**Module 1 : Introduction à TypeScript**

* **Objectif :** Comprendre les bases de TypeScript et son utilité par rapport à JavaScript.

**1.1. Introduction à JavaScript**

* Comprendre les limitations de JavaScript dans les projets à grande échelle.
* Différences entre TypeScript et JavaScript.
* Installation de Node.js et de TypeScript.
* Configuration d’un projet TypeScript avec tsconfig.json.
* Conversion de fichiers .ts en .js.

**1.2. Types de base**

* Types primitifs : number, string, boolean, null, undefined.
* Le mot-clé any et pourquoi l’éviter.
* Inférence de type.
* Les tableaux et les tuples.

**1.3. Compilation et outillage**

* Installer TypeScript avec npm (npm install -g typescript).
* Configurer un environnement TypeScript avec Visual Studio Code.
* Utilisation du compilateur TypeScript (tsc) et options importantes.

**Module 2 : Types avancés et gestion des objets**

* **Objectif :** Apprendre à utiliser les types complexes et gérer des objets en TypeScript.

**2.1. Types complexes**

* Les types personnalisés : type, interface.
* Différences entre interface et type.
* Types littéraux et énumérations (enum).
* Type d’union et d’intersection.
* Les types génériques.

**2.2. Interfaces et objets**

* Définir des interfaces pour structurer les objets.
* Implémentation de plusieurs interfaces.
* Étendre une interface.
* Les classes et héritage en TypeScript.
* Modificateurs d’accès : public, private, protected, readonly.

**Module 3 : Programmation Orientée Objet (POO) avec TypeScript**

* **Objectif :** Appliquer les concepts POO à TypeScript.

**3.1. Classes et objets**

* Définition de classes en TypeScript.
* Constructeurs, propriétés et méthodes.
* Initialisation des membres de classe.
* Classes abstraites et méthodes abstraites.

**3.2. Héritage et polymorphisme**

* Étendre une classe.
* Surcharge et redéfinition des méthodes.
* Utilisation de polymorphisme dans TypeScript.

**3.3. Classes génériques**

* Classes et méthodes génériques.
* Contraintes des types génériques.
* Avantages du typage générique pour les bibliothèques de fonctions.

**Module 4 : Programmation asynchrone en TypeScript**

* **Objectif :** Apprendre à gérer les promesses et l’asynchronisme avec TypeScript.

**4.1. Introduction aux promesses**

* Syntaxe des promesses : Promise, then, catch, finally.
* Conversion d'une fonction JavaScript en TypeScript avec les promesses.
* Le typage des promesses.

**4.2. Utilisation de async / await**

* Gérer les fonctions asynchrones avec async et await.
* Gestion des erreurs dans les fonctions asynchrones (try/catch).

**4.3. Typage des fonctions asynchrones**

* Type des fonctions retournant des promesses.
* Typage avancé pour les fonctions asynchrones.

**Module 5 : Modules et gestion de code**

* **Objectif :** Apprendre à structurer un projet TypeScript en modules.

**5.1. Modules en TypeScript**

* Importation et exportation de modules.
* Importation dynamique.
* Organisation du projet en modules.

**5.2. Gestion des dépendances**

* Gestion des dépendances avec npm et yarn.
* Intégration avec des librairies JavaScript.

**Module 6 : Tests et vérification de type**

* **Objectif :** Comprendre comment tester et garantir la sécurité du code TypeScript.

**6.1. Introduction à la vérification de type**

* Utilisation de tsc pour vérifier la conformité des types.
* Débogage des erreurs de type les plus fréquentes.

**6.2. Tests unitaires**

* Introduction aux tests unitaires avec Jest et TypeScript.
* Utilisation de ts-jest pour écrire des tests TypeScript.
* Écrire des tests pour vérifier le comportement du code.

**Module 7 : Utilisation avancée de TypeScript**

* **Objectif :** Approfondir les fonctionnalités avancées de TypeScript pour des applications robustes.

**7.1. Types conditionnels et utilitaires**

* Les types conditionnels (T extends U ? X : Y).
* Types utilitaires fournis par TypeScript (Partial, Readonly, Pick, Omit).

**7.2. Décorateurs**

* Introduction aux décorateurs.
* Utilisation des décorateurs pour modifier le comportement des classes et des méthodes.
* Décorateurs dans les frameworks comme Angular.

**Module 8 : Intégration avec les frameworks**

* **Objectif :** Utiliser TypeScript avec des frameworks populaires comme Angular et React.

**8.1. TypeScript avec Angular**

* Configuration d’un projet Angular avec TypeScript.
* Types et interfaces dans les services et composants Angular.
* Communication entre les composants avec des types sûrs.

**8.2. TypeScript avec React**

* Configuration de TypeScript dans un projet React.
* Utilisation des interfaces pour typer les props et l’état (state) des composants React.
* Utilisation des types génériques dans les composants fonctionnels.

**Module 9 : TypeScript et Node.js**

* **Objectif :** Utiliser TypeScript avec des applications backend basées sur Node.js.

**9.1. Introduction à Node.js avec TypeScript**

* Initialisation d'un projet Node.js avec TypeScript.
* Utilisation de types pour Express ou d'autres frameworks back-end.
* Création d'une API REST typée avec TypeScript.

**9.2. Typage des bibliothèques**

* Gérer les types dans les bibliothèques JavaScript existantes.
* Utilisation des définitions de types avec @types.

**Module 10 : Outils avancés et optimisation**

* **Objectif :** Maîtriser les outils avancés et les bonnes pratiques pour un code TypeScript de production.

