TypeScript et JavaScript avancé

Typescript

Langage orienté objet **fortement typé** construit "par-dessus" JavaScript
→ "sur-ensemble syntaxique de JavaScript"

Développé par Microsoft

Open source

Variables

Variable muable (à éviter) :

```
let a: number = 3;
a = 4; // OK
```

Variable immuable :

```
const b: number = 3;
b = 4; // Erreur de compilation !
```

Typage

TypeScript est un langage **fortement typé** : chaque variable a un type et il n'est pas possible de mettre une valeur d'un autre type dans une variable.

```
const a: number = 'Test'; // Erreur de compilation !
```

Cette validation à la compilation permet d'améliorer la sécurité du code et sa maintenabilité.

Inférence de type

La définition du type d'une variable ou du retour d'une fonction n'est pas obligatoire si le compilateur TypeScript peut le déterminer (l'inférer) :

```
const a = 'Test'; // a est une string

let b = a; // b est une string

b = 1; // Erreur de compilation !
```

Fonctions

Une fonction TypeScript peut être créée avec le mot-clé **`function`**.

Le type de chacun de ses paramètres doit être précisé, et son type de retour peut être inféré.

```
function sum(a: number, b: number): number {
  return a + b;
}

function stringLength(s: string) { // Type inféré grâce à la valeur du return
  return s.length;
}
```

Higher order functions

En JavaScript (et TypeScript), une fonction peut être passée en paramètre ou être retournée par une autre fonction. On parle alors, pour cette dernière, de **fonction d'ordre supérieur**.

Chaque fonction a un type noté : `(p1: A, p2: B, ...) => T`

```
function sum(a: number, b: number): number {
  return a + b;
}
// sum : (a: number, b: number) => number

function operation(op: (a: number, b: number) => number, a: number, b: number): number {
  return op(a, b);
}
// operation : (op: (a: number, b: number) => number, a: number, b: number) => number

const s = operation(sum, 4, 3); // 7
```

Fonctions lambda / fat arrow functions

Il est possible de stocker dans des variables (temporaires ou pas) des fonctions "anonymes" en utilisant la syntaxe "fat arrow functions" :

```
const sum = (a: number, b: number): number =>
  a + b; // Accolades et return optionnels si une seule instruction

const operation = (op: (a: number, b: number) => number, a: number, b: number): number =>
  op(a, b);

const s = operation(sum, 4, 3); // 7

const m = operation(
  (a: number, b: number) => a * b,
  4,
  3
); // 12
```

Classes

Le mot-clé class permet de créer une classe en TypeScript, de manière similaire à JavaScript :

```
class User {
  firstname: string;
  lastname: string;
  age: number;

  constructor(firstname: string, lastname: string, age: number = 0) {
    this.firstname = firstname;
    this.lastname = lastname;
    this.age = age;
  }
}

const u = new User('Toto', 'Lala', 25);
const u2 = new User('Foo', 'Bar');
```

Interfaces

Les interfaces permettent de définir des structures de données simples sous la forme d'objets JavaScript :

```
interface User {
  firstname: string,
  lastname: string,
  age?: number
}

const u: User = { firstname: 'Toto', lastname: 'Lala', age: 25 };
const u2: User = { firstname: 'Foo', lastname: 'Bar' };
```

Interfaces

Il est possible de créer des fonctions permettant de créer un objet JavaScript du type d'une interface :

```
const User = (
  firstname: string,
  lastname: string,
  age: number = 0
): User => ({
  firstname,
  lastname,
  age
});

const u: User = User('Toto', 'Lala', 25);
const u2 = User('Foo', 'Bar');
```

Alias de types

Lorsqu'un type est trop complexe ou difficile à lire, TypeScript offre la possibilité de créer des alias de type :

```
type UserOrError = Promise<Either<User, Error>>;

type Color = 'BLUE' | 'RED' | 'GREEN';

type UserWithAddress = User & { address: string, city: string, zipCode: string };
```

Génériques

TypeScript fourni la possibilité de définir des classes englobant un type générique :

```
class Request<T> {
  body: T;

  constructor(body: T) {
    this.body = body;
  }
}
```

Génériques

Les fonctions peuvent également être définies avec un type générique :

```
function getBody<T>(request: Request<T>): T {
  return request.body;
}

const getBody = <T extends any>(request: Request<T>): T => request.body;
```

Typage avancé : extends

Une interface peut hériter des propriétés d'un autre avec le mot-clé **`extends`** :

```
interface User {
  firstname: string,
  lastname: string,
  age: number
}

interface UserWithAddress extends User {
  address: string,
  city: string,
  zipCode: string
}
```

