

Moving from JavaScript to TypeScript

Write code that is easier to understand

November 19, 2022 | [Adil Altaf](#) & [Hira Khan](#)



JavaScript



TypeScript

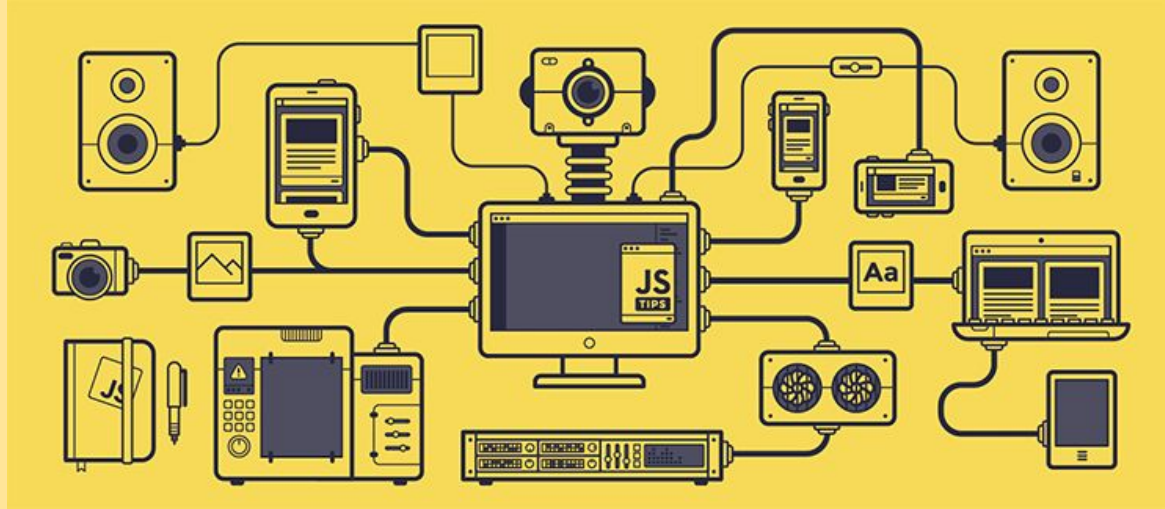
AGENDA

- History of JavaScript
- Pitfalls of Vanilla JavaScript
- TypeScript
- TypeScript Playground
- Local Development
- What TypeScript Isn't

Objective

The objective of this lecture is to understand the need to move from JavaScript to TypeScript.

History of JavaScript



JavaScript created for
Netscape



ECMAScript 3 released



Full support for
ECMAScript 6 in all
browsers



1995

1997

1999

2009 -
2014

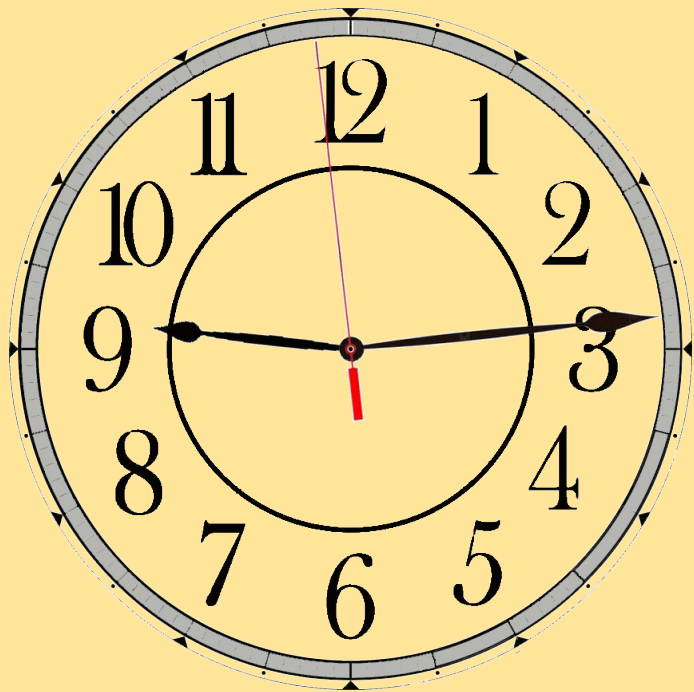
2015 -
2018

ECMAScript 1 released



ECMAScript 5 released
in 2009, full support in





BACKWARDZ COMPATIBLE

Even with regular new language versions, JavaScript has managed to maintain backwards compatibility for decades in varying environments, including browsers, embedded applications, and server runtimes.

Pitfalls of Vanilla JavaScript

Vanilla

Developers often refer to using JavaScript without any significant language extensions or frameworks as “vanilla”: referring to it being the familiar, original flavor.



