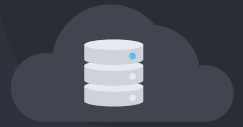
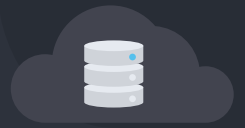
TypeScript #1

Objectives



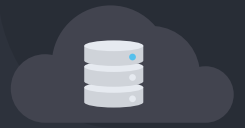
By the end of this session, You will be able to:

1. Understand why TypeScript exists and how it evolved.
2. Learn the difference between static vs dynamic typing.
3. Explore TypeScript's main features (types, inference, interfaces, generics).
4. Learn what transpilers do and how TSC works.
5. Configure TypeScript using tsconfig.json.
6. See how TypeScript fits into modern build workflows



Don't forget our Rules

1. 15 mins late is acceptable, after 15 mins every minute is equal to 1 pound. max(20 pounds)
2. Absence without any excuse -> 2 warnings
3. Missing Task without any excuse -> 1 warning
4. Any unacceptable behavior -> warning



Don't forget our Rules

1. Any correct answer in the session = Mohsens
2. Best member of the week will be awarded next session
3. Best of the month will be awarded 1/12 based on mohsens
4. Best of the Season will be awarded at the end of the season based on mohsens and overall performance

01 Recap

- What happened last Session?
- What did we talk about?
- What was the benefit you got from last session?

Table of Content

Table of Content

Who here hates JS?

</ JS development:

- JavaScript is awesome for building things, but it can get chaotic.
- Suddenly you're debugging for hours, and the bug?
- Something JS didn't warn you about until it exploded at runtime.

How was this problem fixed?

</ TypeScript was the answer

Microsoft introduced TypeScript as a solution:

- Code is safe and type annotated
- Bugs appear early in compile time
- Maintaining Large projects became easier.

What is TypeScript?

</ TypeScript, JavaScript's cousin

TypeScript is a superset of JavaScript, Where it includes static types, and can also be dynamically typed.

TypeScript is also not "Compiled" but rather Transpiled – converted into JavaScript code, but why?

</ Quick fact

- When TypeScript was created, no browser or runtime could execute TypeScript directly.

- So TypeScript had two choices:
  - Option A:
    - Create a new runtime or virtual machine that understands TypeScript → Not practical.
    - It would break compatibility with the entire JS ecosystem.
  - Option B:
    - Compile TypeScript into JavaScript, which browsers already understand.
    - TypeScript chose 100% compatibility with existing JavaScript engines, so it had to be transpiled.

</ Dynamic vs Static Typing

- Dynamic typing means type checking happens at runtime, not during compilation. (e.g. Python, JavaScript)

- While Static typing is that the type checking happens during writing code, so errors appear early on. (e.g. C++, Java)

Quick Competition

Why do we say compilation time even tho JS is interpreted and what is a LSP?

First one to answer => 5 mohsens

Table of Content

TypeScript Basics

1- Types:

```
let Name: string = "Mohsen";
let Age: Number = 28;
let isCool: Boolean = true;
```

You can also define your own type, for example:

```
type Committee = "Backend" | "Linux" | "Frontend";
```

extra- WD-40 of TypeScript:

The Any keyword is used when you don't expect a certain data type, like what JavaScript does.

Example:

```
let age: any;
age = "hello world";
age = 32;
```

Union Types:

```
let age: Number | String = "8";
```

As you can see we can here use the pipe operator “ | ” to declare that a variable may have 2 different data types.

2- Functions:

```
function functionName[Parameters: parameterType]: ReturnType {
    Function body
}
```

Example:

```
function Greet[name: string]: void {
   console.log[`Hello ${name}`]
}
```

Arrow function

```
const fun:()=>string=()=>{
return "hello"
}

const sum:(x:number,y:number)=>number=(x,y)=>{
return x+y
}
```

```
function CalculateArea [Length, Width] {
   return Length*Width;
}

let area = CalculateArea[2,3]; // Works fine
area = CalculateArea[2,"3"];   // Works fine [isn't supposed to]
```

JavaScript

```
function CalculateArea [Length: number, Width: number]: number {
   return Length*Width;
}

let area = CalculateArea[2,3];   // Works fine
area = CalculateArea[2,"3"];    // Compile Time error
```