Typage avancé : union

Un type peut permettre à une variable de contenir une valeur parmis plusieurs grâce à l'opérateur d'union | :

```
type Color = 'BLUE' | 'RED' | 'GREEN';

const c: Color = 'BLUE';

const c2: Color = 'YELLOW'; // Erreur de compilation !
```

Typage avancé : intersection

Un type peut combiner deux autres types ou interfaces grâce au mot-clé `&`, similaire au mot-clé `extends` :

```
type UserWithAddress = User & { address: string, city: string, zipCode: string };
```

Typage avancé : objets dynamiques

Il est possible de définir un type d'objet aux propriétés dynamiques, en spécifiant le **type des clés** et le **type des valeurs** des objets de ce type :

```
type Color = 'BLUE' | 'GREEN' | 'RED';

type Colors = { [key in Color]: string };
// Ou :
type Colors = Record<Color, string>;

// Sans validation des clés : { [key: string]: string } ou Record<string, string>

const colors: Colors = {
  'BLUE': '#0000FF',
  'GREEN': '#00FF00',
  'RED': '#FF0000',
  'YELLOW': '#FFFF00' // Erreur de compilation !
};
```

Spread operator

Permet de "propager" les propriétés d'un objet JavaScript dans un autre lors de sa création :

```
const user: User = {
  firstname: 'Toto',
  lastname: 'Lala',
  age: 25
};

const userWithAddress: UserWithAddress = {
  ...user,
  address: '1 rue Truc',
  city: 'Strasbourg',
  zipCode: '67000'
};
```

Rest operator

Lors de la destructuration des propriétés d'un objet JavaScript ou d'un tableau en différentes variables, permet de récupérer le "reste" des propriétés de l'objet dans une seule variable.

Exemple :

```
const user = {
  firstname: 'Toto',
  lastname: 'Lala',
  age: 25,
  city: 'Paris'
};

const { firstname, lastname, ...infos } = user;

console.log(firstname); // Toto
console.log(lastname); // Lala
console.log(infos); // { age: 25, city: 'Paris' }
```

Imports et exports

Il est d'usage de structurer son code en plusieurs fichiers.

Pour pouvoir utiliser des éléments d'autres fichiers, il est donc nécessaire de les exporter puis de les importer :

```
// folder/file.tsx
export const sum = (a: number, b: number) => a + b;

interface User {
  firstname: string,
  lastname: string,
  age: number
}
export default User; // Un export par défaut maximum par fichier

// index.tsx
import { sum } from 'folder/file.tsx';
import User from 'folder/file.tsx';
```

Imports et exports

Il est possible de renommer un import :

```
import { sum as add } from 'folder/file.tsx';
```

Il est également possible d'importer tous les exports "normaux" et d'y accéder via une variable :

```
import * as fn from 'folder/file.tsx';

fn.sum(3, 4);
```

Exercice

Exercice

Rendez vous sur le site https://www.typescriptlang.org/play et imaginez des structures de données et les fonctions permettant de répondre au besoin suivant :

Une médiathèque est un établissement permettant à des utilisateurs d'emprunter des médias : livres, films ou jeux, chacun avec des caractéristiques différentes (à vous d'imaginer lesquelles).

Un média peut être présent en plusieurs exemplaires et chaque exemplaire ne peut être emprunté que par un seul utilisateur à la fois, mais un média qui ne possède plus d'exemplaire disponible ne peut plus être emprunté.

Autres outils

Node.js

Un serveur web en JavaScript

Première version en 2009

Open source

Basé sur le moteur V8 de Chrome

Basiquement mono-thread

HTTP first

npm

Node Packet Manager (enfin …)

Première version en 2010

Gestionnaire de paquets par défaut de Node.js

Utilisable en CLI

Permet aussi d’exécuter des scripts

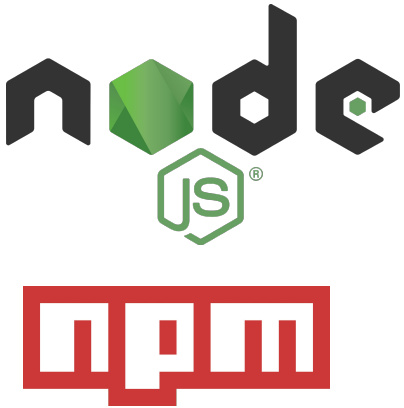
Installation des outils

Node.js

Télécharger la version 16.18.0

https://nodejs.org/

Installe Node.js + NPM



Vérification de l'installation

```
$ node -v
v16.18.0

$ npm -v
8.19.2
```

Exercice

Exercice

Installez Node et npm !