JAVASCRIPT & TYPESCRIPT

Baptiste FAMCHON & Quentin BURG - QSI

AU MENU CE JOUR

Les origines de Javascript

Naissance du JavaScript: de la création à la popularisation du langage

ECMAScript: standards depuis la création du JavaScript

Stranger Things Javascript : assertions pas forcément intuitives ...

Awesome Javascript : quelques clés du langage (scopes, object, array, ...)

Typescript : la revanche de JS

Problématiques adressées: pourquoi TypeScript est devenu indispensable ?

Des primitifs au moins courants: tour d'horizon de ce qui est permis par TS.

JAVASCRIPT ORIGINS

Back to 90’s, à la création de l’internet

Marc Andreessen, créateur de Netscape.

Netscape, un navigateur assez complet (texte, vidéo, son, plugin, images…)

Novateur ! À l'époque, les navigateurs sont majoritairement texte.

Netscape gagne en popularité, gagne en part de marché et devient le navigateur le plus utilisé.

Bill Gates sent le filon, et commence également à mettre de l'internet partout...

"Pourquoi ne pas moi aussi créer mon navigateur ?"

Microsoft souhaite racheter Netscape… Qui ne se laisse pas avoir !

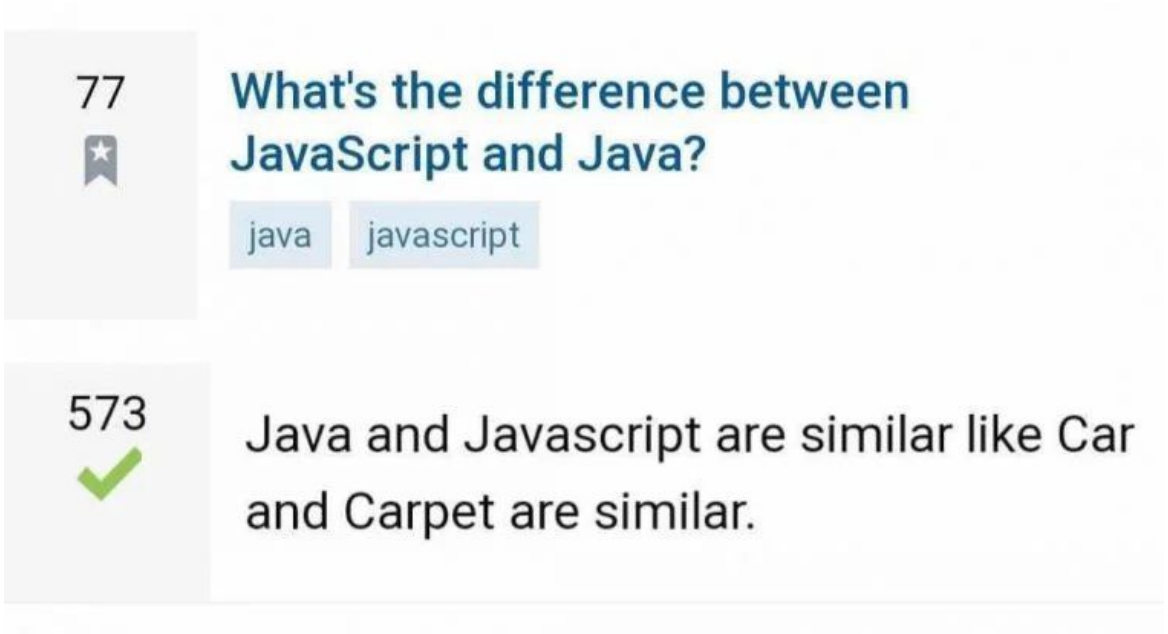
Ils décident alors de développer leur navigateur, qu'ils appelleront…


THE MOST USED
INTERNET BROWSER
TO DOWNLOAD
OTHER BROWSERS
quickmeme.com

Alerte: Microsoft a énormément de part de marché avec ses PC...

Netscape accélérent, les idées fusent: “rendont notre navigateur programmable !”

Brendan Eich est embauché, un crack de l’IT.



