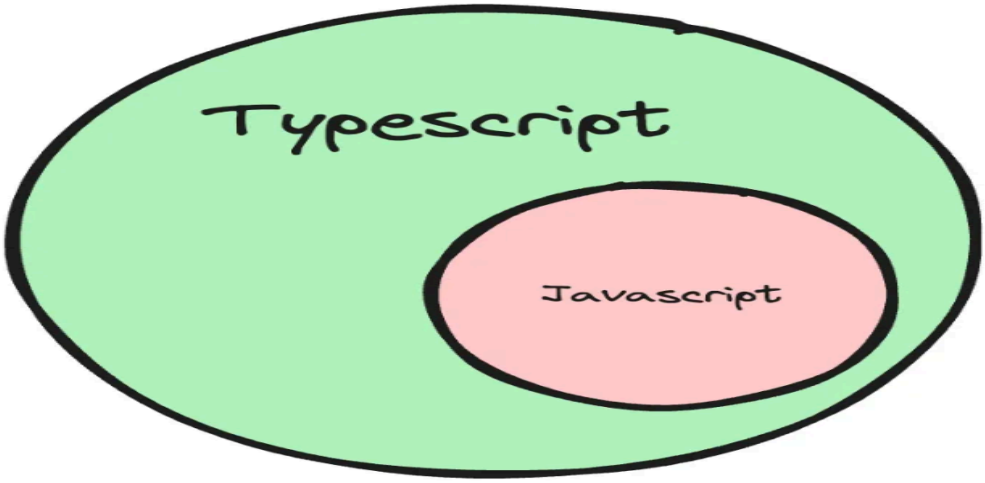
TYPESCRIPT

What is typescript?

TypeScript is a programming language developed and maintained by Microsoft.

It is a strict syntactical superset of JavaScript and adds optional static typing to the language.

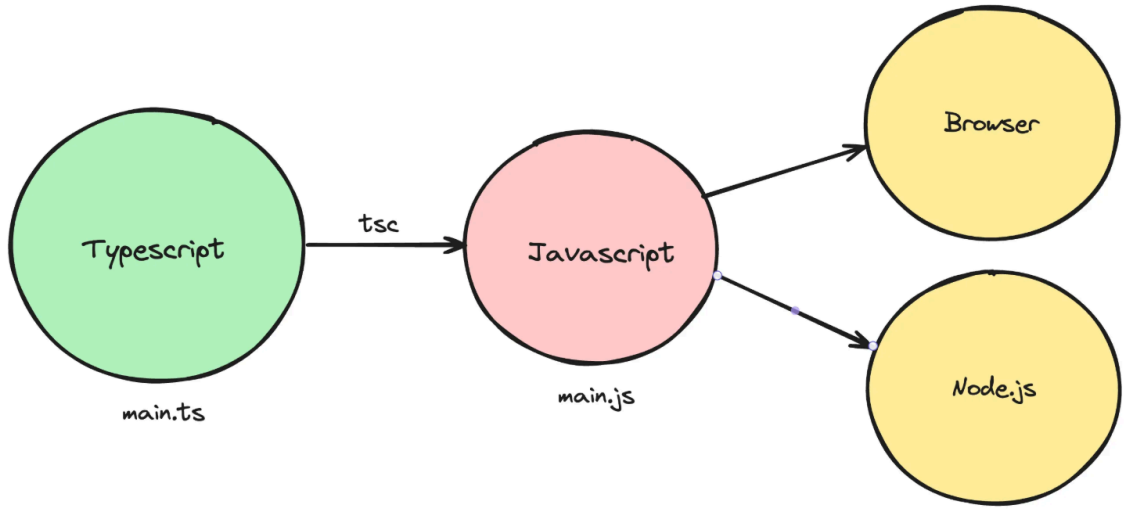


Where/How does typescript code run?

Typescript code never runs in your browser. Your browser can only understand javascript.

