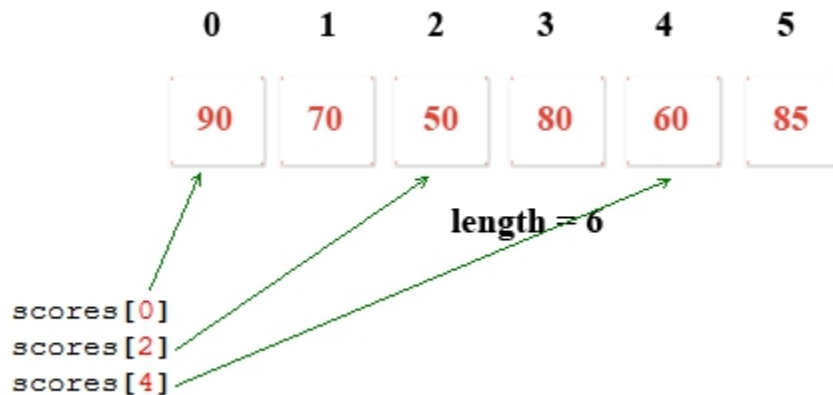


Tableaux

Unidimensionnel Tableau



📁 Créez un fichier : **OneArray.html**

```
<script>  
  // Définir un tableau à une dimension  
  const scores = [ 90 , 70 , 50 , 80 , 60 , 85 ];  
  document.write( scores[ 0 ] + "<br>" );  
  document.write( scores[ 2 ] + "<br>" );  
  document.write( scores[ 4 ] + "<br>" );  
</script>
```

Résultat :

```
90  
50  
60
```

✏ Pour imprimer tous les éléments du tableau

Avec une boucle for :

`scores.length` : la longueur du tableau

```
<script>
  const scores = [ 90 , 70 , 50 , 80 , 60 , 85 ];
  // imprimer toutes les partitions du tableau
  for ( let i = 0 ; i < scores.length; i++) {
    document.write( scores[ i] + "," );
  }
</script>
```

Résultat :

90,70,50,80,60,85,

Avec une boucle foreach

`.foreach()` : "pour chaque" permet d'exécuter une fonction donnée sur chaque élément du tableau

```
<script>
  const scores = [90, 70, 50, 80, 60, 85];

  scores.forEach(function(score) {
    document.write(score);
  });
</script>
```

Résultat:

90,70,50,80,60,85,

Avec les indices :

```
<script>
const scores = [90, 70, 50, 80, 60, 85];

for (let i = 0; i < scores.length; i++) {
```

```
    document.write("Score " + i + " : " + scores[i] + '<br>');  
}  
</script>
```

```
<script>  
const scores = [90, 70, 50, 80, 60, 85];  
  
scores.forEach(function (score, index) {  
    document.write('Score ' + index + ' : ' + score + '<br>');  
});  
</script>
```

Résultat :

Score 0 : 90
Score 1 : 70
Score 2 : 50
Score 3 : 80
Score 4 : 60
Score 5 : 85

Explication foreach :

Syntaxe

```
arr.forEach(callback);
```

Paramètres

- **callback**

La fonction à utiliser pour chaque élément du tableau. Elle prend en compte trois arguments :

- **valeurCourante**
La valeur de l'élément du tableau en cours de traitement.
- **index** (Facultatif)
L'indice de l'élément du tableau en cours de traitement.
- **array** (Facultatif)
Le tableau sur lequel la méthode `forEach` est appliquée.

Extrait de mdn

Callback

En programmation, une **callback** est une **fonction passée en argument à une autre fonction**. La fonction principale (celle qui reçoit l'argument) peut ensuite exécuter la callback à un moment donné. En d'autres termes, une callback est un morceau de code que vous pouvez "donner" à une autre fonction pour qu'elle soit exécutée plus tard.

Exemple simple de callback

Imaginez que vous avez une fonction qui exécute une autre fonction. Voici un exemple :

```
function direBonjour(callback) {  
  console.log("Bonjour !");  
  callback(); // On exécute la fonction passée en argument  
}  
  
function direAuRevoir() {  
  console.log("Au revoir !");  
}  
  
// On passe "direAuRevoir" comme une callback à "direBonjour"  
direBonjour(direAuRevoir);
```

Dans cet exemple :

- `direBonjour` prend une **fonction** comme argument (appelée **callback**).
- Après avoir exécuté `console.log("Bonjour !")`, elle exécute la **callback** (qui est ici `direAuRevoir`).

