Une introduction rapide à TypeScript

Qu'est-ce que TypeScript ?

Définition

TypeScript est un langage de programmation libre et open-source développé et maintenu par Microsoft. Il est un sur-ensemble de JavaScript, c'est-à-dire qu'il est compatible avec JavaScript et qu'il peut être utilisé dans n'importe quel projet JavaScript. Il est principalement utilisé pour le développement d'applications web front-end (Angular, React, Vue.js, etc.) mais peut également être utilisé pour le développement d'applications back-end (Node.js, etc.).

Pourquoi utiliser TypeScript ?

TypeScript est un langage de programmation qui apporte des fonctionnalités supplémentaires à JavaScript. Il permet de développer des applications plus robustes et plus fiables. Il est également plus facile à maintenir et à débugger. Il est également plus facile à lire et à comprendre pour les développeurs qui ne connaissent pas JavaScript.

Quelles sont les fonctionnalités de TypeScript ?

TypeScript apporte plusieurs fonctionnalités supplémentaires à JavaScript :

- **Typage statique** : permet de définir le type d'une variable ou d'une fonction. Par exemple, si vous définissez une variable `x` de type `number`, vous ne pourrez pas lui affecter une valeur de type `string`. Cela permet de détecter des erreurs de programmation plus facilement. Ces erreurs sont détectées à la compilation et non à l'exécution.
- **Classes** : Vous découvrirez ce concept dans ce cours. Une classe est un modèle qui sert de plan pour créer des objets. On peut la comparer à un moule qui définit la structure d'un objet. Chaque classe regroupe des caractéristiques (données) et des actions (fonctions) propres à l'objet qu'elle représente. Par exemple, une classe "Voiture" peut inclure des caractéristiques comme la couleur, la marque et la vitesse, ainsi que des actions comme accélérer ou freiner.
- **Gestion des erreurs** : permet de détecter et de gérer les erreurs de programmation. Par exemple, si vous essayez d'ouvrir un fichier qui n'existe pas, vous obtiendrez une erreur.
- **Gestion des exceptions** : permet d'anticiper et de gérer les erreurs qui surviennent lors de l'exécution d'un programme, évitant ainsi les plantages imprévus. Vous vous focaliserez sur ces deux derniers concepts dans d'autres modules.
- ...

Quelques conventions de nommage

- Les variables doivent être nommées en `camelCase`. Par exemple, `maVariable`. Les variables doivent être nommées de manière à ce que leur nom soit compréhensible par un autre développeur. Par exemple, `maVariable` est plus compréhensible que `x`.

- Les fonctions doivent être nommées en `camelCase`. Par exemple, `maFonction`. Les fonctions doivent être nommées de manière à ce que leur nom soit compréhensible par un autre développeur. Par exemple, `maFonction` est plus compréhensible que `x`.
- Les fichiers doivent être nommés en `kebab-case`. Par exemple, `mon-fichier.ts`. Les fichiers doivent être nommés de manière à ce que leur nom soit compréhensible par un autre développeur. Par exemple, `mon-fichier.ts` est plus compréhensible que `x.ts`.

Quelques conventions de programmation

Il est **recommandé** d'utiliser :

- Des **variables** plutôt que des valeurs en dur.

Exemple : plutôt que d'écrire `console.log("Bonjour")`, il est préférable d'écrire `console.log(bonjour)` avec `bonjour` une variable de type `string` contenant la valeur `"Bonjour"`. Cela permet de faciliter la maintenance du code. Par exemple, si vous souhaitez changer la valeur `"Bonjour"` par `"Salut"`, vous n'aurez qu'à modifier la valeur de la variable `bonjour` et non toutes les occurrences de la valeur `"Bonjour"` dans le code.

- Des **fonctions** plutôt que de copier-coller du code.

Exemple : plutôt que d'écrire `console.log("Bonjour")` et `console.log("Salut")`, il est préférable d'écrire une fonction `direBonjour()` qui affiche `"Bonjour"` et une fonction `direSalut()` qui affiche `"Salut"`. Cela permet de faciliter la maintenance du code. Par exemple, si vous souhaitez changer la valeur `"Bonjour"` par `"Salut"`, vous n'aurez qu'à modifier la valeur de la fonction `direBonjour()` et non toutes les occurrences de la valeur `"Bonjour"` dans le code.

- Des **constantes** plutôt que des variables.