~~Type Checking~~

```
1 function paintPainting(painter, painting) { return painter
2     .prepare()
3     .paint(painting, painter.ownMaterials)
4     .finish();
5 };
```

If you read this code without any context, you will only have vague ideas on how to call the paintPainting function.

You will have no idea what the two parameters (painter, painting) are and how to form these arguments when calling the function.



LOOSE DOCUMENTATION

```
/**
 * Performs a painter painting a particular painting.
 *
 * @param {Painting} painter
 * @param {string} painting
 * @returns {boolean} Whether the painter painted the painting.
 */
function paintPainting(painter, painting) {
  /* ... */
}
```

There is no standard in the JavaScript Language Specification to formally describe the meaning of function parameters, function returns, variables or other constructs.

JS Developers have adopted the JSDoc standard to describe constructs by adding standardized comments placed directly above code.

JSDoc Issues

- JSDoc descriptions can be wrong about code.
- JSDoc comments may become invalidated during a refactor.
- Describing complex objects requires multiple standalone comments to define types and their relationships.
- Impossible to maintain across files in a complex project.

Document your
Javascript code
with JSDoc



```
/**
 * Retrieves a user by email.
 * @async
 * @method
 * @param {String} email - User email
 * @returns {User} User object
 * @throws {NotFoundError} When user not found
 */
const getByEmail = async (email) => {
  // ...
}
```

Weak Tooling

It can be difficult to automate large changes to or gain insights about a codebase.

JavaScript developers are often surprised to see features in typed languages such as C# and Java that allow developers to jump to the place an argument's type was declared.



TypeScript





```
App.tsx  Header.tsx x
2 import logo from './logo.svg';
3
4 interface IProps {
5   userName: string;
6 }
7
8 const Header: SFC<IProps> = props => {
9   return (
10     <header className="App-header">
11       <img src={logo} className="App-logo" alt="logo" />
12       <p>welcome {props.userName}</p>
13     </header>
14   );
15 };
16
17 export default Header;
18
```

TypeScript is 4 things: • Programming Language • Type Checker • Compiler • Language Services



Playground

TS Config ▾

Examples ▾

What's New ▾

Help ▾

Settings

v4.8.2 ▾

Run

Export ▾

Share



```
1  const firstName = "Georgia";  
2  const nameLength = firstName.length();
```



```
1
2
3
4
5
6
7
8
9 const firstName = "Georgia";
10 const nameLength = firstName.length();
```

(property) `String.length`: `number`

Returns the length of a String object.

`length`

This expression is not callable.
Type 'Number' has no call signatures. (2349)

[View Problem \(\xF8\)](#) No quick fixes available



If you tried to run that code in JavaScript, it would crash!

Freedom Through Restriction

TypeScript allows developers to specify what types of values may be provided for parameters and variables.

Some developers find having to explicitly write out how particular areas are supposed to work to be restrictive at first.

By restricting code to only be used in the ways you specify, TypeScript can give you confidence that changes in one area of code won't break other areas of code that use it.



**RESTRICTED
AREA**

```
1 function sayMyName(firstName, lastNameName) {  
2   console.log(`You acting kind of shady, ain't callin' me ${firstName + ' ' + lastNameName}`);  
3 }  
4  
5 sayMyName("Beyoncé", "Knowles");
```

▶ Run


In this example, we have a function that takes in 2 arguments for first name and last name and returns a console log displaying the full name within the sentence.

Here's the output:

```
1  
2 "You acting kind of shady, ain't callin' me Beyoncé Knowles"  
3
```

Let's take a look at this sample code.

```
1 function sayMyName(fullName) {  
2   console.log(`You acting kind of shady, ain't callin' me ${fullName}`);  
3 }  
4  
5 sayMyName("Beyoncé", "Knowles");
```



Now we have refactored the function and instead of providing separate arguments for the first and last names, we provide the full name in one argument.

Here's the output:

```
1  
2 "You acting kind of shady, ain't callin' me Beyoncé"
```

Now, what if we refactor the function to take only 1 argument for the full name and forget to change the function call?

Precise Documentation

This is the same paintPainting function from earlier, except now it's written with TypeScript.

```
1 interface Painter {  
2   finish(): boolean;  
3   ownMaterials: Material[];  
4   paint(painting: string, materials: Material[]): boolean;  
5 }  
6  
7 function paintPainting(painter: Painter, painting: string): boolean { /* ... */}
```

A TypeScript developer reading this code for the first time could understand that painter has at least three properties, two of which are methods. By baking in syntax to describe the “shapes” of objects, TypeScript provides an excellent, enforced system for describing how objects look.

