



INTRODUCTION À TYPESCRIPT



m2information.fr



QU'EST-CE QUE TYPESCRIPT



m2information.fr



DÉFINITION

TypeScript est un langage de programmation développé par **Microsoft** et est un sur-ensemble de **JavaScript**. TypeScript ajoute des fonctionnalités telles que le **typage statique** et des objets orientés sur les **classes**.





AVANTAGES DE TYPESCRIPT

- **Typage statique** pour une meilleure détection d'erreurs
- Support des fonctionnalités **modernes** de JavaScript
- Facilite la création de **projets complexe**
- Intégration **aisée** avec des systèmes de build et des IDE





INSTALLATION ET CONFIGURATION



m2information.fr



TYPES DE BASE EN TYPESCRIPT





TYPES PRIMITIFS

TypeScript supporte les types primitifs suivants :

- **string** : textes
- **number** : nombres (entiers et flottants)
- **boolean** : vrai ou faux
- **null** et **undefined**

Ces types servent à définir les variables et les paramètres des fonctions.





STRING

Pour déclarer une **chaîne de caractères** en TypeScript, utilisez des apostrophes ou des guillemets :

```
let nom: string = 'John Doe';  
let prenom: string = "Jane";
```





OBJETS LITTÉRAUX



m2information.fr



OBJETS LITTÉRAUX

Les **objets littéraux** sont des collections de **paires clé-valeur**.

Ils sont similaires aux **dictionnaires** en Python ou aux **objets JSON**.

```
let objetLiteral = {  
  cle1: "valeur1",  
  cle2: "valeur2",  
  cle3: 42,  
};  
  
console.log(objetLiteral.cle1); // "valeur1"
```





SYNTAXE

Voici la syntaxe pour créer un objet **littéral** en TypeScript :

```
let objet = {  
  cle1: valeur1,  
  cle2: valeur2,  
  // ...  
};
```



m2information.fr



EXEMPLES D'UTILISATION



m2information.fr



TABLEAUX



m2information.fr



TABLEAUX

Les **tableaux** en TypeScript sont des objets utilisés pour stocker plusieurs valeurs dans une seule variable.

Exemple de création de tableau en TypeScript:

```
let fruits: string[] = ["pomme", "banane", "orange"];  
let nombres: Array<number> = [1, 2, 3];
```





TUPLES





TUPLES

Les **tuples** permettent de définir un **tableau de taille fixe** dont les éléments ont des **types prédéfinis**.

```
// Déclaration d'un tuple  
let exempleTuple: [string, number];  
exempleTuple = ['Un exemple de chaîne de caractères', 42];
```





SYNTAXE

Pour déclarer un **tuple** en TypeScript, utilisez la notation : `[type1, type2, type3]`.

```
let tuple: [string, number, boolean];
```





EXEMPLES D'UTILISATION

```
tuple = ["TypeScript", 42, true];  
  
// Accéder aux éléments du tuple  
let nom: string = tuple[0]; // "TypeScript"  
let valeur: number = tuple[1]; // 42  
let estValide: boolean = tuple[2]; // true
```





ENUMÉRATIONS (enum)





ENUMÉRATIONS (enum)

Les énumérations permettent de définir des ensembles de valeurs nommées, facilitant la compréhension et l'utilisation de certaines données.

```
enum Couleur {Rouge, Vert, Bleu}  
let c: Couleur = Couleur.Rouge;
```





SYNTAXE

Pour déclarer une **énumération** en TypeScript, utilisez le mot-clé `enum` suivi du nom de l'énumération et d'une liste de valeurs entre accolades.

```
enum Couleur {  
  Rouge,  
  Vert,  
  Bleu  
}
```





EXEMPLES D'UTILISATION

```
let maCouleur: Couleur = Couleur.Rouge;

switch (maCouleur) {
  case Couleur.Rouge:
    console.log("La couleur est rouge");
    break;
  case Couleur.Vert:
    console.log("La couleur est vert");
    break;
  case Couleur.Bleu:
    console.log("La couleur est bleu");
    break;
}
```





ENUMÉRATIONS AVEC VALEURS CUSTOMISÉES

Les énumérations peuvent être assignées à des **valeurs numériques** ou **chaînes de caractères** spécifiques.

```
enum Status {  
    Actif = "ACTIF",  
    Inactif = "INACTIF",  
    Suspendu = "SUSPENDU"  
}
```