Exemple : plutôt que d'écrire `let x = 1`, il est préférable d'écrire `const x = 1`. Cela permet de faciliter la maintenance du code. Par exemple, si vous souhaitez changer la valeur `1` par `2`, vous n'aurez qu'à modifier la valeur de la constante `x` et non toutes les occurrences de la valeur `1` dans le code. De plus, les constantes ne peuvent pas être modifiées. Cela permet d'éviter les erreurs de programmation.

- Des **commentaires** pour expliquer le code.

Exemple : plutôt que d'écrire `let x = 1`, il est préférable d'écrire `let x = 1 // x contient la valeur 1`. Cela permet de faciliter la maintenance du code. Par exemple, si vous souhaitez changer la valeur `1` par `2`, vous n'aurez qu'à modifier la valeur de la constante `x` et non toutes les occurrences de la valeur `1` dans le code. De plus, les constantes ne peuvent pas être modifiées. Cela permet d'éviter les erreurs de programmation. De plus, les commentaires permettent d'expliquer le code. Par exemple, si vous avez une fonction `direBonjour()` qui affiche `"Bonjour"`, vous pouvez ajouter un commentaire `// Affiche "Bonjour"` pour expliquer ce que fait la fonction. Cependant, il faut veiller à ne pas mettre de commentaires inutiles.

/!\ Il est déconseillé d'utiliser : /!

- Des **variables globales**.

Exemple : plutôt que d'écrire `let x = 1` en dehors d'une fonction, il est préférable d'écrire `let x = 1` dans une fonction. Cela permet de faciliter la maintenance du code. Par exemple, si vous souhaitez changer

la valeur `1` par `2`, vous n'aurez qu'à modifier la valeur de la variable `x` et non toutes les occurrences de la valeur `1` dans le code. De plus, les variables globales sont accessibles depuis n'importe où dans le code. Cela peut poser problème si vous souhaitez modifier la valeur d'une variable globale dans une fonction. Par exemple, si vous avez une variable globale `x` et que vous souhaitez la modifier dans une fonction, vous aurez un problème car la variable `x` sera modifiée dans toutes les fonctions. Si une fonction a besoin d'une valeur en entrée, il est préférable de passer cette valeur en paramètre de la fonction.

Blocs de code

Definition : un **bloc de code** est un ensemble de lignes de code qui sont exécutées les unes après les autres. Par exemple, le code suivant est un bloc de code :

```
console.log("Bonjour");
console.log("Salut");
```

Il est possible de **délimiter les blocs de code** par des accolades. Par exemple, le bloc de code précédent peut être délimité par des accolades :

```
{
  console.log("Bonjour");
  console.log("Salut");
}
```

Variables

Définition

Une variable est une zone mémoire qui contient une valeur. Par exemple, la variable `x` contient la valeur `1` :

```
let x = 1;
```

Différence entre variable et valeur

Une **variable** est comme une boîte qui contient une information. Elle a un nom, qui permet de trouver l'adresse mémoire de la boite, et peut contenir différentes valeurs au cours du programme. Dans l'exemple précédent, le nom permettant de trouver cette "boîte" est `x`.

Une **valeur** est ce qui est réellement stocké dans la variable à un moment donné. Dans l'exemple précédent il s'agit de la valeur `1`.

Types de variables

Il existe plusieurs types de variables :

- `number` : contient un nombre entier ou décimal. Les décimaux sont séparés par un point.
- `string` : contient une chaîne de caractères. Les chaînes de caractères sont délimitées par des guillemets simples ou doubles.
- `boolean` : contient une valeur booléenne (`true` ou `false`).
- `any` : contient une valeur de n'importe quel type.
- `void` : ne contient aucune valeur.
- `null` : contient la valeur `null`.
- `undefined` : contient la valeur `undefined`.
- `unknown` : contient une valeur de n'importe quel type (sauf `null` et `undefined`).
- ...

Dans cette introduction à TypeScript, nous n'utiliserons que les types `number`, `boolean`, `string`.

/!\ Il est interdit dans ce cours d'utiliser les types `any` et `unknown`. /!

