

# JavaScript

## - Méthodes d'itération -

**Groupe des étudiants : CIR1**



# Méthodes d'itération

- Les méthodes d'itération sont utilisées pour appliquer une fonction sur tous les éléments d'un tableau
- Elles permettent de remplacer des boucles explicites par des appels de méthodes
- Cette manière de programmer, appelée "chaînage de méthodes", est très répandue (et naturelle) en JavaScript.
- Les méthodes d'itération prennent en argument une [fonction de rappel](#)

# Fonction de rappel

- Une fonction de rappel (**callback** en anglais) est une fonction qui prend en argument une autre fonction
- Il peut s'agir d'une fonction existante, ou d'une fonction définie "à la volée" dans l'appel de la méthode d'itération.

- Les paramètres dépendent de la méthode d'itération mais la plupart des callbacks sont de la forme :

`callback(value, index, array)`

avec :

- `value` : valeur de l'élément courant
- `index` : indice de l'élément courant
- `array` : le tableau sur lequel la méthode d'itération est appliqué

- Les méthodes d'itération permettent de créer un nouveau tableau à chaque étape de transformation (sans modifier le tableau original)
- Quelques méthodes d'itération
  - `forEach()`
  - `filter()`
  - `map()`
  - `reduce()`

### ■ Exemple 0 :

Boucle explicite

```
for(let city of cities) {  
    console.log(city);  
}
```

Méthode d'itération `forEach()`

```
cities.forEach( function(city) {  
    console.log(city)  
}  
);
```

## Quelques méthodes itératives : `forEach()`

### ■ Exemple 1 :

```
let sites = ["Brest", "Rennes", "Nantes"];

// fonction existante
function printElement(element) {
    console.log(element);
}

sites.forEach( printElement );

/* =>
Brest
Rennes
Nantes
*/
```

## Quelques méthodes itératives : forEach()

### ■ Exemple 2 :

Quand la callback est très spécifique et a peu de chances d'être réutilisable, on peut la définir directement dans l'appel de la méthode d'itération

```
let sites = ["Brest", "Rennes", "Nantes"];

sites.forEach(

    // définition à la volée
    function printSite(site){
        console.log("Site de " + site);
    }

);

/* =>
Site de Brest
Site de Rennes
Site de Nantes
*/
```



## Quelques méthodes itératives : forEach()

### ■ Exemple 2 :

Quand la callback est très spécifique et a peu de chances d'être réutilisable, on peut la définir directement dans l'appel de la méthode d'itération

```
let sites = ["Brest", "Rennes", "Nantes"];
```

```
sites.forEach(
```

```
    // définition à la volée
```

```
    function printSite(site){
```

```
        console.log("Site de " + site);
```

```
    }
```

```
);
```

```
printSite("Lille"); // => ReferenceError: printSite is not  
defined
```

## Quelques méthodes itératives : forEach()

### ■ Exemple 3 :

Le nom de la callback étant inutile, on peut l'omettre dans la définition en utilisant les **fonctions anonymes**

```
let sites = ["Brest", "Rennes", "Nantes"];

sites.forEach(

    // fonction anonyme
    function(site){
        console.log("Site de " + site);
    }

);

/* =>
Site de Brest
Site de Rennes
Site de Nantes
*/
```

## Quelques méthodes itératives : forEach()

### ■ Exemple 4 :

Le code suivant montre une variante où l'on utilise le deuxième paramètre de la callback (indice de l'élément courant)

callback(value, index, array)

```
let sites = ["Brest", "Rennes", "Nantes"];

sites.forEach(

    // fonction anonyme
    function(site, i){
        console.log("Site n°" + (i+1) + " : " + site);
    }

);

/*
Site n°1 : Brest
Site n°2 : Rennes
Site n°3 : Nantes
*/
```

## Quelques méthodes itératives : filter()

- **filter()** construit un nouveau tableau contenant un sous-ensemble des éléments du tableau initial.
- La callback indique (par un booléen) si un élément doit être conservé ou non.  
Paramètres : **(value, index, array)**

### ■ Exemple :

```
let numbers = [42, -2, 100, -3, 0];

let filteredNumbers =
    numbers.filter( function(number) {
        return number >= 0;
    } );

console.log(filteredNumbers); // => [42, 100, 0]
```

Appel n°	number	retour
1	42	true
2	-2	false
3	100	true
4	-3	false
5	0	true

## Quelques méthodes itératives : map()

- `map()` construit un nouveau tableau, contenant les images de chaque élément du tableau de départ par la callback
- On peut l'utiliser faire des conversions ou des opérations mathématiques en masse  
Paramètres : (value, index, array)

