**TypeScript**

**TypeScript**

Typescript is a typed superset of javacript that compiles to plain javascript.

It will always compile a typescript code into a javascript, even a invalid one.

Objects can be declared by using “Interface”.

**Why we should use typescript?**

TypeScript is a superset of JavaScript which primarily provides optional static typing, classes and interfaces. One of the big benefits is to enable IDEs to provide a richer environment for spotting common errors as you type the code.

To get an idea of what I mean, watch Microsoft's introductory video on the language.

For a large JavaScript project, adopting TypeScript might result in more robust software, while still being deployable where a regular JavaScript application would run.

**DATA TYPES:**

**Tuples:**

Tuple types allow you to express an array where the type of a fixed number of elements is known, but need not be the same. For example, you may want to represent a value as a pair of a string and a number.

**let** x: [string, number]; //

Initialize it x = ["hello", 10]; // OK

**ENUM**

**ANY**

**Null and Undefined**

By default null and undefined are subtypes of all other types. That means you can assign null and undefined to something like number.

However, when using the --strictNullChecks flag, null and undefined are only assignable to void and their respective types. This helps avoid many common errors.

As a note: we encourage the use of --strictNullChecks when possible, but for the purposes of this handbook, we will assume it is turned off.

**Fat arrow function**

<https://www.youtube.com/watch?v=Fu0HKDNl0rI>

Each optional property denoted by a  “?”

**Rest functions**

The compiler will build an array of the arguments passed in with the name given after the ellipsis (...), allowing you to use it in your function.

Void 0 and undefined is same.

**Interface**

One of TypeScript’s core principles is that type-checking focuses on the shape that values have. This is sometimes called “duck typing” or “structural subtyping”.

**CLASS:**

What is Class?

A class is an entity that determines how an object will behave and what the object will contain. In other words, it is a blueprint or a set of instruction to build a specific type of object.

**FEATURES OF CLASS:**

**I**NHERIATNCE:

class Animal {

move(distanceInMeters: number = 0) {

console.log(`Animal moved ${distanceInMeters}m.`);

}

}

class Dog extends Animal {

bark() {

console.log('Woof! Woof!');

}

}

const dog = new Dog();

dog.bark();

dog.move(10);

dog.bark();

-------------------------------------------------

class Animal {

name: string;

constructor(theName: string) { this.name = theName; }

move(distanceInMeters: number = 0) {

console.log(`${this.name} moved ${distanceInMeters}m.`);

}

}

class Snake extends Animal {

constructor(name: string) { super(name); }

move(distanceInMeters = 5) {

console.log("Slithering...");

super.move(distanceInMeters);

}

}

class Horse extends Animal {

constructor(name: string) { super(name); }

move(distanceInMeters = 45) {

console.log("Galloping...");

super.move(distanceInMeters);

}

}

let sam = new Snake("Sammy the Python");

let tom: Animal = new Horse("Tommy the Palomino");

sam.move();

tom.move(34);

Each derived class that contains a constructor function must call super()which will execute the constructor of the base class. The example also shows how to override methods in the base class with methods that are specialized for the subclass

**What are the Benefits of TypeScript?**

Ans. TypeScript has following benefits:

* It helps in code structuring
* Use class based object oriented programming
* Impose coding guidelines
* Offers type checking
* Compile time error checking
* Intellisense
* Cross-browser support
* Strict type checking
* Static typing

### Does TypeScript Support All Object Oriented Principles?

**Ans.** The answer is **YES**. There are 4 main principles to Object Oriented Programming: Encapsulation, Inheritance, Abstraction, and Polymorphism. TypeScript can implement all four of them with its smaller and cleaner syntax. Read [Write Object-Oriented JavaScript with TypeScript](http://rachelappel.com/write-object-oriented-javascript-with-typescript).

**Encapsulation**: It describes the idea of bundling data and methods that work on that data within one unit, e.g., a class in Java.

**Polymorphism**: Polymorphism means to process objects differently based on their data type. In other words it means, one method with multiple implementation, for a certain class of action. And which implementation to be used is decided at runtime depending upon the situation (i.e., data type of the object)

* Overloading in simple words means more than one method having the same method name that behaves differently based on the arguments passed while calling the method. This called static because, which method to be invoked is decided at the time of compilation
* Overriding means a derived class is implementing a method of its super class. The call to overriden method is resolved at runtime, thus called runtime polymorphism

**Abstraction**: Its main goal is to handle complexity by hiding unnecessary details from the user.

**What are the Disadvantages of TypeScript?**

Ans. Well, TypeScript is great but there are some disadvantages as well.

* TypeScript is just another way to write JavaScript. It doesn’t fix the problems of JavaScript. It just creates an illusion that it does.
* One more tool to learn.
* TypeScript has dependency on type definition files, if you wish to use any existing JavaScript libraries.
* Quality of type definition files is a concern as how can you be sure the definitions are correct?
* Frequent releases of new versions JavaScript library is also a pain area. Because if their type definition files are not updated then you can’t use them instantly.
* In order to run the application in the browser, a compile step is required to transform TypeScript into JavaScript.
* To use any third party library, definition file is you need. And not all the third party library have definition file available.

**DECLARE KEYWORD:** If we are using any variable or library which doesn’t belong to typescript

Then we use declare. The declare keyword is used for ambient declarations where you want to define a variable that may not have originated from a TypeScript file.

For example, lets imagine that we have a library called myLibrary that doesn’t have a TypeScript declaration file and have a namespace called myLibrary in the global namespace. If you want to use that library in your TypeScript code, you can use the following code:

**declare var myLibrary;**

**INTERVIEW QUESTION LINKS**

* [https://www.onlineinterviewquestions.com/typescript-interview-questions/#.W3I8Ps4zbIU](https://www.onlineinterviewquestions.com/typescript-interview-questions/%23.W3I8Ps4zbIU)
* <http://www.talkingdotnet.com/typescript-interview-questions/>