ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

HONORIS UNITED UNIVERSITIES

Développement Web:

JavaScript

Professeur:

Nouhaila MOUSSAMMI

n.moussammi@emsi.ma

2024/2025

Fonctions avancées

Fonctions avancées

Les paramètres du reste

-> Parfois, nous voudrons également définir des fonctions pouvant accepter un nombre variable d'arguments. Pour faire cela, nous allons devoir utiliser une notation utilisant **...** dans la déclaration des paramètres de la fonction suivi d'un nom qu'on va choisir.

JS cours.js > ...

```
1   let a = 1, b = 2, c = 3, d = 4;
2   function somme(...nombres){
3     let s = 0;
4     for (let nombre of nombres){
5       s += nombre;
6     }
7     return s;
8   }
9
10  console.log(a+' + '+b+' = ' + somme(a, b));
11  console.log(a+' + '+b+' + '+c+' = ' + somme(a, b, c));
12  console.log(a+' + '+b+' + '+c+' + ' +d+' = ' + somme(a, b, c, d));
13
```

```
[Running] node "d:\cours-javascript\cours.js"
1 + 2 = 3
1 + 2 + 3 = 6
1 + 2 + 3 + 4 = 10

[Done] exited with code=0 in 0.791 seconds
```

Fonctions avancées

Les paramètres du reste

Attention !!!

pour que ce type d'écriture fonctionne, il faudra toujours préciser les paramètres classiques en premier et les paramètres du reste en dernier dans la déclaration de la fonction.

```
JS cours.js > ...
1   let nom = 'Giraud', prenom = 'Pierre';
2   function profil(nom, prenom, ...hobbies){
3       let h = '';
4       for(hobbie of hobbies){
5           h += hobbie + ', ';
6       }
7       console.log('Nom : ' + nom + 'Prénom : ' + prenom + 'Hobbies : ' + h);
8   }
9
10  profil('Giraud', 'Pierre');
11  profil('Giraud', 'Pierre', 'Trail');
12  profil('Giraud', 'Pierre', 'Trail', 'Triathlon');
13
```

```
[Running] node "d:\cours-javascript\cours.js"
Nom : GiraudPrénom : PierreHobbies :
Nom : GiraudPrénom : PierreHobbies : Trail,
Nom : GiraudPrénom : PierreHobbies : Trail, Triathlon,

[Done] exited with code=0 in 0.19 seconds
```

Fonctions avancées

L'opérateur de decomposition & Les **paramètres du reste**

->L'opérateur de décomposition, représenté par trois points (...), est un outil puissant en JavaScript pour manipuler des tableaux, des objets, ou des arguments de fonctions.

-> Les paramètres du reste permettent de stocker une liste d'arguments dans un tableau qu'on va ensuite pouvoir manipuler.

-> Les **paramètres du reste** sont utilisés pour **capturer un nombre variable d'arguments** dans un tableau. Cela permet de gérer dynamiquement des paramètres d'une fonction.

-> L'opérateur de décomposition utilise la même syntaxe avec … que les paramètres du reste à la différence qu'on va faire suivre les trois points par le nom d'un tableau existant.

-> L'opérateur de décomposition va alors casser le tableau en une liste d'arguments qui vont pouvoir être utilisés par la fonction.

Fonctions avancées

les paramètres du reste

Exemple 1 : Additionner plusieurs nombres

```javascript
function somme(...nombres) {
    let total = 0;
    for (let i = 0; i < nombres.length; i++) {
        total += nombres[i]; // Ajoute chaque élément du tableau
    }
    return total;
}

console.log(somme(1, 2, 3, 4)); // 10
console.log(somme(5, 7, 10));   // 22
```

Fonctions avancées

JS JavaScript

L'opérateur de decomposition (spread operator ...)

->L'opérateur de décomposition fait **l'inverse des paramètres du reste** : il **transforme un tableau ou un objet en une liste d'éléments individuels**. Cela permet, par exemple, de passer un tableau comme une série d'arguments dans une fonction.

Exemple :

```javascript
const nombres = [1, 2, 3, 4, 5];

function somme(a, b, c, d, e) {
    return a + b + c + d + e;
}

console.log(somme(...nombres)); // 15
```

Ici, ...nombres décompose le tableau [1, 2, 3, 4, 5] en une série d'arguments individuels (1, 2, 3, 4, 5).

Fonctions avancées

JS JavaScript

L'opérateur de decomposition (spread operator ...)

