScript :

L'instruction for

Une boucle `for` répète des instructions jusqu'à ce qu'une condition donnée ne soit plus vérifiée. La boucle `for` JavaScript ressemble beaucoup à celle utilisée en C ou en Java. Une boucle `for` s'utilise de la façon suivante :

```
for ([expressionInitiale]; [condition]; [expressionIncrément])
  instruction
```

Voici ce qui se passe quand une boucle `for` s'exécute :

1. L'expression initiale `expressionInitiale` est exécutée si elle est présente. Généralement, on utilise cette expression pour initialiser un ou plusieurs compteurs dont on se servira dans la boucle. Il est possible d'utiliser des expressions plus complexes si besoin. Elle peut servir à déclarer des variables.
2. L'expression `condition` est évaluée, si elle vaut `true`, les instructions contenues dans la boucle sont exécutées. Si la valeur de `condition` est `false`, la boucle `for` se termine. Si la condition est absente, elle est considérée comme `true`.
3. L'instruction `instruction` est exécutée. Si l'on souhaite exécuter plusieurs instructions, on utilisera un bloc d'instructions (`{ ... }`) afin de les grouper.
4. Si elle est présente, l'expression de mise à jour `expressionIncrément` est exécutée.
5. On retourne ensuite à l'étape 2.

Exemple

La fonction suivante contient une instruction `for` qui compte le nombre d'options sélectionnées dans une liste déroulante (ici, un objet `<select>` permettant une sélection multiple). L'instruction `for` déclare une variable `i` et l'initialise à zéro. Elle vérifie que `i` est bien inférieur au nombre d'options et, pour chaque option, effectue un test conditionnel pour savoir si l'option est sélectionnée puis passe à l'option suivante en incrémentant la variable `i` pour chaque itération.

HTML

```
<form name="selectForm">
  <p>
    <label for="typesMusique"
      >Veuillez choisir des genres musicaux, puis cliquez :</label
    >
    <select id="typesMusique" name="typesMusique" multiple="multiple">
      <option selected="selected">R&B</option>
      <option>Jazz</option>
      <option>Blues</option>
      <option>New Age</option>
      <option>Classique</option>
      <option>Opera</option>
    </select>
  </p>
  <p><button id="btn" type="button">Combien sont sélectionnés ?</button></p>
</form>

<script>
  function quantité(selectObject) {
    let qtéSélectionnée = 0;
    for (let i = 0; i < selectObject.options.length; i++) {
      if (selectObject.options[i].selected) {
        qtéSélectionnée++;
      }
    }
    return qtéSélectionnée;
  }

  let btn = document.getElementById("btn");
  btn.addEventListener("click", function () {
    alert(
      "Nombre d'options choisies : " +
        quantité(document.selectForm.typesMusique),
    );
```

```
  });
</script>
```

L'instruction `do...while`

L'instruction `do...while` permet de répéter un ensemble d'instructions jusqu'à ce qu'une condition donnée ne soit plus vérifiée. (*NdT* : littéralement « do…while » signifie « faire… tant que »). Une instruction `do...while` s'utilise de la façon suivante :

```
do
  instruction
while (condition);
```

`instruction` est exécutée au moins une fois avant que la condition soit vérifiée. Pour utiliser plusieurs instructions à cet endroit, on utilisera une instruction de bloc (`{ ... }`) pour regrouper différentes instructions. Si la `condition` est vérifiée, l'instruction est à nouveau exécutée. À la fin de chaque exécution, la condition est vérifiée. Quand la condition n'est plus vérifiée (vaut `false` ou une valeur équivalente), l'exécution de l'instruction `do…while` est stoppée et le contrôle passe à l'instruction suivante.

Exemple

Dans l'exemple qui suit, la boucle `do` est exécutée au moins une fois et répétée jusqu'à ce que `i` ne soit plus inférieur à 5.

JS

```
let i = 0;
do {
  i += 1;
  console.log(i);
} while (i < 5);
```

L'instruction `while`

Une instruction `while` permet d'exécuter une instruction tant qu'une condition donnée est vérifiée. Cette instruction `while` s'utilise de la façon suivante :

```
while (condition)
  instruction
```

Si la condition n'est pas vérifiée, l'instruction `instruction` n'est pas exécutée et le contrôle passe directement à l'instruction suivant la boucle.

