

Javascript Avancé partie 1

Historique

<https://fr.wikipedia.org/wiki/JavaScript>

Créé par Netscape en 1995

Standardisation en 1997

ECMAScript (Standard ECMA-262)

Version actuelle: [ES2021 \(ES12\)](#)

Utilité



Ou placer le code Javascript

HEAD

```
<!DOCTYPE html>
<html>

<head>
<script>
function myFunction() {
    document.getElementById("demo").innerHTML =
    "Paragraph changed.";
}
</script>
</head>

<body>

<h1>My Web Page</h1>

<p id="demo">A Paragraph</p>

<button type="button" onclick="myFunction()">Try
it</button>

</body>
</html>
```

BODY

```
<!DOCTYPE html>
<html>
<body>

<h1>My Web Page</h1>

<p id="demo">A Paragraph</p>

<button type="button" onclick="myFunction()">Try
it</button>

<script>
function myFunction() {
    document.getElementById("demo").innerHTML =
    "Paragraph changed.";
}
</script>

</body>
</html>
```

Fichier externe Javascript

```
<!DOCTYPE html>|  
<html>  
<body>  
<script src="myScript.js"></script>  
</body>  
</html>
```

Dans le HEAD ou
le BODY

Attention à la mise en cache
lors du développement !!

Types de données

➤ Les différents type de données sont :

➤ Number :

```
var length = 16;
```

➤ Bool :

```
var bool = true;
```

➤ String :

```
var lastName = "Johnson";
```

➤ Array :

```
var cars = ["Sabb", "Volvo", "BMW"];
```

➤ Object :

```
var x = {firstName:"John", lastName="Doe"}
```

➤ Undefined :

```
var person;
```

➤ Null :

```
var nul = null;
```

➤ Function :

```
const func = () => {}
```

typeof de données

```
typeof "John"           // Returns string
typeof 3.14             // Returns number
typeof false            // Returns boolean
typeof [1,2,3,4]        // Returns object
typeof {name:'John', age:34} // Returns object
```

- Array est un object particulier
- Null == undefined mais il ne sont pas du même type

```
null == undefined
```

Opérateurs Javascript

- Les opérateurs utilisés en javascript sont les suivants :
 - +
 - -
 - *
 - /
 - % => reste d'une division
 - ++ => incrément de 1 (exemple `x++` équivaut à `x = x + 1`)
 - -- => décrétement de -1

Expressions contractées Javascript

- Les expressions peuvent se contracter ainsi :
 - $x += y$ est équivalent à $x = x + y$
 - $x -= y$ est équivalent à $x = x - y$
 - $x *= y$ est équivalent à $x = x * y$
 - $x /= y$ est équivalent à $x = x / y$
 - $x \% = y$ est équivalent à $x = x \% y$
 - $x++$ est équivalent à $x = x + 1$
 - $x--$ est équivalent à $x = x - 1$

Comparaison

- `==` égal à
- `!=` différent de
- `<` inférieur à
- `>` supérieur à
- `<=` inférieur ou égal à
- `>=` supérieur ou égal à
- `===` égal en valeur et en type à
- `!==` n'est pas égal en valeur et en type à

Opérateur logique

Operator	Description
&&	and
	or
!	not

Scope & Hoisting

```
worldWide = "global";  
var old = "local";  
let mutable = "block";  
const immutable = "block"
```

	var	let	const
Stockée en global	✓	✗	✗
Portée de la fonction	✓	✓	✓
Portée du block	✗	✓	✓
Ré-assignation	✓	✓	✗
Re-déclaration	✓	✗	✗
Hoisting	✓	✗	✗

Les conditions

```
if (condition) {  
    block of code to be executed if the condition is true  
} else {  
    block of code to be executed if the condition is false  
}
```

```
if (hour < 18) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}
```

```
if (condition1) {  
    block of code to be executed if condition1 is true  
} else if (condition2) {  
    block of code to be executed if the condition1 is false and condition2 is true  
} else {  
    block of code to be executed if the condition1 is false and condition2 is false  
}
```

```
if (time < 10) {  
    greeting = "Good morning";  
} else if (time < 20) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}
```

Switch

Switch

```
switch(expression) {  
    case n:  
        code block  
        break;  
    case n:  
        code block  
        break;  
    default:  
        default code block  
}
```

Boucles

Tant que vrai Faire

```
let n = 0;

while (n < 3) {
  n++;
}
console.log(n);
```

Faire tant que vrai

```
let result = '';
let i = 0;

do {
  i = i + 1;
  result = result + i;
} while (i < 5);
console.log(result);
```

Boucle FOR

```
let str = '';

for(let i = 0; i < 10; i++) {

}
console.log(str);
```

Parcourir un objet

```
const persons = {fname:"john", lname:"Doe", age:25};

for (const property in persons) {
  console.log(`${property}: ${persons[property]}`);
}
```

```
const cars = ["Sabb", "Volvo", "BMW"];
for (const element of cars) {
  console.log(element);
}
```