Mocha sera renommé JavaScript car censé être un compagnon de Java côté navigateur…

C'est un succès ! Il faut délivrer des features, les feedbacks sont intégrés rapidement…

Ce qui donne lieu aux bizarreries JavaScript que l'on connaît aujourd'hui.

JavaScript 1.0 sort en 1995… Et Microsoft sort IE 3.0 en 1996.

Ce navigateur ne contient pas JavaScript, mais JScript… Une copie, avec ses différences.

C'était à prévoir, Microsoft a bombardé le marché d'IE grâce à ses PC.

Ils ont perdus quelques procès, mais ont quand même réussi à tuer Netscape.

Alors racheté par AOL et laissé pour mort.

ECMAScript - Définitions des standards

ES

En **1996**, **Netscape** (avec Javascript), **Microsoft** (avec JScript, qui est une copie de javascript avec quelques modifications) et **Sun** (qui possédait le nom "Java") ont créé le **TC39** (Technical Comittee 39). Son but est de **superviser le développement de la norme ECMAScript** afin de faire évoluer les **standards** de Javascript.

En 2015 sort **ES 6, une révolution** le standard est **adopté par tous les navigateurs !**

Depuis ES6 (2015), le TC39 sort une nouvelle version de Ecmascript tous les ans, en suivant le TC39 process

ECMAScript - TC39 process

ES

Le **TC39 process** est composé de **5 étapes**.

Toutes les propositions d'évolution de la norme ECMAScript doivent passer par ces 5 étapes.

Le comité TC39 doit approuver ces évolutions étape après étape.

ECMAScript - TC39 stages

ES

Stage 0 (Strawperson) : **idée d'évolution non formalisée** venant d'un membre du comité ou un contributeur reconnu.

Stage 1 (Proposal) : l'**idée est formalisée**, avec le problème soulevé, ses potentielles solutions et/ou ses challenges pour résoudre ce problème.

Stage 2 (Draft) : la proposition est **écrite dans une première version à l'aide d'Ecmascript**. Cette version doit décrire le plus possible, la syntaxe ou l'API qui sera développée. A noter qu'à partir de cette étape, si le comité valide la proposition, elle sera développée et éventuellement intégrée dans la spécification officielle.

ECMAScript - TC39 stages

ES

Stage 3 (Candidate) : la proposition est **disponible pour être relue** et le comité est invité à émettre des feedbacks. Elle doit également passée un certain nombre de tests pour être validée.

Stage 4 (Finished) : la **proposition est prête à être intégrer** dans la prochaine version draft de la spécification.

Stranger Things – JavaScript Edition

```
[] == ![];

true == [];

true == ![];

false == [];

false == ![];

!!'false' == !!'true';

!!'false' === !!'true';

true == 'true';
```

```
null == 0;

null > 0;

null >= 0;

1 < 2 < 3;

3 > 2 > 1;

'10' + '2' = ?

'10' - '2' = ?

[] + [] = ?

{} + [] = ?
```

Awesome JS - Prototype language

En OOP, il existe deux types de langages :

- **classes based**, comme Java ou C++
- **prototype based**, comme Javascript

La grande différence entre ces langages réside dans le fait que l'on ne peut pas définir de classe en Javascript, alors qu'en Java oui.

En effet, le mot-clé **class** n'est que du **sucre syntaxique** pour construire un **nouvel objet** avec certaines propriétés, qui seront définies dans le prototype.

On peut faire également de **l'héritage**, notamment avec la méthode **Object.create** qui prend en paramètre un objet sur lequel se baser et retourner un nouvel objet qui aura les mêmes propriétés.

On peut retrouver tout ce qui concerne le prototype d'un objet dans sa propriété **[[Prototype]]**

Awesome JS - Scopes

Plusieurs possibilités pour **déclarer une variable** : **var , let et const**

Historiquement, il n'existait que **var**.
let et **const** ont été ajouté avec **ES6**.

Les différences se situent au niveau de la **portée** de la variable et s'il est possible de la **réassigner**.

