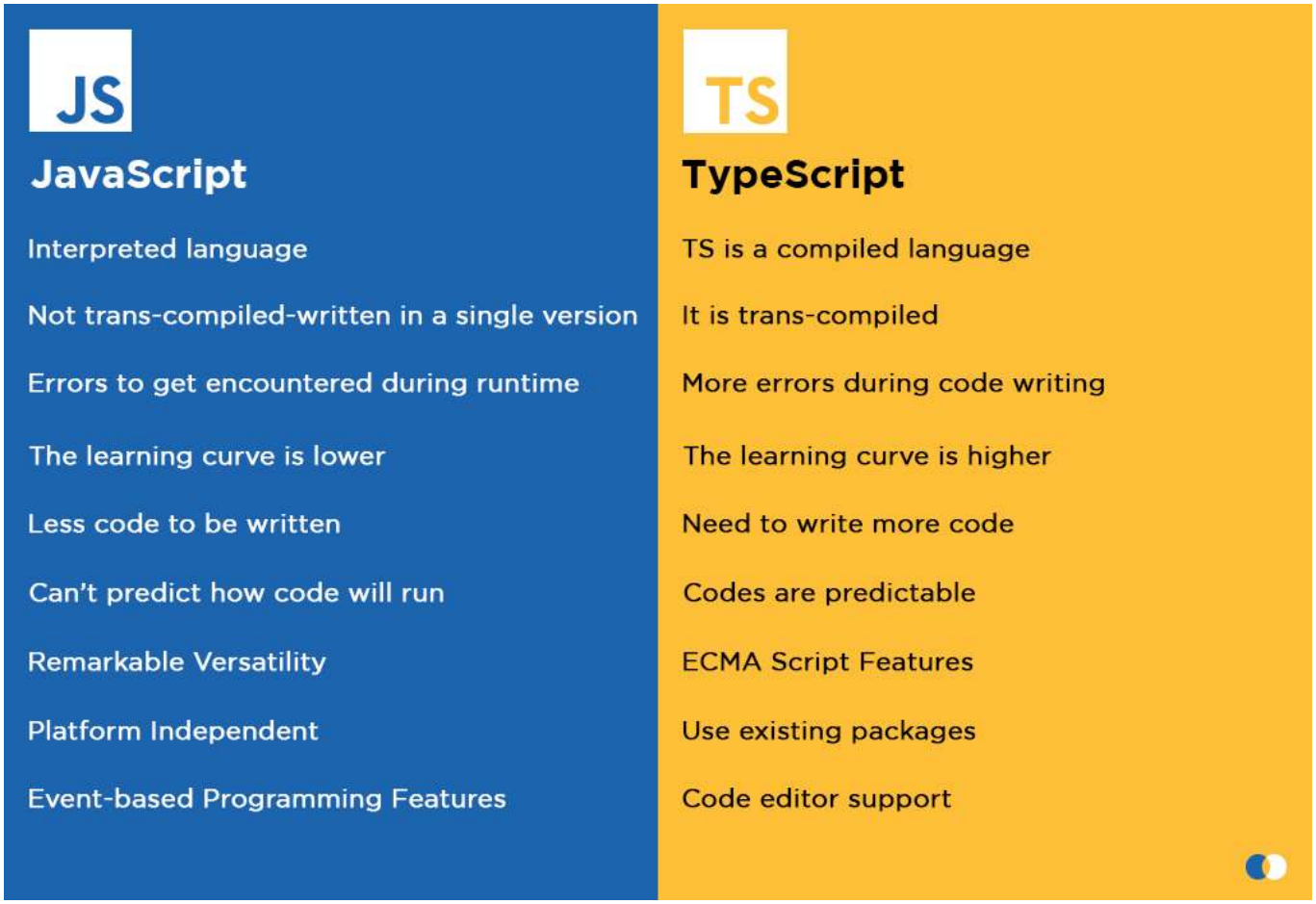
is an open-source object-oriented language developed and maintained by Microsoft, licensed under Apache 2 license. It is a typed superset of Javascript that compiles to plain JavaScript. was developed under Anders Hejlsberg, who also led the creation of the C# language. was first released in October 2012.

