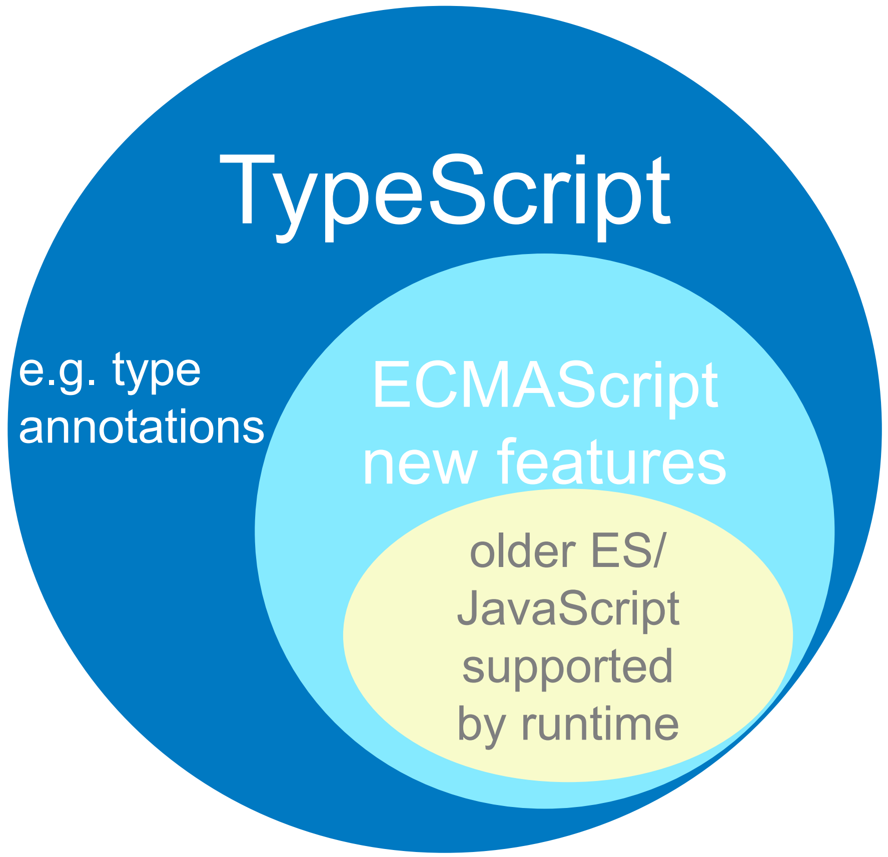
TypeScript usage explained

How TypeScript is related to JavaScript/ECMAScript and what steps are needed

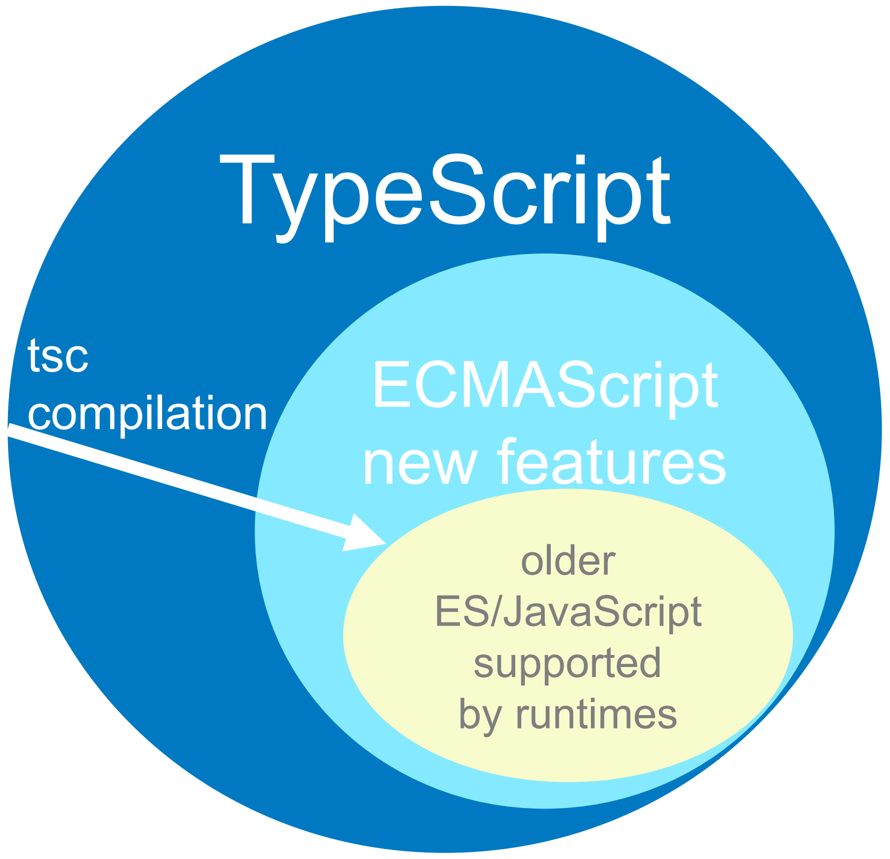
10.2.2025

Juhani Välimäki

Principles as a picture



tsc compilation as the image



Installing TypeScript to your computer

- Assuming you have Node.js already installed to your computer, continue by installing TypeScript, e.g. this might be the correct command:

```
> npm i typescript @types/node
```

Starting to turn JavaScript project to TypeScript

- … or starting TypeScript project from zero (basically same steps!)