- **var** : portée globale (si déclarée en dehors d'une fonction) ou de la fonction et possibilité de réassignement
- **let** : portée d'un bloc ou d'une fonction, possibilité de réassignement
- **const** : portée d'un bloc ou d'une fonction, pas possible de réassigner

💡 *Tips : par défaut j'utilise **const** partout et si j'ai besoin de réassigner une variable (ce qui est plutôt rare), j'utilise **let**. Ne pas utiliser **var** pour éviter les erreurs.*

Awesome JS - Objects

Les objets sont la structure de données la plus utilisée en Javascript.
Ils sont **mutables** et peuvent contenir des **données hétérogènes**.
Ils possèdent un certain nombre de méthodes utilitaires comme **hasProperty** , **keys** ou **entries**.

```
// Déclaration d'un objet - données hétérogènes
const user = {
    name: "John",
    age: 18
};

// Mutabilité
user.age = 20;


Object.entries(user);
// [ [ 'name', 'John' ], [ 'age', 20 ] ]
```

Awesome JS - Functions

Les fonctions peuvent être déclarées de deux manières différentes

```
// Méthode avec le mot-clé function
function isOver18(user) {
    return user.age >= 18;
}

// Méthode avec const - ES6 style
const isOver18 = (user) => user.age >= 18;
```

💡 *A noter qu'en Javascript, une fonction peut* ***retourner une autre fonction*** *et une fonction peut prendre* ***une fonction en paramètre****, ce qui permet d'écrire du code dans un* ***style fonctionnel****.*

```
const canDrinkAlcohol = (condition) => {
    return (user) => condition(user);
}

canDrinkAlcohol(isOver18)(user);
```

Awesome JS - Arrays

Les Arrays sont très utiles en Javascript, les données peuvent être **hétérogènes** et ils peuvent **être redimensionnés**.

Il y a également beaucoup de **méthodes utilitaires** associées aux arrays, qui permettent de ne pas devoir les parcourir avec un **for** ou un **while**

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

// Map
const doubleDumbArray = dumbArray.map(element => element * 2);
// doubledumbArray : [2, 4, 6, 8, 10]


// Reduce
const sumOfDumbArray = dumbArray.reduce((acc, current) => {
    return acc + current;
}, 0);
```

Awesome JS - Classes

Comme vu précédemment, les classes en Javascript n'en sont pas réellement car c'est un l**angage à prototype**, c'est juste du sucre syntaxique.

Comment déclarer une classe ?

```
class User {

    constructor(name, age) {
        this.name = name;
        this.age = age;
    }

    isOver18() {
        return this.age > 18;
    }
}
```

```
class Admin extends User {
    isAdmin() {
        return true
    }
}
```

On peut étendre une classe avec le mot-clé **extends**

TYPESCRIPT

TypeScript - Une évidence

Typescript est apparu **fin 2012** et est développé par **Microsoft.**

TypeScript == JavaScript (sous stéroïdes), on parle de **Superset**

Il vient **rajouter un typage** statique

Tout code JavaScript est un code **TypeScript valide** *(par contre l'inverse n'est pas vraie)*

Le **code TypeScript** est **transpilé** en JavaScript **grâce à tsc**

➢ **Typescript** permet de **définir des types** dans un code TS, qui sera ensuite **transpilé en JS p**our être **exécuté par un navigateur ou un runtime JS** (Node ou Bun par exemple).

Types - Déclaration

Deux possibilités pour déclarer des types Typescript :

interface ou **type**

La différence se situe au niveau de **l'héritage** : une **interface** va pouvoir être **étendue** pour construire une autre interface, alors qu'un **type** est **figé**, il ne peut pas être étendu.

```
// Interface
interface User {
    name: string;
    age: number;
}

// Type
type User = {
    name: string;
    age: number;
}
```

Types - Primitifs

Trois types primitifs principaux : **string**, **number** et **boolean**.