Prerequisites: Basic knowledge of JavaScript is required.



Overview

is an open-source, object-oriented language developed and maintained by Microsoft, licensed under Apache 2 license.

extends JavaScript by adding data types, classes, and other object-oriented features with type-checking. It is a typed superset of JavaScript that compiles to plain JavaScript.

Official website: **https://www.lang.org**

Source code: **https://github.com/Microsoft/**

Version History

0.8	October 2012
0.9	June 2013
1.0	October 2014

2.0	September 2016
3.0	July 2018
4.0 - latest release	August 2020

Why ?

JavaScript is a dynamic programming language with no type system. JavaScript provides primitive types like string, number, object, etc., but it doesn't check assigned values. JavaScript variables are declared using the var keyword, and it can point to any value. JavaScript doesn't support classes and other object-oriented features (ECMA2015 supports it). So, without the type system, it is not easy to use JavaScript to build complex applications with large teams working on the same code.

The type system increases the code quality, readability and makes it easy to maintain and refactor codebase. More importantly, errors can be caught at compile time rather than at runtime.

Hence, the reason to use is that it catches errors at compile-time, so that you can fix it before you run code. It supports object-oriented programming features like data types, classes, enums, etc., allowing JavaScript to be used at scale.

 compiles into simple JavaScript. The compiler is also implemented in and can be used with any browser or JavaScript engines like Node.js. needs an ECMAScript 3 or higher compatible environment to compile. This is a condition met by all major browsers and JavaScript engines today.

Some of the most popular JavaScript frameworks like Angular.js and WinJS are written in .

How to use ?

code is written in a file with .ts extension and then compiled into JavaScript using the compiler. A file can be written in any code editor. A compiler needs to be installed on your platform. Once installed, the command tsc <filename>.ts compiles the code into a plain JavaScript file. JavaScript files can then be included in the HTML and run on any browser.

Compile to JavaScript

Features

- **Cross-Platform:** runs on any platform that JavaScript runs on. The compiler can be installed on any Operating System such as Windows, macOS, and Linux.
- **Object-Oriented Language:** provides powerful features such as Classes, Interfaces, and Modules. You can write pure object-oriented code for client-side as well as server-side development.
- **Static type-checking:** uses static typing. This is done using type annotations. It helps type checking at compile time. Thus, you can find errors while typing the code without running your script each time. Additionally, using the type inference mechanism, if a variable is declared without a type, it will be inferred based on its value.
- **Optional Static Typing:** static typing is optional, if you prefer to use JavaScript's dynamic typing.

- **DOM Manipulation:** Like JavaScript, can be used to manipulate the DOM.
- **ES 6 Features:** includes most features of planned ECMAScript 2015 (ES 6, 7) such as class, interface, Arrow functions etc.

Advantages

1. is an open-source language with continuous development and maintenance by **Microsoft**.
2. runs on any browser or JavaScript engine.
3. is similar to JavaScript and uses the same syntax and semantics. All of 'ts code finally gets converted into JavaScript. This allows a quicker learning curve for front-end developers currently coding in JavaScript.
4. is also closer in syntax to backend languages like Java and Scala. This helps backend developers write front-end code faster.
5. code can be called from an existing JavaScript code. also works with existing JavaScript frameworks and libraries without any issues.
6. The Definition file, with .d.ts extension, provides support for existing JavaScript libraries like Jquery, D3.js, etc. So, code can add JavaScript libraries using type definitions to avail the benefits of type-checking, code autocompletion, and documentation in existing dynamically-typed JavaScript libraries.

7. has support for the latest JavaScript features from **ECMAScript 2015** . It includes features from ES6 and ES7 that can run in ES5-level JavaScript engines like Node.js. This offers a massive advantage of using features from future JavaScript versions in current JavaScript engines.
8. has easy integration with task runner tools like Grunt and Gulp to automate the workflow.

Install

There are three ways to install :

1. **Install as an NPM** package on your local machine or in your project.
2. Install NuGet Package in your .NET or .NET Core project.
3. Install as a Plug-in in your IDE (Integrated Development Environment).

