# New code for old browsers: Babel

One of the coolest things that, in my opinion, has happened in front end development is the concept of transpilation. You can transpile third party languages like Dart or TypeScript or you can take current ES2015+ (currently ES2019 and soon to be ES2020) and make it work in older browsers that don't support the features of the latest ECMAScript features.

In this post we'll worry about converting ES2020 to ES5 using Babel

## Setting up Babel

Babel requires a build system. For this post I've chosen Gulp which is what I normally use.

I assume you already have a `package.json` in your project directory. If you don't run `npm init --yes` to get a quick-start version, you can edit it later.

Run the following command to install Gulp, Babel and related plugins

```
npm i -D gulp \
gulp-load-plugins \
gulp-sourcemaps \
@babel/core \
@babel/preset-env \
gulp-babel
```

Create a `gulpfile.js` file and copy the following code into it.

The code takes all the files and does the following:

- Processes it through Babel using the Babel env preset
- Create a sourcemap for each script that gets transpiled
- Put the resulting scripts and sourcemaps in `src/js`

```
// Require Gulp first
const gulp = require('gulp');
// Lazy load'gulp'ns
// Lazy load pluginsulp-load-plugins')({
  lazy: true,
});

function runBabel() {
  return gulp.src('scripts/*.js')
    .pipe($.sourcemaps.init())
    .pipe($.babel({
      presets: ['@babel/en'scripts/*.js'    .pipe($.sourcemaps.write('.'))
    .pipe(gulp.dest('src/js/'))
}

exports.babel = runBabel;
exports.default = runBabel;
```

The idea is that we'll write the code once and then let Babel convert it to code that will run in older browsers.

## Preset-env

For a while, Babel forced people to transpile all features, whether a browser supported modern JavaScript or not. The Babel team introduced preset-env as an alternative.

The idea is that, using a list of browsers you provide and a configuration for what browsers to transpile for. Babel will only transpiles those parts of your code that your targt browsers don't support, making the resulting code slimmer.

## Preset-modules

After I originally finished writing this, the Babel team introduced env-preset-modules which provides a better way to generate optimized builds for browsers that support modules.

# ECMAScript Modules

As Jake Archibald points out one of the best ways to provide code for both modern browsers (that support all the latest and greatest features) and older browsers without having to write unnecessary code or code that will only run on one version or the other.

Because we used babel to target browsers for transpilation we can be confident that newer browsers will work with our module script and the nomodule version can rely on older technologies.

The HTML looks like this.

```
<script type="module" src="module.mjs"></script>
<script nomodule src="fallbac</script>er></script>
</script>
```

`type="module"` indicates that the associated script will be treated as an ES2015 module and will be ignored by browsers that don't support them.

`nomodule` will run the code as a traditional script. Browsers that support modules also know to ignore any script tag that has the `nomodule` attribute.