

# Liste d'exercices pour apprendre TypeScript

Voici une série d'exercices pour vous aider à maîtriser TypeScript, allant de la découverte des bases aux concepts plus avancés.

---


## 1. Types de base

### Exercice 1.1 : Déclaration de variables avec types

Énoncé : Déclarez une variable `nom` de type `string` et assignez-lui votre nom. Ensuite, déclarez une variable `age` de type `number` et assignez-lui votre âge.

Correction :

typescript

 Copier le code


```
let nom: string = 'VotreNom';  
let age: number = 25;
```

### Exercice 1.2 : Utilisation de types unifiés

Énoncé : Créez une variable `personne` qui peut être soit un `string` pour le nom, soit un `number` pour l'âge. Utilisez l'union type `string | number`.

Correction :

typescript

 Copier le code

```
let personne: string | number;  
personne = 'Alice';  
personne = 30;
```

---


## 2. Interfaces

### Exercice 2.1 : Définir une interface simple

Énoncé : Définissez une interface `Employe` avec les propriétés `nom` (string), `poste` (string) et `salaire` (number). Créez un objet `employee` conforme à cette interface.

Correction :

typescript

 Copier le code

```
interface Employe {  
    nom: string;  
    poste: string;  
    salaire: number;  
}  
  
let employee: Employe = {  
    nom: 'Jean Dupont',  
    poste: 'Développeur',  
    salaire: 3000  
};
```

## Exercice 2.2 : Interface avec méthodes

Énoncé : Ajoutez une méthode `afficherDetails()` à l'interface `Employe`, qui retourne une chaîne de caractères avec les détails de l'employé. Modifiez l'objet `employee` pour implémenter cette méthode.

Correction :

typescript

 Copier le code

```
interface Employe {  
    nom: string;  
    poste: string;  
    salaire: number;  
    afficherDetails(): string;  
}  
  
let employee: Employe = {  
    nom: 'Jean Dupont',  
    poste: 'Développeur',  
    salaire: 3000,  
    afficherDetails() {  
        return `${this.nom} travaille comme ${this.poste} et gagne ${this.salaire} euros.`;  
    }  
};
```

```
    }  
};  
  
console.log(employe.afficherDetails());
```


## 3. Fonctions

### Exercice 3.1 : Fonction avec types de paramètres et de retour

Énoncé : Écrivez une fonction `additionner` qui prend deux paramètres `a` et `b` de type `number` et retourne leur somme. Utilisez le type `number` pour les paramètres et le retour.

Correction :

typescript

 Copier le code


```
function additionner(a: number, b: number): number {  
    return a + b;  
}  
  
console.log(additionner(5, 10)); // Affiche 15
```

### Exercice 3.2 : Fonction avec types optionnels

Énoncé : Modifiez la fonction `additionner` pour que le deuxième paramètre `b` soit optionnel. Utilisez le type `number | undefined` pour le paramètre `b`.

Correction :

typescript

 Copier le code

```
function additionner(a: number, b?: number): number {  
    return b !== undefined ? a + b : a;  
}  
  
console.log(additionner(5)); // Affiche 5  
console.log(additionner(5, 10)); // Affiche 15
```


## 4. Classes

### Exercice 4.1 : Déclaration de classe

Énoncé : Déclarez une classe `Voiture` avec les propriétés `marque` (string) et `modele` (string).  
Ajoutez un constructeur pour initialiser ces propriétés.

Correction :

typescript

 Copier le code


```
class Voiture {  
    marque: string;  
    modele: string;  
  
    constructor(marque: string, modele: string) {  
        this.marque = marque;  
        this.modele = modele;  
    }  
}  
  
let voiture = new Voiture('Toyota', 'Corolla');  
console.log(voiture);
```

### Exercice 4.2 : Méthodes dans une classe

Énoncé : Ajoutez une méthode `afficherDetails` à la classe `Voiture` qui retourne une chaîne de caractères avec la marque et le modèle de la voiture.

Correction :

typescript

 Copier le code

```
class Voiture {  
    marque: string;  
    modele: string;  
  
    constructor(marque: string, modele: string) {  
        this.marque = marque;  
        this.modele = modele;  
    }  
}
```

```
    afficherDetails(): string {
        return `Voiture: ${this.marque} ${this.modele}`;
    }
}

let voiture = new Voiture('Toyota', 'Corolla');
console.log(voiture.afficherDetails());
```


## 5. Généricité

### Exercice 5.1 : Utilisation de génériques

Énoncé : Créez une fonction `identite` qui prend un argument de type générique `T` et retourne cet argument. Utilisez le type générique `T` pour le paramètre et le retour.

