

TypeScript Intro

Let's get started with TypeScript!

1.

History

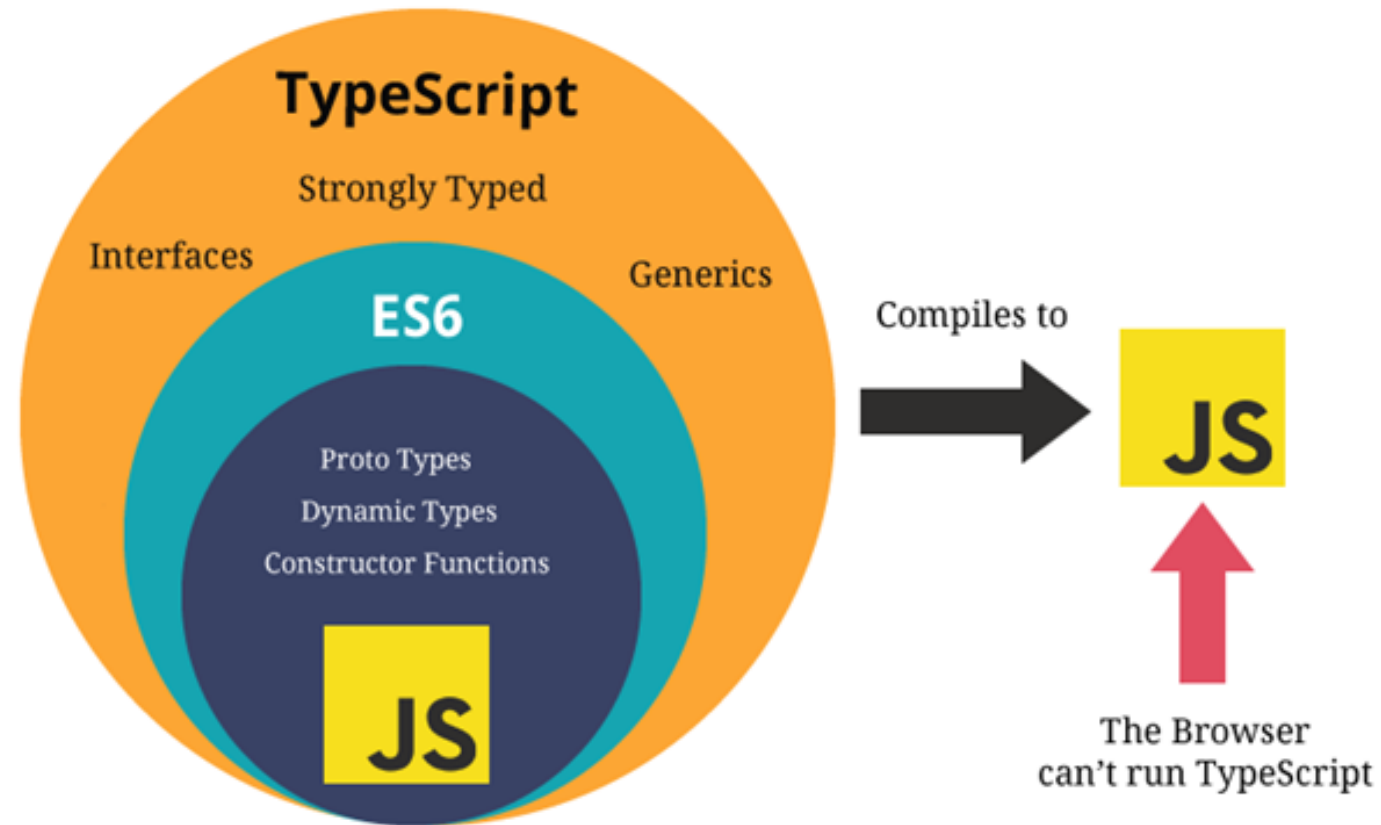
What is
TypeScript and
what is it
about?

