TypeScript

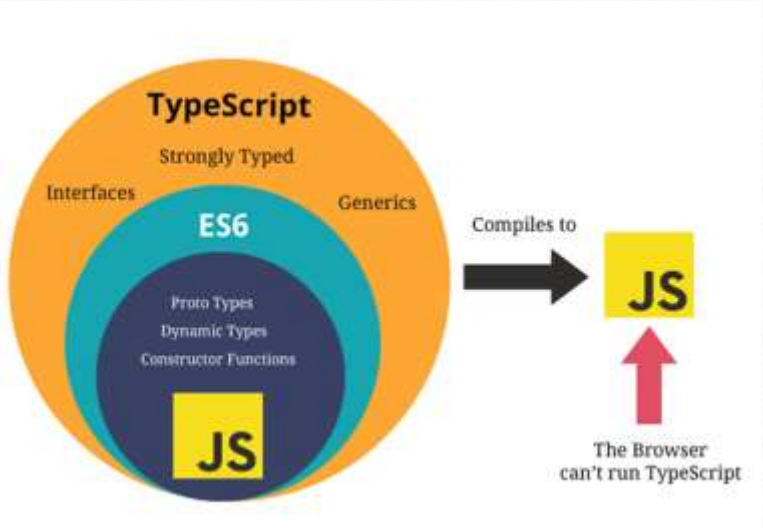
What is TypeScript

- TypeScript is not an entirely new programming language; it's a superset of JavaScript.
- TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.
- It's free and open-source.
- Developed and maintained by Microsoft.
- So, any valid JavaScript code is also a valid TypeScript code. Still, TypeScript has some additional features that do not exist in the current version of JavaScript supported by most browsers.

TS

TypeScript

- Features of TypeScript
  - Strong Typing
  - Object-oriented Features
  - Compile-time errors
  - Great tooling
- Why TypeScript
  - Problems with JavaScript.
  - TypeScript tries to address a lot of problems in JavaScript

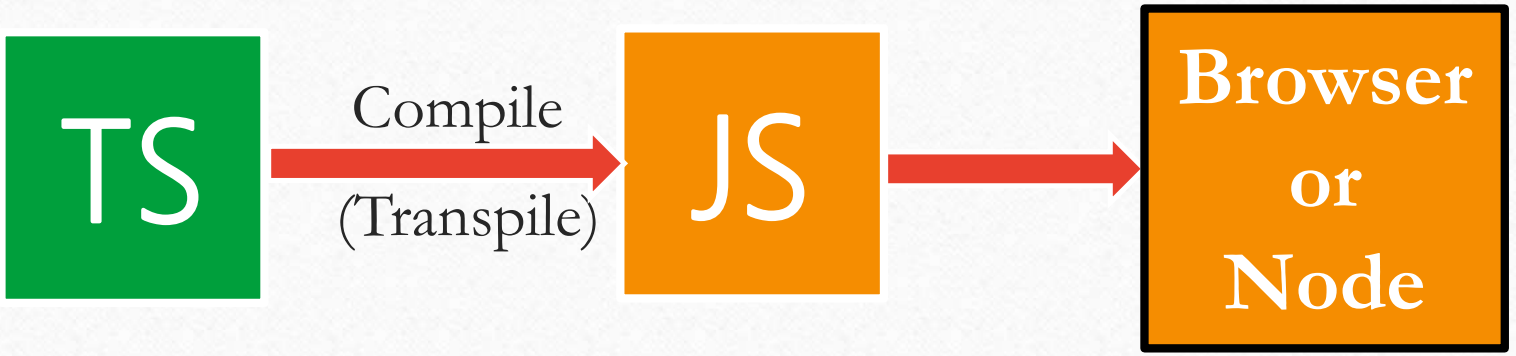


Some problems with JavaScript

1. Lack of Type checking (no enforcement of typing).
   - Objects are loosely structured
2. Application complexity
   - Then why do we use JavaScript?
   - Because browsers can support only JavaScript.(they can understand only JavaScript).
3. New JavaScript features(ES6) may not be supported in older browsers.

TS

Compilation



TypeScript

- A valid TypeScript code is also a valid JavaScript code.
- Syntactic sugar on the top of JavaScript.
- In TypeScript, we can use new ES6 features like let, const, arrow functions, etc., and it converts them into pure or native JavaScript, which can address older versions of browsers.
- So, TypeScript makes building complex applications more manageable.
- Frameworks like Angular and React are using it.

TS

Primitive types

Types that correspond to primitives in Javascript

- number
- string
- boolean
- null
- undefined

Other common types

- Any
- Arrays
- Tuples
- Enums
- Functions
- Objects
- Language types
- Void

Tuple

- Tuple is an array of fixed length, where elements at every position are of predefined types
- For tuples with more than 2 elements you should consider using objects instead for more readability

```
// Tuple to represent person's name and age
const person: [string, number] = ['John', 42];
```

Enum

- Enums allow a developer to define a set of named constants
- TypeScript provides both numeric and string-based enums.

```
enum Direction {
  Up = 1,
  Down,
  Left,
  Right,
}
```

```
enum Color {
  Red = '#D2042D',
  Green = '#21FD5A',
  Blue = '#21E8FD',
  White = '#FFFFFF'
}
```

Function

- When you declare a function, you can add type annotations after each parameter to declare what types of parameters the function accepts.

```
function trasformToGreeting(name: string): string {
  return "Hello, " + name.toUpperCase() + "!!";
}
```

Union type

- A union type is a type formed from two or more other types, representing values that may be any of those types. Each of these types is referred as the union's member.

```
function printId(id: number | string) {
  console.log("Your ID is: " + id);
}
// OK
printId(101);
// OK
printId("202");
// Error
printId({ myID: 22342 });
```

Type alias

- A type alias is a name for any type
- You should use type alias if you want to use a type more than once

```
type Point = {
  x: number;
  y: number;
};

// Exactly the same as the earlier example
function printCoord(pt: Point) {
  console.log("The coordinate's x value is " + pt.x);
  console.log("The coordinate's y value is " + pt.y);
}

printCoord({ x: 100, y: 100 });
```

Interface

- An interface declaration is another way to name an object type

```
interface Point {
  x: number;
  y: number;
}

function printCoord(pt: Point) {
  console.log("The coordinate's x value is " + pt.x);
  console.log("The coordinate's y value is " + pt.y);
}

printCoord({ x: 100, y: 100 });
```

Optional chaining

- The optional chaining operator (?.) enables you to read the value of a property located deep within a chain of connected objects without having to check that each reference in the chain is valid.

```typescript
interface User {
  name: string;
  age: number;
  address?: {
    street: string;
    city: string;
    state: string;
  };
}

const user: User = {
  name: 'John',
  age: 30,
  address: {
    street: 'Main street',
    city: 'New York',
    state: 'NY',
  },
};

function getUserCity(user: User) {
  return user.address?.city;
}
```