TypeScript Tutorial

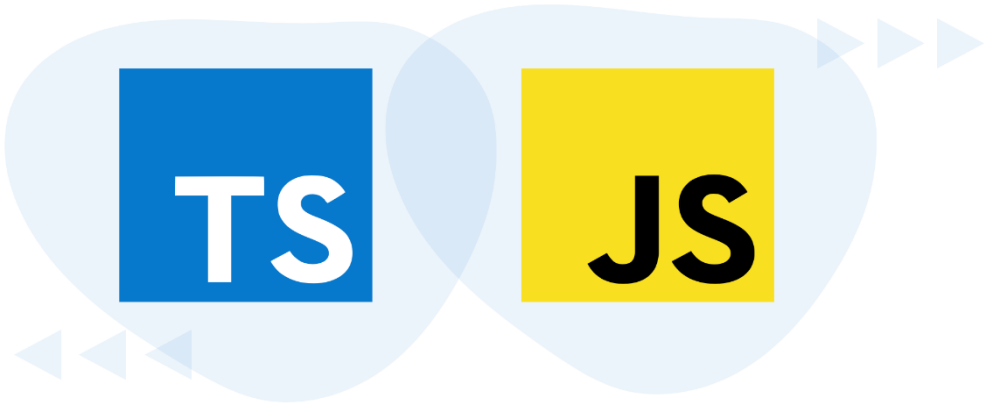
A step-by-step guide to learn TypeScript

What is Typescript?

Typescript is a typed superset of JavaScript and was created by Microsoft in 2012 to aid Javascript developers with large-scale applications. Typescript was designed to help as a structuring mechanism for large codebases because it helps avoid common errors that would slow you down. It makes it easier for teams to work on the same project, as the language allows for modification and produces readable code. If there are multiple developers working on one project, Typescript can prevent a lot of wasted time on debugging.

This language allows us to use our Javascript skills more effectively. In fact, after you compile your code, all the Typescript stuff goes away and produces clean, cross-platform safe Javascript code. On top of being interoperable, Typescript adds unique features, including static typing, interfaces, classes, and more.

TypeScript vs. JavaScript



Javascript is a dynamic scripting language used to make interactive web pages, so it's not designed for complex applications. Typescript, on the other hand, is a static scripting language that is a superset of Javascript, meaning that it is an extra layer on top of your JS code. Typescript was not designed to supersede or replace Javascript. In fact, it never overrides existing behavior. It takes the existing behaviors of Javascript to correct its limitations and leverage common issues with the language.

There are many differences between Typescript and Javascript. Here are just a few:

- TypesScript is an Object oriented programming language whereas JavaScript is a scripting language (with support for object oriented programming).
- TypeScript has static typing whereas JavaScript does not.
- TypeScript uses types and interfaces to describe how data is being used.
- TypeScript has interfaces which are a powerful way to define contracts within your code.
- TypeScript supports optional parameters for functions where JavaScript does not.

Why should you use Typescript?

There are many important benefits to using Typescript. Let's break them down.

Typescript catches mistakes in your JavaScript code earlier on. Typescript has the ability to catch bugs and errors before runtime, so you'll write reliable code and mitigates the pitfalls of JavaScript that are only found at runtime.

- **Transpiling allows you to generate ECMAScript**, and you can specify which version of JavaScript you prefer to use. This means that you can write code that is compatible with old browsers, all while using the newest tools.

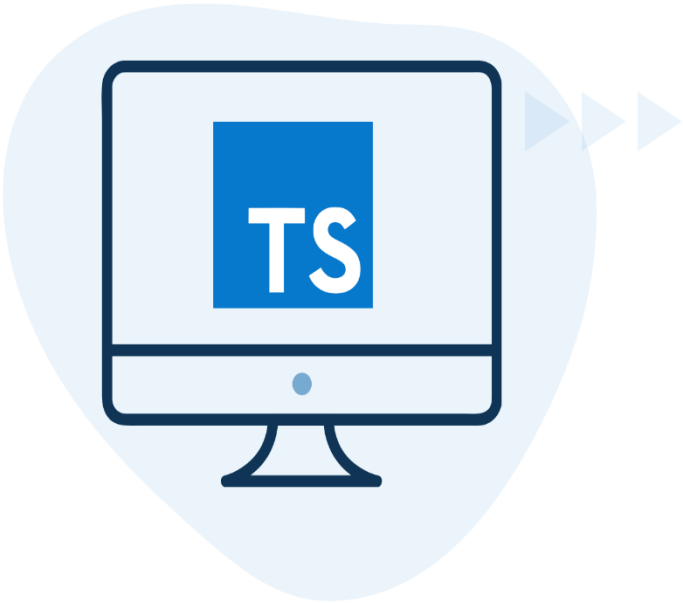
- **Typescript supports JS libraries and API documentation**, including JQuery, BootStrapJS, React, and more. You can use all the familiar tools you already know, so the learning curve isn't too bad.