Predictability

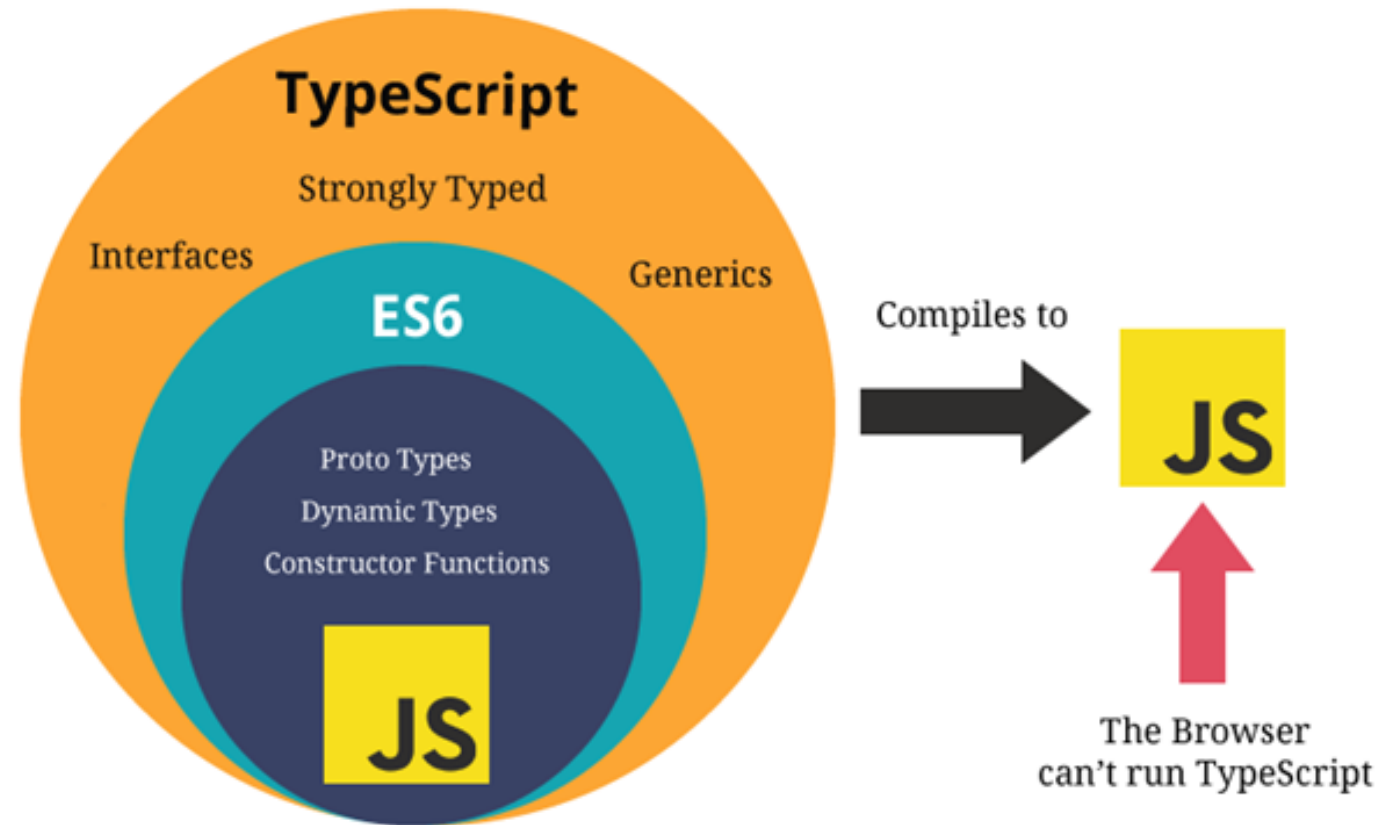
There were some attempts at making JS more **predictable**, however, Microsoft might have proposed the best solution to date.

Typescript, JavaScript's **superset**, saw the light of day for the first time **in 2012**. Its purpose was to **facilitate the development of large applications**.

Superset



Superset



*TypeScript is a **superset** of JavaScript, so it's a layer around JS with more methods.*

Why does TypeScript look partially like C#?

The most publicly recognizable name behind TypeScript is Microsoft Technical Fellow Anders Hejlsberg, **the father of C# and TurboPascal**. But Hejlsberg isn't the one who came up with the idea for TypeScript.

