Programmation Fonctionnelle : des concepts aux applications web (NFP119)

TypeScript : types simples

1. Considérer les déclarations de *variables* suivantes. Quels les types inférés ?

```
var v1 = 2
var v2 = true
var v3 = [1, 3]
var v4 = [true, false, true]
var v5 = [1, 3, 5, 7, 9]
var 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 ?

```
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 ?

```
var [r, s] = [12, 10]
var [r,] = [12, 10]
var [, s] = [12, 10]
var [r, r] = [12, 10]
var u = [1, 2, 3, 4]
var v = [5, 6]
var w = [...u, ...v]
var [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 `explode`, `implode`, `rev`, `hd` and `tl` sont définies ainsi :

```
function explode(s: string): string[] {
    return s.split("");
}
function implode(a: string[]): string {
    return a.join("");
}
function rev(a: string[]): string[] {
    return a.reverse();
}
function hd(a: string[]): string {
    const [h, ...t] = a
    return h;
}
function tl(a: string[]): string[] {
    const [h, ...t] = a
    return t;
}
```

   - La fonction `explode` transforme une chaîne de caractères en un tableau de chaînes de caractères (contenant chacune un seul caractère).
   - La fonction `implode` correspond à la fonction inverse (qui concatène toutes les chaînes de caractères du tableau).
   - La fonction `rev` retourne un tableau.
   - Les fonctions `hd` et `tl` renvoient le premier élément (*head*) et le reste (*tail*) d'un tableau (le reste est obtenu en enlevant la tête).

   Evaluer et afficher le résultat des expressions suivantes :

```
hd(explode("south"))
hd(tl(explode("north")))
hd(rev(explode("east")))
hd(tl(rev(explode("west"))))
```

10. Ecrire les fonctions suivantes en utilisant la récursivité et l'opérateur *spread* :

```
function length<A>(l: A[]): number

function concat<A>(l1: A[], l2: A[]): A[]

function reverse<A>(l: A[]): A[]

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

function filter<A, B>(f: (_: A) => boolean, arr: A[]): A[]
```

Annexe

Installation

Pour compiler, assurez-vous que le fichier de configuration `package.json` est présent et qu'il contient :

```
{
    "name": "test",
    "version": "1.0.0",
    "type": "module",
    "dependencies": {
        "typescript": "^4.9.5"
    }
}
```

Depuis le terminal, lancer alors la commande :

```
npm install
```

Compilation

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 ES2020 main.ts
```

Toutefois, comme le temps de compilation est significatif, il est préférable d'utiliser le mode de compilation incrémental (où le fichier est recompilé automatiquement à chaque modification). La commande est la suivante :

```
npx tsc --watch --strict --target ES2020 main.ts
```

Les options peuvent aussi être rassemblée dans un fichier de configuration `tsconfig.json` contenant donc :

```
{
  "compilerOptions": {
    "target": "ES2020",
    "strict": true
  },
  "files": [
    "main.ts"
  ]
}
```

Il est alors possible de lancer simplement :

```
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").

Execution

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

```
node main.js
```