Itérateur de tableau

forEach :

Il parcourt le tableau et invoque une callback en utilisant chaque valeur comme argument. La callback prend 3 paramètres différents : la valeur dans le tableau, l'index courant et le tableau sur lequel le rappel est appelé. Il existe également une quatrième valeur et c'est celle à utiliser comme this lors de l'exécution de la callback.

```
const cities = ['ROME', 'PARIS', 'LONDON', 'LOS ANGELES', 'VIENNA'];

cities.forEach((value, index, array) => {
  console.log(`${index + 1} ${value}`);
});
//output: 1 ROME, 2 PARIS, 3 LONDON, 4 LOS ANGELES, 5 VIENNA
```


Itérateur de tableau

map :

Très similaire à la méthode `forEach()`. Il itère sur un tableau et prend une callback comme paramètre invoqué sur chaque élément du tableau d'origine. La différence est qu'il renvoie un nouveau tableau qui remplace chaque valeur par la valeur de retour de la fonction de rappel.

```
const numbers = [1, 2, 3, 4, 5];  
  
const squaredNumbers = numbers.map(number => number * number);  
console.log(squaredNumbers);  
//output: [1, 4, 9, 16, 25]
```

Itérateur de tableau

filter :

Il renvoie un nouveau tableau qui ne contient que des éléments du tableau d'origine qui renvoient true lorsqu'ils sont passés à la callback.

```
// Return an array of even values from numbers
const evens = numbers.filter(num => num % 2 === 0);
console.log(evens);
//output: [2, 4, 6, 8, 10]
```

Itérateur de tableau

recude :

C'est une autre méthode qui itère sur chaque élément du tableau, mais cette fois, elle combine cumulativement chaque résultat pour ne renvoyer qu'une seule valeur. La callback est utilisée pour décrire comment combiner chaque valeur du tableau avec le total cumulé. Ceci est souvent utilisé pour le calcul des données stockées dans un tableau. La fonction reduce prend quatre arguments : accumulateur, valeur courante, index courant et tableau source. Elle accepte également une valeur initiale à utiliser comme premier argument du premier appel de la callback.

```
const sum = numbers.reduce((accumulator, currentValue, currentIndex, array) => {  
  |   return accumulator + currentValue;  
}, 0);  
console.log(sum);  
// output: 55
```

Les fonctions

function : mot clé pour définir une fonction

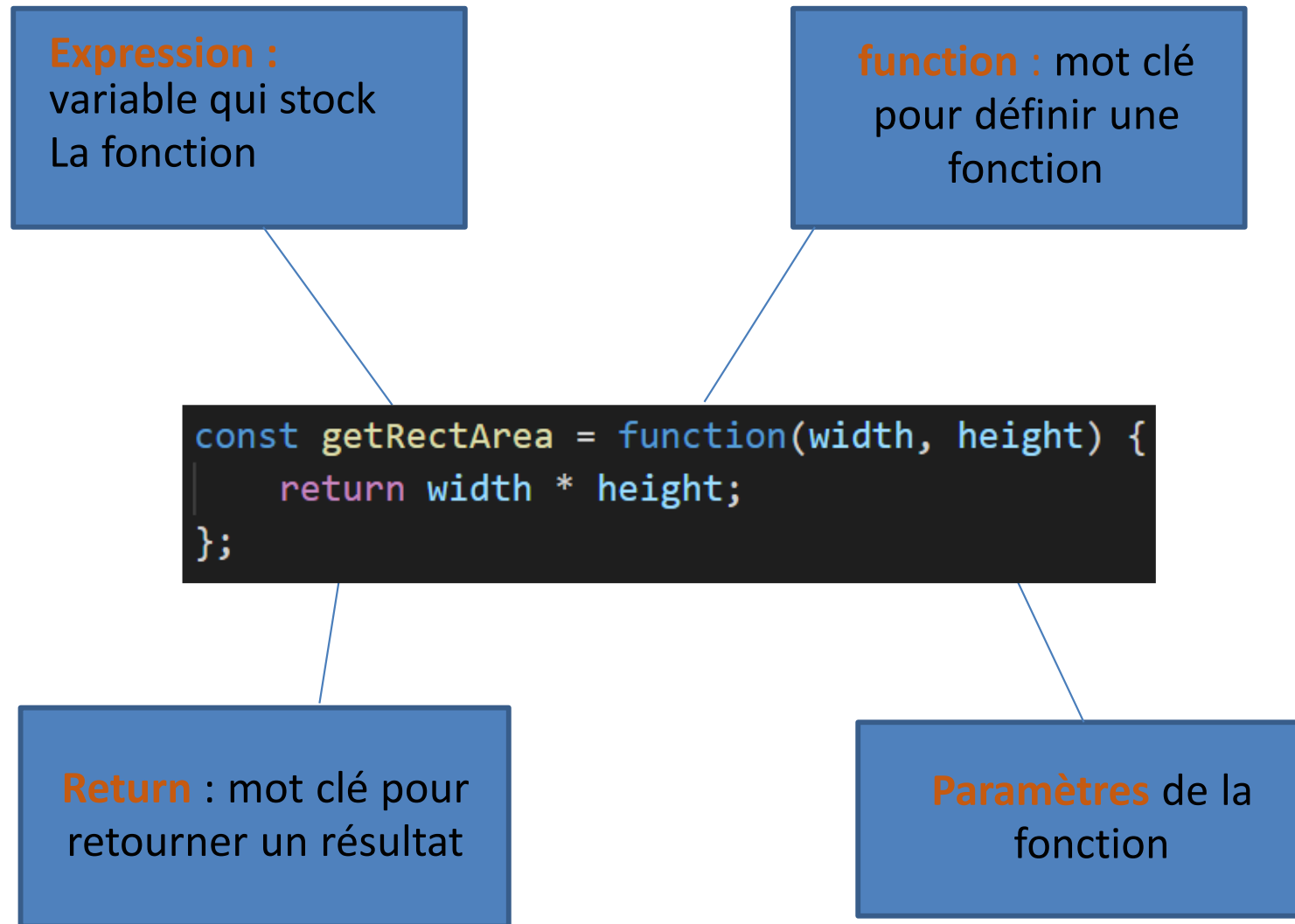
Nom de la fonction
choisi par le développeur

```
function multiplication(p1, p2) {  
    return p1 * p2;  
}
```

