TypeScript Philosophy

In this lesson, we will see TypeScript's philosophy and discuss the goal of the language.

TypeScript has well-documented design goals that inspire developers who use TypeScript to follow its overarching philosophy.

The **main point** is to identify the construct that will produce a runtime error.

Secondly, **TypeScript's goal** is to provide a structuring mechanism that can scale to a large code base. JavaScript on its own tends to be difficult to maintain with a large number of different developers and a substantive amount of code. This second goal embraces the first one which allows modification and being notified of potential issues as quickly as possible. Types self-document the code and conscribed potential values to specific variables.

The **third goal** is not to impose overhead on the produced JavaScript. This goal goes hand in hand with the **fourth goal**, which is to produce clean and recognizable JavaScript. TypeScript's final output is JavaScript code and depending on the version you specified you want the output to be, it will produce the most readable, efficient, and clean code. The result would be humanly usable, which is not only a great way to experiment with TypeScript

and be able to decide to stop and then to continue with JavaScript but is also a great way to debug.

The **fifth goal** is to have a language that can grow well in the future by making it composable and easy to navigate.

The **sixth goal**, in the spirit of being future-compatible, is that TypeScript aims to be aligned with ECMAScript, not only with the current version but also with the future versions.

TypeScript doesn’t try to supersede JavaScript, but being a good citizen, and never overrides existing behavior. Preserving runtime behavior makes TypeScript compatible with all existing JavaScript code. It is also a great tool for an experimenting JavaScript developer who can start using TypeScript with the libraries and functions that they already know. The eighth goal is to avoid adding expression-level syntax, which again preserves a syntax close to JavaScript without making it a completely new language.

TypeScript brings types, which means it needs to have a structured system (for example, interfaces, types, classes). However, the goal is to have this layer removed at runtime, again, ensuring an output completely compatible with ECMAScript.

While Microsoft worked hard to not be a walled garden behind a specific operating system, or browser, or any other constraint like IDE, TypeScript works on both Mac and Windows, and the output works in all browsers since it always follows the ECMAScript standard. It is also compatible with many development tools and doesn’t favor any particular ecosystem. TypeScript offers a tool that allows an external IDE to communicate with it, enabling great extensibility for a third party.

Finally, the last **design goal** is to be backward-compatible with other Microsoft products. Caring about the past is a huge win and a paramount gain for a long-term project. What you are coding today will work in the future, and it is part of the core concept of TypeScript. However, backward compatibility has been true since version 1, and the TypeScript team works with the principle of not causing any substantial breaking change.

with the principle of not causing any substantial breaking change.

It’s important to understand that TypeScript is not trying to create a perfect language on top of a flawed language that is JavaScript. It uses existing JavaScript behaviors and works with them. TypeScript is always swaying between being correct and being productive. The balance is important to keep developers’ productivity high. TypeScript is not trying to be innovative or bring a new way of thinking about how to develop. Instead, it leverages common patterns known by many languages while keeping with the flexibility of JavaScript.