

Javascript

partie 3 suite

Les boucles - suite

Tout d'abord, un peu de lexique - la différence entre « méthode » et « instruction » :

Instruction : JavaScript supporte nativement un ensemble d'instructions. Ces instructions permettent de définir les logiques des algorithmes, le flux des informations, etc. Il existe plusieurs types d'instruction en JavaScript. Ces instructions permettent de :

- **Déclarer une variable** : var, let et const
- **Itérer** : while, for, do while, etc
- **Déclarer une fonction ou class**
- **Contrôler un flux de lecture** : if / else if, continue, break, etc
- etc (return, break, continue...)

Les opérateurs logiques et de comparaison sont également des instructions.

Méthode : Une méthode est une **fonction** associée à un objet, c'est-à-dire **une instruction que l'on peut faire exécuter à un objet.**

Exemple :

<pre>var obj = { nom : "toto", function_prenom(){ return this.nom; } }; console.log(obj.function_prenom());</pre>	<pre>>> var obj = { nom : "toto", function_prenom(){ return this.nom; } }; console.log(obj.function_prenom()); toto ← undefined</pre>
--	--

Pour résumer : une méthode est une fonction associée à un objet et une instruction peut être, entre autres, une action attachée à une méthode (mais peut être aussi plusieurs autres actions, comme par exemple, une opération arithmétique ou la déclaration d'une variable). Donc, comme tu peux voir, tu es déjà familiarisé avec les instructions, sans le savoir. Dès que tu declares une variable (un let, par exemple), tu utilises une instruction...

5.3) forEach :

La **méthode forEach** permet d'itérer sur les propriétés **d'un tableau**. Quand nous parlons que forEach est une méthode, ça veut dire que la boucle forEach appelle une fonction pour passer ces paramètres. Nous allons travailler avec le lexique ES6 et créer des fonctions fléchées. Ne t'inquiète pas, on verra tout ça avec plus de détails dans le Chapitre 4 (Introduction aux fonctions)

Lexique forEach versus for :

```
let items = ["item1", "item2", "item3"];
let copie = [];
```

forEach	for
items. forEach (function(item){ copie.push(item); });	for (var i = 0; i < items.length; i++) { copie.push(items[i]); }

console.table(copie) :

```
>> ▶ let items = ["item1", "item2", "item3"]
    let copie = [];

    items.forEach(function(item){
      copie.push(item);...
```

console.table()

debugger eval code:7:9

(index)	Valeurs
0	item1
1	item2
2	item3

Plus de détails sur le lexique :

```
let ARRAY_A_ITERER = ["item1", "item2", "item3"];
```

```
ARRAY_A_ITERER.forEach(nom_de_la_fonction(valeur, index,  
tous les valeurs de array), this*) => {  
    instruction 1 ;  
    instruction 2 ;  
    etc ;  
})
```

Le nom de la fonction à exécuter pour itérer chaque élément du tableau.

Voici les paramètres de cette fonction:

function(valeur, index, arr)	Argument	Description
	valeur	Obligatoire. La valeur de l'élément itéré
	index	Optionnel. L'index de l'élément itéré
	arr	Optionnel. L'objet tableau auquel appartient l'élément

thisValue

nous verrons cet argument plus en détails dans le prochain chapitre

Exemple :

```
const list = ['a', 'b', 'c']  
  
list.forEach((item, index)  
=> {  
    console.log(item) ;  
    //valeur  
    console.log(index) ;  
    //index  
})
```

```
>> const list = ['a', 'b', 'c']  
    list.forEach((item, index) => {  
        console.log(item) //value  
        console.log(index) //index  
    })  
a  
0  
b  
1  
c  
2  
← undefined
```

Un autre exemple :

```
const list = ['a', 'b', 'c']

list.forEach((item, index, arr) => {
  console.log(item) ;
//valeur
  console.log(index) ;
//index
  console.log(arr) ; //l'array
})
```

```
>> ▶ const list = ['a', 'b', 'c']

      list.forEach((item, index, arr) => {
        console.log(item) //valeur
        console.log(index) //index...
      })
a
0
▶ Array(3) [ "a", "b", "c" ]
b
1
▶ Array(3) [ "a", "b", "c" ]
← undefined
c
2
▶ Array(3) [ "a", "b", "c" ]
```

5.5) L'instruction continue

L'instruction **continue** est utilisée à l'intérieur d'une boucle pour ignorer le reste de l'itération actuelle. L'exécution est reprise à l'itération suivante.

```
let text = '';

for (let i = 0; i < 10; i++) {
  if (i === 3) {
    continue;
  }
  text = text + i;
}

console.log(text);
```

Dans cet exemple, dès que la valeur de la variable *i* est strictement égale à 3, l'instruction **continue** va ignorer l'instruction `text = text + i`. La conséquence est que la valeur 3 ne s'affichera pas dans la console

```
>> ▶ let text = '';

      for (let i = 0; i < 10; i++) {
        if (i === 3) {
          continue;...
        }
      }
012456789
← undefined
```

Un exemple :

```
let x = 0
while (x <= 10)
{
  x++;
  if (x%2 == 1)
  {
    continue;
  }
  console.log(x);
}
```

L'instruction **continue** court-circuitera la boucle while lorsque le reste de la division par 2 est égale à 1 provoquant la non-impression de ce chiffre dans la console – et on n'aura que les nombre paires. La boucle effectuera ses 10 itérations

```
>> ▶ let x = 0
      while (x <= 10)
      {
        x++;
        if (x%2 == 1) ...
```

2

4

6

8

10

5.4) L'instruction break

Tant que l'instruction continue saute l'instruction à l'intérieur de la boucle et force une nouvelle itération, l'instruction break quitte la boucle en définitif (ou la condition, comme nous avons vu antérieurement avec le switch).



Break



Continue

- **Continue** - termine l'exécution de l'itération en cours dans une boucle.
- **Break** - interrompt (arrête) complètement l'exécution d'une boucle.

Prenons cette boucle :

<pre>for (let i = 1; i <= 10; i++){ console.log(i); }</pre>	<pre>>> for (let i = 1; i <= 5; i++){ console.log(i); }</pre> <pre>1</pre> <pre>← undefined</pre> <pre>2</pre> <pre>3</pre> <pre>4</pre> <pre>5</pre>
<pre>for (let i = 1; i <= 5; i++){ if (i == 3) {continue ;} console.log(i); }</pre>	<pre>>> for (let i = 1; i <= 5; i++){ if (i == 3) {continue ;} console.log(i); }</pre> <pre>1</pre> <pre>← undefined</pre> <pre>2</pre> <pre>4</pre> <pre>5</pre>
<pre>for (let i = 1; i <= 5; i++){ if (i == 3) {break ;} console.log(i); }</pre>	<pre>>> for (let i = 1; i <= 5; i++){ if (i == 3) {break ;} console.log(i); }</pre> <pre>1</pre> <pre>← undefined</pre> <pre>2</pre>