```
//string
const username: string = "John";

//number
const age: number = 20;

//boolean
const isOver18: boolean = true;
```

⚠️ *Éviter d'utiliser les types avec une majuscule (String), ce sont des objets pour créer de nouveaux éléments JS.*

Types - Composés

Type UNION (aussi appelé type OU ou type SOMME) avec un " **|** " (pipe)

Type INTERSECTION (aussi appelé type ET ou type PRODUIT) avec un " **&** "

Union discriminée = une union de plusieurs intersections avec **au moins une propriété en commun**, appelée le **discriminant**

🤔 *Dans l'exemple, le type Animal est une union discriminée car la propriété "specie" est le discriminant*

```ts
// Object type
type User = {
    name: string;
};

// Array
type Users: Array<User>;

// Union type
type Staff = Employee | Manager

// Intersection type
type Cat = Animal & { specie: "cat" }

// Discriminated union.
// Property `specie` is the discriminant
type Cat = { specie: "cat", name: "Grumpy" }
type Dog = { specie: "dog", breed: "Pinscher" }

type Animal = Cat | Dog;
```

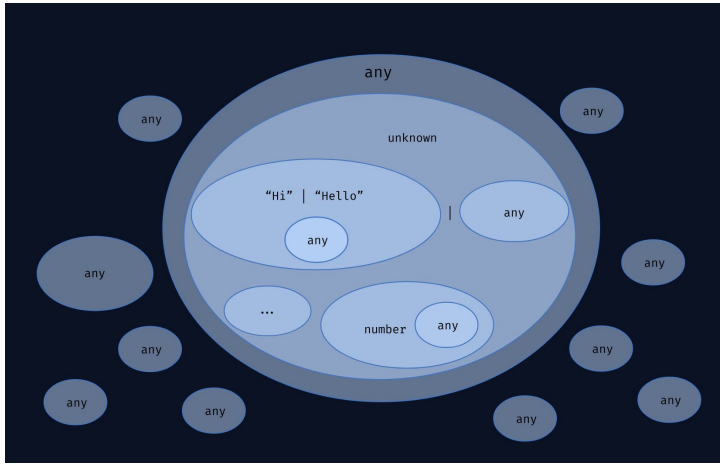
Types - Special type : Any

Type any = désactive le type-checking de TS

A éviter au maximum car peut générer des erreurs si l'on ne fait pas attention

🛠️ On peut configurer TS pour qu'il remonte une erreur lorsqu'on écrit any, avec l'option strict à true.

🛠️ Si vous voulez avoir un typage plus fort, vous pouvez configurer TS pour qu'il vous remonte une erreur lorsqu'une variable n'est pas inférée (implicit any), l'option est noImplicitAny à true



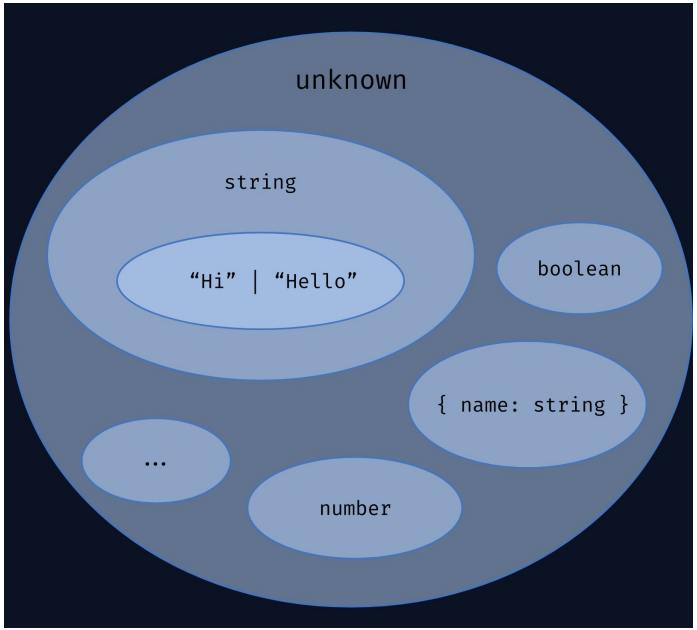
Source [type level typescript]

```
{
  "strict": true,
  "noImplicitAny": true
}
```

Types - unknown VS never

unknown contient **tous les types** utilisables en TS (string, number, ...)

never ne contient **aucun type**, c'est l'ensemble vide



Source [type level typescript]

Ressources

MDN : l'incontournable documentation sur le web

Doc Typescript : pour y trouver toutes les infos concernant Typescript

Type Level Typescript : pour des explications sur le fonctionnement des types TS, avec des exercices

Youtube Matt Pocock : tips sur les types Typescript

Des question ?