En résultat, la sortie est :

```
Bonjour !  
Au revoir !
```

Utilisation de callback dans un `forEach`

Dans le cas de `forEach`, la callback est une fonction qui est exécutée **pour chaque élément** du tableau. La méthode `forEach` s'attend à recevoir une callback en argument, et elle l'exécutera une fois pour chaque élément du tableau.

Voyons un exemple avec un tableau de scores :

```
<script>
const scores = [90, 70, 50, 80, 60, 85];

// On définit une callback qui affiche chaque score
function afficherScore(score) {
  document.write("Score:", score);
}

// On passe la callback "afficherScore" à la méthode "forEach"
scores.forEach(afficherScore);
</script>
```

Explication du code :

1. Le **tableau** `scores` contient les notes des étudiants.
2. La **callback** `afficherScore` est une fonction qui prend un score et l'affiche dans la console.
3. Le **forEach** parcourt chaque élément du tableau `scores`, et pour chaque élément, il exécute la callback `afficherScore`.

Cela affiche les résultats comme suit :

```
Score: 90
Score: 70
Score: 50
Score: 80
Score: 60
Score: 85
```

Utilisation d'une fonction anonyme comme callback dans `forEach`

Au lieu de définir une fonction séparée comme `afficherScore`, vous pouvez passer une **fonction anonyme** directement dans `forEach`. Une fonction anonyme est une fonction sans nom qui est définie là où elle est utilisée.

Voici l'exemple précédent, mais avec une fonction anonyme en tant que callback :

```
<script>
const scores = [90, 70, 50, 80, 60, 85];

scores.forEach(function(score) {
  console.log("Score:", score);
});
</script>
```

Cela fait exactement la même chose, mais on ne définit pas une fonction séparée. Cette manière d'écrire est souvent utilisée lorsque la fonction est simple et n'a pas besoin **d'être réutilisée ailleurs**.

Utilisation d'une fonction fléchée (lambda) comme callback dans **forEach**

En JavaScript moderne, on peut encore simplifier cela avec une **fonction fléchée** (lambda), comme ceci :

```
const scores = [90, 70, 50, 80, 60, 85];

scores.forEach((score) => {
  console.log("Score:", score);
});
```

C'est la version la plus concise, qui fait exactement la même chose. On utilise ici une **fonction fléchée** à la place de la fonction anonyme classique.

Pourquoi utiliser des callbacks ?

Les **callbacks** sont utiles parce qu'elles permettent de rendre le code plus **flexible**. Vous pouvez décider quelle action effectuer sur chaque élément d'un tableau, simplement en passant une nouvelle fonction en tant que callback à **forEach**. Cela permet de créer des fonctions plus générales et réutilisables.

Exemple d'une autre callback avec **forEach** :

Supposons que vous voulez **doubler chaque score** au lieu de simplement l'afficher. Vous pouvez définir une autre callback pour cela :

```
let scores = [90, 70, 50, 80, 60, 85];

function doublerScore(score) {
  console.log("Double du score:", score * 2);
}

scores.forEach(doublerScore);
```

Ici, la méthode **forEach** exécute la **callback doublerScore**, qui double chaque score avant de l'afficher.

MEMO

Fonction callback = fonction placer en paramètre d'une autre fonction
Fonction anonyme = fonction qui n'a pas de nom
Fonction fléchée = fonction anonyme avec une syntaxe plus compacte

Case à Cocher Sélectionner Tout

<input type="checkbox"/> Select All	Book Name
<input type="checkbox"/>	Easy Learning JavaScript
<input type="checkbox"/>	Easy Learning Python

