# **What is TypeScript?**

TS is a language on top of JS i.e., everything possible in JS is available in TS as well – hence it is a superset of JS.

It provides 2 strong advantages –

* It transpiles TS into JS so advanced ES features can be used which are not available to all web browsers and does that by providing polyfill. It acts as a combination of other static checkers combined with babel.
* It can enforce static typing to catch potential issues earlier in SDLC hence reducing the need for some unit tests and can analyze the code to find runtime errors at design time.

TS was made public on October 1st 2012, but inception started at Redmond 2 years prior as an internal product (2010) at Microsoft. The project is open-source, hosted in GitHub, and under Apache2 license.

Many Microsoft products use TS like Teams, Visual Studio Team Services (VSTS), Office online version, VS Code etc. Google has been using TS since Angular 2. Slack has migrated their JS codebase to TS. And many other companies use TS.

TS was created by Andres Hejlsberg, and currently manages it as well. He is also the creator of Turbo Pascal, Delphi, C#

TS is all about having a strongly-typed language at design time, but the code produced doesn’t contain any type. The type gets erased because TS removes types, interfaces, aliases etc. at transpilation and ends up with a common JS file, which means the final design must not rely on the type dynamically at runtime. We must remember that type and interface are not available at runtime.

# **Why to use TypeScript?**

The following topics are covered here –

* TS is fast
* Transpiling allows us to generate ES
  + TS lets us use the libraries and frameworks that we already know as a client-side developer
* TS uses NPM
* TS brings static typing
* TS is easier to refactor due to static typing
* TS is easier to maintain
* TS has a mechanism to define an existing JS library to have good support
* TS has excellent intellisense
  + TS reduces the number of unit tests
* TS mitigates potential pitfalls that can only be found at runtime
* TS is a low risk to take

TypeScript provides many advantages for client-side developers. In this section, we will see many reasons we should use TypeScript.

TypeScript is not a completely new language. It is easier to pick up than some other alternatives since we can jump in with a JavaScript background and learn how to use the enhancement that TypeScript offers.

TS is fast

Even if it has a compilation phase called “transpile,” it scales well with large codebases. We do not need to transpile every TypeScript file, since we can transpile a subset, like a file that has changed, or a directory.

Being fast is crucial to the development flow. JavaScript has the advantage of being a runtime language and having a middle ground which is not a burden; it reduces the friction for people who want fast results in their browser. Similarly, automatic build on files change by a third party is available. The combination of TypeScript and other tools transform the experience into a quasi-seamless flow.

Transpiling allows to generate ECMAScript

TypeScript lets us specify which version of JS is desired. This degree of detail means that we can generate JavaScript compatible with a very old browser or a new one, or with a feature that is planned to be available but not yet there. The output is a different JavaScript, depending on which target version. This feature allows us to use modern TypeScript syntax; for example, async is not fully supported by all browsers today, but TypeScript can target a version of ECMAScript that doesn’t support it.

TypeScript will remain the same since it borrows the ECMAScript standard syntax and the produced JavaScript will be different depending on the target. An old target version will provide polyfill which is less performant but will still produce the same behavior for the user.

Targeting a newer version leverages the native browser support of features, which will be performant, and will also produce a clean JavaScript code. In the example of async, it would use a *promise* approach with an older version of ECMAScript that doesn’t support it natively, but use the async syntax directly with the newest version.

TS allows usage of well-known libraries and frameworks

We can use JQuery, MomentJS, BootStrapJS, React, etc. There is no constraint. The interoperability is a huge win because we can transpose existing expertise without learning equivalences. Aside from lowering the barrier to entry, it makes TypeScript have the same ecosystem that JavaScript owns.

TS uses NPM

NPM gives access to millions of available libraries and has been tested for many years, and these libraries are accessible with a known and appreciated mechanism. Avoiding a custom tool to access libraries simplifies the jump to TypeScript by not having to learn a new language and a new tool. The same known NPM commands work with TypeScript to access definition files as well.

TS brings static typing

To not be strict remains an available option. It’s a wise choice when a hybrid model may be the only viable option.