Javascript is the runtime language (the thing that actually runs in your browser/nodejs runtime)

Typescript is something that compiles down to javascript



Typescript compiler

tsc is the official typescript compiler that you can use to convert Typescript code into Javascript

There are many other famous compilers/transpilers for converting Typescript to Javascript. Some famous ones are -

1.esbuild

2.swc

The tsc compiler:

Let's bootstrap a simple Typescript Node.js application locally on our machines

Step 1 - Install tsc/typescript globally

```
npm install -g typescript
```

Step 2 - Initialize an empty Node.js project with typescript

```
mkdir node-app
cd node-app
npm init -y
npx tsc --init
```

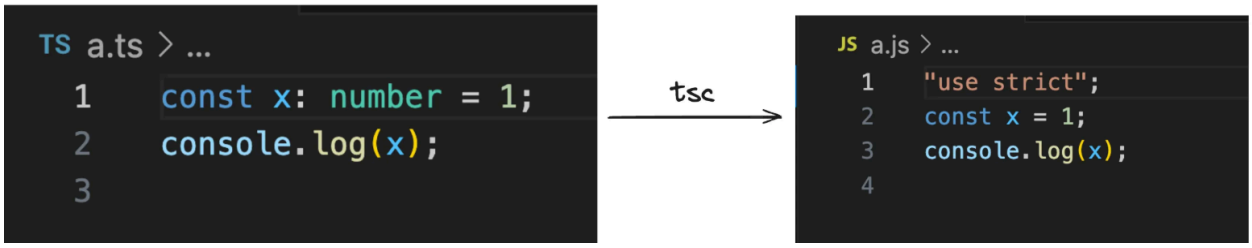
Step 3 - Create a a.ts file

```ts
const x: number = 1;
console.log(x);
```

Step 4 - Compile the ts file to js file

```
tsc -b
```

Step 5 - Explore the newly generated index.js file



Step 6 - Try assigning x to a string

Make sure you convert the `const` to `let`

```ts
let x: number = 1;
x = "harkirat"
console.log(x);
```

Step 7 - Try compiling the code again

```
tsc -b
```

This is the high level benefit of typescript. It lets you catch type errors at compile time

Basic Types in TypeScript:

Typescript provides you some basic types number, string, boolean, null, undefined.

Let's create some simple applications using these types -

Problem 1 - Hello world

Write a function that greets a user given their first name.

Argument - firstName

Logs - Hello {firstName}

Doesn't return anything

Solution

```
function greet(firstName: string) {
    console.log("Hello " + firstName);
}

greet("harkirat");
```

Problem 2 - Sum function

Write a function that calculates the sum of two functions

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

console.log(sum(2, 3));
```

The tsconfig file:

The tsconfig file has a bunch of options that you can change to change the compilation process.

Some of these include

1. target

The target option in a tsconfig.json file specifies the ECMAScript target version to which the TypeScript compiler will compile the TypeScript code.

To try it out, try compiling the following code for target being ES5 and es20201. target

```ts
const greet = (name: string) => `Hello, ${name}!`;
```

▼ Output for ES5

```ts
"use strict";
var greet = function (name) { return "Hello, ".concat(name, "!"); };
```

▼ Output for ES2020

```ts
"use strict";
const greet = (name) => `Hello, ${name}!`;
```

2. rootDir

Where should the compiler look for .ts files. Good practise is for this to be the src folder

3. outDir

Where should the compiler look for spit out the .js files.

4. noImplicitAny

Try enabling it and see the compilation errors on the following code -

```
const greet = (name) => `Hello, ${name}!`;
```

5. removeComments

Weather or not to include comments in the final js file

Interfaces:

1. What are interfaces

How can you assign types to objects? For example, a user object that looks like this -

```
const user = {
    firstName: "harkirat",
    lastName: "singh",
    email: "email@gmail.com".
    age: 21,
}
```

To assign a type to the `user` object, you can use `interfaces`

```
interface User {
    firstName: string;
    lastName: string;
    email: string;
    age: number;
}
```

2. Implementing interfaces

Interfaces have another special property. You can implement interfaces as a class.

Let's say you have an personinterface -

```
interface Person {
    name: string;
    age: number;
    greet(phrase: string): void;
}
```

You can create a class which `implements` this interface.

