15-Minute TypeScript Introduction for Computer Science Students

What is TypeScript? (2 minutes)

TypeScript is an open-source language developed and maintained by Microsoft. It's a **syntactic superset of JavaScript**, meaning any valid JavaScript code is also valid TypeScript code. The key differentiator? TypeScript adds **static typing** to JavaScript.

So, what is it? The simplest definition is: **TypeScript is JavaScript with syntax for static types.**

> It's a "superset" of JavaScript, which means any valid JavaScript code is also valid TypeScript code. Think of it this way: JavaScript is the alphabet. TypeScript is the alphabet plus a grammar book. It doesn't change the language, it just adds rules to help you write it better.

Why Bother Using It? (2 minutes)

You might be wondering, "Why add complexity to JavaScript?" There are 4 huge benefits:

1. **Catch Errors Early:** TypeScript catches common bugs while you're typing, not when your app is crashing for a user.
2. **Amazing Tooling:** Code editors like VS Code can provide better autocompletion, refactoring, and error-checking because they understand the "shape" of your data.
3. **Readability and Maintenance:** Your code becomes self-documenting. It's easier for you (and your teammates) to understand what functions expect and what they return, which is critical for large projects.
4. **Scalability:** As projects grow, static typing helps manage complexity and reduce bugs.

Core Concepts in Action

Let's skip the slides and see it in action. The best way to understand TypeScript is to see the problems it solves.

Basic Types

Imagine this classic JavaScript bug. We have a function to add two numbers.

```javascript
// file: add.js
function add(a, b) {
  return a + b;
}

console.log(add(5, 10));   // Prints 15. Correct!
console.log(add("5", 10)); // Prints "510". Bug! 🐛
```

JavaScript happily concatenates the string "5" and the number 10. This is a common source of runtime errors. Now, let's fix this with TypeScript. All we do is add type annotations.

```
// file: add.ts
function add(a: number, b: number): number {
  return a + b;
}

console.log(add(5, 10));

// The line below will now show an error in your editor!
console.log(add("5", 10));
```

Before we even run the code, our editor is screaming at us. If we try to compile this .ts file, we get a clear error: Argument of type 'string' is not assignable to parameter of type 'number'. We just caught a bug at compile-time instead of runtime.

Getting Started: Installation & Basic Compilation (3 minutes)

1. Installation (using npm - Node Package Manager):

```
npm install -g typescript
```

This command installs the TypeScript compiler (`tsc`) globally on your system.

2. Your First TypeScript File:

Create a file named `hello.ts` and add the following code:

```
// hello.ts
function greet(person: string) {
    return "Hello, " + person;
}

let user = "Alice";
console.log(greet(user));

// What if we try to pass a number?
// let age = 30;
// console.log(greet(age)); // This would cause a compile-time error!
```

3. Compiling TypeScript to JavaScript:

Open your terminal in the same directory as `hello.ts` and run:

```
tsc hello.ts
```

This will generate a `hello.js` file:

```
// hello.js (automatically generated by tsc)
function greet(person) {
    return "Hello, " + person;
}
var user = "Alice";
console.log(greet(user));
// What if we try to pass a number?
// let age = 30;
// console.log(greet(age)); // This would cause a compile-time error!
```

Notice that the type annotations (`: string`) are removed in the compiled JavaScript. This is because browsers only understand JavaScript. TypeScript is a *compile-time* tool.

4. Running the JavaScript:

```
node hello.js
```

You should see: `Hello, Alice`

Core Concepts: Type Annotations (5 minutes)

Type annotations are how we tell TypeScript the expected type of a variable, function parameter, or function return value.

1. Primitive Types:

```
let myName: string = "Bob";
let age: number = 25;
let isStudent: boolean = true;
let anything: any = "can be anything"; // Avoid 'any' when possible!
let noValue: null = null;
let notDefined: undefined = undefined;
```

2. Array Types:

```
let numbers: number[] = [1, 2, 3];
let names: Array<string> = ["Alice", "Bob"]; // Generic array type
```

3. Function Types:

```
// Parameters with types, and return type
function add(a: number, b: number): number {
    return a + b;
}

// Optional parameters (use '?' after the parameter name)
function logMessage(message: string, userName?: string): void {
    if (userName) {
        console.log(`${userName}: ${message}`);
    } else {
        console.log(message);
    }
}

// Default parameters
function multiply(a: number, b: number = 2): number {
    return a * b;
}

console.log(add(5, 3)); // 8
logMessage("Hello TypeScript!");
logMessage("Welcome!", "Student");
console.log(multiply(5));   // 10 (5 * 2)
console.log(multiply(5, 3)); // 15 (5 * 3)
```

Advanced Concepts: Interfaces & Classes (5 minutes)

1. Interfaces:

Interfaces are a powerful way to define the "shape" of an object. They are crucial for defining contracts within your application.

```
// Define an interface for a Person object
interface Person {
    firstName: string;
    lastName: string;
    age?: number; // '?' makes 'age' an optional property
}

// Create an object that adheres to the Person interface
function greetPerson(person: Person): string {
    return `Hello, ${person.firstName} ${person.lastName}!`;
}

let user1: Person = { firstName: "Jane", lastName: "Doe" };
let user2: Person = { firstName: "John", lastName: "Smith", age: 30 };

console.log(greetPerson(user1)); // Hello, Jane Doe!
console.log(greetPerson(user2)); // Hello, John Smith!
```

```
// This would cause a compile-time error because 'firstName' is missing:
// let invalidUser: Person = { lastName: "Bloggs" };
```

2. Classes:

TypeScript supports object-oriented programming with classes, similar to Java or C#.

```
class Greeter {
    greeting: string; // Property

    constructor(message: string) { // Constructor
        this.greeting = message;
    }

    // Method
    greet(): string {
        return "Hello, " + this.greeting;
    }
}

let greeter = new Greeter("world");
console.log(greeter.greet()); // Hello, world

class Student extends Greeter {
    fullName: string;
    constructor(firstName: string, middleInitial: string, lastName:
string) {
        super(`student ${firstName} ${lastName}`); // Call parent
constructor
        this.fullName = `${firstName} ${middleInitial} ${lastName}`;
    }

    getStudentName(): string {
        return this.fullName;
    }
}

let student = new Student("Mister", "T", "TypeScript");
console.log(student.greet());         // Hello, student Mister TypeScript
console.log(student.getStudentName()); // Mister T TypeScript
```

Why Learn TypeScript?

- **Industry Demand:** Many modern web development frameworks (Angular, React, Vue) heavily use or recommend TypeScript.
- **Large-scale Applications:** Essential for building robust, maintainable applications, especially in teams.
- **Bridging JavaScript & Academia:** Helps bring the rigor of statically typed languages to the flexible world of JavaScript.

Next Steps:

- **Official TypeScript Handbook:** https://www.typescriptlang.org/docs/handbook/intro.html
- **Playground:** Experiment directly in your browser: https://www.typescriptlang.org/play
- **Set up a project with `tsconfig.json`:** Learn how to configure the TypeScript compiler for larger projects.

That's your 15-minute crash course! TypeScript is a powerful tool that will make your JavaScript development more reliable and enjoyable.