ENSEIRB-Matmeca
2024-2025

1ère année - Filière Info
PG104

TD n°8 - Techniques : typage en Typescript

▷ Le langage Typescript (`https://www.typescriptlang.org`) est un sur-langage d'Ecmascript qui permet (entre autres) d'ajouter des annotations de type aux expressions du langage. Les fichiers Typescript ont une extension `.ts` pour les différencier des fichiers `.js`. Le langage est fourni avec un compilateur `tsc` qui transforme les fichiers `.ts` en `.js`.
Ce langage s'installe simplement comme une bibliothèque Node.js :

```
npm install typescript
```

▷ Pour utiliser le compilateur Typescript, deux options possibles :

— (*conseillé*) Utiliser le compilateur en mode *watch* (surveillance) :

```
npx tsc --project ⟨tsconfig.json⟩ --watch
```

Ceci a pour effet de lancer un serveur `tsserver` qui se charge une seule fois en mémoire, puis s'exécute à chaque modification d'un fichier de code. Il faut lui dédier un terminal pour le garder lancé en continu.

— (*déconseillé*) Compiler les fichiers un par un à la main :

```
npx tsc ⟨file.ts⟩ --outFile ⟨file.js⟩
```

Cette façon de faire a le mauvais goût d'*ignorer* les fichiers de configuration, en plus d'être assez lente, car le compilateur est redémarré à chaque compilation. Mais elle est plus proche d'un fonctionnement comme avec `gcc`.

Rappelons que toutes les compilations sont censées être faites depuis *la racine du dépôt*.

Exercice 1: Rencontre du 3ème type

Dans cet exercice, nous allons reprendre un exercice réalisé précédemment, en s'exerçant à ajouter les types. Le but de cet exercice est donc de construire un fichier Typescript contenant des indications de types aussi précises que possible.

> ▷ Le code de correction dans l'exercice "Fonctions et Dérivées" est fourni dans les sources.

1. Comment écrit-on en Typescript le type d'une fonction prenant un nombre en paramètre et renvoyant un nombre ?
2. Typer la fonction `myPolynomial` qui évalue le polynôme suivant :

$$x \mapsto 3x^2 + 4.7$$

3. Typer la fonction `differentiate2`, ainsi que les 2 valeurs de test `specDiff2OnH` et `specDiff2OnF`.
4. Typer la valeur `differentiateCurry`, ainsi que les 2 valeurs de test `specCurryCos` et `specCurrySin`.

Exercice 2: Des listes et des types

Dans cet exercice, nous allons reprendre un exercice réalisé précédemment, en s'exerçant à ajouter les types. Le but de cet exercice est donc de construire un fichier Typescript contenant des indications de types aussi précises que possible.

> ▷ Un ensemble de fonctions non typées sur les listes chaînées est fourni dans les sources. Pour cet exercice, il est demandé de *ne pas* faire les `import`s usuels sur les listes.

Pour commencer, nous nous proposons de typer les fonctions sur les paires pointées. Le type des paires pointées est le suivant :

```
// The type for pointed pairs (T, U are type variables)
type PointedPair<T, U> = {
    car: T;
    cdr: U;
};
```

Ce type est *générique*, au sens où les variables `T` et `U` sont des types inconnus, qui peuvent être remplacés par n'importe quel type concret. Pour typer une fonction générique, il faut utiliser la syntaxe suivante :

```
// Creates a pointed pair with elements 'aCar' and 'aCdr'
function cons<T, U>(aCar: T, aCdr: U) : PointedPair<T, U> {
    return { car: aCar, cdr: aCdr };
}
```

1. Ajouter les indications de types sur les fonctions `car` et `cdr`.

Le type sur les listes est un type plus complexe, parce qu'il est récursif. Néanmoins, il est possible de le faire coller à la définition formelle des listes. Nous proposons d'utiliser le type suivant :

```ts
// The type for the linked lists (T is a type variable)
type List<T> = undefined | { car: T, cdr: List<T> };
```

▷ Dans cet exemple, il n'est pas possible (pour une limitation de Typescript sur les types récursifs) d'utiliser directement `type List<T> = undefined | PointedPair<T, List<T>>;`

Une liste est donc une *disjonction* de deux choix possibles : soit un objet vide, soit une paire pointée. Nous fournissons dans les sources quelques fonctions à typer sur les listes.

2. Ajouter les indications de types sur les fonctions `isEmpty`, `isNonEmpty`, `head` et `tail`.
   Qu'est-il nécessaire d'ajouter à ces fonctions si on veut les typer correctement `head` et `tail` ?
3. Ajouter les indications de type sur les fonction `listMap` et `listFoldR` (il y a une solution utilisant deux variables de types).

Exercice 3: Sagesse de l'Antiquité

Considérons une base de données de personnages fournis avec leurs noms, dates de naissance et de mort. Pour simplifier [1], nous nous limiterons à un simple tableau de triplets disponible dans les sources, et dont voici un court extrait :

```ts
type Person = { name: string, birth: number, death: number };
const db : Array<Person> = [
    { name: "Thales", birth: -625, death: -547 },
    { name: "Anaximandre", birth: -600, death: -546 },
    { name: "Heraclite", birth: -544, death: -480 },
]
```

L'idée de cet exercice consiste à manipuler cette base de données en utilisant des méthodes d'ordre supérieur pour la parcourir (`map`, `reduce` ...). On pourra chercher l'inspiration en regardant la liste des méthodes disponibles pour les tableaux.

1. Extraire de cette base de données la liste des noms dans l'ordre alphabétique.
2. Extraire de cette base la personne née le plus tôt chronologiquement.
3. Extraire de cette base la durée de vie la plus longue de toutes ces personnes.
4. Extraire de cette base la liste des personnes mortes à un âge pair.

1. Il existe des bibliothèques Node.js pour se connecter à des bases de données classiques (c.f. par exemple `https://github.com/mysqljs/mysql` ou `https://github.com/brianc/node-postgres`).