TypeScript is actually the product of a team of about 50 people, headed by Microsoft Technical Fellow Steve Lucco.



Some release history

- 0.8: 1 October 2012
- 1.4: 20 January 2015
 - union types, let and const declarations, template strings, type guards, type aliases
- 3.7: 5 November 2019
 - Optional Chaining, Nullish Coalescing
 - Nullish Coalescing arrives with ECMAScript2020 (June 2020)
- 4.4: 26 August 2021
 - Control Flow Analysis of Aliased Conditions and Discriminants, Symbol and Template String Pattern Index Signatures

Some release history

- 4.5: 17 November 2021
 - Type and Promise Improvements, Supporting lib from node_modules, Template String Types as Discriminants, and es2022 module
- 4.6: 28 February 2022
 - Constructors Before super(), Control Flow Analysis for Destructured Discriminated, Improved Unions, target es2022
- 4.8: 25 August 2022 (current 4.8.4)
 - Improved Intersection Reduction, Union Compatibility, and Narrowing Improved Inference for inferring Types in Template String Types --build, --watch, and --incremental Performance Improvements

Version 4.9

- 4.9: 15 November 2022
 - satisfies Operator
 - Unlisted Property Narrowing with the in Operator
 - Auto-Accessors
 - Checks for Equality on NaN
 - File-Watching with File System Events
 - Editor: Remove Unused Imports & Sort Imports
 - And some more minor changes...

Version 5.0

- 5.0: 16 March 2023
 - Decorators
 - const Type Parameters
 - All enums are Union enums
 - Exhaustive switch/case completions
 - Many more adoptions and corrections

Version 5.4

- 5.4: 6 March 2024
 - Preserved Narrowing in Closures
 - The NoInfer Utility Type
 - Object.groupBy and Map.groupBy
 - Support for require() calls in
 - --moduleResolution bundler and
 - --module preserve Checked Import Attributes and Assertions
- Quick Fix for Adding Missing Parameters
- Auto-Import Support for Subpath Imports

Version 5.5

- 5.5: 20 June 2024
 - Inferred Type Predicates
 - Control Flow Narrowing for Constant Indexed Accesses
 - The JSDoc @import Tag
 - Support for ECMAScript Set Methods
 - Isolated Declarations
 - The transpileDeclaration API



What is ECMA or ECMAScript?

ECMAScript (/ˈɛkməskript/) (or ES) is a **general-purpose programming language**, standardized by Ecma International according to the document ECMA-262.

It is a JavaScript standard meant to ensure web pages' interoperability across different browsers.