Stronger Tooling



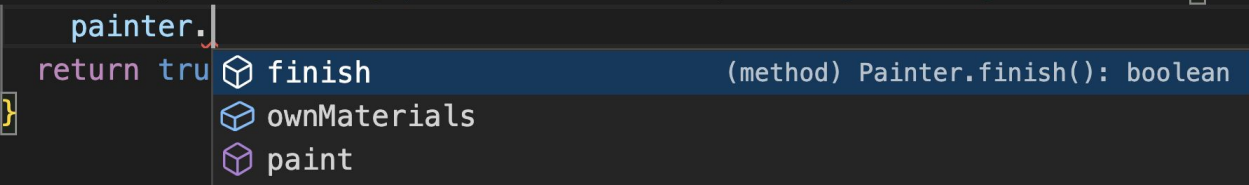
Using TypeScript, IDEs such as VS Code, gain much deeper insights into your code and are able to provide tools such as surface intelligent suggestions as you type.



If you've used VS Code to write JavaScript before, you might have noticed that it suggests "autocompletions" as you write code with built-in types of objects like strings. If, say, you start typing the member of something known to be a string, TypeScript can suggest all the members of the string.

```
1  "test".
2
3  charAt      (method) String.charAt(pos: number): string
4  charCodeAt
5  codePointAt
6  concat
7  endsWith
8  includes
9  indexOf
10 lastIndexOf
11 length
12 localeCompare
13 match
14 matchAll
```

In this example, as soon as we type `painter.`, TypeScript is able to tell us the properties of a painter based on the `Painter` interface.

```
interface Painter {  
  finish(): boolean;  
  ownMaterials: Material[];  
  paint(painting: string, materials: Material[]): boolean;  
}  
  
function paintPainting(painter: Painter, painting: string): boolean {  
  painter.  
  return true  
}
```



-  finish (method) Painter.finish(): boolean
-  ownMaterials
-  paint

Compiling Syntax

The TypeScript compiler takes in TypeScript code, checks the types, and generates the equivalent JavaScript code.

Let's take a look at the following TypeScript code (left-side). The TypeScript compiler generates the equivalent JavaScript code (right-side).

```
1  const artist = "Augusta Savage";  
2  console.log({ artist });
```

```
"use strict";  
const artist = "Augusta Savage";  
console.log({ artist });
```

Local Development



Installing TypeScript

To install the latest version of TypeScript globally, run the following command:

```
npm i -g typescript
```

Check TypeScript Compiler Version

Now you can run the TypeScript Compiler command (tsc). To make sure it's installed properly, run it with the --version flag

```
tsc --version
```

Initiate a New TypeScript Project

To start a new TypeScript project, create a new folder and run this command to create a new *tsconfig.json* file:

```
tsc --init
```

The *tsconfig.json* file declares the configuration that TypeScript uses when analyzing your code.

Create File and Run the TypeScript Compiler

Create a new file named *index.ts* and add the following line of code:

```
console.log("Nothing is worth more than laughter.");
```

Now run the TypeScript Compiler using the `tsc` command and provide the name of the file.

```
tsc index.ts
```

Create File and Run the TypeScript Compiler

There's no property such as blub in the console, so the compiler returns an error.

```
index.ts:2:9 – error TS2339: Property 'blub' does not exist on type 'Console'.
```

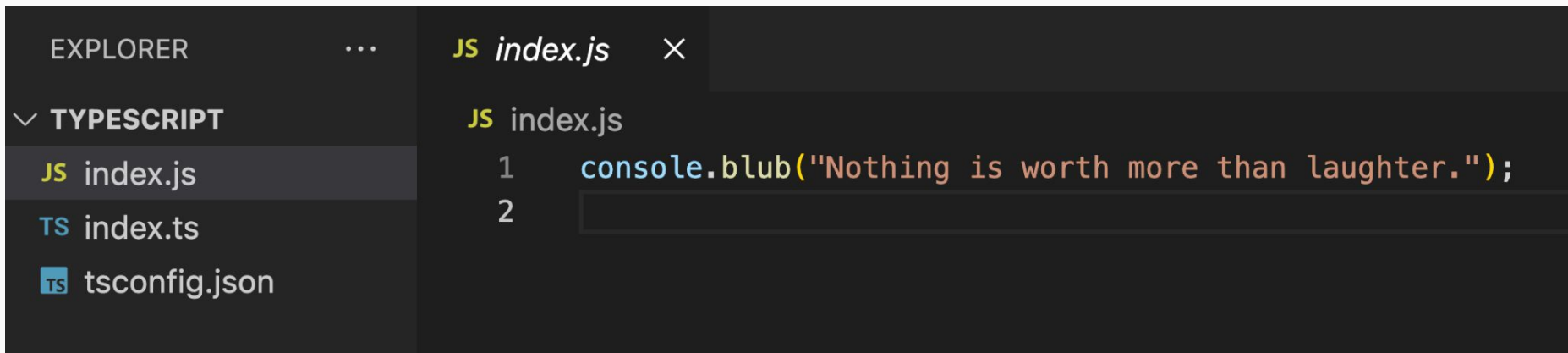
```
2 console.blub("Nothing is worth more than laughter.");
```

~~~~~