When starting a new project, it is recommended to enable the strict mode which will enforce type. Nevertheless, having TypeScript letting us progressively add type is a nice touch for people who are just starting, to be able to bring TypeScript into an existing JavaScript project.

The hybrid option can introduce static type slowly without having to halt existing development because of a huge migration. We can go at our pace. In fact, TypeScript can run against existing JavaScript code and even provide inference and a minimum set of validation in the boundary of what is realizable.

TS is easier to refactor due to static typing

If we rename a member, it can be found everywhere it is used and hence be renamed everywhere.

Changing a type will highlight incompatibility straight in the IDE we use or during compilation. This is true with many features like changing if a member is optional or not, or adding, removing or modifying parameters of a function.

Navigation inside TypeScript code is a breeze since we can navigate to references of functions or members since the IDE can link usage of specific types and their content.

TS is easier to maintain

This is because reading the code is easier than JavaScript. For example, an object that initializes with an option would not explicitly define all the potential options in JavaScript.

Normally, if we are new to a piece of code that uses a parameter that is an object, we need to look inside the function and follow it as much as we can. We could also look at the unit tests or the documentation if they are available.

In all cases, this is time-consuming. With TypeScript, we can look at the type, click on it, and see the definition. It’s quick, self-documented, and provides insight without digging.

TS has mechanism to define existing JS library to have good support

It’s called a “definition file” which we will see in this course. The definition file is optional, but when provided to JavaScript library code, it brings Intellisense as well as type, to untyped code.

This increases the productivity of developers by reducing potential typos and bringing documentation to their fingertips. This also reduces losing focus by switching to external documentation.

TS has excellent intellisense

As mentioned, existing JavaScript libraries can get TypeScript typing which allows them to have code-completion capability. Meanwhile, TypeScript is fully Intellisense-supported by default, which means that every type defined can be used with an experience that provides type members, function arguments, types and return, etc.

TS reduces number of unit tests

Checking for types (structures) or expected members on the object, or a type that passed by parameter for undefined and null value are all cases that TypeScript checks on the compilation with static types.

Less code means less code to maintain. The unit tests only focus on meaningful things like logic or actual algorithms.

TS mitigates potential pitfalls that can be found during runtime only

JavaScript has many quirks and TypeScript mitigates potential pitfalls that could only be found at runtime by exposing them at development time.

TypeScript lets us fine-tune the rigidity of how it harnesses quirks for us. For example, the arithmetic with JavaScript is pretty loose. Also, specific values can be interpreted during value comparison.

TS is a low risk to take

Finally, TypeScript is a low risk to take since it produces human-readable JavaScript which is like an exit door wide open to start and stop in the future.

It’s low risk since it is open-source, which means that in the event that Microsoft stops maintaining the language, anyone can jump on it. That being said, Microsoft, Google, and other big corporations have invested millions in TypeScript, which should also be a good indicator for smaller companies.

Finally, the risk is low because the learning curve is gentle for a JavaScript developer.

# **Understanding benefits of TypeScript with React**

The following topics are discussed here –

* Using CodeSandbox
* Consuming & refactoring a component prop without TS
* Consuming & refactoring a component prop with TS
* Wrap up

Using CodeSandbox

CodeSandbox is a browser-based IDE that supports React and TypeScript apps. When you create a project in CodeSandbox, the basic configuration is automatically done, so we can start writing code straight away. It’s an excellent tool for learning React with TypeScript!

We are going to use CodeSandbox in some of the lessons in this course to try out different aspects of React with TypeScript.

The following link takes us to CodeSandbox – <https://codesandbox.io/>

Consuming and refactoring a component prop without TS

If a child component expects some props and we don’t provide it in parent component while calling it, then we will not see the issue/error until we run the code. Also, what is the type of props needed (string, number or object? If object then what are the property names?), IDE provides no intellisense for it.

And if we change a prop name in child component, the references in the parent component doesn’t gets changed automatically and hence we will miss the error until the code is executed.

Consuming and refactoring a component prop with TS

The problems mentioned above are highlighted and changed automatically while using TS with React.

Wrap up

TypeScript has many benefits when using it to develop React apps –

* Sophisticated type checking
* Accurate IntelliSense
* Accurate code refactoring
* Accurate code navigation

These benefits increase our productivity.