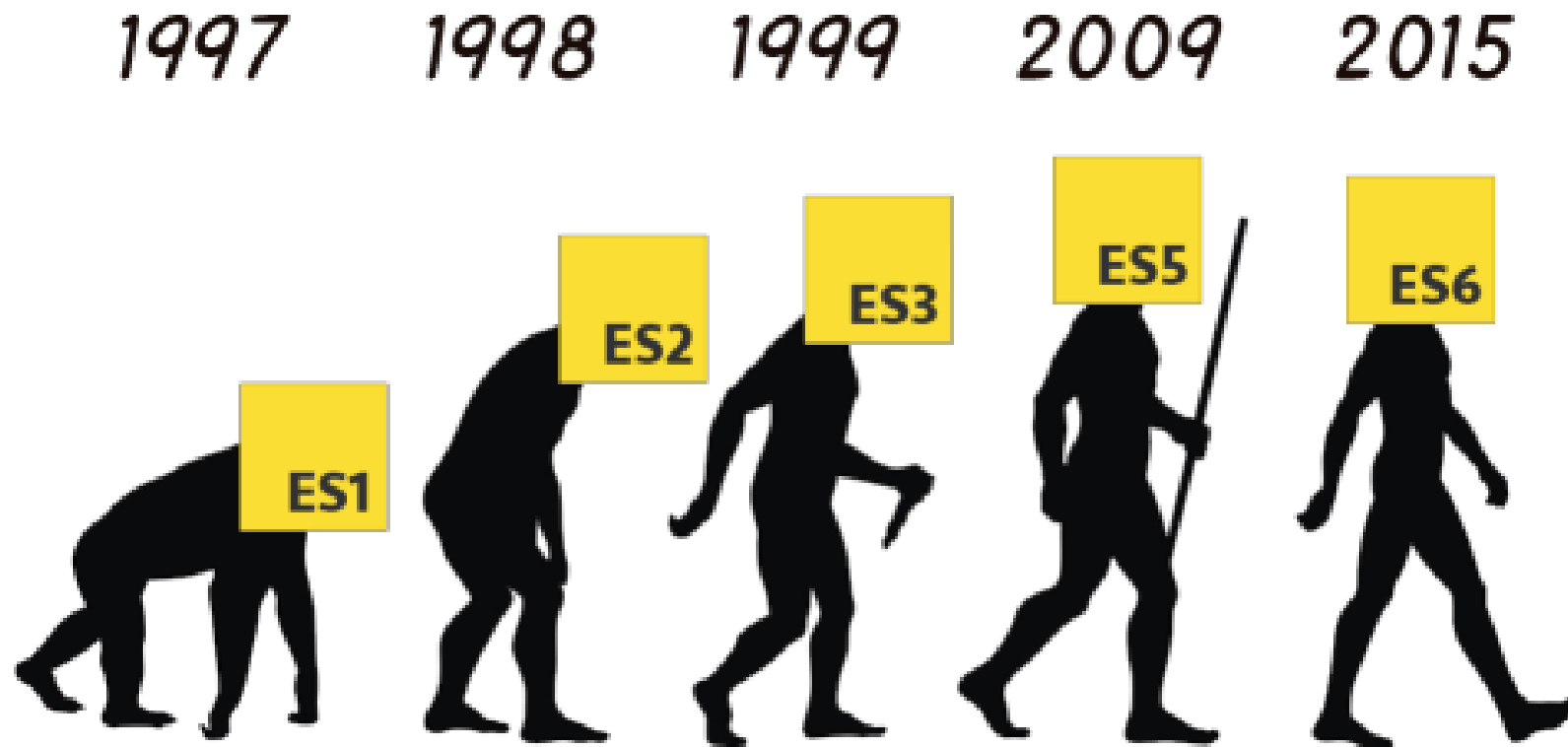
Which ECMAScript is currently in use?



Mostly in use is ES6, which is also named ECMAScript 2015 and is released since **June 2015**.

Notes: ECMA stands for "European Computer Manufacturers Association".

ES6 is already 6 years old!



This is another great point of **TypeScript** you can already "play" with features you might see in later ECMAScript versions or **not at all**.

ES6 and newer

- ECMAScript 2016 (ES7)
 - `Array.prototype.includes` and the exponentiation operator (`**`)
- ECMAScript 2017 (ES8)
- ECMAScript 2018 (ES9)
 - Support for spread in destructuring
- ECMAScript 2019 (ES10)
 - Support for `flat` & `flatMap` array methods
- ECMAScript 2021 (ES12)
 - `export *` as namespace, Optional chaining operator, Nullish coalescing operator
- ECMAScript 2022 (ES13)
 - String: `replaceAll`, Promise: `any`, Logical OR, AND and Logical nullish assignment

Question to the audience

- Why does TypeScript tend to make your life easier?
- What is the relation of TypeScript and ECMAScript?

Solution

- Why does TypeScript tend to make your life easier?
 - Predictability and more
- What is the relation between TypeScript and ECMAScript?
 - TypeScript is a superset of ECMAScript and enhances it

2.

Improvement

How
TypeScript
improves
your code

Recap: Predictability

There were some attempts at making JS more **predictable**.

However, Microsoft might have proposed the best solution (to date).

Typescript, JavaScript's **superset**, saw the light of day for the first time in 2012.
Its purpose was to **facilitate the development of large applications**.

