TypeScript Essentials

A Practical Introduction for JavaScript Developers

(Don't Worry! You can stick with JavaScript)

What is TypeScript?

A Superset of JavaScript

TypeScript extends JavaScript by adding static type definitions and new features while maintaining full compatibility with JavaScript code.

Compiles to JavaScript

The TypeScript compiler (tsc) converts your .ts files into standard JavaScript that runs in any environment—browsers, Node.js, or anywhere else JavaScript runs.

Think of TypeScript as JavaScript with additional safety features that help you catch errors *before* your code runs rather than during runtime.

Why Use TypeScript?

1

Static Type Checking

Catches type-related errors during development rather than at runtime, reducing bugs in production.

2

Enhanced Developer Experience

Better autocomplete, inline documentation, and intelligent code navigation in your IDE.

3

Improved Maintainability

Makes refactoring safer and helps new team members understand your code more quickly.

4

Better for Larger Projects

Scales more effectively as your application grows, making complex codebases more manageable.

Setting Up Your Development Environment

Install Node.js

Download and install from nodejs.org

Verify with: `node --version`

Install TypeScript

Run: `npm install -g typescript`

Verify with: `tsc --version`

Set Up VS Code (optional)

Download VS Code from code.visualstudio.com

Extensions to install:

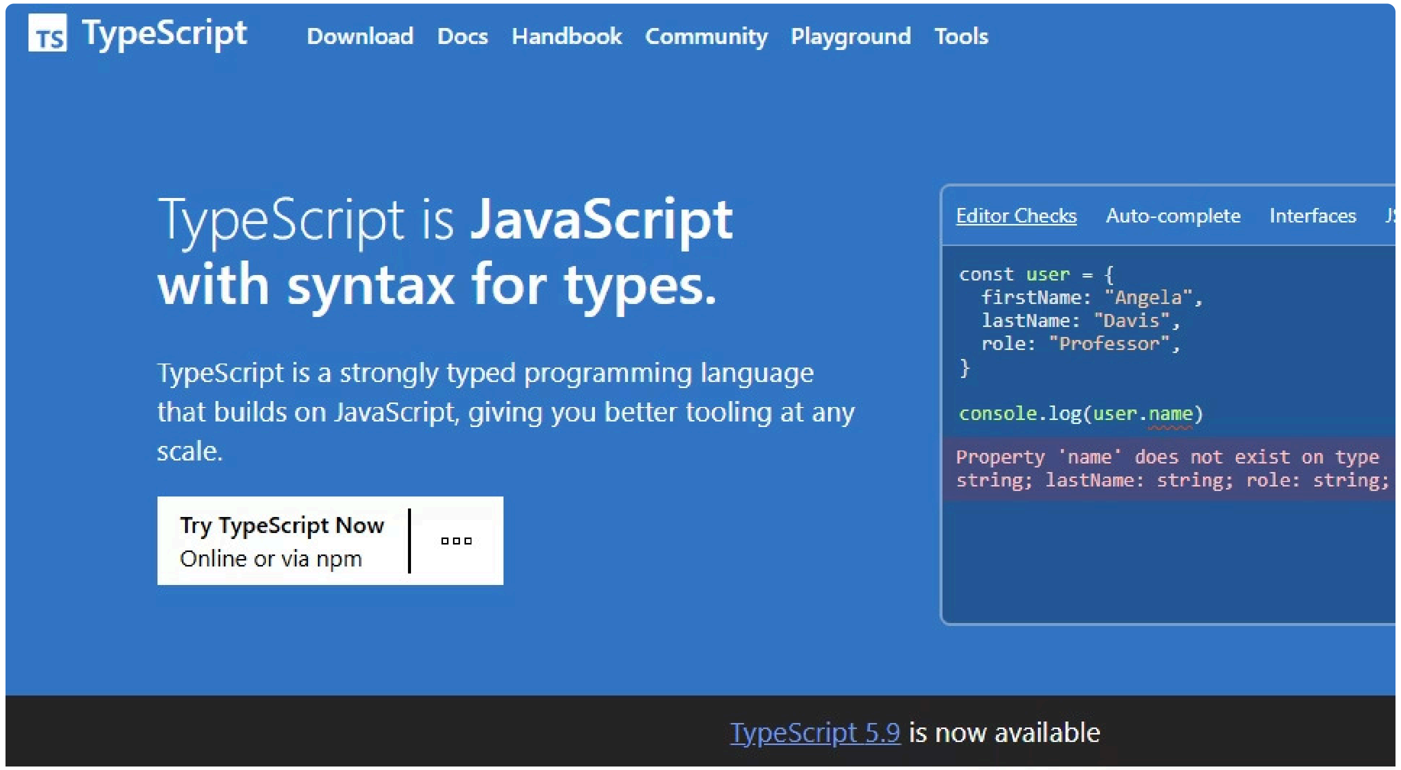
- TypeScript Language Features (built-in)
- ESLint for additional linting

Quick test

typescriptlang

JavaScript With Syntax For Types.

