TypeScript: Scaling Up JavaScript

Saray Chak, Web Developer

Agenda

- What is TypeScript?
- Installation
- Hello World
- Why TypeScript?
- Basic Type
- Function & Class
- Interface
- Generic
- Enum
- Who Use TypeScript?
- Conclusion
- Q&A
- References

What is TypeScript?

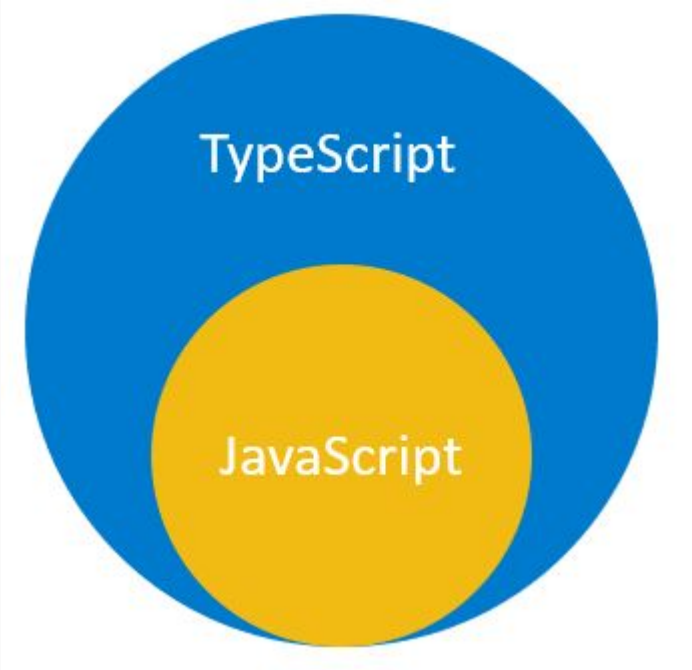
JavaScript Superset

Add new features & advantages to JavaScript

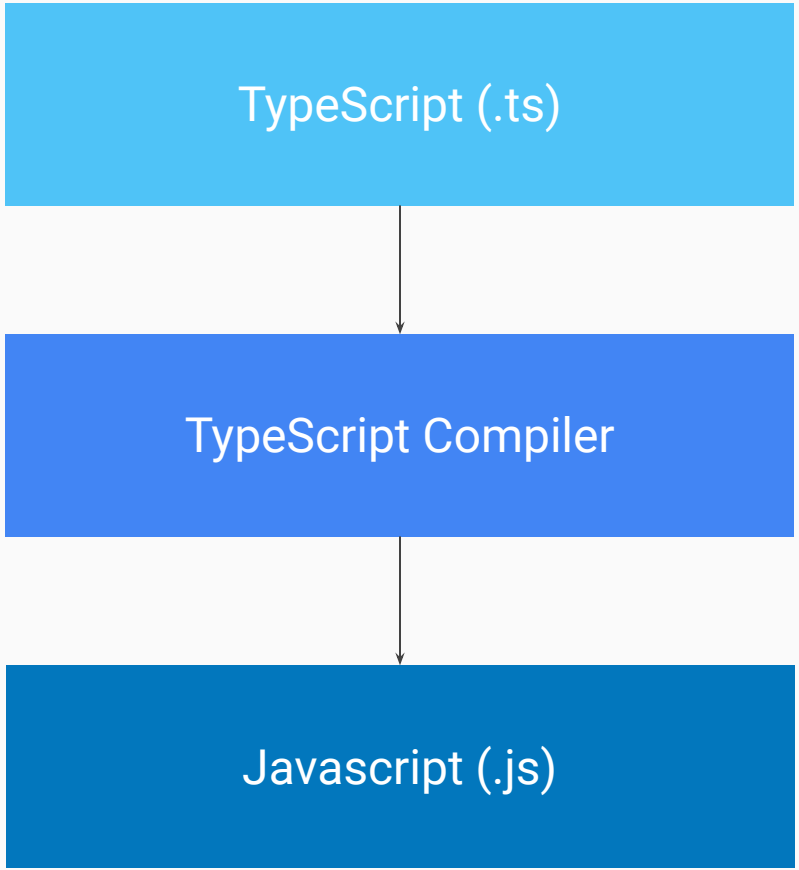
Language building up on JavaScript

Browser CAN’T execute it!