Install using NPM

NPM (Node.js package manager) is used to install the package on your local machine or a project. Make sure you have Node.js install on your local machine. If you are using JavaScript frameworks for your application, then it is highly recommended to install Node.js.

To install or update the latest version of , open command prompt/terminal and type the following command:

```
npm install -g
```

The above command will install globally so that you can use it in any project. Check the installed version of using the following command:

```
tsc -v
```

Execute the following command to install the to your local project as dev dependency.

```
npm install  --save-dev
```

1. **npm install -g typescript@latest**
2. **tsc –v**
3. **npm install --save-dev typescript**
4. **npx tsc –init**
5. **tsc index.ts**
6. **node index.js**

In , just like in JavaScript, there are several basic data types that you can use to define variables. adds static typing and type annotations to these types, enhancing the ability to catch errors at compile time. Here are the basic data types in :

1. boolean

The boolean type represents a logical value that can be either true or false.

```
let isDone: boolean = true;

console.log(`Is done: ${isDone}`); // Output: Is done: true
```

2. number

The number type is used to represent both integers and floating-point numbers.

```
let decimal: number = 10;
let hex: number = 0xff;
let binary: number = 0b1010;
let octal: number = 0o12;
```

```
console.log(`Decimal: ${decimal}, Hex: ${hex}, Binary: ${binary}, Octal: ${octal}`);
```

- **Hexadecimal** is a base-16 numbering system. It uses digits 0-9 and letters A-F (or a-f) to represent values from 0 to 15.
- Numbers in hexadecimal are prefixed with 0x or 0X.
- In this example, hex is set to 0xff, which represents 255 in decimal:
    - f in hexadecimal is 15 in decimal.
    - 0xff means (15 * 16^1) + (15 * 16^0), which equals 255.

- **Binary** is a base-2 numbering system that uses only two digits: 0 and 1.
- Binary numbers are prefixed with 0b or 0B.
- In this example, binary is set to 0b1010, which represents 10 in decimal:
    - The rightmost 1 is 1 * 2^0 = 1.
    - The next 0 is 0 * 2^1 = 0.
    - The next 1 is 1 * 2^2 = 4.
    - The leftmost 1 is 1 * 2^3 = 8.
    - Adding these up: 8 + 0 + 2 + 0 = 10.

- **Octal** is a base-8 numbering system that uses digits 0 to 7.
- Octal numbers are prefixed with 0o or 0O.
- In this example, octal is set to 0o12, which represents 10 in decimal:
    - The rightmost 2 is 2 * 8^0 = 2.
    - The next 1 is 1 * 8^1 = 8.
    - Adding these up: 8 + 2 = 10.
    -
    - • decimal is a regular base-10 number 10.
    - • hex is a hexadecimal number 0xff, equivalent to 255 in decimal.
    - • binary is a binary number 0b1010, equivalent to 10 in decimal.
    - • octal is an octal number 0o12, also equivalent to 10 in decimal.

3. string

The string type is used for text data. You can use single quotes ('), double quotes ("), or template literals (`).

```
let firstName: string = "Vikas";

let lastName: string = "S";

let fullName: string = `${firstName} ${lastName}`;

console.log(`Full Name: ${fullName}`); // Output: Full Name: Vikas S
```

4. array

The array type can be declared in two ways: by specifying the type of elements followed by [], or using the generic array type **Array<elementType>.**

```
let list: number[] = [1, 2, 3, 4, 5];

let stringList: Array<string> = ["one", "two", "three"];

console.log(`Number List: ${list}`);

console.log(`String List: ${stringList}`);
```

5. tuple

Tuples are used to express an array with a fixed number of elements whose types are known but not necessarily the same.

```
let tuple: [string, number];
tuple = ["hello", 42]; // OK
console.log(`Tuple: ${tuple}`); // Output: Tuple: hello,42
```

6. enum

Enums are a way to define a set of named constants. provides an enum type with a few choices.

```
enum Color {

  Red = 1,
```

```
  Green,

  Blue,

}

let colorName: string = Color[2];

console.log(`Color at position 2: ${colorName}`); // Output: Color at position
2: Green
```