Le test de la condition s'effectue avant d'exécuter `instruction`. Si la condition renvoie `true` (ou une valeur équivalente), `instruction` sera exécutée et la condition sera testée à nouveau. Si la condition renvoie `false` (ou une valeur équivalente), l'exécution s'arrête et le contrôle est passé à l'instruction qui suit `while`.

Pour pouvoir utiliser plusieurs instructions dans la boucle, on utilisera une instruction de bloc (`{ ... }`) afin de les regrouper.

Exemple 1

La boucle `while` qui suit permet d'effectuer des itérations tant que `n` est inférieur à 3 :

JS

```
let n = 0;
let x = 0;
while (n < 3) {
  n++;
  x += n;
}
```

À chaque itération, la boucle incrémente `n` et ajoute la valeur de `n` à `x`. `x` et `n` prendront ainsi les valeurs suivantes :

- Après la première itération : `n` = 1 et `x` = 1
- Après la deuxième itération : `n` = 2 et `x` = 3
- Après la troisième itération : `n` = 3 et `x` = 6

Une fois la troisième itération effectuée, la condition `n < 3` n'est plus vérifiée, par conséquent, la boucle se termine.

Exemple 2

Attention à éviter les boucles infinies. Il faut bien s'assurer que la condition utilisée dans la boucle ne soit plus vérifiée à un moment donné. Si la condition est toujours vérifiée, la boucle se répétera sans jamais s'arrêter. Dans l'exemple qui suit, les instructions contenues dans la boucle `while` s'exécutent sans discontinuer car la condition est toujours vérifiée :

JS

```
while (true) {
  console.log("Coucou monde !");
}
```

L'instruction `label`

Un `label` (ou étiquette) permet de fournir un identifiant pour une instruction afin d'y faire référence depuis un autre endroit dans le programme. On peut ainsi identifier une boucle

On utilise un label de la façon suivante :

```
label:
  instruction
```

La valeur de `label` peut être n'importe quel identifiant JavaScript valide (et ne doit pas être un mot réservé pour le langage). L'`instruction` peut être n'importe quelle instruction JavaScript valide (y compris un bloc).

Exemple

Dans cet exemple, on utilise un label `memoBoucle` pour identifier une boucle `while`.

JS

```
memoBoucle: while (memo == true) {
  faireQQC();
}
```

> **Note :** Pour plus de détails sur cette instruction, voir la page de la référence JavaScript pour `label`.

L'instruction `break`

L'instruction `break` est utilisée pour finir l'exécution d'une boucle, d'une instruction `switch`, ou avec un label.

- Lorsque `break` est utilisé sans label, il provoque la fin de l'instruction `while`, `do-while`, `for`, ou `switch` dans laquelle il est inscrit (on finit l'instruction la plus imbriquée), le contrôle est ensuite passé à l'instruction suivante.
- Lorsque `break` est utilisé avec un label, il provoque la fin de l'instruction correspondante.

La syntaxe de cette instruction possède donc deux formes :

1. `break;`
2. `break label;`

La première forme permet d'interrompre la boucle la plus imbriquée (ou le `switch`) dans laquelle on se trouve. La seconde forme interrompt l'exécution d'une instruction identifiée par un label.

Exemple 1

Dans l'exemple qui suit, on itère sur un tableau grâce à une boucle jusqu'à trouver un élément dont la valeur est `valeurTest` :

JS

```
for (i = 0; i < a.length; i++) {
  if (a[i] === valeurTest) {
    break;
  }
}
```

Exemple 2

Ici, on utilise `break` des deux façons : avec une instruction représentée par un label et sans.

JS

```
let x = 0;
let z = 0;
labelAnnuleBoucle: while (true) {
  console.log("Boucle externe : " + x);
  x += 1;
  z = 1;
  while (true) {
    console.log("Boucle interne : " + z);
    z += 1;
    if (z === 10 && x === 10) {
      break labelAnnuleBoucle;
    } else if (z === 10) {
      break;
    }
  }
}
```

L'instruction `continue`

L'instruction `continue` permet de reprendre une boucle `while`, `do-while`, `for`, ou une instruction `label`.

- Lorsque `continue` est utilisé sans label, l'itération courante de la boucle (celle la plus imbriquée) est terminée et la boucle passe à l'exécution de la prochaine itération. À la différence de l'instruction `break`, `continue` ne stoppe pas entièrement l'exécution de la boucle. Si elle est utilisée dans une boucle `while`, l'itération reprend au niveau de la condition d'arrêt. Dans une boucle `for`, l'itération reprend au niveau de l'expression d'incrément pour la boucle.
- Lorsque `continue` est utilisé avec un label, il est appliqué à l'instruction de boucle correspondante.

