"TypeScript is like JavaScript but with no surprises." This is how developers often describe the open-source programming language introduced in 2012 by Microsoft. As for 2020, the close relative of JS is among the top ten most wanted programming languages, according to GitHub. In this article, we'll explore what fuels the popularity of TypeScript and the factors that make it less attractive in the eyes of engineers and businesses.

What is TypeScript? TypeScript vs JavaScript

Technically, TypeScript is a variant of JavaScript, the number one language for many years in a row. First rolled out in 1995, JS brought interactivity to previously lifeless websites and completely changed the front-end development landscape. Currently, 95 percent of websites take advantage of JavaScript on the client side, with many of them running JS on the back end as well. It is also used for mobile application development.

Popular as it is, JS doesn't come without drawbacks. The major concern is that this language wasn't initially meant for building enterprise-level applications. And that's where TypeScript steps in. Kin to JS in terms of syntax and semantics, it brings a number of additional tools that increase productivity and facilitate the creation of large codebases.

For this reason, Google chose TS to build Angular, targeted at large-scale applications, and made it the primary language for developing Angular apps. Microsoft's solution is also compatible with all JS libraries and frameworks and can be used to create software with React, Vue, or any other technology within the JS ecosystem. This versatility along with great tooling makes TS one of the most loved languages, according to the 2019 Developer Survey by Stack Overflow.

The 2019 list of most loved languages. Source: Stackoverflow

Because browsers aren't currently able to read TypeScript code, it must be converted to plain JavaScript before it's released. After automated *transpiling* — the process of translating one high-level language into another — the code can be executed in any browser or server equipped with *JavaScript engines*. The latter, in turn, compiles JS into machine-level code.

There are two quick ways to get started with TS, suggested by its official website:

- via NPM — Node.js package manager or
- by installing plugins for Visual Studio, a full-fledged IDE from Microsoft.

The following tutorials will help you dive deeper into TypeScript and learn how to use it.

Overview — descriptions of all existing TS releases (up to the 3.7 version)

Migrating from JavaScript — a document teaching you how to convert JS files to TS files

React & Webpack — s short introduction to teaming up TS with React and

webpack
Handbook — tutorials covering all essential concepts of TS, including types, variable declarations, classes, functions, generic, enums, etc.
Declaration files — a comprehensive guide to writing a high-quality TypeScript Declaration File
Project Configuration — manuals that give answers to questions about compiling and integrating with build tools like Babel, Browserify, Grunt, and others
With all the above in mind, let's inspect the advantages offered by TypeScript in more detail.

TypeScript pros: what makes TypeScript a good fit for large projects

TypeScript inherits major pros of JavaScript, but also offers additional benefits coming from static typing and other concepts specific to TS. They prove especially useful when it comes to large-sized codebase and distributed teams working on the same project.

Aspects of TypeScript most loved by developers. Source: The State of Javascript, 2018

Optional static typing

JavaScript is a **dynamically typed** language which means that the types are checked, and datatype errors are spotted only at the runtime. Runtime type checking is not, per se, a disadvantage: It offers more flexibility, enabling program components to adapt and change on the fly. But the larger the project and the team, the more undefined variables are added and the more potential mistakes the code amasses.
TypeScript introduces optional strong **static typing**: Once declared, a variable doesn't change its type and can take only certain values. The compiler alerts developers to type-related mistakes, so they have no opportunity to hit the production phase. This results in less error-prone code and better performance during execution.
But static typing is not only about catching bugs. It also gives the code more structure, makes it self-documenting and more readable, speeds up debugging and refactoring. All told increases productivity across a large team.
It's important to note that TS doesn't force declaring types everywhere. Developers are free to change the level of type strictness in different parts of the project. This approach distinguishes TS from other statically typed languages and allows you to find the right balance between flexibility and correctness.

Early spotted bugs

Researchers found that TypeScript detects 15 percent of common bugs at the compile stage. Far from a 100 percent result, this amount is still significant enough to save developers time and let them focus on correcting mistakes in the logic — rather than catching common bugs. Pushing the code through a compiler also decreases the volume of quality assurance and testing activities.

Predictability

With TypeScript, everything stays the way it was initially defined. If a variable is declared as a string, it will always be a string and won't turn into a Boolean. This enhances the likelihood of functions working the way initially intended.

Readability

Due to adding strict types and other elements that make the code more self-expressive (so-called *syntactic sugar*), you can see the design intent of developers who originally wrote the code. It's especially important for distributed teams working on the same project. A code that speaks for itself can offset the lack of direct communication between team members.

Rich IDE support

Information about types makes editors and Integrated development environments (IDE) much more helpful. They can offer features like code navigation and autocompletion, providing accurate suggestions. You also get feedback while typing: The editor flags errors, including type-related as soon as they occur. All this helps you write maintainable code and results in a significant productivity boost.