TypeScript extends JavaScript by adding types to the language. TypeScript speeds up your development experience by catching errors and providing fixes before you even run your code.



TypeScript is **JavaScript with syntax for types.**

TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale.

```
const user = {
  firstName: "Angela",
  lastName: "Davis",
  role: "Professor",
}

console.log(user.name)
Property 'name' does not exist on type
string; lastName: string; role: string;
```

TypeScript 5.9 is now available

Your First TypeScript Program

Step 1: Create a TypeScript file

```
// hello.ts
function greet(name: string) {
  return `Hello, ${name}!`;
}

console.log(greet("TypeScript Beginner"));
```

Step 2: Compile your TypeScript code

```
tsc hello.ts
```

Step 3: Run the compiled JavaScript

```
node hello.js
```

What happened:

1. We created a `.ts` file with a typed parameter
2. The TypeScript compiler (`tsc`) generated JavaScript
3. We ran the JavaScript with Node.js

This simple example demonstrates the TypeScript workflow—write TypeScript, compile to JavaScript, then run the JavaScript.

Basic Types: Numbers, Strings, and Booleans

Type Annotations

```
// Explicit type annotations
let age: number = 25;
let name: string = "Alice";
let isActive: boolean = true;
```

Type Inference

```
// TypeScript can infer types
let age = 25;          // number
let name = "Alice";    // string
let isActive = true;   // boolean
```

TypeScript's type system helps prevent common errors like:

```
let age: number = 25;
age = "twenty-five"; // Error: Type 'string' is not
 // assignable to type 'number'
```

Arrays and Tuples

Arrays

Arrays hold multiple values of the same type:

```
// Array of numbers
let scores: number[] = [85, 92, 78];

// Alternative syntax
let names: Array = ["Alice", "Bob"];

// Array with type inference
let items = [1, 2, 3];  // number[]
```

Tuples

Tuples are arrays with fixed length and ordered types:

```
// Define a tuple type
let person: [string, number, boolean] =
 ["Alice", 25, true];

// Accessing tuple elements
console.log(person[0]); // "Alice"
console.log(person[1]); // 25

// Error if types don't match
person = [25, "Alice", true]; // Error!
```

Objects and Interfaces

```
// Define an interface
interface User {
  id: number;
  name: string;
  email: string;
  age?: number;  // Optional property
  readonly createdAt: Date;  // Can't be modified after creation
}

// Use the interface
const newUser: User = {
  id: 1,
  name: "Alice Smith",
  email: "alice@example.com",
  createdAt: new Date()
};

// TypeScript ensures object structure compliance
newUser.phone = "555-1234";  // Error: Property 'phone' does not exist
newUser.createdAt = new Date(2020, 1, 1);  // Error: readonly property
```

Interfaces define the **shape** of objects, helping you ensure consistent structure throughout your application.

Functions and Type Safety

Basic Function Types

```
// Parameter and return type annotations
function add(a: number, b: number): number {
  return a + b;
}

// Arrow function with types
const multiply = (a: number, b: number): number => {
  return a * b;
};
```

Optional and Default Parameters

```
// Optional parameter (note the ?)
function greet(name: string, greeting?: string): string {
 return greeting ? `${greeting}, ${name}!` : `Hello,
${name}!`;
}

// Default parameter
function greet2(name: string, greeting: string =
"Hello"): string {
 return `${greeting}, ${name}!`;
}
```

Union Types and Type Guards

Union Types

Allow a variable to hold more than one type:

```
// Can be either a string or number
let id: string | number;

id = 101;     // Valid
id = "abc123"; // Also valid
id = true;    // Error: not a string or number
```

Type Guards

Help TypeScript narrow down types at runtime:

```
function formatId(id: string | number): string {
  // Type guard with typeof
  if (typeof id === "string") {
    // TypeScript knows id is a string here
    return id.toUpperCase();
  } else {
    // TypeScript knows id is a number here
    return `ID-${id.toString().padStart(6, '0')}`;
  }
}
```

Type guards let you safely work with union types by checking the type at runtime, giving TypeScript the information it needs to type-check your code properly.