📁 Créez un fichier: **SelectCheckBox.html**

`document.getElementsByName()` : obtenir un tableau d'éléments avec le même nom
`checked` : si la case est cochée?


```
<table width="400" border="1">
  <tr>
    <td>
      <input type="checkbox" onclick="checkAll(this)" />
      Select All</td>
    <td>Book Name</td>
  </tr>
  <tr>
    <td><input type="checkbox" name="book" /></td>
    <td>Easy Learning JavaScript</td>
  </tr>
  <tr>
    <td><input type="checkbox" name="book" /></td>
    <td>Easy Learning Python </td>
  </tr>
</table>
<script language="JavaScript">
  const bookInputs= document.getElementsByName( "book" );

  function checkAll(element) {
    for ( let i= 0 ; i< bookInputs.length; i++){
      bookInputs[i].checked= element.checked;
    }
  }
</script>
```

✏ Reformuler la fonction avec un foreach

`document.querySelectorAll()` : permet d'attraper des éléments à partir d'un sélecteur CSS

```
const bookInputs= document.querySelectorAll( "input#book" );
```

 Reformuler la fonction avec un `querySelector` et un `foreach`

MEMO

querySelector et querySelectorAll

`document.querySelector(selector)`: Retourne le premier élément correspondant au sélecteur CSS.

`document.querySelectorAll(selector)`: Retourne une list (NodeList) de tous les éléments correspondant au sélecteur CSS.

```
document.querySelector('.class-name') // retourne un élément
document.querySelectorAll('p') // retourne une NodeList
```

getElementBy ...

`document.getElementById(id)`: Retourne l'élément avec l'ID spécifié.

`document.getElementsByClassName(className)`: Retourne une HTMLCollection d'éléments avec la classe spécifiée.

`document.getElementsByTagName(tagName)`: Retourne une HTMLCollection d'éléments avec le tag spécifié.

`document.getElementsByName(name)`: Retourne une NodeList d'éléments avec l'attribut name spécifié.

```
document.getElementById('unique-id')
document.getElementsByClassName('class-name')
document.getElementsByTagName('div')
document.getElementsByName('input-name')
```

Comparaison

- `querySelector` et `querySelectorAll`: Plus flexibles, utilisent des sélecteurs CSS.
- `getElementById`: Le plus rapide pour sélectionner un élément unique par ID.
- `getElementsByClassName`, `getElementsByTagName`, `getElementsByName`: Retournent des **collections live** (mises à jour automatiquement).

Note

Les méthodes `getElement(s)By...` retournent des **collections live**, tandis que `querySelectorAll` retourne une **collection statique**.

Exercice

Implémenter une fonction de filtrage qui surligne les tâches contenant le texte saisi par l'utilisateur.

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Filtrage de Liste de Tâches</title>
  <style>
    .highlight { background-color: yellow; }
  </style>
</head>
<body>
  <h1>Ma Liste de Tâches</h1>
  <input type="text" id="filterInput"
    placeholder="Filtrer les tâches"
    oninput="filterTasks()">
  <ul id="taskList">
    <li>Finir le projet JavaScript</li>
    <li>Faire les courses</li>
    <li>Appeler un ami</li>
    <li>Préparer la présentation</li>
    <li>Lire un chapitre du livre JavaScript</li>
    <li>Faire de l'exercice</li>
    <li>Planifier les vacances</li>
    <li>Réviser le code du projet</li>
  </ul>

  <script>
    function filterTasks() {
      // TODO: Implémentez cette fonction
      // 1. Récupérer la valeur du champ de filtre
      // 2. Récupérer tous les éléments de la liste de tâches
      // 3. Pour chaque tâche :
      //    - Si le texte de la tâche contient la valeur du filtre,
      //    ajouter la classe 'highlight'
      //    - Sinon, retirer la classe 'highlight'
    }
  </script>
</body>
```

```
</html>
```

Pour vous aider :

<code>classList.add()</code> : ajoute une classe à un élément
<code>classList.remove()</code> : supprime une classe à un élément
<code>classList.toggle()</code> : supprime une classe si elle est là ou ajoute une classe (comme un interrupteur)

Dans un string

<code>includes()</code>	vérifie si une sous chaîne existe dans une chaîne
<code>indexOf()</code>	renvoie l'index (la position) du premier caractère de la sous-chaîne recherchée, ou <code>-1</code> si la sous-chaîne n'est pas trouvée.