Initialize the (JS or new) project as TS project

```
> tsc --init      or  npx tsc --init
```

- Creates the tsconfig.json file to the project root. Example console output below from 2025:

```
Created a new tsconfig.json with:
  target: es2016
  module: commonjs
  strict: true
  esModuleInterop: true
  skipLibCheck: true
  forceConsistentCasingInFileNames: true
```

tsconfig.json file:

- E.g.
- source folder (for TS files)
- Output / dist folder (for compiled JS files)
- How strict TypeScript should be required/followed?
- What version of ES should the output be? ES 2020?
- The module mechanism to use? e.g.
  - ES6 (2015):   export / import   instead of
  - CommonJS: module.exports / require

tsconfig.json file (An example from 2024)

```
{
  "compilerOptions": {
    "target": "es2022",
    "module": "nodenext",
    "outDir": "dist",
    "strict": true
  },
  "include": [
    "src"
  ]
}
```

Package.json npm run/build etc scripts changed

- …scripts won’t be using JS tools anymore, but use TS tools like tsc compiler
- Or e.g. tsc-watch could look for changing .ts files and compile them automatically to .js files

Renaming source files from .js to .ts

- … and start using TypeScript / ECMAScript features (e.g. according to the list on course materials)

You can use TypeScript and modern ECMAScript

- … as TypeScript compiler tsc still makes compilation to older ECMAScript understood by the runtime(s)
- **ECMAScript**, some tricky features: https://github.com/valju/JS_ES_Features/blob/master/ES_advanced/ES_advanced_or_tricky_features.md
- **TypeScript**, some useful features: https://github.com/valju/JS_ES_Features/blob/master/TS_basics/TS_in_a_fullstack_project.md
- More?
  - Look at the course pages
  - Search web for TypeScript and ECMAScript **cheat sheets**

E.g. create your own complex datatypes with ‘interface’

- So that those types can be used in code as types
- ‘interface’ is better option than ‘type’ as interface can be extended to create more subtypes

Take into use new versions of libraries, and their @-type modules

- Then your code that uses and calls the library code will be also valid TypeScript,
- as the types imported can be used for checking the datatypes compilation time (and in e.g. VS Code source code writing/saving time)

Biome checker that forces to 1. use TS features and 2. to use them correctly

This seems to be correct way to run **biome** in Windows computers, **crlf** : (Linux & Mac: change crlf to **cr**)

1. First you might need to fix and **rewrite formatting** of files for your enviroment (indentation and line-endings)
2. Second you can just **check** (for **other problems**/hints than formatting)
3. Third would also apply = **write those changes** to the files

```
npx @biomejs/biome format --write --max-diagnostics=200 --line-ending=crlf ./src
```

```
npx @biomejs/biome check  --max-diagnostics=200 --line-ending=crlf ./src
```

```
npx @biomejs/biome check --apply --max-diagnostics=200 --line-ending=crlf ./src
```

# 200 here means it will each time only notice/fix first 200 probs!

biome.json example configuration files (March 2024)

A React Material UI frontend

```
{
  "formatter": {
    "indentStyle": "space"
  },

  "linter": {
    "enabled": true,
    "rules": {
      "correctness": {
        "useExhaustiveDependencies": "off"
      },
      "style": {
        "noUselessElse": "off"
      }
    }
  }
}
```

A Node/Express/Knex/MariaDB backend

```
{
  "files": {
    "ignore": ["dist/"]
  },
  "formatter": {
    "indentStyle": "space"
  },
  "javascript": {
    "formatter": {
      "quoteStyle": "single"
    }
  },
  "linter": {
    "enabled": true,
    "rules": {
      "style": {
        "noUselessElse": "off"
      }
    }
  }
}
```

(Probably not perfect configs, but fixed some needed issues, and worked for us. Consult biome documentation for more)

Install the TS versions of libraries, with their type definition modules

- Random command example:

```
> npm i --save express @types/node @types/react @types/react-dom @types/jest
```

Understand compilation-time vs run-time

- - 1. Compilation time: **tsc** (TypeScript compiler) **.ts** ⇒ **.js**
- - 2. Runtime: run the **.js**, e.g. with **node**, **nodemon**, **pm2** or so

- See also how all TS tools are in **devDependencies** in **package.json**, for development time steps and processes. Whereas JS tools and modules are in **dependencies** for the running time / production.

- (2. or use the **ts-node** for combining the compilation and running as a one, bit slower, step)