```
Found 1 error.
```

# TypeScript Compiler

Please note that tsc created an index.js for you with contents including the console.blub.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar is open, showing a project structure under the 'TYPESCRIPT' folder. It contains three files: 'index.js' (JavaScript file), 'index.ts' (TypeScript file), and 'tsconfig.json' (TypeScript configuration file). The main editor area on the right displays the content of 'index.js'. The file is named 'index.js' and contains a single line of code: `console.blub("Nothing is worth more than laughter.");`. The code is syntax-highlighted, with 'console' in blue, 'blub' in orange, and the string in quotes in orange. The line number '1' is visible on the left side of the editor.

```
EXPLORER  ...  JS index.js  X

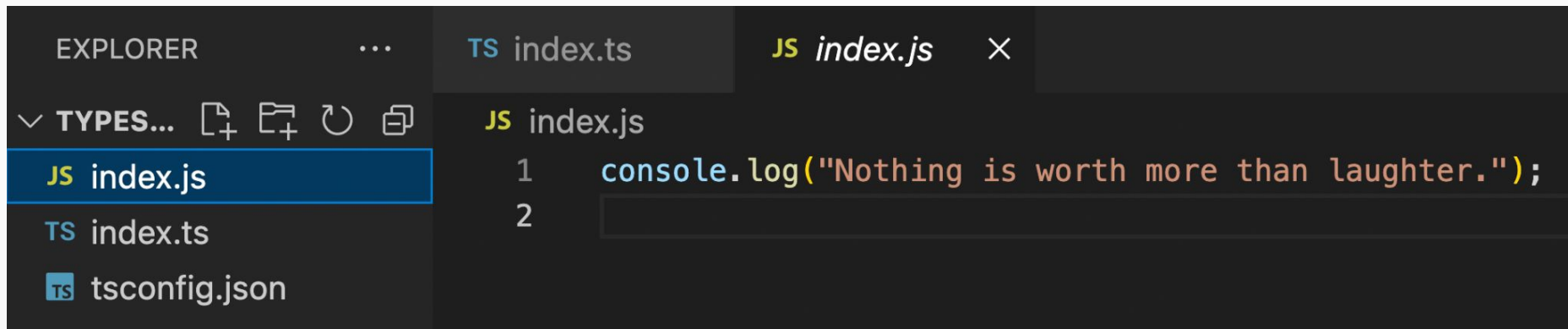
  v TYPESCRIPT
    JS index.js
    TS index.ts
    TS tsconfig.json

  JS index.js
  1  console.blub("Nothing is worth more than laughter.");
  2
```



# TypeScript Compiler

Now fix the code in index.ts to use console.log and run tsc again.





**What TypeScript is NOT**

# Remedy for Bad Code

TypeScript is not a remedy for badly structured code.

It helps you structure JavaScript code, but other than enforcing type safety, TypeScript doesn't enforce any opinions on what that structure should look like.

TypeScript is not an opinionated framework. So you can write code using whatever architectural patterns you're used to from JavaScript, and TypeScript will support them.

# Extensions to JavaScript

TypeScript's design goals explicitly state that it should:

- Align with current and future ECMAScript proposals
- Preserve runtime behavior of all JavaScript code

TypeScript does not try to change how JavaScript works.

Its creators have tried very hard to avoid adding new code features that would add to or conflict with JavaScript. These tasks are the responsibility of TC39, the technical committee that works on ECMAScript itself.

# Slower than JavaScript

The only changes TypeScript makes to code are if you ask it to compile your code down to earlier versions of JavaScript to support older runtime environments such as Internet Explorer 11.

TypeScript adds some time to building your code because it must be compiled down to JavaScript before most environments, such as browsers and Node.js, will be able to run it.

Most build pipelines are generally set up so that the performance hit is negligible, and slower TypeScript features such as analyzing code for likely mistakes are done separately from generating runnable application code files.

# Finished Evolving

The web is nowhere near finished evolving, and so neither is TypeScript.

The TypeScript language is constantly receiving bug fixes and feature additions to match the ever-shifting needs of the web community. The basic tenets of TypeScript will remain about the same, but error messages, fancier features, and editor integrations will improve over time.

# Summary

In this section, we have covered:

- A history of JavaScript
- Pitfalls of JS
- Intro to TypeScript
- Advantages of TypeScript
- Getting Started with TypeScript
- What TypeScript Isn't

# Assignment

<https://www.learningtypescript.com/from-javascript-to-typescript/the-typeinator/>