Why Use TypeScript?

In this lesson, we will see why one should conisder using TypeScript in the first place.
Let's begin!

WE'LL COVER THE FOLLOWING

- **TypeScript is fast**
- **Transpiling allows you to generate ECMAScript...**
- **TypeScript lets you use the libraries and frameworks you already know as a client-side developer**
- **TypeScript uses a popular tool called NPM**
- **TypeScript brings static typing**
- **TypeScript is easier to refactor due to static typing**
- **TypeScript is easier to maintain**
- **TypeScript has a mechanism to define an existing JavaScript library to have good support**
- **TypeScript has excellent Intellisense**
- **TypeScript reduces the number of unit tests**
- **TypeScript mitigates potential pitfalls that could only be found at runtime**
- **TypeScript is a low risk to take**

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

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

TypeScript is fast

Even if it has a compilation phase called “transpile,” it scales well with large codebases. You do not need to transpile every TypeScript file, since you 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 with a habit of having 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 you to generate ECMAScript...

...and TypeScript lets you specify which version is desired. The degree of detail means that you 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. This feature allows you to use modern TypeScript syntax; for example, `async` is not fully supported by all browsers today and targeting 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 and will be performant and will also produce a clean JavaScript. In the example of `async`, it would use a Promise approach with an older version of ECMAScript that doesn’t support it natively but would use the `async` syntax directly with the newest version.

directly with the newest version.

TypeScript lets you use the libraries and frameworks you already know as a client-side developer

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

TypeScript uses a popular tool called NPM

NPM gives access to millions of libraries available and has been tested for many years, and they 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.

TypeScript brings static typing

To not be strict remains an available option. It's worth a wise choice at a moment when a hybrid model may be the only viable option. It is recommended when starting a new project to enable the strict mode which will enforce type. Nevertheless, having TypeScript letting you 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. You can go at your 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.

TypeScript is easier to refactor due to static typing

If you rename a member, it one can be found everywhere it is used and hence rename every place. Changing a type will highlight incompatibility straight in the IDE you 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 you can navigate to references of functions or members since IDE can link

usage of specific types and their content.

TypeScript 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 you are new to the piece of code using a parameter that is an object, you need to look up inside the function and follow as much as you can, which can be deep, what is used inside the object. You could also look at the unit tests, or the documentation if they are available. In all cases, this is time-consuming. With TypeScript, you can look at the type, click on it and see the definition. It's quick, self-documented, and provides insight without digging.

TypeScript has a mechanism to define an existing JavaScript library to have good support

It's called "definition file" which we will see in this book. 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 reduces losing focus by switching to external documentation.

TypeScript has excellent Intellisense

As mentioned, existing JavaScript libraries can get TypeScript typing and allow them to be with 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.

TypeScript reduces the 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.

TypeScript mitigates potential pitfalls that could only

be found at runtime

JavaScript has many quirks, and TypeScript mitigates potential pitfalls that could only be found at runtime by exposing them at development time. TypeScript lets you fine-tune the rigidity of how it harnesses quirks for you. For example, the arithmetic with JavaScript is pretty loose; also, specific values can be interpreted during value comparison.

TypeScript is a low risk to take

Finally, TypeScript is a low risk to take since it produces a 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 could 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 a smaller company that is not a small project maintained by some part-time developers. Finally, the risk is low because the learning curve is gentle for a JavaScript developer.