Introduction à JavaScript et TypeScript

JavaScript et TypeScript

TypeScript est un langage de programmation basé sur JavaScript. Tout code écrit en TypeScript doit être compilé en JavaScript afin de pouvoir être utilisé. TypeScript, permet entre autres, d'ajouter du support pour le typage des variables, des méthodes et des fonctions et d'assurer une syntaxe constante, peu importe la plateforme visée.

Dans ce document, seuls les sectiosn sur les classes et le typage des variables est spécifique à TypeScript. Tout le reste peut aussi être utilisé en JavaScript.

Types de données

Booléens

```
true
false
0 == false // true
1 == true  // true
```

String

Les chaînes de caractères contiennent de nombreuses fonctions de recherche et d'extraction de sous-chaînes.

```
"abc"
'abc'

"abc".length // 3
"abc".charAt(0) // "a"
"abc"[1] // "b"
"abc" + "def" // "abcdef"
"a" + 1 // "a1"
```

Number

En JavaScript, il n'y a pas de type séparé pour les entiers et les nombres à virgule. Tous les nombres sont stockés en virgule flottante 64 bits. C'est l'équivalent de `double` en Java.

```
11 + 2 // 13
7 - 8  // -1
3 * 4  // 12
3 / 2  // 1.5
```

NaN (Not a Number)

Il s'obtient lorsqu'on tente de faire des opérations mathématiques impossibles ou de convertir une chaîne de caractères en `Number`. Voir la section Conversion de nombres pour plus de détails.

```
0 / 0 // NaN
```

Array

Les valeurs qui y sont contenues sont ordonnées. Elles peuvent être de type différents, mais il s'agit d'une mauvaise pratique.

```
var array = ["a", "b", "d"]
var array2 = [1, "a", true]

array[0] // "a"
array[2] = "c"
array.length // 3
array.push("d") // 4
array.pop() // "d"
array.unshift("z") // 4
array.shift() // "z"
```

Object

Ce sont des paires de clés valeurs non-ordonnées. C'est l'équivalent de `Map` en Java. Les clés et les valeurs peuvent être de différents types. Notez que les clés qui sont des chaînes de caractères n'ont pas besoin d'être entre guillemets lors de l'initialisation.

```
var object = {"a":1, "b":2, "c":3}
var object2 = {a:1, true:"oui", 12:14, obj:{"b":2}}

object.a // 1
object["a"] // 1
object.d = 4

object2[true] // "oui"
object2["true"] // "oui"
object2.obj.b // 2
```

undefined

Le type est utilisé pour représenter une valeur qui n'existe pas.

```
var obj = {"a":1, "b":2};

obj.c // undefined
```

null

Similaire à `undefined`, `null` représente une valeur qui n'existe pas de façon délibérée.

```
var obj = {"a":1, "b":2, "c": null};

obj.c // null
```

Fonctions

```
function x2(x) {
    return x * x;
}

x2(2); // 4
```

Les fonctions ne sont pas obligées d'avoir un nom. Ce sont des fonctions anonymes.

```
setTimeout(function() {
    // S'exécute dans 5000 ms
}, 5000)
```

En JavaScript, les fonctions sont aussi des objets. On peut les déclarer dans des variables ou les passer en paramètre.

```
var x2 = function(x) {
    return x * x;
}

x2(3) // 9
```

Il existe une syntaxe alternatives pour les fonctions. Ce sont les fonctions flèche (arrow functions). Il existe des différences entre les fonctions « régulières » et les fonctions flèches, mais les détails ne sont pas pertinents pour le moment.

```
var x2 = (x) => {
    return x * x;
}

x2(4) // 16
```

Structures de contrôle

Syntaxe des conditions

```
if (x > 10) {
    // Do stuff
}
else if (x < 0) {
    // Do stuff
}
else {
    // Default
}
```

Syntaxe des boucles

```
while (true) {
    // Do stuff
}
```

Syntaxe des boucles for

```
for (var i = 0; i < 10; i++) {
    // Do stuff
}
```

Fonctions et classes utiles

Accès à la console

```
console.log("Bonjour")
console.log(1, 2, 3)

console.error("N'existe pas")
```

Conversion et vérification de nombres

```
parseInt("2") // 2
parseInt("2abc") // 2
parseInt("abc2") // NaN

parseFloat("1.25") // 1.25

isNaN(NaN) // true
isNaN(2) // false
isNaN("2") // false
isNaN("2abc") // true
```

Fonctions mathématiques

Il existe de nombreuses méthodes et constantes disponibles dans la classe `Math`.

```
Math.round(2.5) // 3
Math.floor(2.5) // 2
Math.ceil(2.5) // 3

Math.max(1, 2, 3, 4) // 4
Math.min(1, 2) // 1
```

Date

Il existe de nombreux constructeurs et méthodes disponibles dans la classe `Date`.

```
new Date() // Retourne le temps au moment de la création
new Date("01-03-2020") // Utilise le format MM-JJ-AAAA mais peut changer selon la plateforme
new Date(2020, 00, 03) // En format numérique, les jours débutent à 0, mais les mois à 1
new Date(2020, 00, 03) > new Date(2020, 00, 04) // false
```

Classes

La définition des classes en TypeScript est présentée. La documentation de TypeScript contient plus de détails.

`this` doit toujours être utilisé pour accéder aux propriétés et aux méthodes de la classe.