- **Usages de l'opérateur de decomposition:**

Utilisé **dans l'appel** d'une fonction ou pour copier/concaténer des structures

```javascript
const arr = [1, 2, 3];
console.log(...arr); // Décompose [1, 2, 3] en 1, 2, 3
```

a. Appeler une fonction avec un tableau

```javascript
const valeurs = [2, 4, 6];
console.log(Math.max(...valeurs)); // 6
```

...valeurs passe 2, 4, 6 comme arguments à Math.max.

Fonctions avancées

JS JavaScript

L'opérateur de decomposition (spread operator ...)

b. Copier un tableau

```javascript
const original = [1, 2, 3];
const copie = [...original];

console.log(copie); // [1, 2, 3]
```

...original décompose les éléments du tableau pour les copier dans un nouveau tableau.

Fonctions avancées

JS JavaScript

L'opérateur de decomposition (spread operator ...)

c. Concaténer des tableaux

javascript

```javascript
const arr1 = [1, 2];
const arr2 = [3, 4];
const concatenation = [...arr1, ...arr2];

console.log(concatenation); // [1, 2, 3, 4]
```

Fonctions avancées

JS JavaScript

L'opérateur de decomposition (spread operator ...)

->L'opérateur de décomposition fait **l'inverse des paramètres du reste** : il **transforme un tableau ou un objet en une liste d'éléments individuels**. Cela permet, par exemple, de passer un tableau comme une série d'arguments dans une fonction.

JS cours.js > ...

```
1   let tb1 = [3, 5, 1, 32];
2   let tb2= [64, -5, 17];
3
4
5   console.log('Plus grand nombre de tb1 : ' + Math.max(...tb1));
6   console.log('Plus grand nombre de tb1 et tb2 : ' +Math.max(...tb1, ...tb2) );
7
8
```

```
[Running] node "d:\cours-javascript\cours.js"
Plus grand nombre de tb1 : 32
Plus grand nombre de tb1 et tb2 : 64
```

Fonctions avancées

Les fonctions fléchées JavaScript

JS JavaScript

Il existe quatre syntaxes différentes nous permettant de créer une fonction en JavaScript. On va ainsi pouvoir créer une fonction en utilisant :

- une déclaration de fonction ;
- une expression de fonction ;
- une fonction fléchée ;.

Fonctions avancées

Les fonctions fléchées JavaScript

❖ une déclaration de fonction ;

Jusqu'à présent, nous avons principalement utilisé des déclarations de fonctions. La syntaxe de déclaration de fonction est la suivante :

```
JS cours.js > ...
1   function disBonjour(){
2       console.log('Bonjour');
3   }
4
5   disBonjour();
6
```

```
[Running] node "d:\cours-javascript\cours.js"
Bonjour

[Done] exited with code=0 in 0.168 seconds
```

Fonctions avancées

Les fonctions fléchées JavaScript

JS JavaScript

- **une expression de fonction ;**

Pour créer une expression de fonction, nous allons utiliser une syntaxe similaire à celle de déclaration à la différence qu'on va cette fois-ci directement assigner notre fonction à une variable dont on choisira le nom.

```
JS cours.js > ...
1   let disBonjour= function(){
2       console.log('Bonjour');
3   };
4
5   disBonjour();
```

```
[Running] node "d:\cours-javascript\cours.js"
Bonjour

[Done] exited with code=0 in 0.21 seconds
```

Généralement, lorsqu'on crée une fonction de cette manière, on utilise **une fonction anonyme** qu'on assigne ensuite à une variable. Pour appeler une fonction créée comme cela, on va pouvoir utiliser la variable comme une fonction, c'est-à-dire avec un couple de parenthèses après son nom.

Fonctions avancées

Les fonctions fléchées JavaScript

- **une expression de fonction ;**

Les fonctions anonymes :

Une fonction anonyme est une fonction qui n'a pas de nom explicite. Ces fonctions sont souvent utilisées lorsqu'on a besoin de définir des comportements simples et temporaires, sans avoir besoin de réutiliser cette fonction ailleurs dans le code.

Les fonctions anonymes sont souvent utilisées comme argument à d'autres fonctions, qui les appellent plus tard.

cours.js > ...

```
1   // Utilisation d'une fonction anonyme comme callback
2   setTimeout(function() {
3       console.log("Ce message s'affiche après 2 secondes.");
4   }, 2000);
5
```

Fonctions avancées

JS JavaScript

Les fonctions fléchées JavaScript

❖ **Les fonctions anonymes :**

Une **fonction anonyme** est une fonction sans nom. Elle est souvent utilisée comme :
•**Argument** pour d'autres fonctions (comme dans les callbacks).
•**Expression de fonction** assignée à une variable.