Expression arithmétique

Il est possible d'utiliser des expressions arithmétiques dans TypeScript. Les opérateurs arithmétiques sont :

- `+` : addition
- `-` : soustraction
- `*` : multiplication
- `/` : division
- `%` : modulo

La priorité des opérateurs est la suivante : `*` et `/` ont la priorité sur `+` et `-`. Par exemple, l'expression `1 + 2 * 3` est évaluée comme `1 + (2 * 3)`. Il est possible d'utiliser des parenthèses pour changer la priorité des opérateurs. Par exemple, l'expression `(1 + 2) * 3` est évaluée comme `(1 + 2) * 3`.

Déclarer une variable

Il est possible de déclarer une variable en utilisant le mot-clé `let`. Par exemple, la variable `x` est déclarée avec la valeur `1` :

```
let x = 1;
```

Ici le type de la variable `x` est `number` car la valeur `1` est un nombre. Le type a été déduit automatiquement par TypeScript. Il est possible de déclarer une variable avec un type explicite :

```
let x: number = 1;
```

Cependant, si le type est déduit automatiquement, il est préférable de ne pas le déclarer explicitement. On utilisera cette syntaxe uniquement si le type n'est pas déduit automatiquement.

```

```

```
let x = 1; // Type déduit automatiquement
let y: number; // Type déclaré explicitement
```

Pratique du cours imposée par le linter

Lors de la déclaration d'une variable, il faut que le type soit déduit automatiquement ou déclaré explicitement. Il est interdit de déclarer une variable sans type.

```
let x = 1; // Type déduit automatiquement
let y: number; // Type déclaré explicitement
let z; // Erreur : type non déclaré
```

Initialisation d'une variable et affectation d'une valeur

Il faut qu'une variable reçoive une valeur avant sa première utilisation. On peut initialiser une variable en lui donnant une valeur lors de la déclaration :

```
let x = 1;
```

On peut aussi initialiser une variable en lui donnant une valeur après la déclaration :

```
let b: boolean;

b = true;
```

La valeur d'une variable peut être modifiée à tout moment :

```
let x = 1;

x = 2;
```

Utiliser une variable

Une variable peut être utilisée dans une expression :

```
let x = 1;

console.log(x + 1);
```

Utiliser des constantes

Il est possible de déclarer une constante en utilisant le mot-clé `const`. Par exemple, la constante `x` est déclarée avec la valeur `1` :

```
const x = 1;
```

Une constante ne peut pas être modifiée :

```
const x = 1;

x = 2; // Erreur : la constante ne peut pas être modifiée
```

Fonctions

Déclaration d'une fonction

Il est possible de déclarer une fonction en utilisant le mot-clé `function`. Par exemple, la fonction `add` prend deux paramètres `a` et `b` et renvoie la somme de ces deux paramètres :

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

Remarques :

- Le type de la valeur renvoyée par la fonction est déclaré après les deux points `: number`.
- Il faut utiliser le mot-clé `return` pour renvoyer une valeur.
- Il faut indiquer le type de chaque paramètre de la fonction.
- Il faut indiquer le type de la valeur renvoyée par la fonction.
- Le type de la valeur renvoyée par la fonction peut être `void` si la fonction ne renvoie aucune valeur. Dans ce cas, il faut utiliser le mot-clé `return` sans valeur.

Utiliser une fonction

Une fonction peut être appelée en utilisant son nom et en passant les paramètres entre parenthèses :

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

console.log(add(1, 2));
```

Visibilité des variables

Les variables déclarées dans une fonction ne sont pas visibles en dehors de cette fonction :

```
function add(a: number, b: number): number {
  let x = a + b;
  return x;
}

console.log(x); // Erreur : la variable x n'est pas visible en dehors de la fonction
```

Conditions

Opérateurs de comparaison

Il est possible d'utiliser les opérateurs de comparaison suivants :

- `===` : égalité
- `!==` : différence
- `<` : inférieur
- `<=` : inférieur ou égal
- `>` : supérieur
- `>=` : supérieur ou égal

Opérateurs logiques

Il est possible d'utiliser les opérateurs logiques suivants :

- `&&` : ET
- `||` : OU
- `!` : NON

Expressions booléennes

Il est possible d'utiliser les valeurs booléennes suivantes :

- `true` : vrai
- `false` : faux

et de construire des expressions booléennes avec les opérateurs de comparaison et les opérateurs logiques. Par exemple :

```
let x = 1;
let y = 2;
```

```
console.log(x === 1 && y === 2); // true
console.log(x === 1 && y === 3); // false
console.log(x === 1 || y === 3); // true
console.log(!(x === 1)); // false
```

Condition `if`

Il est possible d'utiliser une condition `if` pour exécuter du code si une condition est vraie :

