What is TypeScript?

Why would you use TypeScript?

TypeScript is a JavaScript superset

TypeScript is a JavaScript superset

It’s an “extension” to the JavaScript language

The core syntax & features are the same
but extra features are added

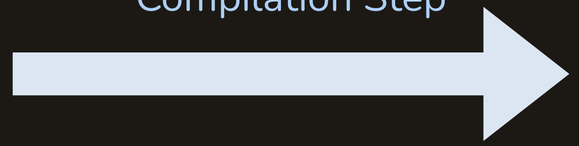
Most importantly: Strict & static typing

You Can’t Execute TypeScript Code

Unless you’re working with a runtime that has built-in TypeScript support (e.g., Bun, Deno)

TypeScript Code

```
function calculate(input: number) {
  return input + 10;
}
```



JavaScript Code

```
function calculate(input) {
  return input + 10;
}
```

To run TypeScript code in other environments (e.g., the browser), you must **compile it to standard JavaScript code**

About This Course

Essentials

Assigning & Inferring Types

Built-in Types & Essential Types

Custom Types

Advanced

Classes & Interfaces

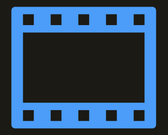
Generic & Derived Types

Decorators

Small Demos & Code Snippets

Demo Projects

How To Get The Most Out Of This Course



Watch the Videos

Optimize playback speed

Watch multiple times

Follow curriculum, at least for the first sections

Code Along & Practice

Only practice makes perfect!

Deviate from course code

Use TypeScript in your future (dummy) projects

Ask & Answer

Try solving problems on your own first

Use AI or Google if you can’t find a solution

Ask & Answer in the Q&A section

Classes & Interfaces

Creating & Using Classes

Constructor Functions & Properties

Methods

Inheritance

Interfaces

Maximilian Schwarzmüller

What Are Classes?

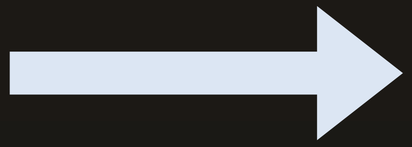
Classes are blueprints for objects

Class Definition

```
class User {
  name: string;
  age: number;

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

  greet() {
    console.log('Hi, I am ' + this.name);
  }
}
```



Instantiated Object

```
const max = new User('Max', 38);

max.greet(); // Hi, I am Max


const fred = new User('Fred', 35);

fred.greet(); // Hi, I am Fred
```

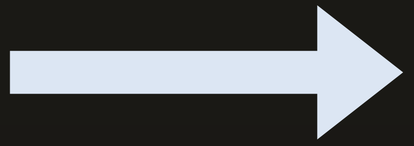
What Are Interfaces?

Object type definitions & **contracts** that can be implemented by classes

Interface Definition

```
interface Authenticatable {
  email: string;
  password: string;

  login(): void;
  logout(): void;
}
```



Usage as Object Type

```
let authUser: Authenticatable;

authUser = {
  email: 'test@example.com',
  password: 'abc1',
  login() { … },
  logout() { … }
};
```

Implementation as Contract

```
class User implements Authenticatable {
  constructor(
    public email: string,
    public password: string
  ) {}

  login() { … }

  logout() { … }
}
```

Advanced TypeScript - Demo Project

Building a Linked List Data Structure

Work with Classes & Generics

Maximilian Schwarzmüller

Understanding Linked Lists

A data structure that stores connected values

Every Node links to the Node next to it (but to no other Node in the overall list)

Decorators

What & Why

Creating Class, Method & Property Decorators

Decorator Factories

Official Decorators vs Experimental Decorators

Maximilian Schwarzmüller

What Are Decorators?

Metaprogramming

Code that interacts with other code

TypeScript Supports Two Kinds Of Decorators

ECMAScript Decorators

Built-into JavaScript

Can be used without TypeScript

Experimental Decorators

Previously planned implementation

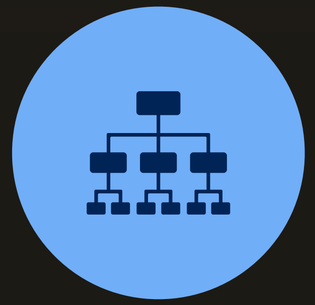
Did not make it into JavaScript

Only supported by TypeScript

Requires “experimentalDecorators” configuration

Types of Decorators

- Class Decorators
- Method Decorators
- Field Decorators
- Getter Decorators
- Setter Decorators



Modules & Namespaces

Splitting Code Into Multiple Files

Maximilian Schwarzmüller

Two Options For Splitting Code

Namespaces & File Bundling

Use “`namespace`” syntax to group related code

Per-file or bundled compilation is possible

ES Modules

Use official ES Modules import / export syntax

Bundling via build tools (e.g., Webpack) is still possible