Exemple 1 : Fonction anonyme comme callback

javascript

```javascript
setTimeout(function () {
  console.log("Exécuté après 2 secondes !");
}, 2000);
```

Fonctions avancées

JS JavaScript

Les fonctions fléchées JavaScript

❖ **Les fonctions anonymes :**

Exemple 2: Assignée à une variable

javascript

```javascript
const add = function (a, b) {
  return a + b;
};

console.log(add(2, 3)); // 5
```

Fonctions avancées

Les fonctions fléchées JavaScript

JS JavaScript

Déclarations de fonctions vs expressions de fonctions

- Dans la grande majorité des cas, il est plus pratique d'utiliser une déclaration de fonction qu'une expression de fonction pour créer une fonction en JavaScript.

- En effet, lorsqu'on crée des fonctions en utilisant des déclarations de fonctions, celles-ci vont être immédiatement disponibles dans le script ou dans l'espace dans lequel elles ont été créées, ce qui n'est pas le cas pour les expressions de fonctions.

- Cela est dû au fait que les déclarations de fonctions sont les premiers éléments recherchés et lus par le JavaScript lorsqu'un script est exécuté. Les expressions de fonctions, au contraire, vont être lues dans l'ordre de leur écriture dans le script.

Fonctions avancées

Les fonctions fléchées JavaScript

JS JavaScript

Déclarations de fonctions vs expressions de fonctions

```
JS cours.js > ...
1   disBonjour(); //Ceci fonctionne
2
3
4   function disBonjour(){
5       console.log('Bonjour');
6   }
7
8   disAuRevoir(); //Ceci ne fonctionne pas
9
10  let disAuRevoir = function(){
11      console.log('Au revoir');
12  }
13
```

```
[Running] node "d:\cours-javascript\cours.js"
Bonjour
d:\cours-javascript\cours.js:8
disAuRevoir(); //Ceci ne fonctionne pas
^

ReferenceError: Cannot access 'disAuRevoir' before initialization
    at Object.<anonymous> (d:\cours-javascript\cours.js:8:1)
    at Module._compile (internal/modules/cjs/loader.js:1085:14)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1114:10)
    at Module.load (internal/modules/cjs/loader.js:950:32)
    at Function.Module._load (internal/modules/cjs/loader.js:790:12)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:75:12)
    at internal/main/run_main_module.js:17:47

[Done] exited with code=1 in 1.229 seconds
```

Fonctions avancées

Les fonctions fléchées JavaScript

❖ Les expressions de fonctions fléchées : syntaxe et intérêts

En ES6, une version plus concise des fonctions anonymes a été introduite : les fonctions fléchées.

Les fonctions fléchées sont des fonctions qui possèdent une syntaxe très compacte, ce qui les rend très rapide à écrire. Les fonctions fléchées utilisent le signe => qui leur a donné leur nom à cause de sa forme de flèche.

Les fonctions fléchées n'ont pas besoin du couple d'accolades classique aux fonctions pour fonctionner et n'ont pas besoin non plus d'une expression return puisque celles-ci vont automatiquement évaluer l'expression à droite du signe => et retourner son résultat.

JS cours.js > ...

```
1   /*Expression de fonction classique :
2   let somme = function(a, b) {
3       return a + b;
4   };
5   */
6
7   //Equivalent en fonction fléchée :
8   let somme = (a, b) => a + b;
9
10
11  console.log(somme(1, 2));
12
```

```
[Running] node "d:\cours-javascript\cours.js"
3

[Done] exited with code=0 in 0.667 seconds
```

Fonctions avancées

Les fonctions fléchées JavaScript

❖ Les expressions de fonctions fléchées : syntaxe et intérêts

Si la fonction fléchée n'a besoin que d'un argument pour fonctionner, alors on pourra également omettre le couple de parenthèses.

```
JS cours.js > ...
1   /*Expression de fonction classique :
2   let double = function(n){
3       return n * 2
4   }
5   */
6
7   //Equivalent en fonction fléchée :
8   let double = n => n * 2;
9
10
11  console.log(double(3));
12
```

Fonctions avancées

JS JavaScript

Les fonctions Auto-Invoquée (IIFE) (Immediately Invoked Function Expression)

Une **fonction auto-invoquée (IIFE)** est une fonction qui est **définie et immédiatement exécutée**. Contrairement à une fonction anonyme, elle ne dépend pas d'un autre contexte pour être appelée.