```
let x = 1;

if (x === 1) {
  console.log("x est égal à 1");
}
```

Condition `if else`

Il est possible d'utiliser une condition `if else` pour exécuter du code si une condition est vraie, et du code différent si la condition est fausse :

```
let x = 1;

if (x === 1) {
  console.log("x est égal à 1");
} else {
  console.log("x est différent de 1");
}
```

Condition `if else if else`

Il est possible d'utiliser une condition `if else if else` pour exécuter du code si une condition est vraie, du code différent si une autre condition est vraie, et du code différent si aucune des conditions n'est vraie :

```
let x = 1;

if (x === 1) {
  console.log("x est égal à 1");
} else if (x === 2) {
  console.log("x est égal à 2");
} else {
  console.log("x est différent de 1 et de 2");
}
```

Il est possible d'utiliser plusieurs conditions `if else if else` les unes à la suite des autres. Par exemple :

```
let x = 1;

if (x === 1) {
  console.log("x est égal à 1");
} else if (x === 2) {
  console.log("x est égal à 2");
} else if (x === 3) {
  console.log("x est égal à 3");
} else {
  console.log("x est différent de 1, 2 et 3");
}
```

Il est possible d'utiliser une condition `if else if else` sans condition `else` :

```
let x = 1;

if (x === 1) {
  console.log("x est égal à 1");
} else if (x === 2) {
  console.log("x est égal à 2");
} else if (x === 3) {
  console.log("x est égal à 3");
}
```

Opérateur ternaire

Il est possible d'utiliser l'opérateur ternaire pour exécuter du code si une condition est vraie, et du code différent si la condition est fausse :

```
let x = 1;

x === 1 ? console.log("x est égal à 1") : console.log("x est différent de 1");
```

Boucles

Boucle `while`

Il est possible d'utiliser une boucle `while` pour répéter un bloc de code tant qu'une condition est vraie :

```
let x = 0;

while (x < 3) {
```

```
  console.log(x);
  x++;
}
```

Boucle `do while`

Il est possible d'utiliser une boucle `do while` pour répéter un bloc de code tant qu'une condition est vraie :

```
let x = 0;

do {
  console.log(x);
  x++;
} while (x < 3);
```

Attention : une variable utilisée dans une boucle `do while` doit être déclarée avant la boucle car elle n'est pas visible en dehors de la boucle. Par exemple :

```
let x = 0;

do {
  let y = 0;
  console.log(x);
  x++;
} while (x < 3);

console.log(y); // Erreur : la variable y n'est pas visible en dehors de la boucle
```

Cela vaut aussi pour la condition d'arrêt de la boucle :

```
let x = 0;

do {
  let y = x / 2;
  console.log(x);
  x++;
} while (x < 10 && y < 5); // Erreur : la variable y n'est pas visible en dehors du
corps de la boucle
```

Boucle `for`

Il est possible d'utiliser une boucle `for` pour répéter un bloc de code un nombre défini de fois :

```
for (let x = 0; x < 3; x++) {
  console.log(x);
}
```

- `let x = 0` : initialisation de la variable `x` à 0
- `x < 3` : condition d'arrêt de la boucle
- `x++` : incrémentation de la variable `x` de 1

Les variables déclarées dans une boucle `for` ne sont pas visibles en dehors de la boucle. Par exemple :

```
for (let x = 0; x < 3; x++) {
  console.log(x);
}

console.log(x); // Erreur : la variable x n'est pas visible en dehors de la boucle
```

Lien entre boucle `for` et boucle `while`

Une boucle `for` est équivalente à une boucle `while` :

```
for (let x = 0; x < 3; x++) {
  console.log(x);
}
```

```
let x = 0;

while (x < 3) {
  console.log(x);
  x++;
}
```

L'inverse n'est pas nécessairement vrai.

Chaînes de caractères

Type

Le type d'une chaîne de caractères est `string` :

```
let x = "Hello";
console.log(typeof x); // string
```

Literal

Il est possible de déclarer une chaîne de caractères avec des guillemets simples ou doubles :

```
let x = "Hello";
let y = "World";
let z: string;

z = "Bonjour Monde";
```

La chaîne de caractères peut contenir des guillemets simples ou doubles :