Correction :

typescript

 Copier le code

```
function identite<T>(arg: T): T {
    return arg;
}


console.log(identite<number>(123)); // Affiche 123
console.log(identite<string>('Hello')); // Affiche 'Hello'
```

### Exercice 5.2 : Génériques avec interfaces

Énoncé : Définissez une interface `Boîte<T>` avec une propriété `contenu` de type `T`. Créez une instance de `Boîte` avec un nombre et une autre avec une chaîne de caractères.

Correction :

typescript

 Copier le code

```
interface Boîte<T> {
    contenu: T;
}
```

```
let boiteNombre: Boite<number> = { contenu: 123 };
let boiteTexte: Boite<string> = { contenu: 'Hello' };

console.log(boiteNombre);
console.log(boiteTexte);
```


## 6. Modules

### Exercice 6.1 : Export et import

Énoncé : Créez un fichier `math.ts` contenant une fonction `additionner` qui retourne la somme de deux nombres. Importez cette fonction dans un autre fichier et utilisez-la.

`math.ts`

typescript

 Copier le code

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

`app.ts`

typescript

 Copier le code

```
import { additionner } from './math';

console.log(additionner(5, 10)); // Affiche 15
```

### Exercice 6.2 : Export par défaut

Énoncé : Dans le fichier `utils.ts`, exportez une fonction `multiplié` par défaut qui prend deux nombres et retourne leur produit. Importez cette fonction dans un autre fichier.

`utils.ts`


typescript

 Copier le code

```
export default function multiplié(a: number, b: number): number {  
    return a * b;  
}
```

app.ts

typescript

 Copier le code

```
import multiplié from './utils';  
  
console.log(multiplié(5, 10)); // Affiche 50
```


## 7. Décorateurs

### Exercice 7.1 : Décorateur de classe

Énoncé : Créez un décorateur de classe `logClass` qui affiche un message dans la console chaque fois qu'une instance de la classe est créée.

Correction :

typescript

 Copier le code


```
function logClass(constructor: Function) {  
    console.log(`Classe ${constructor.name}instanciée`);  
}  
  
@logClass  
class Exemple {  
    constructor() {  
        console.log('Instance créée');  
    }  
}  
  
const instance = new Exemple();
```

### Exercice 7.2 : Décorateur de méthode

Énoncé : Ajoutez un décorateur `logMethod` à la méthode `somme` de la classe `Calcul` qui affiche le nom de la méthode et ses arguments dans la console lorsqu'elle est appelée.

Correction :

typescript

 Copier le code

```
function logMethod(target: any, key: string, descriptor: PropertyDescriptor) {
  const originalMethod = descriptor.value;
  descriptor.value = function(...args: any[]) {
    console.log(`Méthode ${key} appelée avec arguments: ${args}`);
    return originalMethod.apply(this, args);
  };
  return descriptor;
}

class Calcul {
  @logMethod
  somme(a: number, b: number): number {
    return a + b;
  }
}

const calc = new Calcul();
console.log(calc.somme(5, 10));
```


## 8. Types avancés

### Exercice 8.1 : Types conditionnels

Énoncé : Créez un type `TypeA` qui est `string` et un type `TypeB` qui est `number`. Utilisez un type conditionnel pour définir un type `Valeur` qui est `TypeA` si `T` est `string`, sinon `TypeB`.

Correction :

typescript

 Copier le code

```
type TypeA = string;
type TypeB = number;
```



```
type Valeur<T> = T extends string ? TypeA : TypeB;

let valeurString: Valeur<string> = 'Bonjour';
let valeurNumber: Valeur<number> = 42;

console.log(valeurString); // Affiche 'Bonjour'
console.log(valeurNumber);
```

## Exercice 8.2 : Types mappés

Énoncé : Créez un type mappé `Readonly<T>` qui rend toutes les propriétés de `T` en lecture seule. Testez-le avec une interface `Point` ayant des propriétés `x` et `y`.

Correction :

```
typescript Copier le code

type Readonly<T> = {
  readonly [P in keyof T]: T[P];
};

interface Point {
  x: number;
  y: number;
}

let point: Readonly<Point> = { x: 10, y: 20 };

// point.x = 15; // Erreur : Impossible d'assigner à 'x' car c'est une propriété en lecture seule
console.log(point);
```

Ces exercices couvrent les concepts clés de TypeScript, vous permettant de développer des compétences de base à avancées dans ce langage. Assurez-vous de bien comprendre chaque exercice avant de passer au suivant pour construire progressivement vos compétences.