2023 / 1

JavaScript

Flavien Bigi

lesentrecodeurs.com

flavien@lesentrecodeurs.com

hAppywAy#8369

Le plan

1. Pourquoi JavaScript
2. Les variables
   - types, scope, concat, operator etc
3. Les boucles
4. Structures de controle
5. Procédures et fonctions
6. DOM et events

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.

John Woods, Software Engineer

Pourquoi JavaScript

Historiquement les technologies Web
HTML, CSS
JavaScript

Issu du navigateur Netscape (Brendan Eich, 1995), numéro 1 au milieu des années 1990. Détrôné par MicroSoft avec Internet Explorer (qui utilise JScript et non JavaScript), il revient très fort avec Firefox (2002) puis Chrome (2008).

Interprété ou compilé à la volée par le navigateur. Pour en apprendre d'avantage sur la compilation de Javascript

Spécificité Javascript

Langage le plus populaire du moment

Web moderne et dynamique

Accessible**

Très grande communauté

Très permissif

Synchrone/Asynchrone

Sa force

Modifier dynamiquement les éléments dans le DOM : Document Object Model

Le DOM

Et maintenant ?

JavaScript pour développer sur bureau, mobile et tablette.
React, Angular 2+, Vue etc

Possibilité de développer des applications serveurs.
Nodejs, Deno, Express, Meteor, etc

https://2020.stateofjs.com/en-US/

Domaine d’utilisation

😟 **IoT**	
😡 **Systèmes embarqués**	
😄 **Applications Desktop**	*Electron*
😄 **Web**	*Angular, React, Vue, Hapi, ExpressJs*
😄 **Jeux-vidéo**	*Phaser, Threejs*
😄 **Mobile**	*React Native, NativeScript*
😡 😟 **ML**	*TensorflowJs*

Évolution du langage

jQuery

-

ES6 = ES2015**

ES7 = ES2016

ES8 = ES2017

ES9 = ES2018

ES10 = ES2019

ES11 = ES2020

ES12 = ES2021

ECMAScript

Les variables

mot clé nom de la variable “=”, opérateur d’affectation

```
var jeSuisUneVariable = "varaible gobale"; // variable globale

function uneFonction() {
  var jeSuisUneVariable = 1;
  // une var peut être redéclarée
  // ici elle sera locale
  console.log(jeSuisUneVariable);
}

uneFonction();
console.log(jeSuisUneVariable);
```

Le scope est la portée de la variable

Scope de var : global ou fonction

Les variables

```
function anyFunction () {
  const uneVariableEnLectureSeule = 32;
  try {
    uneVariableEnLectureSeule = 44;
  } catch (e) {
    console.error(e); // error
  }
  console.log(uneVariableEnLectureSeule); // 32
}

anyFunction();
if (true) {
  const autreVariableEnLectureSeule = "OUI";
  console.log(autreVariableEnLectureSeule); // OUI
}

console.log(autreVariableEnLectureSeule); // error
```

Scope de const et let : global, fonction et bloc

Les variables

```
function anyFunction () {
  let uneVariable = "BIM";
  try {
    uneVariable = "BAM";
  } catch (e) {
    console.error(e); // pas d'erreur, réaffectation
  }
  console.log(uneVariable); // BAM
}

anyFunction();
if (true) {
  let variableDansScopeIf = "BOOM";
  console.log(variableDansScopeIf); // OUI
}

console.log(variableDansScopeIf); // error
```

const et **let** sont apparus depuis ES6 pour limité la vie d'une variable dans un bloc. En bloc
étant défini par { ... }

Les variables et scopes

```
var msg = "Bijour Var Out";

function greet () {
    let msg = "Bijour Let In";
    console.log(msg);
}

greet();
```

Résultat

```
Bijour Let In
```

Les variables et scopes

```javascript
let msg = "Out";

console.log(msg);

function greet () {
    msg = "In";
    console.log(msg);
}

console.log(msg);
greet();
console.log(msg);
```

Résultat

```
Out
Out
In
In
```

Les variables et scopes

```js
let msg = "Out";

console.log(msg);

function greet () {
    msg = "In";
    const msg2 = "Const In";
    console.log(msg + msg2);
}

console.log(msg);
greet();
console.log(msg + msg2);
```