What is TypeScript?



What is TypeScript?



Installation

The following tools you need to setup to start with TypeScript:
- Node js
- TypeScript compiler
- IDE (VsCode)

```
npm install -g typescript
```

```
tsc --v
```

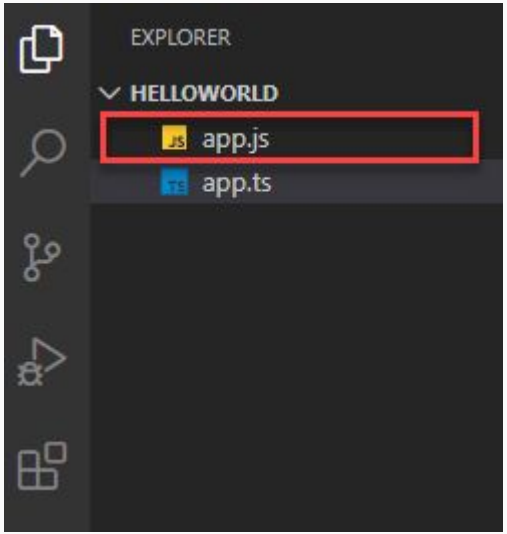
```
Version 4.0.2
```

Hello World

```
let message: string = 'Hello, World!';
console.log(message);
```

compile the app.ts file

```
tsc app.ts
```




```
node app.js
```

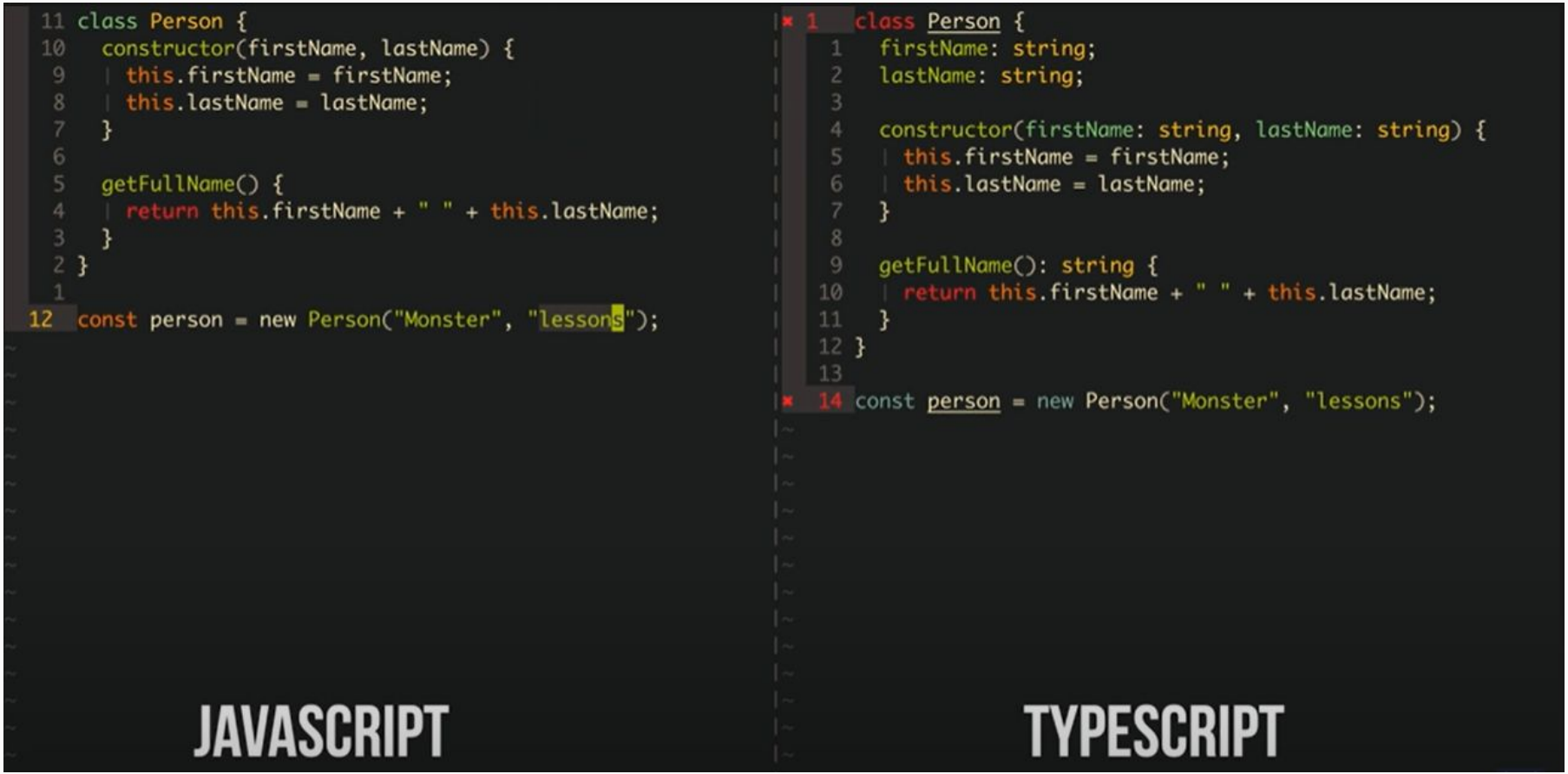
You will see the output as
Hello, World!

