**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