

## TypeScript - types simples

1. Considérer les déclarations de *variables* suivantes. Quels sont les types inférés dans VS Code ?

```
let v1 = 2
let v2 = true
let v3 = [1, 3]
let v4 = [true, false, true]
let v5 = [1, 3, 5, 7, 9]
let v6 = ["andrew", "ben", "john", "peter"]
```

2. Rajoutez les types inférés explicitement dans les déclarations ci-dessus.
3. Pouvez-vous rajouter les instructions suivantes après la première ligne ? Justifiez votre réponse.

```
v1 = 3
v1 = true
```

4. Transformez la déclaration de v1 comme suit. Que constatez-vous ?

```
let v1: (number | boolean) = 2
```

5. Considérer les déclarations de *constantes* suivantes. Sont-elles correctes ? Pour celles qui sont correctes, quelles sont les constantes définies, leur types et leur valeurs ?

```
const c1: number = 2
const c2: boolean = true
const c3: [number, number] = [5, 7]
const c4: [string, boolean, string] = ["two", true, "y"]
const c5: number[] = [5, 7]
const c6: string[] = ["andrew", "ben", "john"]
```

6. Tester les déclarations précédentes sans les informations de type. Que remarquez-vous ?
7. Considérer les déclarations de variables suivantes. Quelles sont les variables définies, leur types et leur valeurs ?

```
let [r1, s1] = [12, 10]
let [r2,] = [12, 10]
let [, s2] = [12, 10]
let [r3, r3] = [12, 10]
let u = [1, 2, 3, 4]
let v = [5, 6]
let w = [...u, ...v]
let [h, ...t] = w
```

8. Que calculent les fonctions suivantes :

```
function fone(x: number): [number, number, number] { return [x, x, x] }
function ftwo(x: number) { return [x, x, x] }
```

```

function fthree(x: number, y: number): [string, number] {
    return [(x.toString() + "h"), (y + 1)]
}

function ffour(x: number, y: number) {
    return [(x.toString() + "h"), (y + 1)]
}

```

9. Les fonctions génériques `hd` et `tl` renvoient le premier élément (*head*) et le reste (*tail*) d'un tableau (le reste est obtenu en enlevant la tête).

```

function hd<A>(items: A[]): A {
    const [h, ...t] = items
    return h
}

function tl<A>(items: A[]): A[] {
    const [h, ...t] = items
    return t
}

```

Tester ces fonctions sur des exemples.

10. Ecrire les fonctions suivantes en utilisant uniquement des définitions par cas (éventuellement récursives) et l'opérateur *spread* :

```

function is_empty<A>(items: A[]): boolean

function concat<A>(items1: A[], items2: A[]): A[]

function reversed<A>(items: A[]): A[]

function map<A, B>(f: (_: A) => B, items: A[]): B[]

function filter<A>(f: (_: A) => boolean, items: A[]): A[]

```

## Exécution du code TypeScript

Le programme `main.ts` peut être exécuté directement par `node`<sup>1</sup> depuis la version 22.18.0 :

```
node main.ts
```

1. <https://nodejs.org/en/learn/typescript/run-natively>

## Annexe

### Compilation en JavaScript

Vous devez d'abord installer le compilateur TypeScript ainsi (ou globalement avec l'option `-g`) :

```
npm install typescript
```

Un programme exemple `main.ts` peut ensuite être compilé ainsi (l'option `--strict` active un maximum de contrôles et l'option `--target` détermine la version JavaScript du code généré) :

```
npx tsc --strict --target ES2022 main.ts
```

Les options peuvent aussi être rassemblées dans un fichier de configuration `tsconfig.json` contenant par exemple :

```
{
  "compilerOptions": {
    "target": "ES2022",
    "strict": true,
    "erasableSyntaxOnly": true
  }
}
```

Il devient alors possible de compiler tous les fichiers TypeScript du répertoire simplement ainsi :

```
npx tsc
```

Il peut être pratique d'utiliser le mode de compilation incrémental (où le fichier est recompilé automatiquement à chaque modification). La commande est la suivante :

```
npx tsc --watch
```

Ce fichier de configuration `tsconfig.json` est aussi utilisé pour compiler automatiquement depuis VS Code quand on sauvegarde le fichier (en lançant `run task "tsc watch"`).

### Exécution du code JavaScript

Le programme `main.js` ainsi généré peut ensuite être exécuté par `node` :

```
node main.js
```

### Installation d'un REPL TypeScript

Il est parfois pratique de disposer d'un REPL permettant d'exécuter directement du code TypeScript (sans passer par la phase de compilation JavaScript). C'est possible en installant d'abord `deno`<sup>2</sup> par exemple ainsi (ou globalement avec l'option `-g`) :

```
npm install deno
```

et en lançant ensuite le REPL `deno` à la place de `node`.

---

2. [https://docs.deno.com/runtime/getting\\_started/installation](https://docs.deno.com/runtime/getting_started/installation)