3 Reasons why you should choose TypeScript

3 Reasons why you should choose TypeScript

1. TypeScript is more reliable
 1. Helps during coding
 2. Easier to refactor
 3. You need to think more before acting

3 Reasons why you should choose TypeScript

1. TypeScript is more reliable
 1. Helps during coding
 2. Easier to refactor
 3. You need to think more before acting
2. TypeScript is more explicit
 1. TypeScript forces you to focus on how a system is built
 2. It gives you a clear look at how different parts interact with each other
 3. It forces you to keep the context in mind

3 Reasons why you should choose TypeScript

1. TypeScript is more reliable
 1. Helps during coding
 2. Easier to refactor
 3. You need to think more before acting
2. TypeScript is more explicit
 1. TypeScript forces you to focus on how a system is built
 2. It gives you a clear look at how different parts interact with each other
 3. It forces you to keep the context in mind
3. TypeScript and JavaScript are interchangeable
 1. You can use it more or less. However, you prefer

TypeScript vs JavaScript

TypeScript vs JavaScript

TypeScript

- TS is an object-oriented scripting language
- Dependent language (compiles to JavaScript)
- Compiled language cannot be directly executed in a browser
- It can be statically typed
- Better structured and concise
- Has a .ts extension
- A fair choice for complex projects

TypeScript vs JavaScript

TypeScript

- TS is an object-oriented scripting language
- Dependent language (compiles to JavaScript)
- Compiled language cannot be directly executed in a browser
- It can be statically typed
- Better structured and concise
- Has a .ts extension
- A fair choice for complex projects


JavaScript

- JS is an object-oriented scripting language
- Independent language (can be interpreted and executed)
- An interpreted language executed directly in a web browser
- Dynamically typed
- More flexible since you are not limited by the type of system
- It has a .js extension
- Good to work with small, simple projects

Install TypeScript

To start everything, we like to program a greeter and evolve it a bit.

To enable our tests, we will need to install TypeScript globally (if not already done):



```
1 npm install -g typescript
```

Note: -g installs globally

First example - JavaScript



```
1 function my_sum(a, b) {  
2   return a + b;  
3 }  
4  
5  
6 let a = 4;  
7 let b = "5";  
8  
9  
10 my_sum(a, b);
```

First example - TypeScript



```
1 function my_sum(a: number, b: number) {  
2     return a + b;  
3 }  
4  
5  
6 let a = 4;  
7 let b = "5";  
8  
9  
10 my_sum(a, b);
```

First example - Oh Snap!

First example - Oh Snap!

Argument of type 'string' is not assignable to parameter
of type 'number'.

First example - Oh Snap!

**Argument of type 'string' is not assignable to parameter
of type 'number'.**

But wait. Is that not cool?

Before running the code my IDE tells me that something is wrong.

First example

```
function my_sum(a: number, b: number) {  
    return a + b;  
}
```

```
let a = 4;  
let b = "5";
```

```
my_sum(a, b);
```

TS2345: Argument of type 'string' is not assignable to parameter of type 'number'.

[Change my_sum\(\) signature](#)

[More actions...](#)

```
function my_sum(  
    a: number,  
    b: number): number
```

first_example.ts

This example is very simple, but you can already see where this is going.

First example - Fixed

First example - Fixed

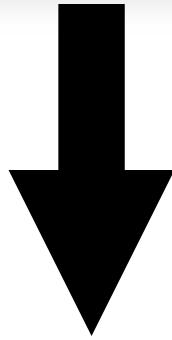


```
1 let b: number = "5" // Type '"5"' is not assignable to type 'number'.
```

First example - Fixed



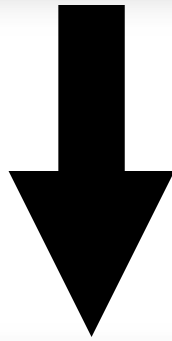
```
1 let b: number = "5" // Type '"5"' is not assignable to type 'number'.
```



First example - Fixed



```
1 let b: number = "5" // Type '"5"' is not assignable to type 'number'.
```



```
1 let b: number = 5 // Everything ok.
```

3.

Tooling

Use TypeScript

Helpers

We have already seen that TypeScript is a **compiled language**, therefore we need a way to compile TypeScript (TS-Files) efficiently to JavaScript.