Why TypeScript?

There are three main reasons to use TypeScript:

- TypeScript adds a strongly-type system to help you avoid many problems with dynamic types in JavaScript.
- TypeScript always point out the compilation errors at the time of development.
- TypeScript implements the future features of JavaScript ES Next so that you can use them today.
- Write code better with architecture.

Why TypeScript?



Why TypeScript?

Basic Type

TypeScript inherits the built-in types from JavaScript. TypeScript types is categorized into:

- Primitive type
- Objective Type

Primitive types

The following illustrates the primitive types in TypeScript:

Name	Description
`string`	represents text data
`number`	represents numeric values
`boolean`	has true and false values
`null`	has one value: null
`undefined`	has one value: `undefined` . It is a default value of an uninitialized variable
`symbol`	represents a unique constant value

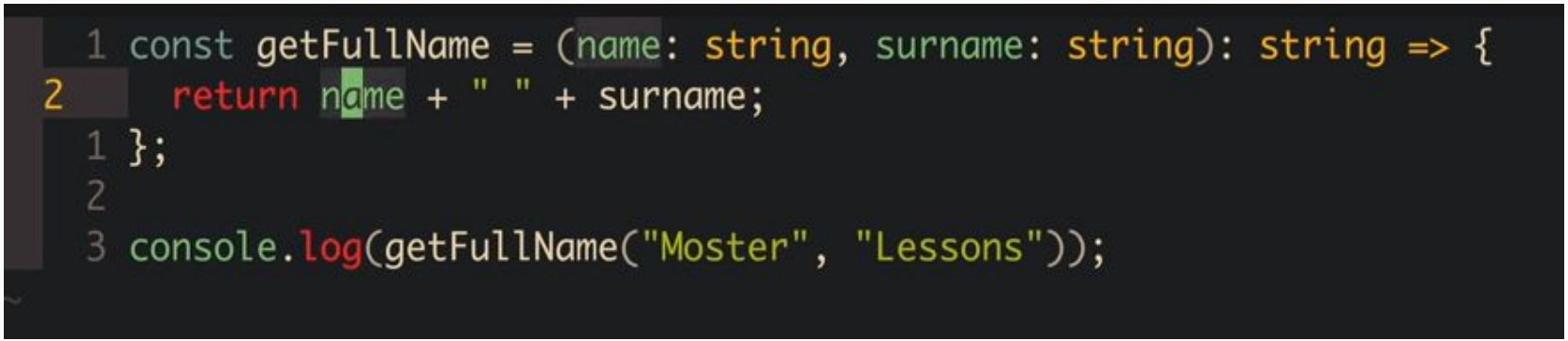
Basic Type

Object types:

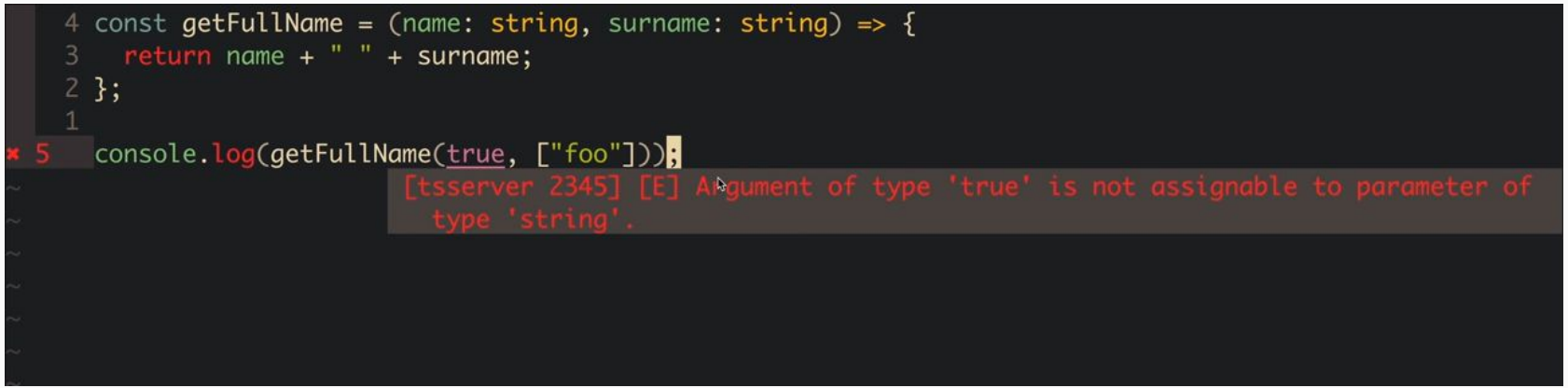
- Function
- Arrays
- Classes
- Objects
- Tuples
- Enum

Function

TypeScript functions are the building blocks of readable, maintainable, and reusable code.



Function


4 const getFullName = (name: string, surname: string) => {
3 return name + " " + surname;
2 };
1
5 console.log(getFullName(true, ["foo"]));
[tsserver 2345] [E] Argument of type 'true' is not assignable to parameter of type 'string'.

Function: overloading

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

function addStrings(a: string, b: string): string {
    return a + b;
}
```

Function: overloading

```
function add(a: number | string, b: number | string): number | string {
    if (typeof a === 'number' && typeof b === 'number')
        return a + b;

    if (typeof a === 'string' && typeof b === 'string')
        return a + b;
}
```

Function: overloading

Overloading function

```
function add(a: number, b: number): number;
function add(a: string, b: string): string;
function add(a: any, b: any): any {
    return a + b;
}
```

```
                function add(a: number, b: number): number (+1 overload)
let result = add(10, 20);
```

Class

ES5

```
function Person(ssn, firstName, lastName) {
    this.ssn = ssn;
    this.firstName = firstName;
    this.lastName = lastName;
}
```

```
Person.prototype.getFullName = function () {
    return `${this.firstName} ${this.lastName}`;
}
```

```
let person = new Person('171-28-0926','John','Doe');
console.log(person.getFullName());
```

Class

ES6

```
class Person {
    ssn;
    firstName;
    lastName;

    constructor(ssn, firstName, lastName) {
        this.ssn = ssn;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    getFullName() {
        return `${this.firstName} ${this.lastName}`;
    }
}
```

```
let person = new Person('171-28-0926','John','Doe');
console.log(person.getFullName());
```

Class

TypeScript

```typescript
class Person {
    ssn: string;
    firstName: string;
    lastName: string;

    constructor(ssn: string, firstName: string, lastName: string) {
        this.ssn = ssn;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    getFullName(): string {
        return `${this.firstName} ${this.lastName}`;
    }
}
```

```typescript
let person = new Person(171280926, 'John', 'Doe');
```

Class: inheritances

```
class Person {
    constructor(private firstName: string, private lastName: string) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
    getFullName(): string {
        return `${this.firstName} ${this.lastName}`;
    }
    describe(): string {
        return `This is ${this.firstName} ${this.lastName}.`;
    }
}
```

```
class Employee extends Person {
    constructor(
        firstName: string,
        lastName: string,
        private jobTitle: string) {

        // call the constructor of the Person class:
        super(firstName, lastName);
    }
}
```

Interface

```
function getFullName(person: {
    firstName: string;
    lastName: string
}) {
    return `${person.firstName} ${person.lastName}`;
}

let person = {
    firstName: 'John',
    lastName: 'Doe'
};

console.log(getFullName(person));
```

Interface

```
interface Person {
    firstName: string;
    lastName: string;
}
```

```
function getFullName(person: Person) {
    return `${person.firstName} ${person.lastName}`;
}

let john = {
    firstName: 'John',
    lastName: 'Doe'
};

console.log(getFullName(john));
```

Interface: extend one interface

```
interface Mailable {
    send(email: string): boolean
    queue(email: string): boolean
}
```