Notez que le support pour les accesseurs et les mutateurs (get et set) est natif à TypeScript. Attention toutefois à ne pas utiliser le même nom pour l'accesseur ou le mutateur (ex.: radius) et la propriété (ex.: _radius).

```
abstract class Form {
    abstract get area():number;
    abstract toJSON():string;
}

class Circle extends Form {
    private _radius:number = NaN;

    constructor(theRadius:number) {
        super();
        this.radius = theRadius;
    }

    get area():number {
        return Math.PI * this.radius * this.radius;
    }

    get radius():number {
        return this._radius;
    }

    set radius(theRadius:number) {
        if (theRadius < 0) {
            throw "Radius can't be under 0";
        }

        this._radius = theRadius;
    }

    toJSON():string {
        return {"radius": this._radius};
    }
}
```

```
var c:Circle = new Circle(2);

circle.area // 12.566370614359172
circle.radius = 3
circle.radius // 3
circle.toJSON() //{"radius": 3}
```

Imports

Avec TypeScript, il est possible d'utiliser des classes, des fonctions et des variables déclarées dans d'autres fichiers. Il faut toutefois les exporter.

```
export const array:Array<number> = [1, 2, 3];

// Disponible aussi avec les fonctions flèches
export var x2 = function(x:number):number {
    return x * x;
}

export class Employee {
    ...
};
```

Par la suite, on peut les importer avec `import` ou `require`. `import` vérifie la présence du fichier à la compilation tandis `require` le fait à l'exécution. Le chemin d'accès du fichier est relatif à partir du fichier qui fait l'import.

```
import * from "./file";

file.array.length;
file.x2(x);
```

```
import * as f from "./file";

f.array.length;
f.x2(2);
```

```
import {x2} from "./file";

x2(2);
```

```
import {x2 as y2} from "./file";

y2(2);
```

JSON

Le JavaScript Object Notation est un format de représentation des données qui s'inspire des objets et tableaux JavaScript. Il utilise les types suivants : `Object`, `Array`, `Number`, `Boolean`, `String` (guillemets uniquement) et `null`.

```
{
    "forms": [
        {
            "type": "circle",
            "radius": 12,
            "visible": true
        },
        {
            "type": "square",
            "height": 100,
            "width": 100,
            "visible": false
        }
    ]
}
```

JavaScript offre des méthodes natives pour manipuler le JSON.

```
JSON.stringify(...) //Retourne une chaîne de caractères
JSON.parse(...) //Retourne des types JavaScript
```

La méthode toJSON() est utilisée pour convertir des objets en JSON. Si cette méthode n'est pas définie, JavaScript exporte toutes les propriétés de votre objet.

```
class Circle extends Form {
    private _radius:number = NaN;
    ...
    toJSON():string {
        return {"radius": this._radius};
    }
}
```

```
public getForm(req: Request, res: Response, next: NextFunction) {
    var c:Circle = new Circle(12);
    res.status(201).send({"form": c}); //toJSON appelée automatiquement
}
```

Gestion des erreurs

La gestion des erreurs est limitée en JavaScript. Seul le type `Error` existe. Toutefois,vous pouvez définir vos propres types d'erreurs.

```
class NotFound extends Error {
    private _code:number = NaN;

    constructor(message:string) {
        super(message);
        this._code = 404;
    }

    get code():number {
        return this._code;
    }

    toJSON():object {
        return {"code": this._code, "message": this.message};
    }
}
```

```
try {
    throw new NotFound("Not found");
}
catch (error) {
    if (error instanceof NotFound) {
        // Do stuff
    }
    else {
        // Do stuff
    }
}
```

Promesses

Une promesse représente une valeur qui n'est pas encore connue ou disponible, par exemple, le résultat d'un appel vers un site web ou de n'importe quelle opération asynchrone. Une promesse peut être en attente, tenue ou rompue. Il s'agit d'un proxy (patron GoF).

Il est possible de créer des promesses, mais vous aurez plus souvent à les consommer.

```
maPromesse.then(fonctionEnCasDeSucces, fonctionEnCasEchec)
```

```
maPromesse.then(fonctionEnCasDeSucces)
          .catch(fonctionEnCasEchec)
          .then(fonctionAppeleePeuImporteLeResultat)
```

```
fetch("https://monsiteweb.ca").then(function(res)
{
    console.log(res); // Objet de type Response
}, function(err)
{
    console.log(err); // Message d'erreur
});
```

Async/await

Puisque les promesses sont asynchrones, elles sont souvent utilisée avec la déclaration `async` et l'opérateur `await`.

```
async function doFetch() {
    var result = await fetch("https://monsiteweb.ca");
    console.log(result);
}

doFetch();
```

Notez l'ordre d'exécution des `console.log`.

```
async function doFetch() {
    console.log("Deuxième");
   var result = await fetch("https://monsiteweb.ca");
   console.log(result); // Quatrième
}

console.log("Premier");
doFetch();
console.log("Troisième")
```

```
"Premier"
"Deuxième"
"Troisième"
Response {...}
```

Pour en savoir plus

JavaScript

- https://javascript.info/

TypeScript

- https://riptutorial.com/fr/typescript (en français)
- https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes-oop.html

Références

Patrick Stillhart. 2019. JavaScript in a nutshell. Repéré à https://wiki.stillh.art/page/SyAWy65o/JavaScript-in-a-nutshell

Microsoft. s.d. TypeScript Documentation. Repéré à https://www.typescriptlang.org/docs

Jennifer Fu. 2016. A Handy Guide to Export and Import Modules for JavaScript and TypeScript. Repéré à https://medium.com/better-programming/a-handy-guide-to-export-and-import-modules-for-javascript-and-typescript-6cff8e47d554

Mozilla and individual contributors. 2020. Utiliser les promesses. Repéré à https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Utiliser_les_promesses

Mozilla and individual contributors. 2019. async function. Repéré à https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/async_function