7. unknown

The unknown type represents any value. This is similar to any, but is safer because you cannot perform any operations on an unknown type without first narrowing it.

```
let notSure: unknown = 4;

console.log(`Unknown value (number): ${notSure}`); // Output: Unknown value
(number): 4

notSure = "maybe a string";

console.log(`Unknown value (string): ${notSure}`); // Output: Unknown value
(string): maybe a string
```

8. any

The any type allows you to opt out of type-checking, effectively disabling type checking for that variable.

```
let anything: any = "Hello";
anything = 42;
anything = true;
console.log(`Any type value: ${anything}`); // Output: Any type value: true
```

9. void

The void type is used for functions that do not return a value.

```
function warnUser(): void {

  console.log("This is a warning message");
```

```
}

warnUser(); // Output: This is a warning message
```

10. null and undefined

In , null and undefined are actual values, and they have their own types: null and undefined.

```
let u: undefined = undefined;
let n: null = null;
console.log(`Undefined: ${u}, Null: ${n}`); // Output: Undefined: undefined,
Null: null
```

11. never

The never type represents the type of values that never occur. It's often used for functions that throw exceptions or have infinite loops.

```
function error(message: string): never {
  throw new Error(message);
}

// Uncommenting this will terminate the program due to the thrown error
// error("Something went wrong!");
```

12. object

The object type represents non-primitive types, such as arrays, functions, and objects.

```
let someObject: object = { name: "Ajay", age: 25 };

console.log(`Object: ${JSON.stringify(someObject)}`); // Output: Object:
{"name":"Ajay","age":25}
```

Variable

follows the same rules as JavaScript for variable declarations. Variables can be declared using: var, let, and const.

var

Variables in can be declared using var keyword, same as in JavaScript. The scoping rules remains the same as in JavaScript.

let

To solve problems with var declarations, ES6 introduced two new types of variable declarations in JavaScript, using the keywords let and const. , being a superset of JavaScript, also supports these new types of variable declarations.

The let declarations follow the same syntax as var declarations. Unlike variables declared with var, variables declared with let have a block-scope. This means that the scope of let variables is limited to their containing block, e.g. function, if else block or loop block. Consider the following example.

```
let num1:number = 1;

function letDeclaration() {
    let num2:number = 2;

    if (num2 > num1) {
        let num3: number = 3;
        num3++;
    }

    while(num1 < num2) {
        let num4: number = 4;
        num1++;
    }

    console.log(num1); //OK
    console.log(num2); //OK
```

```
    //console.log(num3); //Compiler Error: Cannot find name 'num3'
   // console.log(num4); //Compiler Error: Cannot find name 'num4'
}

letDeclaration();
```

In the above example, all the variables are declared using let. num3 is declared in the if block so its scope is limited to the if block and cannot be accessed out of the if block. In the same way, num4 is declared in the while block so it cannot be accessed out of while block. Thus, when accessing num3 and num4 else where will give a compiler error.

The same example with the **var** declaration is compiled without an error.

```
var num1:number = 1;

function varDeclaration() {
    var num2:number = 2;

    if (num2 > num1) {
        var num3: number = 3;
        num3++;
    }

    while(num1 < num2) {
        var num4: number = 4;
        num1++;
    }

    console.log(num1); //2
    console.log(num2); //2
    console.log(num3); //4
    console.log(num4); //4
}
```

```
varDeclaration();
```

Advantages of using let over var

1) Block-scoped let variables cannot be read or written to before they are declared.

```
//console.log(num1); // Compiler Error: error TS2448: Block-scoped variable
'num' used before its declaration
//let num1:number = 10 ;

console.log(num2); // OK, Output: undefined
var num2:number = 10 ;
console.log(num2);
```

In the above example, the compiler will give an error if we use variables before declaring them using let, whereas it won't give an error when using variables before declaring them using var.

Const

Variables can be declared using const similar to var or let declarations. The const makes a variable a constant where its value cannot be changed. Const variables have the same scoping rules as let variables.

```
const numc:number = 100;
numc = 200; //Compiler Error: Cannot assign to 'num' because it is a constant
or read-only property
```