Résultat

```
Out
Out
InConst In
C:\projects\js\web1\varAndScope.js:13
console.log(msg + msg2);
                  ^

ReferenceError: msg2 is not defined
```

Les types

String
Number
Boolean
Null => rien
Undefined => non défini
Symbol (ES6)
BigInt (> 2^53) (ES6)
Object

Opérateurs arithmétiques

Opérateur	Nom de l'opération associée
+	Addition
–	Soustraction
*	Multiplication
/	Division
%	Modulo (reste d'une division euclidienne)
**	Exponentielle (élévation à la puissance d'un nombre par un autre)

Opérateurs d’affectation

Opérateur	Définition
+=	Additionne puis affecte le résultat
-=	Soustrait puis affecte le résultat
*=	Multiplie puis affecte le résultat
/=	Divise puis affecte le résultat
%=	Calcule le modulo puis affecte le résultat

Opérateurs d’incrémentation

Exemple (opérateur + variable)	Résultat
`++x`	Pré-incrémentation : incrémente la valeur contenue dans la variable x, puis retourne la valeur incrémentée
`x++`	Post-incrémentation : retourne la valeur contenue dans x avant incrémentation, puis incrémente la valeur de $x
`--x`	Pré-décrémentation : décrémente la valeur contenue dans la variable x, puis retourne la valeur décrémentée
`x--`	Post-décrémentation : retourne la valeur contenue dans x avant décrémentation, puis décrémente la valeur de $x

Concaténation

```
const prefix = "En JS, la concaténation prend se fait ";
const suffix = "avec le signe +, qui est aussi l'opérateur d'addition";

const preSuffix = prefix + suffix;
console.log(preSuffix);



// En ES6, on a corrigé ca avec ` et ${}
console.log(`${prefix}${suffix}`);
```

Opérateurs de comparaison

En javascript, les égalités sont parfois étonnantes, ceci vient principalement de la coercion

```
// Avec les triple =, on compare la valeur mais aussi son type
console.log( "1 == \"1\"", 1 == "1" ); // true
console.log( "1 === \"1\"", 1 === "1" ); // false
console.log( "1 === 1", 1 === 1 ); // true
console.log( "{} === {}", {} === {} ); // false ???
console.log( "[] === []", [] === [] ); // false
console.log( "[] == ![]", [] == ![] ); // true ???
console.log( "[] == []", [] == [] ); // false
console.log( "1 !== 1", 1 !== 1 ); // false
console.log( "1 > 1", 1 > 1 ); // false
console.log( "1 >= 1", 1 >= 1 ); // true
// ATTENTION, nombreux problèmes du à la coercion
```

Structures de contrôle

```
const min = 3;
const max = 99;

if (min > max) {
  console.log("error");
} else if (min === max) {
  console.log("equal");
} else {
  console.log("looks great");
}
```

```
const min = 3;
const max = 99;

switch (min < max) {
  case false:
    switch (min === max) {
      case true:
        console.log("equal");
        break;
      default:
        console.log("error");
    }
    break;
  default:
    console.log("looks great");
}
```

Dans un contexte booléen if(monTest), toute valeur en js est évaluée à true en dehors de

- false
- 0
- “”
- null
- undefined
- NaN => Not a Number

qui sont elles falsy

Les boucles

Mot clé / keyword

for (let x = ..., condition d'arrêt, incrémentation ou décrémentation) { // code }

```
const classMatesNumber = 31;

for (let i = 0; i < classMatesNumber; i++) {
  console.log("Élève " + i);
}
```

```
// ES6
console.log("==== for of");
for (const number of arrayOfNumbers) {
  console.log(number);
}

console.log("==== for in");
for (const key in arrayOfNumbers) {
  console.log(arrayOfNumbers[key]);
}
```

Mot clé / keyword

for (const value of values) { // code }

for (const keyOfValue in values) { // code }

<u>on parcourt toute ma boucle</u>

Les boucles

Mot clé / keyword
do { // code } while (condition d'arrêt);

```
const classMatesNumber = 31;
let i = 0;
do {
  console.log("Élève " + i);
  i++;
} while (classMatesNumber > i);
```

Mot clé / keyword
while (condition d'arrêt) { // code } ;

```
const classMatesNumber = 31;
let i = 0;
while (classMatesNumber > i) {
  console.log("Élève " + i);
  i++;
}
```