JavaScript - La boucle while

La boucle `while` répète un bloc de code **tant qu'une condition est vraie**. Contrairement au `for`, on ne sait pas forcément à l'avance combien de tours auront lieu — c'est l'outil idéal pour 'continuer jusqu'à ce que quelque chose arrive'.

5TTR 6TTR niveau

La syntaxe

```
while (condition) {
    // instructions répétées tant que la condition est vraie
}
```

C'est plus simple qu'un `for` : **une seule condition**, vérifiée avant chaque tour.

```
let i = 1;

while (i <= 5) {
    console.log(i);
    i++;
}
// 1 2 3 4 5
```

Déroulement :

1. La condition est testée.
2. Si vraie → le bloc s'exécute → retour à l'étape 1.
3. Si fausse → fin de la boucle.

TU DOIS MODIFIER LA CONDITION DANS LE CORPS DE LA BOUCLE. Si la condition reste vraie indéfiniment, c'est la boucle infinie : page bloquée, onglet à fermer. Toujours s'assurer qu'on **se rapproche** de la sortie.

`for` ou `while` ?

C'est la question classique. La règle simple :

- `for` : *"Je vais répéter N fois"* ou *"je vais parcourir un tableau"*. Le nombre de tours est connu, ou borné par la taille de quelque chose.
- `while` : *"Je continue **jusqu'à ce que** ..."*. Le nombre de tours dépend de ce qui se passe pendant la boucle.

Situation	Choix recommandé
Afficher les 10 premiers nombres	`for`
Parcourir un tableau	`for...of`
Demander un mot de passe tant qu'il est incorrect	`while`
Lancer un dé jusqu'à obtenir un 6	`while`
Lire un fichier ligne par ligne jusqu'à la fin	`while`

Techniquement, **tout `for` peut s'écrire avec un `while`** et vice-versa. Le choix est une question de lisibilité.

Exemple typique : ressaisir tant que c'est invalide

```
let motDePasse = prompt("Mot de passe ?");

while (motDePasse !== "azerty123") {
    motDePasse = prompt("Incorrect. Mot de passe ?");
}

alert("Bienvenue !");
```

Combien de tours ? Aucune idée — ça dépend de l'utilisateur. Avec `for`, on devrait fixer une limite arbitraire ; avec `while`, on exprime exactement l'intention.

Exemple : lancer un dé jusqu'à un 6

```
let lancer = 0;
let tentatives = 0;

while (lancer !== 6) {
    lancer = Math.floor(Math.random() * 6) + 1;
    tentatives++;
    console.log(`Lancer ${tentatives} : ${lancer}`);
}
```

```
9
10  console.log(`Il a fallu ${tentatives} lancers.`);
```

`Math.random()` renvoie un nombre entre 0 et 1. `Math.floor(Math.random() * 6) + 1` donne un entier entre 1 et 6.

La variante `do...while`

Une version qui exécute **le bloc d'abord**, puis vérifie la condition. Garantit **au moins un tour**.

```
1  let reponse;
2
3  do {
4      reponse = prompt("Tape 'oui' pour continuer");
5  } while (reponse !== "oui");
```

Avec `while` classique, si la condition est fausse dès le départ, le bloc n'est jamais exécuté. Avec `do...while`, on l'exécute au moins une fois. Utile pour les saisies utilisateur : on veut **toujours** poser la question au moins une fois.

Sécurité : la limite anti-boucle-infinie

Quand on prototype, on n'est pas toujours sûr que la condition deviendra fausse. Un garde-fou utile :

```
1  let lancer = 0;
2  let tentatives = 0;
3
4  while (lancer !== 6 && tentatives < 1000) {
5      lancer = Math.floor(Math.random() * 6) + 1;
6      tentatives++;
7  }
8
9  if (tentatives >= 1000) {
10     console.log("Bug : on a tourné 1000 fois sans obtenir 6 !?");
11 }
```

La limite (1000 ici) joue le rôle de coupe-circuit. Dès que ça part en vrille, on sort proprement au lieu de bloquer la page.

`break` dans un `while`

Comme dans le `for`, `break` arrête la boucle immédiatement.

```
while (true) {                // boucle a priori infinie
    let saisie = prompt("Mot magique ?");
    if (saisie === "abracadabra") {
        break;                // sortie quand le mot est bon
    }
    console.log("Non, essaie encore.");
}
```

`while (true)` est un raccourci pour *"boucle jusqu'à un `break`"*. À utiliser avec parcimonie — c'est puissant mais facile à oublier d'arrêter.

À retenir

- `while (condition) { ... }` — répète **tant que** la condition est vraie.
- Idéal quand on ne sait pas à l'avance combien de tours.
- **Toujours** s'assurer que quelque chose dans le corps fait évoluer la condition, sinon : boucle infinie.
- `do { ... } while (condition)` — exécute au moins une fois avant de tester.
- `break` permet de sortir d'un coup, même depuis un `while (true)`.
- Si tu écris un `while`, demande-toi : *"qu'est-ce qui va arrêter cette boucle ?"*. Si tu ne peux pas répondre, tu as un bug.