Return : mot clé pour retourner un résultat

Paramètres de la fonction

Expression de fonction



Fonctions fléchées

Syntaxe :

```
([param] [, param]) => {  
  instructions  
}
```


```
(param1, param2, ..., param2) => expression  
// équivalent à  
(param1, param2, ..., param2) => {  
  return expression;  
}
```

```
// Parenthèses non nécessaires quand il n'y a qu'un seul argument  
param => expression
```

```
// Une fonction sans paramètre peut s'écrire avec un couple  
// de parenthèses  
() => {  
  instructions  
}
```

```
var a = [  
  "We're up all night 'til the sun",  
  "We're up all night to get some",  
  "We're up all night for good fun",  
  "We're up all night to get lucky"  
];  
  
// Sans la syntaxe des fonctions fléchées  
var a2 = a.map(function (s) { return s.length });  
// [31, 30, 31, 31]  
  
// Avec, on a quelque chose de plus concis  
var a3 = a.map( s => s.length);  
// [31, 30, 31, 31]
```

Les objets : principe

Object	Propriétés	Méthodes
	<code>car.nom = Fiat</code> <code>car.modele = 500</code> <code>car.poids = 850kg</code> <code>car.couleur = white</code>	<code>car.demarrer()</code> <code>car.conduire()</code> <code>car.tourner()</code> <code>car.eteindre()</code>

Les objets : syntaxe

Nom de l'objet

Propriété


```
var personne = {  
  prenom: "Angel",  
  nom: "Di Maria",  
  age: 28,  
  couleurYeux: "marron",  
  getNomPrenom : function() {return this.nom + " " + this.prenom;}  
};
```

Méthode

Les objets : syntaxe

```
var personne = {  
  prenom:"Angel",  
  nom:"Di Maria",  
  age:28,  
  couleurYeux:"marron",  
  getNomPrenom : function() {return this.nom + " " + this.prenom;}  
};
```


Appelle d'une propriété



A blue box labeled 'Appelle d'une propriété' has a line pointing to a bracket. The bracket groups two code snippets, each preceded by a green vertical bar.

- `personne.nom;`
- `personne["nom"];`

Appelle d'une méthode



A blue box labeled 'Appelle d'une méthode' has a line pointing to a bracket. The bracket groups two code snippets, each preceded by a green vertical bar.

- `objectName.methodName()`
- `personne.getNomPrenom()`

Exercices

- **UC First**: 1ère lettre en MAJ
 - hello world => Hello world
- **Capitalize** : 1ère lettre de chaque mot en MAJ
 - hello world => Hello World
- **camelCase** : Capitalize + coller les mots
 - hello world => HelloWorld
- **snake_case**: Joindre les mots par des Underscores
- **l33t speak** : Cryptage (uniquement les voyelles)
 - anaconda => 4n4c0nd4
 - A=>4, E=>3, I=>1, O=>0 ,U=> (_), Y=>7
- **findLongestWord** : retourne le mot le plus long d'une phrase.
 - Le chemin le plus cours n'est pas toujours le meilleur => toujours

Pour chaque exercice, renvoyer « Merci de mettre une string valable » si le type n'est pas une string ou si la chaine est vide.