```
class Employee implements Person {
    name: string;
    age: number;

    constructor(n: string, a: number) {
        this.name = n;
        this.age = a;
    }

    greet(phrase: string) {
        console.log(`${phrase} ${this.name}`);
    }
}
```

Rectangle and Circle classes

```
class Rectangle extends Shape {
  name = "Rectangle";

  constructor(public width: number, public height: number) {
    super();
  }

  // Implement the abstract method
  calculateArea(): number {
    return this.width * this.height;
  }
}

// Another subclass implementing the abstract class
class Circle extends Shape {
  name = "Circle";

  constructor(public radius: number) {
    super();
  }

  // Implement the abstract method
  calculateArea(): number {
    return Math.PI * this.radius * this.radius;
  }
}
```

Types:

What are types?

Very similar to interfaces , types let you aggregate data together.

```ts
type User = {
  firstName: string;
  lastName: string;
  age: number
}
```

1. Unions

Let's say you want to print the id of a user, which can be a number or a string.

```ts
type StringOrNumber = string | number;

function printId(id: StringOrNumber) {
  console.log(`ID: ${id}`);
}

printId(101); // ID: 101
printId("202"); // ID: 202
```

2. Intersection

What if you want to create a type that has every property of multiple types/ interfaces

```ts
type Employee = {
  name: string;
  startDate: Date;
};

type Manager = {
  name: string;
  department: string;
};

type TeamLead = Employee & Manager;

const teamLead: TeamLead = {
  name: "harkirat",
  startDate: new Date(),
  department: "Software developer"
};
```

Arrays in TS

If you want to access arrays in typescript, it's as simple as adding a [] annotation next to the type

Example 1

Given an array of positive integers as input, return the maximum value in the array

```ts
function maxValue(arr: number[]) {
    let max = 0;
    for (let i = 0; i < arr.length; i++) {
        if (arr[i] > max) {
            max = arr[i]
        }
    }
    return max;
}

console.log(maxValue([1, 2, 3]));
```

Example 2

Given a list of users, filter out the users that are legal (greater than 18 years of age)

```
interface User {
  firstName: string;
  lastName: string;
  age: number;
}
```

Solution

```
interface User {
  firstName: string;
  lastName: string;
  age: number;
}

function filteredUsers(users: User[]) {
  return users.filter(x => x.age >= 18);
}

console.log(filteredUsers([{
  firstName: "harkirat",
  lastName: "Singh",
  age: 21
}, {
  firstName: "Raman",
  lastName: "Singh",
  age: 16
}, ]));
```

Enums

Enums (short for enumerations) in TypeScript are a feature that allows you to define a set of named constants.

The concept behind an enumeration is to create a human-readable way to represent a set of constant values, which might otherwise be represented as numbers or strings.

Example 1 - Game

Let's say you have a game where you have to perform an action based on weather the user has pressed the `up` arrow key, `down` arrow key, `left` arrow key or `right` arrow key.

```
function doSomething(keyPressed) {
  // do something.
}
```

The best thing to use in such a case is an `enum`.

```
enum Direction {
  Up,
  Down,
  Left,
  Right
}

function doSomething(keyPressed: Direction) {
  // do something.
}

doSomething(Direction.Up)
```

2. What values do you see at runtime for Direction.UP ?

Try logging Direction.Up on screen

