Typescript

Presented by

Sagar
Java Consultant

Typescript

- JavaScript was introduced as a language for the client side.

- The development of Node.js has marked JavaScript as an emerging server-side technology.

- TypeScript offers all of JavaScript's features, and an additional layer on top of these:
  TypeScript's "type" system. [data types].

- For example in creating a variable and assigning it to a particular value, TypeScript will use the value as its type.

```
2    let str:string;
3    str="Hello";
4    console.log(str);
```


javatechie.sagar@gmail.com

Typescript Features

- TypeScript code is not understandable by the browsers.
- The code is written in Type Script.
- Then converted into Java Script.
- This process is known as Trans-piling – tsc is a transpiler
- Java Script code can be converted into Type Script Code … .js to .ts
- With TypeScript, developers can use existing JavaScript code, incorporate popular JavaScript libraries.

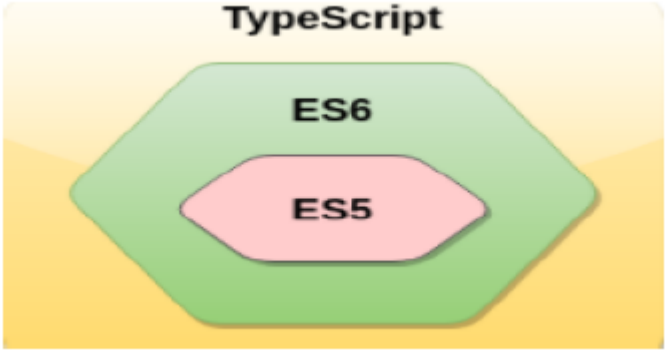
javatechie.sagar@gmail.com

Why to use Typescript?

- TypeScript supports object-oriented programming features such as classes, interfaces, inheritance etc.
- TypeScript provides the error-checking feature at compilation time.
- TypeScript supports all JavaScript libraries because it is the superset of JavaScript.

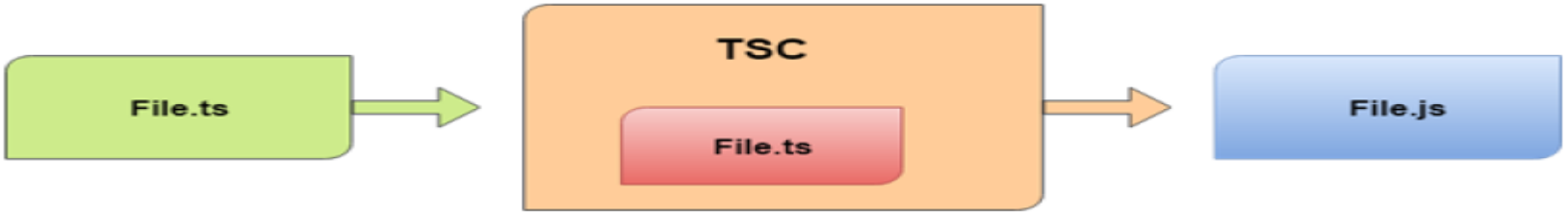
Typescript supports ECMA Scripts

- TypeScriptsupportsthelatestJavaScriptfeatures,includingECMAScript2015.



- TypeScript can not run directly on the browser.

- It needs a compiler to compile the file and generate it in Java Script file, which can run directly on the browser.



Typescript Data Types

- **string** represents string values like "Hello, world“
- **number** is for numbers like 42.
- JavaScript does not have a special runtime value for integers, so there’s no equivalent to int or float - everything is simply number
- **boolean** is for the two values true and false

** Demo

Typescript Arrays

- A TypeScript array is an ordered list of data.
- To declare an array that holds values of a specific type, use the following syntax:

let arrayName: type[];

For example, the following declares an array of strings:

let skills: string[] = [];

Add one or more strings to the array:

```
skills[0] = "Problem Solving";
skills[1] = "Programming";
```

Also use the push() method:

skills.push('Software Design');

or **Demo

```
let skills = ['Problem Sovling','Software Design','Programming'];
skills.sort();
```


javatechie.sagar@gmail.com

Typescript Tuple

A tuple works like an array with some additional considerations:

- The number of elements in the tuple is fixed.
- The types of elements are known, and need not be the same.

For example, you can use a tuple to represent a value as a pair of a `string` and a `number`:

```
let skill: [string, number];
skill = ['Programming', 5];
```

The order of values in a tuple is important. If you change the order of values of the `skill` tuple to `[5, "Programming"]`, you'll get an error:

```
let skill: [string, number];
skill = [5, 'Programming'];
```

Error:

```
error TS2322: Type 'string' is not assignable to type 'number'.
```

Typescript Enum

An enum is a group of named constant values. Enum stands for enumerated type.

To define an enum, you follow these steps:

- First, use the `enum` keyword followed by the name of the enum.
- Then, define constant values for the enum.

- The following example creates an enum that represents the months of the year:

```
enum Month { Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };
```

**Demo

Typescript Any

- **any** is used to represent any data type.

- Ex :

```
1  let result: any;
2  result = 1;
3  console.log(result);
4  result = 'Hello';
5  console.log(result);
6  result = [1, 2, 3];
7  console.log(result);
```

** Demo

Typescript void

- **void** – denotes the absence any data type at all.
- Typically, void is used as return type of a function that returns nothing.

```
function log(message): void {
    console.log(message);
}

log("Hello");
```

** Demo

Typescript undefined and null

- The value **undefined** means value is not assigned.
- The value **null** indicates that does not have a value.

```
1  let value: any;
2  console.log(value);  //undefined
3
4  value=null           //Setting its value to null
5  console.log(value);
```

** Demo

Typescript Type inference

- Type inference describes where and how TypeScript infers (concludes) types when explicitly mentioned them.

```
1    function setCounter(max=100) {
2        console.log(max);
3    }
4    setCounter();
```

- In this example, TypeScript infers the type of the max parameter to be number.

** Demo

Typescript typecasting

- Type casting, also known as type assertions or type conversion.

```
function multiply(num1: number, num2: number): string {
  return (num1 * num2).toString();
}

const result = multiply(5, 10);
const convertedResult = result as unknown as number;
console.log(convertedResult); // Output: 50
```

** Demo

let vs var

- **let** – 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.

  Ex:

```
let employeeName = "John";
// or
let employeeName:string = "John";
```

  **Demo

- **var** – Variables declared with **var** have a non-block scope.

  This means that the scope of **var** variables is not limited to their containing block, e.g. function, if else block or loop block.

  Ex:

```
var employeeName = "John";
// or
var employeeName:string = "John";
```

  **Demo

Typescript 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.

Ex:

```
const num:number = 100;
num = 200;
```

```
Cannot assign to 'num' because it is a constant. ts(2588)
const num: any
View Problem (Alt+F8)    Quick Fix... (Ctrl+.)
num = 200;
```

** Demo