Enums and Literal Types

Enums

Define a set of named constants:

```
// Numeric enum
enum Direction {
  North = 0,
  East = 1,
  South = 2,
  West = 3
}

// String enum
enum ApiStatus {
  Success = "SUCCESS",
  Error = "ERROR",
  Pending = "PENDING"
}

let status: ApiStatus = ApiStatus.Success;
```

Literal Types

Restrict values to specific literals:

```
// String literal type
type Alignment = "left" | "center" | "right";

// Function that only accepts specific values
function setAlignment(align: Alignment) {
 // ...
}

setAlignment("left"); // Valid
setAlignment("center"); // Valid
setAlignment("top"); // Error!
```

Classes in TypeScript

```
class Person {
 // Properties with access modifiers
 private readonly id: number;
 public name: string;
 protected age: number;

 // Constructor
 constructor(id: number, name: string, age: number) {
 this.id = id;
 this.name = name;
 this.age = age;
 }

 // Method
 public greet(): string {
 return `Hello, my name is ${this.name}`;
 }
}

// Inheritance
class Employee extends Person {
 private department: string;

 constructor(id: number, name: string, age: number, department: string) {
 super(id, name, age); // Call parent constructor
 this.department = department;
 }

 public getDetails(): string {
 // Can access protected property from parent
 return `${this.name}, ${this.age}, ${this.department}`;
 }
}
```

Generics Basics

Generics allow you to create reusable components that work with *any* data type while maintaining type safety.

Generic Functions

```
// A generic function
function identity(arg: T): T {
  return arg;
}

// Calling with explicit type
let num = identity(42);

// Type inference (preferred)
let str = identity("hello");
```

Generic Classes

```
// A generic class
class Box {
 private contents: T;

 constructor(value: T) {
 this.contents = value;
 }

 getContents(): T {
 return this.contents;
 }
}

// Create instances
let numberBox = new Box(123);
let stringBox = new Box("hello");
```

Working with External Libraries

Installing Type Definitions

Most popular libraries have type definitions available in the DefinitelyTyped repository:

```
npm install axios         # 
Install library
npm install @types/axios    # 
Install type definitions
```

Libraries with Built-in Types

Many modern libraries include TypeScript definitions:

```
npm install react         # 
Types included!
import { useState } from 
'react';  # Fully typed
```

Custom Type Declarations

For libraries without types, create a `.d.ts` file:

```
// my-library.d.ts
declare module 'untyped-
library' {
 export function 
doSomething(value: string): 
number;
 export const VERSION: 
string;
}
```

Common TypeScript Errors and Solutions

Error Message	Likely Cause	Solution
`Property does not exist on type...`	Accessing a property that doesn't exist in the type definition	Update the interface/type or fix the property name
`Type 'X' is not assignable to type 'Y'`	Trying to assign an incompatible value to a variable	Use the correct type or add type conversion
`Cannot find module...`	Missing imports or type definitions for external modules	Install @types package or create custom type declarations
`Parameter 'x' implicitly has an 'any' type`	Missing type annotations with strict mode enabled	Add explicit type annotations to parameters

Debugging Tip: When facing cryptic errors, try breaking down complex expressions into smaller, explicitly typed variables to pinpoint the issue.

TypeScript Configuration (tsconfig.json)

Basic Configuration

```
{
  "compilerOptions": {
    "target": "es2016",
    "module": "commonjs",
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "esModuleInterop": true
  },
  "include": ["src/**/*"],
  "exclude": ["node_modules"]
}
```

Key Compiler Options

- `target`: JavaScript version to compile to
- `module`: Module system (commonjs, es6, etc.)
- `outDir`: Output directory for compiled files
- `strict`: Enable strict type-checking
- `noImplicitAny`: Disallow implicit 'any' types
- `strictNullChecks`: More rigorous null/undefined checks

Create a config file with: tsc --init

Next Steps and Resources

Advanced Topics

- Utility Types (Partial, Readonly, etc.)
- Mapped and Conditional Types
- Decorators and Metadata
- TypeScript with React/Angular/Vue

Official Resources

- TypeScript Handbook (typescriptlang.org)
- TypeScript Playground (online editor)
- DefinitelyTyped (GitHub repository)

Community Support

- Stack Overflow TypeScript tag
- TypeScript Discord/Reddit communities
- TypeScript Weekly newsletter

Remember: The best way to learn TypeScript is to **practice building real projects**!