Exemple d'utilisation :

```
let monStatus: Status = Status.Actif;  
  
if (monStatus === Status.Actif) {  
    console.log("Le statut est actif");  
}
```





UNION DE TYPES



m2information.fr



UNION DE TYPES

TypeScript permet la combinaison de plusieurs types dans une seule variable. Cela permet d'autoriser plusieurs types possibles pour une seule variable.

```
let variable: string | number;  
  
variable = "Bonjour"; // autorisé  
variable = 42; // autorisé  
variable = true; // erreur de type
```





SYNTAXE

Utilisez le symbole `|` pour séparer les différents **types possibles**.

```
let variable: type1 | type2 | type3;
```



m2information.fr



EXEMPLES D'UTILISATION

```
// Combinaison de types string et number
let nomOuAge: string | number;

nomOuAge = "John"; // valide
nomOuAge = 25; // valide
```

```
// Combinaison de types boolean et null
let estActifOuNul: boolean | null;

estActifOuNul = true; // valide
estActifOuNul = null; // valide
```

```
// Union de types pour un tableau
let tableauMixte: (number | string)[];

tableauMixte = [1, "deux", 3]; // valide
```





DÉCLARATION DE FONCTIONS



m2information.fr



DÉCLARATION DE FONCTIONS

Les **fonctions** sont des blocs de code **réutilisables** qui effectuent une action spécifique.

```
function nomDeLaFonction(parametre1: type, parametre2: type): returnType {  
  // Corps de la fonction  
}
```



m2information.fr



SYNTAXE

Pour déclarer une fonction en **TypeScript**, utilisez la syntaxe suivante:

```
function nomFonction(param1: type, param2: type): returnType {  
  // code de la fonction  
}
```



m2information.fr



EXEMPLES D'UTILISATION

```
function somme(a: number, b: number): number {  
  return a + b;  
}  
  
let resultat = somme(5, 3);
```





PARAMÈTRES OPTIONNELS



m2information.fr



PARAMÈTRES OPTIONNELS

Dans **TypeScript**, il est possible de rendre un paramètre de fonction optionnel en ajoutant un `?` après le nom du paramètre.

```
function saluer(nom: string, age?: number) {  
  if (age) {  
    console.log(`Bonjour ${nom}, vous avez ${age} ans.`);  
  } else {  
    console.log(`Bonjour ${nom}`);  
  }  
}  
  
saluer("Alice"); // "Bonjour Alice"  
saluer("Bob", 30); // "Bonjour Bob, vous avez 30 ans."
```





SYNTAXE

```
function example_function(param1: number, param2?: string) {  
  // Code here  
}
```

Dans cet exemple, `param1` est un **paramètre obligatoire** de type `number` et `param2` est un **paramètre optionnel** de type `string`.



EXEMPLES D'UTILISATION

```
// Appel de la fonction avec tous les paramètres
example_function(42, "Hello");

// Appel de la fonction sans paramètre optionnel
example_function(42);

// La fonction peut vérifier si le paramètre optionnel est défini
function example_function(param1: number, param2?: string) {
  if (param2) {
    console.log(`param2: ${param2}`);
  } else {
    console.log("param2 is not defined");
  }
}
```





PARAMÈTRES PAR DÉFAUT



m2information.fr

PARAMÈTRES PAR DÉFAUT

Les **paramètres par défaut** permettent de définir une valeur par défaut pour un argument de fonction si celui-ci n'est pas fourni lors de l'appel de la fonction.

```
function saluer(nom: string, salutation: string = "Bonjour") {  
  console.log(`${salutation}, ${nom} !`);  
}  
  
saluer("Sophie"); // Affichera "Bonjour, Sophie !"  
saluer("Lucas", "Salut"); // Affichera "Salut, Lucas !"
```



SYNTAXE

Pour définir un **paramètre par défaut** en TypeScript, utilisez la syntaxe suivante :

```
function fonction(nom: string, age: number = 0): string {  
  // Code de la fonction  
}
```

Dans cet exemple, le paramètre `age` a une **valeur par défaut** de 0.





m2information.fr