There are two "helpers" which are very common to do this:

- TSC: The TypeScript Compiler
- TSConfig: The TypeScript Configuration

TypeScript Compiler

The TypeScript compiler is itself written in TypeScript and compiled into JavaScript. It is licensed under the **Apache License 2.0**.



TSConfig

A TSConfig file in a directory indicates that the directory is the root of a TypeScript or JavaScript project. The TSConfig file can be either a **tsconfig.json** or **jsconfig.json**, both have the same set of config variables.

A categorized overview of all compiler flags

- The root fields for letting TypeScript know what files are available
- The compilerOptions fields this is the majority of the document
- The watchOptions fields for tweaking the watch mode
- The typeAcquisition fields for tweaking how types are added to JavaScript projects

Notes: <https://www.typescriptlang.org/tsconfig>

TSConfig Example

```
1 {  
2   # Some typical compiler options  
3   "compilerOptions": {  
4     "target": "es2020",  
5     "moduleResolution": "node"  
6   },  
7   "watchOptions": {  
8     "watchFile": "useFsEvents",  
9     "watchDirectory": "useFsEvents",  
10    "excludeDirectories": ["**/node_modules", "_build"],  
11    "excludeFiles": ["build/fileWhichChangesOften.ts"]  
12  }  
13 }
```

TSConfig Example

```
1 {
2   # Some typical compiler options
3   "compilerOptions": {
4     "target": "es2020",
5     "moduleResolution": "node"
6   },
7   "watchOptions": {
8     "watchFile": "useFsEvents",
9     "watchDirectory": "useFsEvents",
10    "excludeDirectories": ["**/node_modules", "_build"],
11    "excludeFiles": ["build/fileWhichChangesOften.ts"]
12  }
13 }
```

TSConfig Example


```
1 {  
2   # Some typical compiler options  
3   "compilerOptions": {  
4     "target": "es2020",  
5     "moduleResolution": "node"  
6   },  
7   "watchOptions": {  
8     "watchFile": "useFsEvents",  
9     "watchDirectory": "useFsEvents",  
10    "excludeDirectories": ["**/node_modules", "_build"],  
11    "excludeFiles": ["build/fileWhichChangesOften.ts"]  
12  }  
13 }
```

TSConfig Example

```
1 {  
2   # Some typical compiler options  
3   "compilerOptions": {  
4     "target": "es2020",  
5     "moduleResolution": "node"  
6   },  
7   "watchOptions": {  
8     "watchFile": "useFsEvents",  
9     "watchDirectory": "useFsEvents",  
10    "excludeDirectories": ["**/node_modules", "_build"],  
11    "excludeFiles": ["build/fileWhichChangesOften.ts"]  
12  }  
13 }
```

TSConfig Example - Cont.

Question: Look at the following snippet, what could be the problem?




```
1 {  
2   # Some typical compiler options  
3   "compilerOptions": {  
4     "target": "es2020",  
5     "moduleResolution": "node"  
6   }  
7 }
```

TSConfig Example - Cont.

Target tells the TypeScript Compiler (TSC) against which target system it compiles.

Since not all browsers understand ES2020 right away, it is probably better to compile against ES2015 (ES6), which is still the best supported.



```
1 {  
2   # Some typical compiler options  
3   "compilerOptions": {  
4     "target": "es2015",  
5     "moduleResolution": "node"  
6   }  
7 }
```


Still a problem?

Well, since IE has changed a lot - but you will still need to have a look if your options are supported:

The Optional Chaining Operator (??)

The `?.` operator returns `undefined` if an object is `undefined` or `null` (instead of throwing an error).






