JDeDev

Formation Javascript

Partie 1 : Bases de JS

En partenariat avec

LiveCampus
Apprentissage connecté

Julien Demettre EIRL ▸ Expert Windev Freelance
+33 (0)6 49 24 41 62 ▸ julien@jdedev.fr ▸ www.jdedev.fr
Siège social ▸ 125 Rue de Viller, 54300 LUNEVILLE - FRANCE
SIRET ▸ 832 481 220 00024 | APE ▸ 5829C

JDeDev

Sommaire

LIER UN FICHIER JAVASCRIPT À UN HTML

Par défaut le navigateur charge une page HTML, cette page fera elle-même référence à différents fichiers (images, css, js,etc…)

Pour lier un fichier javascript à un html on utilise la balise script avec l'attribut src.

Vous pouvez mettre cette balise script en fin de page HTML pour éviter les soucis de chargement du DOM mais cela ne fournis pas un beau code HTML, le script devrait faire partie de l'entête du document et non du corp.

HTML5 nous fournis pour cela un attribut bien pratique : defer qui permet de lier le fichier js dans l'entête et de charger se fichier en différé (après la page HTML).

Voici une structure de base de page HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="./css/style.css">
    <script src="./js/index.js" defer></script>
</head>
<body>

</body>
</html>
```

LES VARIABLES

Il existe 3 manières de déclarer une variable en javascript :

```
var x=5;
let y=true;
const nom="Julien";
```

var est une ancienne manière d'écrire du JS, mieux vaut ne plus l'utiliser, elle pose des soucis de scope.

let est une variable normale, ce n'est pas super intuitif mais var était déjà pris.

const définit une constante : variable non modifiable (attention avec les tableaux et les objets)

À noter : le ; en fin de ligne n'est pas obligatoire en JS

LE TYPAGE

Javascript est un langage interprété, ce qui signifie qu'il n'est pas compilé mais exécuté directement dans le navigateur grâce au moteur d'exécution Javascript intégré à celui-ci.

Javascript est également faiblement typé, ce qui signifie que nous n'avons pas besoin de définir le type d'une variable le moteur d'exécution la déduis en fonction de son contenu. Vous pouvez également changer le type d'une variable en changeant son contenu

```
let x = 5

console.log(x)
console.log(typeof x);

x = "toto"
console.log(typeof x);
```

Les types simples en JS :

number -> Tous les nombres avec ou sans virgule, y compris NaN

```
let x = 5
```

string -> Toutes les chaines de caractères avec ‘ ou ‘’ ou `

```
let x = "test"
```

boolean -> true ou false

```
let x = true
```

undefined -> Les variables non définies et les variables dont la valeur est undefined

```
let x = undefined
```

Les types avancés :

function -> Toutes les fonctions nommés, anonymes, fonctions fléchés ou les classes

```
let x = ()=>{}
```

object -> Les tableaux, les objets, les objets de classe ou null

```
let x = []
```

Arithmétiques

Les opérateurs arithmétiques servent à faire des calculs sur des nombre, on retrouve + - * ou %

L'opérateur % est l'opérateur pour effectuer un modulo (reste de la division entière)

```
console.log(10%3);
```

Le reste de la division entière de 10/3 est 1

Relationnels

Les opérateurs relationnels permettent de faire des comparaisons et renvoient un booléen (true ou false). On retrouve :

- < inférieur
- <= inférieur ou égal
- > supérieur
- >= supérieur ou égal
- == valeur égale
- != valeur différente
- === valeur et type égal
- !== valeur ou type différent

Ces opérateurs sont très utiles pour les conditions et les boucles

Logiques

Les opérateurs logiques permettent d'effectuer des calculs de logique booléenne, il en existe 3 :

&&

résultat	a	b
false	false	false
false	false	true
false	true	false
true	true	true

||

résultat	a	B
false	false	false
true	false	true
true	true	false
true	true	true

!

resultat	a
false	true
true	false

Les opérateurs unaires

Ces opérateurs n'ont besoin que d'une valeur pour faire un calcul :

- ++ ajoute 1 (ex : a++)
- -- retire 1 (ex : a--)
- - inverse le signe (ex : -a)
- + conserve le signe et convertit en nombre (ex : +a)

Les opérateurs d'assignation

Ces opérateurs permettent d'assigner la valeur de droite de l'opération dans la variable de gauche :

- = simple assignation (ex : a=5)
- += addition/assignation (ex : a+=5)
- -=
- *=
- /=
- %=

L'opérateur de concaténation

Pour fusionner deux chaines (ou concaténer) en JS on utilise l'opérateur + ce qui pose problème entre les additions et la concaténation. La règle est simple, si dans une partie du calcul il y a une chaine, l'ensemble du calcul est une concaténation

```
console.log(1 + "2" + 3);
```

Pour résoudre le souci on peut utiliser l'opérateur unaire +

```
console.log(1 + (+"2") + 3);
```

L'opérateur conditionnel ternaire

Cet opérateur est assez spécial, il permet de réaliser une condition simple (if/else) et d'assigner la valeur induite à une variable

```
console.log((10 < 5) ? "if" : "else");
```

Cet opérateur est assez difficile a bien comprendre mais permet quelques raccourcis sur la taille du code

LES BOUCLES ET LES CONDITIONS

Les conditions (if, else if, else)

Les conditions permettent de créer des branches d'exécution différentes du code en fonction d'une condition donnée.

