COESIONE ITALIA 21-27 PIEMONTE

Cofinanziato dall'Unione europea

REGIONE PIEMONTE

FinTech Software Developer

Programmazione WEB - HTML | CSS | Javascript

Docente: Shadi Lahham

Intervento realizzato da ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

Typescript

Introduction

Shadi Lahham - Web development

Typescript

Typescript

Typescript is a superset of Javascript

- valid JavaScript code is also valid TypeScript code
- builds on top of JavaScript adding new features
- maintains full compatibility with JavaScript
- intuitive and easy to learn, minimal learning curve
- incremental upgrade is a major benefit
    - easy to gradually migrate code from JavaScript to TypeScript
    - developers can start by gradually adding types to JavaScript code
    - additional Typescript features can be added when and as needed

Quickstart

Quick Typescript

Quickly try in the
TS Playground
Quick introduction
TypeScript for JavaScript Programmers and The Basics
Quick setup
TypeScript Tooling in 5 minutes

Simple types

primitives

Explicit Types

main.ts

```ts
let str: string = 'hello';
let num: number = 42;
let bool: boolean = true;
```

main.js

```js
"use strict";
let str = 'hello';
let num = 42;
let bool = true;
```

Protection from type errors

```
let clientName: string = 'james';
clientName = 88;
// TypeScript compiler throws a type assignment error

// function param example
function printMessage(message: string) {
  console.log(message);
}

// Correct usage
printMessage('Good morning!'); // prints "Good morning!"

// Incorrect usage
printMessage(42);

// TypeScript will throw a compilation error
// Error: Argument of type 'number' is not assignable to parameter of type 'string'
```

Undefined & null

```
let x: number | undefined = undefined; // variable can be a number or undefined
let y: string | null = null; // variable can be a string or null

function printName(name?: string) {
  console.log(name || 'Anonymous');
}

printName(); // prints "Anonymous"
printName('Alice'); // prints "Alice"
printName(undefined); // prints "Anonymous"

// Incorrect usage
printName(null);

// TypeScript will throw a compilation error
// Error: Argument of type 'null' is not assignable to parameter of type 'string | undefined'
```

Void

```
function logMessage(message: string): void {
  console.log(message);
}

const result: void = logMessage('Hello, world!'); // result is undefined
```

Arrays

There are multiple ways to declare arrays in TypeScript

- Using the type[] syntax
- Using the Array<type> syntax

```
let myNumbers: number[] = [1, 2, 3];
let myStrings: Array<string> = ['hello', 'world'];

// same result
let moreNumbers: Array<number> = [1, 2, 3];
let moreStrings: string[] = ['hello', 'world'];
```

```
// the first is more common, because it's shorter
// the second will become clearer after discussing generics
```

Arrays - type protection

```
let myArray = [1, 2, 3]; // myArray is inferred to be of type number[]
myArray.push(16); // OK
myArray.push('16'); // error: argument of type 'string' not assignable to parameter of type 'number'
myArray = [1, 2, 3, 'hello', 'world']; // error: Type 'string[]' is not assignable to type 'number[]'

let myArray1: number[] = [1, 2, 3];
let myArray2: Array<string> = ['hello', 'world'];
myArray1 = myArray2; // error: Type 'string[]' is not assignable to type 'number[]'.
```

Basic object type

```
// Object with specific properties
let entity: { name: string; age: number } = {
  name: 'John',
  age: 30
};

// Type checking
entity = { name: 'Mary', age: '40' }; // error: type 'string' is not assignable to type 'number'
```

Passing objects to functions

```
// Function that takes an object with specific properties
function printPerson(person: { name: string; age: number }) {
  console.log(`Name: ${person.name}, Age: ${person.age}`);
}

// Type checking
printPerson({ name: 'John', age: 30 });
printPerson({ name: 'Mary' }); // error: property 'age' is missing in type '{ name: string; }'
```