TypeScript

3- Arrays

```
let arr =[1,2,3,"hi",true]; //same as
let ar:any[] =[1,2,3,"hi",true];
let array:(string | number )[]  =[1,2,3,"hi"];
let a:(string )[]  =["yomna","hi"];
let nums:(number )[]  =[1,2,3];
```

3- Arrays

```
let tuple: [string, number, boolean] = ["Yomna", 25, true];
let readonlyArr: readonly number[] = [1, 2, 3];
let nums: Array<number> = [1, 2, 3];
let names: Array<string> = ["Yomna", "Ali"];
let mixed: Array<number|string> = [1, "hi"];
```

4- Generic Types:

```
function identity<T>(value: T): T {
  return value;
}

let num = identity<number>(42);     // T is number
let str = identity<string>("hello"); // T is string
let auto = identity(true);          // T inferred as boolean
```

This is like generalizing what a function can do to multiple data types.

5- Interfaces:

Think of them as Structs in C++

```
interface member {
    name: String;
    age: Number;
    committee: String;
}
```

When this code compiled into JS , it will be:

```
var mem = {
  name: "Mohsen",
  age: 28,
  committee: "BackEnd" };
console.log(mem);
```

There is no interface in JavaScript, and TypeScript annotations (: Member) are removed after compilation

Table of Content

6- Intersection Types: “Type Aliases”

```
type Id = { id: number };
type Name = { name: string };
type User = Id & Name;
let person:User={id:10,name:"yomna"}
```

Here use the ampersand operator “ & ” to declare that a variable will have the attributes of all types declared.

Another Quest

What is the main differences between **interface** and **type alias** ?

First one to answer => 5 mohsens

7-classes

Class = a blueprint for creating **objects.**

Can contain:

1. **Properties** [object variables].
2. **Constructor** [function executed when an object is created].
3. **Methods** [functions inside the object].

TypeScript: Type vs Interface vs Class vs Object

Type	Interface	Class	Object
Compile-Time Only	Compile-Time Only	Run time	Run time
Disappears after compilation as it is unique for TS	Disappears after compilation as it is unique for TS	Exists at runtime, in the JS code	Exists at runtime, in the JS code
A way to define shapes and aliases for type checking	A contract that defines object structure	A blueprint for creating objects with behavior	Actual data stored in memory

What is partial type ?

In TypeScript, Partial<Type> is a **utility type** that makes **all properties of a type optional**.

```
interface User {
 id: number;
 name: string;
 email: string;
}
const user1: User = {
 id: 1,
 name: "Yomna",
 email: "yomna@example.com"
};
```

```
const user2: User = {
 id: 2,
 name: "Ali"
}; // Error: Property 'email' is missing
BUTT!
const partial_User: Partial<User> = {
 id: 3
}; // Works, name & email are optional

## We can use ? => it means optional attribute
name?: "Ali"// Works, name is optional
```

Table of Content

Table of Content

</ Top-down vs Bottom-up flows

Top-Down

```
interface User {
  id: number;
  name: string;
  address: Address;
  roles: Role[];
}

interface Address {
  city: string;
  street: string;
}
type Role = "admin" | "member" | "guest";
```

Bottom-up

```
interface Id {
  id: number;
}
interface Name {
  name: string;
}
type Role = "admin" | "member" | "guest";

interface User extends Id, Named {
  roles: Role[];
}
```

Bottom up	Top down
Start small then build up	Start big then break down
Build all pieces then assemble	Write what you want first then define how later

</ Shape Compatibility

If we define multiple interfaces that have common attributes we can assign an object of the more general interface to an object of the narrower scope interface.

Shape Compatibility example

```
const emp: Employee = {
  name: "Bianka",
  age: 5,
  employeeId: 101
};

const person: Person = emp; //
works
```

```
interface Person {
  name: string;
  age: number; }
interface Employee {
  name: string;
  age: number;
  employeeId: number; }
const p: Person = {
  name: "Bianka",
  age: 5 };
const e: Employee = p; // Error
```

But it can work in 2 cases! +2 mohsens

Table of Content

Hands On

Break