DÉCOUVERTE DE TYPESCRIPT

RAPPELS JAVASCRIPT

- javascript est un langage avec un **typage dynamique**
- cela ne veut pas dire que les variables ne sont pas typées
- A tout moment, le type d'une variable est celui de la valeur qu'il contient : `string`, `number`, `boolean`, `object` ... ou `any` si la variable est déclarée sans contenu.

RAPPELS JAVASCRIPT

- javascript est un langage **interprété**
- les opérations possibles sur une variable dépendent de son type courant
- une opération impossible dans un script déclenchera une erreur au moment de son interprétation par le navigateur.

RAPPELS JAVASCRIPT

Par exemple:

```
const a = 2;
const b = a.toLowerCase();
```

affichera dans la console du navigateur :

```
Uncaught TypeError: a.toLowerCase is not a function
```

Un des intérêts de typescript c'est la détection de ce type d'erreur plus tôt, lors d'une étape de "compilation", ou encore mieux, quand on écrit le code, à l'aide d'un environnement de développement adapté.

TYPESCRIPT

- typescript est un langage, développé par l'entreprise Microsoft
- c'est une surcouche de javascript
- un navigateur ne peut pas interpréter le typescript
- le code typescript est transpilé ("compilé") en javascript
- c'est le moteur javascript qui exécute
- typescript apporte un **typage statique**, gage de maintenabilité (documentation du code) et de développement simplifié (détection d'erreurs, autocomplétion dans l'IDE).

Documentation TypeScript (avec *Playground*)

TYPESCRIPT

- les principaux types primitifs sont `string`, `number` et `boolean`

```
const nom: string = 'toto';
const age: number = 18;
const estUnEtudiant: boolean = true;
```

- le type `undefined` représente le fait qu'une variable n'est pas initialisée.
- le type et la valeur `null` existent mais il est recommandé de limiter leur utilisation.

TYPESCRIPT

Exemple de détection d'erreur par un IDE qui prend en charge typescript :

```
const a = 2;
const b = a.toLowerCase();
```

Property 'toLowerCase' does not exist on type '2'. ts(2339)

TYPESCRIPT

- les tableaux sont de type `Array` et leur contenu est typé.

```
let nombres: number[] = []; // ou let nombres: Array<number> =[]
nombres.push(42);
nombres.push('42'); // erreur de compilation
```

- les propriétés des objets littéraux sont également typées. Le ? après le nom d'une propriété signale que celle-ci est optionelle.

```
const person: {
  firstname?: string,
  lastname: string
} = {
  lastname: 'Dupont'
};
```

TYPESCRIPT

- pour une variable sans précision de type, il existe le type spécial any.😈

```
let v: any =  2;
v = 'hello'; // compile
const n: number = v; // compile
```

- le type unknown est préférable. Il accepte n'importe quelle valeur mais ne peut pas être affecté aux autres types sans vérification (sauf any).

```
let v: unknown =  2;
let n: number = 0;
v = 'hello'; // compile
n = v; // erreur de compilation
if(typeof v === "number") n = v; // compile
```

TYPESCRIPT

- l'union de types est possible pour une variable qui pourra prendre des valeurs de plusieurs types.

```
let v: number | undefined;
```

- l'union de valeurs est possible pour une variable qui pourra prendre un nombre restreint de valeurs (sans passer par des enum).

```
let v: 'toto' | 'tutu' = 'toto';
```

TypeScript déduit les types des variables des valeurs utilisées pour les initialiser.👍

```
let x = 3;
```

TYPESCRIPT

Typez explicitement les retours de fonctions (même si inférence de type possible).👍

- utiliser le type `void` pour une fonction sans valeur de retour (`undefined`)
- le ? après le nom du paramètre signale que celui-ci est optionnel

```
function getFavoriteNumber(): number {
  return 20;
}

function sayHello(firstname: string, lastname?: string ): void {
  console.log(`hello ${firstname} ${(lastname)? lastname : ''}`);
}
```

- il est possible de donner une valeur par défaut à un paramètre

```
function sayHello(firstname: string, lastname: string = 'Anonyme'): void {
  console.log(`hello ${firstname} ${lastname}`);
}
```

TYPESCRIPT

- les interfaces permettent de définir la structure que doit avoir un objet (= contrat sur la structure) qu'il s'agisse d'une instance de classe ou d'un objet littéral. Les noms des interfaces sont utilisés comme des types.
- le mot-clé readonly permet de dire que la valeur de la propriété ne pourra pas changer après avoir été initialisée.

TYPESCRIPT

```
interface Personne {
    readonly id: string;
    nom: string;
    prenom: string;
}

function nomComplet(p: Personne): string {
  p.id = '0';// erreur de compilation
  return `${p.prenom} ${p.nom}`;
}

console.log(nomComplet({ id: '0', prenom: 'Tom', nom: 'Pouce' }));

console.log(nomComplet({ id: '1', prenom: 'Tom' }));
// erreur de compilation
```