Optional properties

```
// Object with optional properties
let person: { name: string; age?: number } = {
  name: 'John'
};

// Type checking
person = { name: 'Ann', age: 34 }; // OK
person = { name: 'Mary', age: '40' }; // error: type 'string' is not assignable to type 'number'
```

Function parameter type annotations

```
// parameter type annotations
function greet(name: string) {
  console.log(`Hello, ${name}!`);
}

const alternateGreet = (name: string) => console.log(name); // as arrow function
```

Function return type annotations

```
// return type annotations
function multiply(a: number, b: number): number {
  return a * b;
}

const mul = (a: number, b: number): number => a * b; // as arrow function

// use void for functions with no documented return value
function speak(word: string): void {
  console.log(word);
}

const talk = (word: string): void => console.log(word); // as arrow function
```

Union types

```
// defining a union type for a variable
let employeeId: number | string;
employeeId = 'S188D7LM';
employeeId = 1927599;
employeeId = false; // error: type 'boolean' is not assignable to type 'string | number'

// defining a union type for a function parameter
function printID(id: number | string): void {
  console.log(`ID: ${id}`);
}
printID(1927599); // output: "ID: 1927599"
printID('S188D7LM'); // output: "ID: S188D7LM"
```

Interface example

```typescript
interface Entity {
  name: string;
  age: number;
}

// Object with specific properties
let entity: Entity = {
  name: 'John',
  age: 30
};

// Type checking
entity = { name: 'Mary', age: '40' }; // error: Type 'string' is not assignable to type 'number'
```

Interface used with functions

```
// Interface with specific properties
interface Person {
  name: string;
  age: number;
}

// Object that implements the Person interface
const john: Person = { name: 'John', age: 30 };

// Function that takes an object of type Person
function printPerson(person: Person) {
  console.log(`Name: ${person.name}, Age: ${person.age}`);
}

// Type checking
printPerson(john);
printPerson({ name: 'John', age: 30 });
printPerson({ name: 'Mary' }); // error: property 'age' is missing in type '{ name: string; }'
```

Type aliases

```
// alias for primitive types
type MyNumber = number;
let x: MyNumber = 42;
let y: number = x; // this is allowed, x and y are the same type

// alias type unions
type Id = number | string;
let employeeId: Id = 'S188D7LM';
employeeId = 1927599;
employeeId = false; // error: type 'boolean' is not assignable to type 'Id'

// a slightly more complex type union
type StringLike = string | (() => string);
let friend: StringLike = 'sam';
let getFriend: StringLike = () => {
  return 'adam';
};
```

Initialized numeric enum

```
// fully initialized enum
enum StatusCode {
  OK = 200,
  NotFound = 404,
  ServerError = 500
}

console.log(StatusCode.OK); // Output: 200
console.log(StatusCode.NotFound); // Output: 404
console.log(StatusCode.ServerError); // Output: 500
```

String enum

```
// string enum
enum LogLevel {
  Error = 'ERROR',
  Warn = 'WARN',
  Info = 'INFO',
  Debug = 'DEBUG'
}

console.log(LogLevel.Error); // Output: 'ERROR'
console.log(LogLevel.Info); // Output: 'INFO'
console.log(LogLevel.Debug); // Output: 'DEBUG'
```

Note: enums should be numeric or string; don't mix types if you don't have a good reason

Your turn

1. Rewrite

- Create a basic typescript project that contains
  - src/ dist/ folders
  - a tsconfig.json file
  - a main.ts file
  - an index.html file that links to the compiled .js file in the dist/ folder

- Choose any of your previous Javascript exercises or projects
  - Rewrite it using typescript
  - Think where typescript can improve the safety of your code or simplify it
  - Compile your code and make sure it works like the original version, or maybe better

References

Documentation - Everyday Types
Documentation - Object Types
Documentation - More on Functions
Handbook - Enums

References

Unions and Intersections
Classes
Documentation - Generics