- **Typescript introduces static typing** to structure your code and improve object-oriented programming techniques. The static typing feature of Typescript also makes the code easier to refactor, since you can navigate to references of functions members.

- **Typescript uses NPM**, which gives you access to millions of reliable libraries. This also makes it far easier to learn Typescript, as you don't have to make custom tools to access libraries.

- **Typescript is easier to maintain.** The language is generally easier to read and access. The built-in self-documentation makes it easier to check on types and objects in your code.
- **Typescript makes it easier to use React, Angular, and Vue.** Typescript integrates well with these frameworks, particularly React, which has been described as a perfect fit with Typescript. The usage of Typescript is not mandatory for these frameworks, but it can add productivity and ease.

Steps to learn TypeScript



Now that you have a grasp on the basics, we're going to teach you everything you need to know to get started with Typescript today.

Step 1: Setting up Typescript

Install TypeScript

You can get access to Typescript either by installing TS Visual Studio Plugin or using NPM (Node Package Manager).

After installing NPM, write the following command in terminal to install TS.

```
npm install -g typescript
```

To check what version you are using, run the following command line in a shell

```
tsc -v
```

TypeScript Compilers

To compile TS code, we run the command `tsc filename.ts`. This will create a JS file of the same name, so we can eventually use it on the browser.

Step 2: Exploring TypeScript types

Types

As the name suggests, everything in Typescript deals with types. Since Typescript is the typed version of JS, we can specify types to variables when they are declared. This makes your code more scalable and reliable, and you can check that your code runs properly before runtime.

If you've worked with Javascript before, you know that it has eight types: string, number, null, undefined, object, symbol, bigint, and boolean. Javascript is dynamically typed, which means that it doesn't know the type of your variable until runtime and variables can change their type. Even if we change them intentionally, errors and bugs often arise. Typescript helps with this problem by adding static types to the code.

There are three categories of types in Typescript: `any`, `Built-in`, and `User-defined`.

- The `any` type is a superset of all Typescript data types, and it is the loosest one available to us. It means that a variable can be of any type. If we use this type, it will override type checking.
- `Built-in` types include number, string, boolean, undefined, null, and void.
- `User-defined` types include enum, array, interface, class, and tuple.

Let's dive into each of those a bit more and how to use Typescript types.

Assigning types

To assign a type in Typescript, you need a colon `:`, the name of the type, an equal sign `=`, and the value of that variable. Let's look at an example.

```
let variableName: typeScriptType = value;
```

Number

Typescript supports decimal, hexadecimal, octal, and binary literal. In Typescript, all numbers are floating-point values.

```
let num: number = 0.444;
let hex: number = 0xbeef;
let bin: number = 0b0010;
```

Boolean

Boolean values function just like they do in Javascript.

```
let yes: boolean = true;
let no: boolean = false;
```

Array

In Typescript, arrays are a collection of the same object. You can declare a typed array in two ways, either with the datatype followed by [ ], or the generic array approach with `Array<elemType>`.

You can also assign multiple types to one array using the `|` operator or create a multidimensional array to save one array into another array using the `[ ]` operator.

```
const arr3: (Date| string[])[] = [new Date(), new Date(), ["1", "a"]];
```

Tuple

Tuples are a lot like arrays, but we can define the type of data that are stored in each position. Tuple types enable you to make organized arrays. You can express an array when you know the type of a fixed number of elements and predefine your types in order.

```
let numberTuple: [number, number, number];
```

Void

Void is a subtype of `undefined`. It is a return type that can be substituted with different types when needed. Void is used when we are returning functions. It essentially tells us that a function will return undefined. This ensures that a function does not return a value.

Enum

Enums allow us to define a set of named predefined constants. These are defined with the enum keyword. You can define a numeric or a string enum.

```
enum MyStringEnum {
    ChoiceA = "A",
    ChoiceB = "B",
}
```

String

Typescript follows the same syntax of Javascript with double or single quotes around text. You can also use the backtick character to use multiple lines or the `${expression}` to enable evaluated operations inside a string.

```
let w = "Value1";
let x = "this is a string with the value " + w;
let y = 'this is a string with the value ' + w;
let z = `this is a string ${w}`;
console.log(w,x,y,z)
```

Step 3: Basics of variables

Like most programming languages, we use variables to store values, such as a string, Boolean, number, or expression. In TS, we can define a variable using `var`, `let`, and `const`. There are some issues that arise when we use `var`. For example, a variable declared with `var` inside a function is function-scoped but global-scoped when declared outside a function. This can lead to errors in the JavaScript code.