- Exemple :

```
let numbers = [42, -2, 100, -3, 0];

let squaredNumbers =
  numbers.map( function(number) {
    return number**2;
  } );

console.log(squaredNumbers); // => [1764, 4, 10000, 9, 0]
```

Appel n°	number	retour
1	42	1764
2	-2	4
3	100	10000
4	-3	9
5	0	0

- **reduce()** agrège toutes les valeurs du tableau en une seule en utilisant un accumulateur (dont la valeur initiale est fournie en deuxième argument)
- On peut l'utiliser pour calculer des statistiques sur des données, par exemple min, max, moyenne, etc.  
**Paramètres** : (**acc**, **value**, **index**, **array**), où acc est la valeur courante de l'accumulateur

- Exemple :

```
let numbers = [42, -2, 100, -3, 0];

let minNumber =
  numbers.reduce( function(min, number){ // fonction agrégation
                    return Math.min(min, number); },
                    Infinity // valeur initiale accumulateur
  );

console.log(minNumber); // => -3
```

Appel n°	min	number	retour
1	Infinity	42	42
2	42	-2	-2
3	-2	100	-2
4	-2	-3	-3
5	-3	0	-3

## Fonctions fléchées

- Les fonctions fléchées permettent de définir des fonctions anonymes avec une notation allégée :
  - les accolades et les mots clés `function`/`return` sont implicites
  - l'expression derrière la flèche est évaluée et sa valeur est renvoyée
- Exemple 1 :

Fonction "classique"

```
function (n) {
    return n % 2 === 0;
}
```

Fonction fléchée :

```
(n) => n % 2 === 0;
```

## Fonctions fléchées

- Les fonctions fléchées permettent de définir des fonctions anonymes avec une notation allégée :
  - les accolades et les mots clés function/return sont implicites
  - l'expression derrière la flèche est évaluée et sa valeur est renvoyée

### ■ Exemple 2 :

```
hello = (a) => {
  if(a>0) return ">0";
  else if (a<0) return "<0";
  else return "=0";
}
console.log(hello(-5))
```

### ■ Exemple 3 :

```
f = (a) => (a<10) ? 'valid' : 'invalid';
```



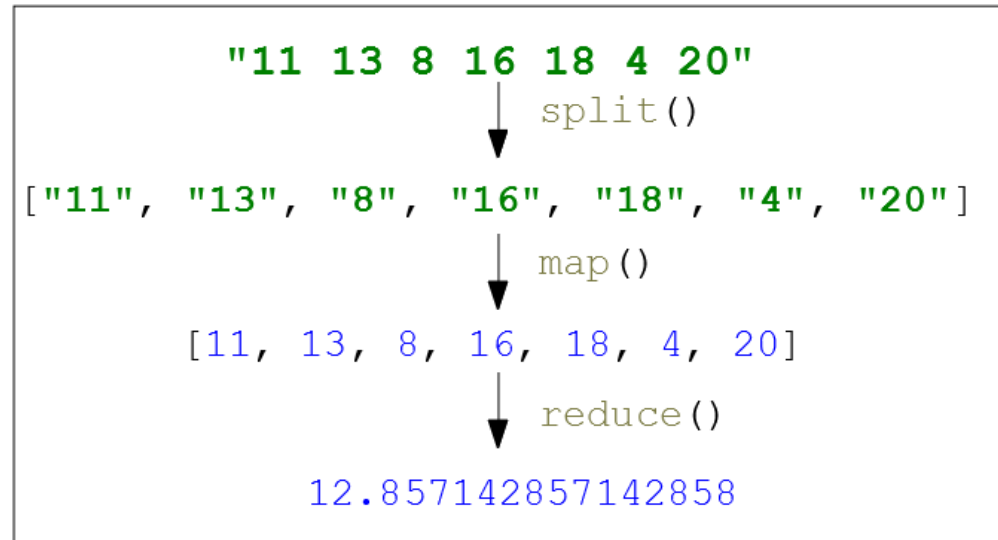
## Exemple synthétique

```
let grades = "11 13 8 16 18 4 20";

let average =
    grades.split(" ")
        .map( function(grade) { return parseInt(grade); } )
        .reduce( function(avg, grade, index, array) {
            return avg + (grade/array.length); } , 0);

console.log(average); // => 12.857142857142858
```

## Exemple synthétique



```

let average =
  grades.split(" ")
    .map( function(grade) { return parseInt(grade); } )
    .reduce( function(avg, grade, index, array) {
      return avg + (grade/array.length); } , 0);

console.log(average); // => 12.857142857142858
  
```