Syntaxe générale

```javascript
(function () {
  // Code ici
})();
```

Ou avec une fonction fléchée :

```javascript
(() => {
  // Code ici
})();
```

Fonctions avancées

JS JavaScript

Les fonctions Auto-Invoquée (IIFE) (Immediately Invoked Function Expression)

Exemple 1 : Isolation de variables

javascript

```javascript
var globalVariable = "Je suis globale";

(function () {
  var localVariable = "Je suis locale";
  console.log(globalVariable); // "Je suis globale"
  console.log(localVariable);  // "Je suis locale"
})();

console.log(typeof localVariable); // "undefined"
```

Dans cet exemple, `localVariable` n'est accessible que dans l'IIFE.

Fonctions avancées

JS JavaScript

Les fonctions Auto-Invoquée (IIFE) (Immediately Invoked Function Expression)

Exemple 2 : Initialisation immédiate

javascript

```javascript
(function () {
  console.log("Cette fonction est exécutée immédiatement !");
})();
```

Cela affichera immédiatement :

bash

```
Cette fonction est exécutée immédiatement !
```

Fonctions avancées

JS JavaScript

Les fonctions Auto-Invoquée (IIFE) (Immediately Invoked Function Expression)

Exemple 3 : Paramètres dans une IIFE

On peut passer des arguments à une IIFE.

```javascript
(function (name) {
  console.log(`Bonjour, ${name}!`);
})("Nouhaila");
```

Sortie :

```
Bonjour, Nouhaila!
```

Fonctions avancées

JS JavaScript

Les fonctions Auto-Invoquée (IIFE) (Immediately Invoked Function Expression)

Exemple 4 : IIFE avec une fonction fléchée

javascript

```javascript
(() => {
  console.log("IIFE avec une fonction fléchée !");
})();
```

Fonctions avancées

Gestion du délai d'exécution en JavaScript

La fonction prédéfinie : setTimeout()

La méthode setTimeout() permet d'exécuter une fonction ou un bloc de code après une certaine période définie (à la fin de ce qu'on appelle un « timer »).

Il va falloir passer deux arguments à cette méthode :

1. une fonction à exécuter
2. un nombre en millisecondes qui représente le délai d'exécution de la fonction (le moment où la fonction doit s'exécuter à partir de l'exécution de setTimeout()).

cours.js > ...

```
1   // Affiche un message après 3 secondes (3000 millisecondes)
2   setTimeout(function() {
3       console.log("Ce message apparaît après 3 secondes.");
4    }, 3000);
5
```

```
[Running] node "d:\cours-javascript\cours.js"
Ce message apparaît après 3 secondes.

[Done] exited with code=0 in 3.181 seconds
```

Fonctions avancées

Gestion du délai d'exécution en JavaScript

La fonction prédéfinie : setInterval()

La méthode setInterval() permet d'exécuter une fonction ou un bloc de code en l'appelant en boucle selon un intervalle de temps fixe entre chaque appel.

Cette méthode va prendre en arguments le bloc de code à exécuter en boucle et l'intervalle entre chaque exécution exprimée en millisecondes.

cours.js > ...

```
1   // Affiche un message toutes les 2 secondes (2000 millisecondes)
2   setInterval(function() {
3       console.log("Ce message apparaît toutes les 2 secondes.");
4   }, 2000);
5
6
```

Fonctions avancées

Gestion du délai d'exécution en JavaScript

La fonction prédéfinie : clearInterval()

La méthode clearInterval() permet d'annuler l'exécution en boucle d'une fonction ou d'un bloc de code définie avec setInterval().

Pour que cette méthode fonctionne, il va falloir lui passer en argument l'identifiant retourné par setInterval().

JS cours.js > ...

```
1   // Déclare une variable pour stocker l'identifiant de l'intervalle
2   const intervalID = setInterval(function() {
3       console.log("Ce message apparaît toutes les 2 secondes.");
4     }, 2000);
5
6     // Arrête l'intervalle après 10 secondes
7     setTimeout(function() {
8       clearInterval(intervalID); // Arrête l'exécution répétée
9       console.log("L'intervalle a été arrêté.");
10    }, 10000);  // 10 000 millisecondes = 10 secondes
11
```

```
[Running] node "d:\cours-javascript\cours.js"
Ce message apparaît toutes les 2 secondes.
Ce message apparaît toutes les 2 secondes.
Ce message apparaît toutes les 2 secondes.
Ce message apparaît toutes les 2 secondes.
L'intervalle a été arrêté.

[Done] exited with code=0 in 10.18 seconds
```