With Microsoft Visual Studio as the most popular and natural environment for running TypeScript, it is also supported by many other EDIs including

- WebStorm, the intelligent JavaScript IDE;
- Eclipse, an integrated IDE offering a plugin for TS development;
- Visual Studio Code, a lightweight cross-platform editor by Microsoft;
- Atom, a cross-platform text editor; and
- CATS, an open source TypeScript development environment.

Fast refactoring

Refactoring or updating the app without changing its behavior is needed to keep the codebase robust and maintainable. TypeScript makes this important process less painful. With IDEs knowing much about your code, you are equipped with navigation tools like "find all references" or "go to definition." Besides, a lot of mistakes are spotted automatically. For example, if you rename a function and later forget to change the name somewhere, TS will alert you to the issue. This simplifies and accelerates refactoring, which is especially beneficial when you deal with large portions of the codebase.

Power of OOP

TypeScript supports concepts from class-based object-oriented programming (OOP) like classes, interfaces, inheritance, and more. The OOP paradigm facilitates building a well-organized scalable code, and this advantage becomes more obvious as your project grows in size and complexity.

Cross-platform and cross-browser compatibility

Every device, platform, or browser that runs JavaScript works with TypeScript as well — after the compiler translates it into vanilla JS. Usually, IDEs and editors supporting TypeScript come with a built-in TS compiler (tsc), which can be invoked from the command line. TS allows for converting a part of the codebase or the whole app at once by adding a configuration file

called tsconfig.json to the proper root directory.

Huge talent pool

The number of JavaScript developers hits 11.4 million — almost twice as much as the number of engineers using Java or Python. Of course, not all of them know TypeScript. But since TS relies on the JS syntax and semantics, the learning curve for JS devs will be smooth. All in all, companies switching to TS will hardly face the talent shortage.

❓

The number of active software developers detected in 2019.
Source: Developer Economics

Support from the tech world

TypeScript is an open-source language with a steadily growing community. The very fact that it is loved by millions of developers and backed by Microsoft conveys confidence that the technology won't be abandoned, and you always can seek (and get!) assistance from the wide TypeScript community. Many popular software products — such as Asana and Slack — switched to TypeScript to manage and maintain their large codebases.

TypeScript cons: what problems it creates

There is nothing perfect in this world, and TypeScript is no exception. For the most part, its weaknesses originated from its own strengths.

❓

Aspects of TypeScript most hated by developers. Source: The State of Javascript, 2018

Not true static typing

Developers mastering C#, C++ or Java often argue that TypeScript is not a true statically typing language. Apart from the fact that this feature is optional for TS, eventually it is transpiled into untyped JavaScript, so there is always the risk of weird type conversions at runtime.

One more JavaScript to learn

In the previous section we admitted that it would be relatively easy for JS developers to learn TS. However, despite this similarity, they still need to invest time and effort in learning types and some TS-specific constructs. So, if your company plans to move to TypeScript and have JS devs with no previous experience with TS, they won't hit 100% productivity immediately.

Bloated code

Enhanced code readability brought by previously mentioned syntactic sugar and type annotations has its downsides. First, you have to write more code and this has the potential of slowing down the development process. Second, additional annotations make TS files larger than those written in plain JS. The difference in the number of code lines depends on numerous factors. The good thing is that all these extra features will disappear after being transpiled. In the end, the browser will execute the plain JavaScript.

Adding extra step — transpiling

As we wrote before, browsers can't interpret the TypeScript code, so you need

to transpile it to JavaScript before running. However, this process is highly automated and doesn't require a lot of additional time. In total, the downside of this step is far less significant than its benefits: The compiler spots flaws and flags them before they can cause any damage.

It's getting better all the time

TypeScript continues to evolve, with its 3.8 version coming soon. At the time of writing, the release candidate was already available, and the production release will come with the same set of upgrades plus some critical bug fixes. So, what major improvements does the fresh version bring?

New syntax for type-only imports and exports. This update is supposed to give developers more control over how the compiler imports and elides (omits) some elements.

Support for ECMAScript private fields. ECMAScript or ES for short is a specification that defines rules and guidelines for scripting languages, and it lies at the core of JS. In 2015, ES introduced the concept of a class to JavaScript — roughly, it's a template that defines how the function of a certain type (or class) should behave.

By default, ES classes are public, which means they can be accessed and modified outside the class. The implementation of private fields — data slots, available from inside the class only — enables programmers to create a kind of protective cover, secluding internal details and preventing unwelcome interventions. This feature is already supported in Node.js 12, Chrome 74, and Babel.

"Fast and Loose" incremental checking. The new compiling option reveals if the file needs rechecking or rebuilding, based on changes made to it. The update is supposed to reduce build time and thus favor large projects.

Other minor additions and changes will serve the same global idea — to bring productivity, reliability, and predictability to a somewhat chaotic JavaScript world.