```
later(email: string, after: number): void
```

```
interface FutureMailable extends Mailable {
    later(email: string, after: number): boolean
}
```

Generic

```
function getRandomNumberElement(items: number[]): number {
    let randomIndex = Math.floor(Math.random() * items.length);
    return items[randomIndex];
}
```

```
let numbers = [1, 5, 7, 4, 2, 9];
console.log(getRandomNumberElement(numbers));
```

```
function getRandomStringElement(items: string[]): string {
    let randomIndex = Math.floor(Math.random() * items.length);
    return items[randomIndex];
}
```

```
let colors = ['red', 'green', 'blue'];
console.log(getRandomStringElement(colors));
```

Generic

Using the any type

```
function getRandomAnyElement(items: any[]): any {
    let randomIndex = Math.floor(Math.random() * items.length);
    return items[randomIndex];
}
```

```
let numbers = [1, 5, 7, 4, 2, 9];
let colors = ['red', 'green', 'blue'];

console.log(getRandomAnyElement(numbers));
console.log(getRandomAnyElement(colors));
```

Generic

TypeScript Generic comes to rescue

```
function getRandomElement<T>(items: T[]): T {
    let randomIndex = Math.floor(Math.random() * items.length);
    return items[randomIndex];
}
```

```
let numbers = [1, 5, 7, 4, 2, 9];
let randomEle = getRandomElement(numbers);
console.log(randomEle);
```

```
let numbers = [1, 5, 7, 4, 2, 9];
let returnElem: string;
returnElem = getRandomElement(numbers);  // compiler error
```

Generic: constraints

```
function merge<U, V>(obj1: U, obj2: V) {
    return {
        ...obj1,
        ...obj2
    };
}
```

```
let person = merge(
    { name: 'John' },
    { age: 25 }
);

console.log(result);
```

```
{ name: 'John', age: 25 }
```

Generic: constraints

```
let person = merge(
    { name: 'John' },
    25
);

console.log(person);
```

```
{ name: 'John' }
```

Generic: constraints

```
function merge<U extends object, V extends object>(obj1: U, obj2: V) {
    return {
        ...obj1,
        ...obj2
    };
}
```

```
let person = merge(
    { name: 'John' },
    25
);
```

```
Argument of type '25' is not assignable to parameter of type 'object'.
```

Generic: interface

```
interface Pair<K, V> {
    key: K;
    value: V;
}
```

```
let month: Pair<string, number> = {
    key: 'Jan',
    value: 1
};

console.log(month);
```

Enum

Without enum

```
const statuses = {
  notStarted: 0,
  inProgress: 1,
  done: 2,
};

console.log(statuses.inProgress);
```

With enum

```
enum Status {
  NotStarted,
  InProgress,
  Done,
}

console.log(Status.InProgress);
```

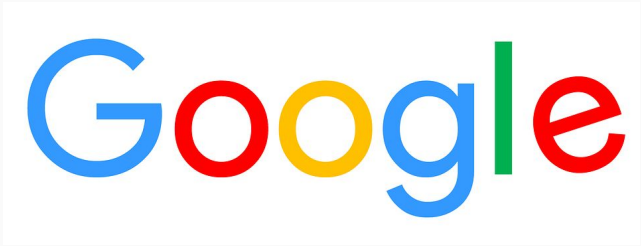
Enum

```
enum Status {
  NotStarted,
  InProgress,
  Done,
}

let notStrartedStatus: Status = Status.NotStarted;

notStrartedStatus = "foo";
                    [tsserver 2322] [E] Type '"foo"' is not assignable to type 'Status'.
```

Who use TypeScript?



Conclusion

- TypeScript simplifies JavaScript code making it easier to read and understand.
- It gives us all the benefits of ES6, plus more productivity.
- Help us avoid painful bugs by type checking.
- Structural, rather than nominal

Any Questions?

References

- prototype inheritance
- ES6 allowed you to define a class
- https://www.typescripttutorial.net/
- https://www.typescriptlang.org/
- TypeScript type annotations