Edge

Example

```
// Create an object:
const car = {type:"Fiat", model:"500", color:"white"};
// Ask for car name:
document.getElementById("demo").innerHTML = car?.name;
```

Try it Yourself »

The optional chaining operator is supported in all browsers since March 2020:

				
Chrome 80	Edge 80	Firefox 74	Safari 13.1	Opera 67
Feb 2020	Feb 2020	Mar 2020	Mar 2020	Mar 2020

Exercise - my_sum


Please try the following:




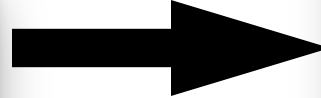
```
1 cd typescript/01_intro/exercises  
2 tsc my_sum.ts
```

Exercise - my_sum

Since this is a very easy example, the JavaScript file will be straightforward:




```
1 function my_sum(a: number, b: number) {  
2     return a + b;  
3 }  
4  
5 let a = 4;  
6 let b = 5;  
7  
8 my_sum(a, b);
```



```
1 function my_sum(a, b) {  
2     return a + b;  
3 }  
4 var a = 4;  
5 var b = 5;  
6 my_sum(a, b);
```



some cases



```
1 function my_sum(a, b) {  
2     return a + b;  
3 }  
4 let a = 4;  
5 let b = 5;  
6 my_sum(a, b);
```

Exercise - my_sum

But sometimes, even small scripts change a lot. Please try the following:



```
1 cd typescript/01_intro/exercises  
2 tsc my_sum.ts --target es5
```

and



```
1 cd typescript/01_intro/exercises  
2 tsc my_sum.ts --target es2020
```

To use the project-level tsconfig (which globs all .ts files) use the following command:



```
1 tsc --project tsconfig.json
```

Exercise - arrow es2020


How does an arrow function look with es2020:



```
1 const square = (a) => {  
2   return a * a;  
3 };
```

Exercise - arrow es5

And how with es5:



```
1 var square = function (a) {  
2   return a * a;  
3 };
```

Exercise - arrow conclusion

The change might seem small, but we can clearly see that the **Arrow-Function** is as a **feature not available to es5** and therefore the output is different.

Linting

Linting is a process by a linter program that analyzes source code in a particular programming language and flags potential problems like syntax errors, deviations from a prescribed coding style, or using of unsafe constructs.

Question: Do you have linting in place? What is your baseline?

Linting

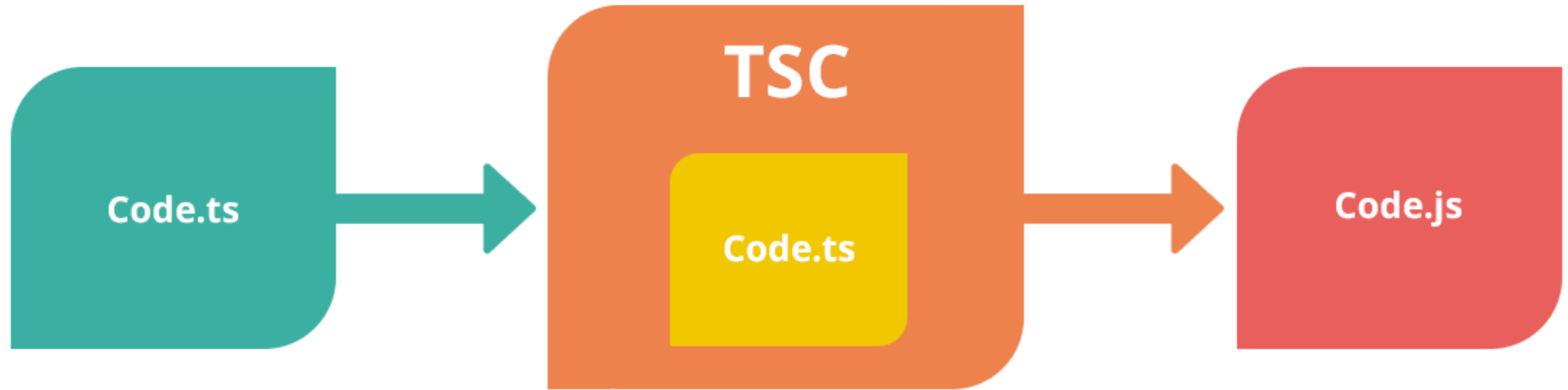
I will talk about linting later to not distract you too much right now. But if you like to you can already have a look:

<https://prettier.io/>

<https://eslint.org/>

What about TSLint? TSLint has been the recommended linter in the past **but now TSLint is deprecated** and [ESLint](#) is taking over its duties.

Conclusion



End

That was all for this chapter