L'instruction `continue` s'utilise donc de la façon suivante :

1. `continue;`
2. `continue label;`

Exemple 1

Dans l'exemple qui suit, on utilise une boucle `while` avec une instruction `continue` qui est exécutée lorsque `i` vaut 3. Ici, `n` prendra donc les valeurs 1, 3, 7 et 12.

JS

```
let i = 0;
let n = 0;
while (i < 5) {
  i++;
  if (i === 3) {
    continue;
  }
  n += i;
  console.log(n);
}
// 1, 3, 7, 12
```

Exemple 2

Dans l'exemple suivant, on a une instruction étiquetée `vérifIetJ` qui contient une autre instruction étiquetée `vérifJ`. Si l'instruction `continue` est utilisée, le programme reprend l'exécution au début de l'instruction `vérifJ`. Chaque fois que `continue` est utilisé, `vérifJ` réitère jusqu'à ce que sa condition renvoie `false`. Lorsque c'est le cas, le reste de l'instruction `vérifIetJ` est exécuté.

Si `continue` utilisait l'étiquette `vérifIetJ`, le programme continuerait au début de l'instruction `vérifIetJ`

JS

```
let i = 0;
let j = 8;

vérifIetJ: while (i < 4) {
  console.log("i : " + i);
  i += 1;
```

```
  vérifJ: while (j > 4) {
    console.log("j : " + j);
    j -= 1;
    if (j % 2 === 0) {
      continue vérifJ;
    }
    console.log(j + " est impaire.");
  }
  console.log("i = " + i);
  console.log("j = " + j);
}
```

L'instruction `for...in`

L'instruction `for...in` permet d'itérer sur l'ensemble des propriétés énumérables d'un objet. Pour chaque propriété, JavaScript exécutera l'instruction indiquée. Cette instruction s'utilise de la façon suivante :

```
for (variable in objet) {
  instruction
}
```

Exemple

La fonction suivante prend comme argument un objet et le nom de cet objet. Elle parcourt ensuite les propriétés de l'objet et renvoie une chaîne de caractères qui liste les propriétés avec leurs noms et leurs valeurs respectives :

JS

```
function afficherProps(obj, nomObj) {
  var result = "";
  for (var i in obj) {
    result += nomObj + "." + i + " = " + obj[i] + "\n";
  }
  result += "\n";
  return result;
}
```

Pour un objet `voiture` dont les propriétés sont `fabricant` et `modèle`, `result` serait :

JS

```
voiture.fabricant = Ford;
voiture.modèle = Mustang;
```

Les tableaux (arrays) et `for...in`

Bien qu'il soit tentant d'utiliser cette instruction pour parcourir les éléments d'un objet `Array`, cela peut avoir des comportements inattendus. En effet, `for...in` permet de parcourir les propriétés définies par l'utilisateur ainsi que les éléments de tableau. Ainsi, si l'on modifie un objet `Array` en lui ajoutant des propriétés et/ou des méthodes, la boucle `for...in` renverra le nom de ces nouvelles propriétés en plus des indices des éléments du tableau. C'est pourquoi, il est préférable d'utiliser une boucle `for` avec les indices du tableau pour parcourir ses éléments.

L'instruction `for...of`

L'instruction `for...of` crée une boucle qui fonctionne avec les objets itérables (qui incluent `Array`, `Map`, `Set`, l'objet `arguments`, etc.). La boucle appelle un mécanisme d'itération propre à l'objet utilisé et elle parcourt l'objet et les valeurs de ses différentes propriétés.

```
for (variable of objet) {
  instruction
}
```

Dans l'exemple suivant, on illustre la différence entre une boucle `for...of` et une boucle `for...in`. `for...in` parcourt les noms des propriétés d'un objet alors que `for...of` parcourt les **valeurs** des propriétés :

JS

```
let arr = [3, 5, 7];
arr.toto = "coucou";

for (let i in arr) {
  console.log(i); // affiche 0, 1, 2, "toto" dans la console
}

for (let i of arr) {
```

```
  console.log(i); // affiche 3, 5, 7 dans la console
}
```

This page was last modified on 3 août 2023 by MDN contributors.