```
let x=5
if(x==0){
    console.log("x=0");
}else if(x<10){
    console.log("x<10");
}else {
    console.log("x est au moins égal à 10");
}
```

L'écriture de multiples conditions (enchainement des else if) peut rendre le code peu lisible

Les conditions (switch case)

Une condition switch peut être plus adaptée si votre code doit tester de nombreuses conditions (ex : test pour savoir quelle touche du clavier est appuyée).

Attention, pour sortir d'un case, il faut utiliser l'instruction break, sinon le code continuera au case suivant

```
switch (x) {
    case 0:
        console.log("x=0");
        break;
    case 1:
    case 2:
    case 3:
```

```
        console.log("x<4");
        break
    default:
        console.log("x est au moins égal à 4");
        break;
}
```

La boucle tant que

La boucle Tant que est utile si l'on ne sait pas quand doit s'arrêter la boucle, comme lors de la lecture d'un fichier ligne à ligne. Il existe cependant un risque de boucle infinie. Il existe deux manières de réaliser une boucle Tant que :

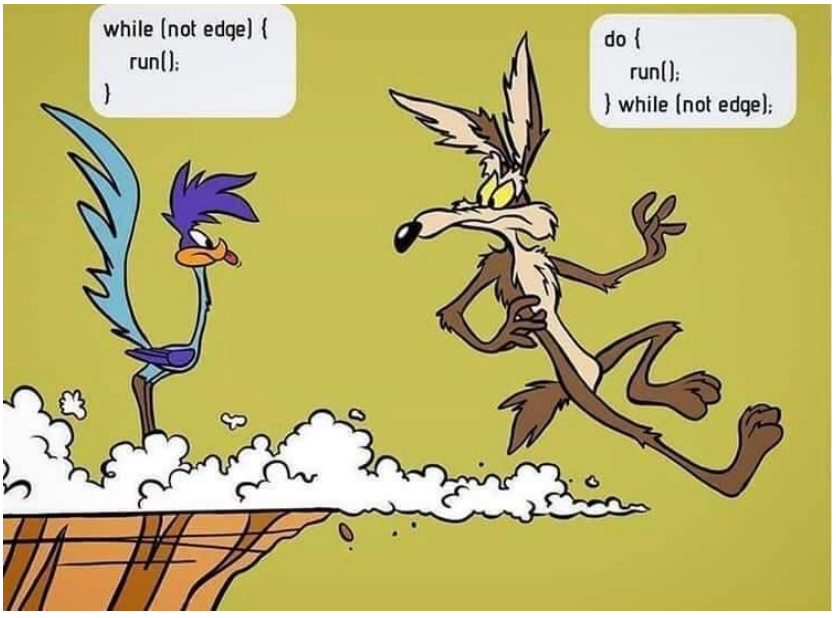
- On peut choisir d'exécuter une première fois la boucle puis de vérifier si sa condition de reboucle est réunie

```
let x = 5
do {
    x++
    console.log("x:" + x + " inférieur à 5");
} while (x < 5)
```

- Ou alors commencer par vérifier la condition puis exécuter la boucle

```
let x = 5
while (x < 5) {
    x++
    console.log("x:" + x + " inférieur à 5");
}
```

À noter : il faut toujours faire attention lorsque l'on utilise une boucle do … while



La boucle for

La boucle for est très utile lorsque l'on peut prévoir à l'avance de la fin d'une boucle, par exemple dans le cadre du parcours d'un tableau. Il existe 4 manières de réaliser une boucle for :

- Le for classique

```
let fruits = ["pomme", "poire", "banane", "fraise"]
for (let i = 0; i < fruits.length; i++) {
    const fruit = fruits[i]
    console.log(fruit);
}
```

- Le for in

```
let fruits = ["pomme", "poire", "banane", "fraise"]
for (let i in fruits) {
    const fruit = fruits[i]
    console.log(fruit);
}
```

- Le for of

```
let fruits = ["pomme", "poire", "banane", "fraise"]
for (let fruit of fruits) {
    console.log(fruit);
}
```

- Le foreach

```
let fruits = ["pomme", "poire", "banane", "fraise"]
fruits.forEach(fruit => {
    console.log(fruit);
})
```

Les 4 manières de faire amènent au même résultat, en termes de performances pure, le for classique est le plus rapide.

LES FONCTIONS

Une fonction permet de définir un bloc de code que nous pourrons exécuter plus tard. Réaliser des fonctions est très utile pour éviter de répéter du code à plusieurs endroits

Une fonction peut contenir : un nom, des paramètres et elle peut renvoyer une valeur

```
function addition(a,b){
    return a+b
}
```

Mais une fonction peut ne pas renvoyer de valeur

```
function addition(a,b){
    console.log(a+b);
}
```

Une fonction peut également ne pas avoir de paramètres

```
function addition(){
    console.log(3+5);
}
```

Elle peut même ne pas avoir de nom

```
const addition = function (a,b){
    return a+b
}
```

Voir ne pas utiliser le mot clé function (cas de la fonction fléché)

```
const addition = (a, b) => {
    return a + b
}
```

ENTRAINEZ VOUS

Inscrivez vous sur la plateforme Tainix

Résolvez ces 3 problèmes (en javascript) :

https://tainix.fr/sandbox/play/PIERRE_FEUILLE_CISEAUX

https://tainix.fr/sandbox/play/SAC_1

https://tainix.fr/sandbox/play/SURVIVAL_1