## Transpilation and Transpilers

## Transpilation means taking source code written in a high-level programming language and convert it into code written in another high-level programming language. We need special programs (softwares) to do that and these programs are called transpilers (or transcompilers).

[Typescript](https://www.typescriptlang.org/) (TS) is a JavaScript superset (sometimes called syntactic sugar) that supports types, a feature that Javascript does not have. There are other languages that do the same thing as Typescript, Coffescript or Elm for example, but TS is the most popular as of today. Typescript needs to be converted into Javascript and the transpiler used for that is called tsc.

## Difference Between Babel And WebPack

Babel and Webpack are apple and banana. One never compares them side-by-side as they solve different problems. This post explains their different concepts.

**Babel**

Babel is simply a translator, who translates your 'fancy' (ES6+) JS code into 'not-so-fancy' (ES5) ones that browser (front-end) or Node.js (back-end) understands.

Why we speak fancier than browser and Node.js? Because we can't wait to use the latest and greatest, even before they are officially supported.

Below is a fancy code that most developers write today. Despite of how fancy it is, our browser / Node.js has no idea what it's talking about. *(Note: Some Node.js higher versions have ES6 support now.)*

// ES6 syntax

import moment from 'moment';

export default () => moment().format("YYYY Do MM");

And this is why we need Babel to translate above into the equivalent not-so-fancy code below, that our browser / Node.js actually understands.

// ES5 syntax

const moment = require('moment')

function getDateString() {

const date = moment();

return date.format("YYYY Do MM");

}

exports.default = getDateString;

That's why Babel is sometimes called a *transpiler*.

It's worth noting that Babel is commonly used for both front- and back-end. Why do I mention this? Because Webpack is front-end only (in most cases).

**Webpack**

If Babel is a translator for JS, you can think of Webpack as a mega-multi-translator that works with all kinds of languages (or assets). For example, Webpack often runs Babel as one of its jobs. Another example, Webpack can collect all your inline CSS styles in your Javascript files and bundle them into one.

**Why do we need such a monster for front-end, but not back-end?**

Because front-end has many kinds of assets such as CSS, SASS, images, fonts and is way more complex and dynamic than back-end which only has JS. And in the end of day we need to somehow package all variety of assets into a small file that our users' browser can download at page load time. This is also known as minify and uglify. You see, back-end has none of the above requirement.

Another important reason is that front-end doesn't work with modules (again, in most cases). Modules are built-in features of Node.js, not browsers. Nowadays developers are so used to npm install, import and export JS modules in front-end, as it allows us to better organize code and share packages. **But in reality they are only syntactic sugars, and it's Webpack's job to figure out all the dependencies among all the modules that we use in the code, and compile them into one big chunk of JS code that the browser actually understands.**

When do we use Webpack in back-end? A good use case is to support SSR (Server-Side Rendering).

**To sum up,**

* Backend: we use Babel so that we can use the fanciest JS syntax (ES6/7) with Node.js.
* Frontend: we use Webpack (which uses Babel and other things) to compile JS code and many other assets into a few small bundle files that our users can download when they first load our webpage. For example, [create-react-app uses Webpack and Babel](https://github.com/facebook/create-react-app" \l "get-started-immediately) when creating your app.