The keyword `let` solves this problem by setting the variable's lifespan at the block where it was declared. Similarly, `const` solves the same problem as `let`,

but it can only be initialized once when it is declared. Typescript will make sure no value can be set.

Variables in Typescript follow similar syntactic rules as many other programming languages.

- They can be comprised of lower and uppercase letters of the alphabet
- They cannot begin with a digit
- They can include special characters, such as `$` or `@`.

Step 4: Commenting in TypeScript

Comments in TS use the same syntax as Javascript Double slash for single-line comments Slash stars to open a block of comments Star slash to close a block of comments

Typescript introduces a special syntax. If you add `/*!`, Typescript will keep the comment while transforming into Javascript. This enables you to keep copyright at the top of a TS file that needs to be generated in JS.

```
let x = 1; // This is a single line comment

/* This can be spread on
multiple
lines */
let y = 2;
```

Step 5: Type Inference

Type Inference is what the compiler uses to determine different types. It is smart enough to figure out types from their values. That way, you won't have to specify your types in your code. This a powerful feature of Typescript that allows you to manipulate types and combine them.

The Typescript inference features can infer types in the following scenarios:

- When variables are declared and initialized
- When default values are set as parameters
- When the function return types are determined

Step 6: Functions

Typescript does not make any major changes to the function-scoped core of Javascript. However, Typescript does enhance functions with strong signatures we can use to define parameters and return types.

We declare a function using the `function` keyword. You can also use the `fat arrow` to make a function without the keyword. This does not change with Typescript. We can use Typescript types for function arguments. We use a colon to do so. Take a look at an example:

```
function functionWithParameters(arg1: string, arg2: number){}
```

Typescript functions fall into two categories: **function expressions** or **function declarations**. A function declaration is when a function is defined by not assigning it to a variable while a function expression is assigned to a variable.

In Typescript, you can specify the type of a function with `this` keyword. To do so, you use the `this` followed by a colon, followed by the type of the function signature.

Step 7: Mapped Type

This functionality enables you to create a new type from an existing one. For example, you could have an existing interface keep all the same members but change into read-only or optional. Before the mapped type, we would have to create an extra interface to reflect the final state we want, which can pollute the code and lead to issues.

And without the mapped type, every interface would require a separate function, so things can get out of control quickly. Thanks to the custom logic of a mapped type in Typescript, we don't have to deal with those issues.

There are different mapping functions in Typescript: partial, nullable, pick, omit, record, extract, exclude, and ReturnType.

Step 8: Objects and OOP

Typescript supports object-oriented programming and adds new features to improve upon Javascript's OOP functionality. Typescript supports the use of classes by using the `class` keyword. Think of this like a template of objects. Let's take a look at an example:

```
class class_Name{
    field;
    method;
}
```

This will generate the following JavaScript code:

```
// Generated by typescript 1.8.10
var Person = (function () {
    function Person() {
    }
    return Person;
}());
```

Typescript introduced new ways of using objects. There are many different object types in Typescript: `Object`, `object`, and `{object}`. Typescript can create an object using curly brackets, and you must define its members at initialization. It's a quicker way to organize your data, and you do not need a name since it's not a structural language.

Step 9: Type Checking and Assertions

Let's look at how we can check that our variable has the right type. Here are the two most common approaches.

Instanceof

This operator can check for custom types not defined by Javascript. Below, we first write a custom type, make an instance of it, and check that it is indeed the right variable.

```
class Dog{
 name: string;
 constructor(data: string) {
  this.name = data;
 }
}
let dog = new dog('Rover')
if(dog instanceof Dog){
 console.log(`${dog.name} is a dog`)
}
```

Typeof

This operator can check for basic datatypes. Below, we make a string variable, use the typeof command to check it against another variable and then print the result.

```
let myObject = { name: "test" };
let myOtherObject: typeof myObject; // Borrow type of the "myObject"
```

```
myOtherObject = { name: "test2" };
type TypeFromTypeOf = typeof myOtherObject; // Borrow
```

Sometimes, we need to cast our variables to a datatype, commonly when you are using a general type and need to make it more concrete. There are a few different ways to do this. Let's discuss two popular approaches.

As Keyword

Use the as keyword after the name of the variable and end it with the desired datatype.

```
let str: any = 'This is a String'
let strLength = (str as string).length
```

< > Operator

We can also cast our variables by using the `< >` operator. This has the same effect on our code but implements a simpler syntax.

```
let str: any = 'This is a String'
let strLength = (<string>str).length
```