```
enum Direction {
  Up,
  Down,
  Left,
  Right
}

function doSomething(keyPressed: Direction) {
  // do something.
}

doSomething(Direction.Up)
console.log(Direction.Up)
```

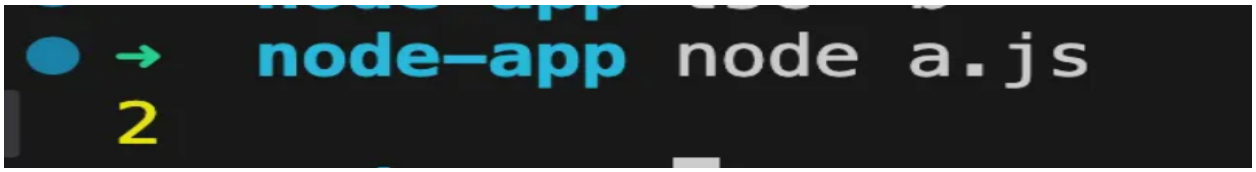
This tells you that by default, enums get values as 0 , 1, 2

3. How to change values?

```
enum Direction {
  Up = 1,
  Down, // becomes 2 by default
  Left, // becomes 3
  Right // becomes 4
}

function doSomething(keyPressed: Direction) {
  // do something.
}

doSomething(Direction.Down)
```



4. Can also be strings

```
enum Direction {
  Up = "UP",
  Down = "Down",
  Left = "Left",
  Right = 'Right'
}

function doSomething(keyPressed: Direction) {
  // do something.
}

doSomething(Direction.Down)
```

5. Common usecase in express

```
enum ResponseStatus {
  Success = 200,
  NotFound = 404,
  Error = 500
}

app.get('/', (req, res) => {
  if (!req.query.userId) {
      res.status(ResponseStatus.Error).json({})
  }
  // and so on...
    res.status(ResponseStatus.Success).json({});
})
```

Generics:

Generics are a language independent concept (exist in C++ as well).

Let's learn it via an example

1. Problem Statement

Let's say you have a function that needs to return the first element of an array. Array can be of type either string or integer.

How would you solve this problem?

```
function getFirstElement(arr: (string | number)[]) {
  return arr[0];
}

const el = getFirstElement([1, 2, 3]);
```

What is the problem in this approach?

1.User can send different types of values in inputs, without any type errors

```
function getFirstElement(arr: (string | number)[]) {
  return arr[0];
}

const el = getFirstElement([1, 2, '3']);
```

Typescript isn't able to infer the right type of the return type

```
function getFirstElement(arr: (string | number)[]) {
  return arr[0];
}

const el = getFirstElement(["harkiratSingh", "ramanSingh"]);
console.log(el.toLowerCase())
```

2. Solution - Generics:

Generics enable you to create components that work with any data type while still providing compile-time type safety.

Simple example -

```
function identity<T>(arg: T): T {
  return arg;
}

let output1 = identity<string>("myString");
let output2 = identity<number>(100);
```

3. Solution to original problem

Can you modify the code of the original problem now to include generics in it?

```
function getFirstElement<T>(arr: T[]) {
  return arr[0];
}

const el = getFirstElement(["harkiratSingh", "ramanSingh"]);
console.log(el.toLowerCase())
```

Did the issues go away?

1.User can send different types of values in inputs, without any type errors

```
function getFirstElement<T>(arr: T[]) {
  return arr[0];
}

const el = getFirstElement<string>(["harkiratSingh", 2]);
console.log(el.toLowerCase())
```

2.Typescript isn't able to infer the right type of the return type

```
function getFirstElement<T>(arr: T[]) {
  return arr[0];
}

const el = getFirstElement(["harkiratSingh", "ramanSingh"]);
console.log(el.toLowerCase())
```

Exporting and importing modules:

TypeScript follows the ES6 module system, using import and export statements to share code between different files. Here's a brief overview of how this works:

1. Constant exports

math.ts

```
export function add(x: number, y: number): number {
  return x + y;
}

export function subtract(x: number, y: number): number {
  return x - y;
}
```

main.ts

```
import { add } from "./math"

add(1, 2)
```

2. Default exports

```
export default class Calculator {
  add(x: number, y: number): number {
    return x + y;
  }
}
```

```
import Calculator from './Calculator';

const calc = new Calculator();
console.log(calc.add(10, 5));
```