```
let x = "Hello 'World'";
let y = 'Hello "World"';
```

La chaîne vide est déclarée avec deux guillemets simples ou doubles :

```
let x = "";
let y = "";
```

Concaténation

Il est possible de concaténer des chaînes de caractères avec l'opérateur `+` :

```
let x = "Hello";
let y = "World";

console.log(x + " " + y); // Hello World
```

Longueur

Il est possible de connaître la longueur d'une chaîne de caractères avec la propriété `length` :

```
let x = "Hello World";

console.log(x.length); // 11
```

Accès à un caractère

Il est possible d'accéder à un caractère d'une chaîne de caractères avec l'opérateur `[]` :

```
let x = "Hello World";

console.log(x[0]); // H
console.log(x[1]); // e
console.log(x[2]); // l
console.log(x[3]); // l
console.log(x[4]); // o
console.log(x[5]); // " "
console.log(x[6]); // W
console.log(x[7]); // o
console.log(x[8]); // r
console.log(x[9]); // l
console.log(x[10]); // d
```

Accès à un caractère avec une variable

Il est possible d'accéder à un caractère d'une chaîne de caractères avec une variable :

```
let x = "Hello World";

for (let i = 0; i < x.length; i++) {
  console.log(x[i]);
}
```

Boucle `for of`

Il est possible d'utiliser une boucle `for of` pour parcourir une chaîne de caractères :

```
let x = "Hello World";

for (let c of x) {
  console.log(c);
}
```

On utilisera plutôt une boucle `for of` pour parcourir une chaîne de caractères si l'indice n'est pas nécessaire et que l'on doit parcoure la chaîne de caractères entièrement caractère par caractère.

Non mutabilité

Une chaîne de caractères est non mutable :

```
let x = "Hello World";

x[0] = "h"; // Erreur : une chaîne de caractères est non mutable
```

Si on veut modifier une chaîne de caractères, il faut construire une nouvelle chaines de caractères. Par exemple, la fonction ci-dessous permet de remplacer toutes les occurrences d'un caractère par un autre caractère :

```
function replaceAll(str: string, search: string, replacement: string): string {
  let result = "";

  for (let c of str) {
    if (c === search) {
      result += replacement;
    } else {
      result += c;
    }
  }

  return result;
}
```

Entrées / Sorties

Affichage à l'écran

Affichage par défaut

L'affichage de données se fait via la fonction `console.log` :

```
console.log();
```

Comportement : la fonction affiche un retour à la ligne.

On peut passer à la fonction un argument de type `number` ou `string` qui sera affiché suivi d'un retour à la ligne :

```
console.log(42);
console.log("Hello, World!");
```

On peut passer une expression qui sera évaluée avant l'affichage :

```

```

```
let nombre = 42;
let chaîne = "Hello, World!";

console.log(nombre);
console.log(chaîne);
```

On peut passer plusieurs arguments qui seront affichés séparés par un espace :

```
let nombre = 42;
console.log("2 *", nombre, "=", 2 * nombre);
```

Affichage sans retour à la ligne

Vous pouvez utiliser la fonction `écrire` pour afficher sans retour à la ligne :

```
function écrire(...args: string[]): void {
  Deno.stdout.writeSync(new TextEncoder().encode(args.join(" ")));
}

for (let i = 0; i < 5; i++) {
  écrire("Itération", String(i), ":");
  for (let j = 0; j < 10; j++) écrire(" " + j);
  écrire("\n");
}
```

Lecture au clavier

La lecture de données au clavier se fait via la fonction `prompt`. La fonction prend un argument qui est le message affiché à l'écran et retourne une chaîne de caractères correspondant aux caractères saisis jusqu'à un retour à la ligne (non inclus). Si vous souhaitez lire un nombre, il faudra convertir la chaîne de caractères retournée en `number` à l'aide de la fonction `Number`.

Voici un exemple d'utilisation :

```
let nom = prompt("Quel est ton nom ? ");
console.log("Ton nom est", nom);

let âge = Number(prompt("Quel est ton âge ? "));
console.log("Ton âge est", âge);
```

Si l'utilisateur n'entre rien, la fonction ne retourne pas une chaîne vide mais `null`. Il est donc recommender de tester la valeur retournée avant de l'utiliser ou d'utiliser la fonction suivante :

```ts
function lire(message: string): string {
  let valeur = prompt(message);
  return valeur ?? "";
}
```