**10.1. Linting et formatage du code**

* Utilisation de ESLint pour vérifier les erreurs de code.
* Utilisation de Prettier pour formater le code TypeScript.

**10.2. Configuration avancée de TypeScript**

* Optimisation du fichier tsconfig.json pour les environnements de production.
* Utilisation de Webpack ou d'autres bundlers avec TypeScript.

**Conclusion : Pratiques de développement avec TypeScript**

* Bonnes pratiques pour écrire du code TypeScript maintenable et évolutif.
* Stratégies pour adopter TypeScript dans des projets existants.
* Mise à jour continue des compétences avec les nouvelles versions de TypeScript.

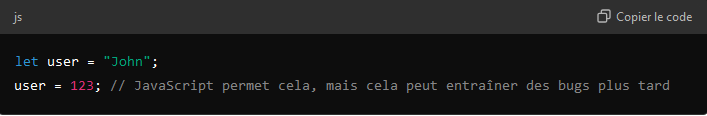
**Objectifs généraux :**

* Maitriser les bases de TypeScript et ses avantages par rapport à JavaScript.
* Appliquer les concepts de la programmation orientée objet avec TypeScript.
* Écrire du code TypeScript sûr, robuste et maintenable.
* Utiliser TypeScript dans des projets front-end et back-end.

### ****1.1. Introduction à JavaScript****

#### Limitations de JavaScript dans les projets à grande échelle

JavaScript est dynamique, ce qui signifie qu’il n’y a pas de vérification de type lors de l'écriture du code. Cela peut rendre difficile la gestion des gros projets, car les erreurs de type ne sont détectées qu’à l'exécution. Exemple :

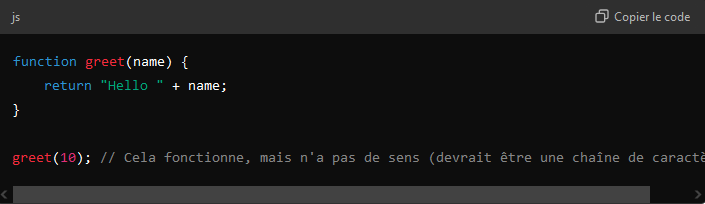


Avec TypeScript, les types sont vérifiés à la compilation, ce qui aide à éviter des erreurs avant même que le code ne s’exécute.

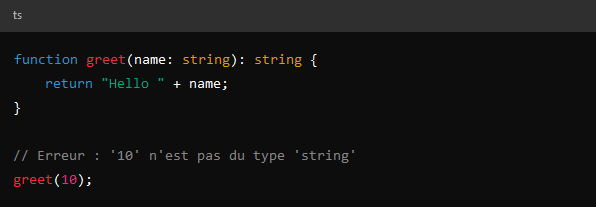
#### ****Différences entre TypeScript et JavaScript****

**JavaScript** est un langage dynamique, sans types, tandis que **TypeScript** ajoute des types statiques. En TypeScript, vous pouvez définir explicitement le type d’une variable, et cela améliore la sécurité du code.

Exemple en JavaScript :



Exemple en TypeScript :



**Installation de Node.js et TypeScript**

1. **Installer Node.js** : Vous devez d’abord installer Node.js (il inclut npm, le gestionnaire de paquets). Vous pouvez le télécharger ici : <https://nodejs.org>.
2. **Installer TypeScript globalement** : Une fois Node.js installé, vous pouvez installer TypeScript globalement avec npm : npm install -g typescript

**Installer TypeScript localement** :

**Créer un projet avec package.json** :

* Dans le répertoire de votre projet, exécutez la commande suivante pour initialiser un projet Node.js : npm init -y
* Installez TypeScript en tant que dépendance de développement dans votre projet :

npm install typescript --save-dev

Cela installe TypeScript localement dans le dossier node\_modules, et vous pourrez l'utiliser via les scripts définis dans votre package.json ou via npx.

1. **Vérifier l'installation locale** :

* Pour vérifier que TypeScript est bien installé localement et compiler des fichiers .ts :

npx tsc –init

Cela créera le fichier tsconfig.json dans votre projet.

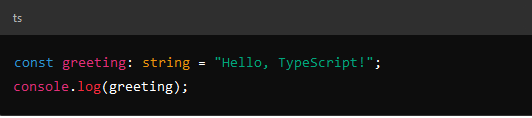
### Exécuter TypeScript dans l'environnement virtuel du projet

Une fois TypeScript installé localement dans le projet, vous pouvez exécuter le compilateur TypeScript via npx, qui permet de l'utiliser sans installation globale : npx tsc

Cela va compiler vos fichiers .ts en .js tout en utilisant la version locale de TypeScript dans votre projet.

**Étapes pour créer et compiler un fichier .ts :**

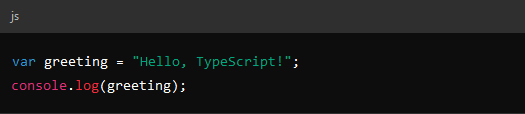
1. **Créer un fichier TypeScript :**
   * Créez un fichier nommé app.ts dans votre projet avec du code TypeScript, par exemple :



1. **Compiler le fichier TypeScript en JavaScript :**

* Utilisez la commande suivante pour compiler votre fichier app.ts en JavaScript : npx tsc app.ts

Cela génère un fichier app.js dans le même répertoire avec le code JavaScript compilé :



1. **Exécuter le fichier JavaScript avec Node.js :**

* Exécutez le fichier JavaScript généré avec Node.js : node app.js

