

## React JS 🥰

=====

1. React JS is a Library
2. Using React we can Build Small Scale, medium and Complex or Enterprise Applications
3. React is used to develop only User Interface
4. React does not provide any features to implement routing, validations, api integration
5. React Application are fast in initial load

## Angular 🎉

=====

1. Angular is a Framework
2. Angular is preferred for Complex and enterprise app
3. Angular can develop entire frontend app
4. Angular Provides inbuilt functionalities to implement validation, api integration, design, routing etc
5. Angular app are slow in initial load

## Javascript 😊

=====

Javascript is a Loosely or Dynamic Typed, Synchronous, single-threaded Programming Language.

## Loosely Typed or Dynamic Types 👍

=====

JavaScript is called a loosely typed or dynamically typed language because you don't have to declare the type of a variable when you create it, and a variable can hold any type of data at any time.

### What does that mean?

```
let x = 10;      // x is a number
x = "hello";    // now x is a string
x = true;       // now x is a boolean
```

- The variable `x` changes type as needed — this is what makes it *loosely typed*.
- You don't need to say `int x = 10` like you would in a *strongly typed* language like Java or C++.

### Why is it useful?

- It makes coding faster and more flexible.
- You can write quick scripts without worrying about types.

But there's a catch:

It can lead to weird bugs or unexpected behavior if you're not careful. For example:

```
console.log("5" - 1); // 4 (JS converts "5" to 5)
console.log("5" + 1); // "51" (JS treats this as string concatenation)
```

This kind of type juggling is powerful but also a source of confusion — one of the reasons developers use TypeScript, which adds optional static typing on top of JavaScript.

## How is JavaScript Synchronous and Single Threaded?

By default, JavaScript is **synchronous** and **single-threaded**.

That means:

- It can only do one thing at a time.
- It executes code line by line, in the exact order it's written.
- It waits for each line to finish before moving on to the next.

 Example:

```
console.log("Start");  
console.log("Middle");  
console.log("End");
```

✓ Output:

```
Start  
Middle  
End
```

JavaScript runs this top to bottom, synchronously, like reading instructions step by step.

⚠ What's the catch?

Let's say you add a slow task, like this:

```
function wait() {  
  let start = Date.now();  
  while (Date.now() - start < 3000) {  
    // do nothing, just block  
  }  
}  
  
console.log("Start");  
wait(); // blocks for 3 seconds  
console.log("End");
```

🕒 Output:

```
Start  
(wait 3 seconds)  
End
```

Even if nothing is happening, JavaScript waits — because it's synchronous. It can't do anything else until that line finishes.

Moving from JavaScript (JS) to TypeScript (TS) has become increasingly popular in modern software development. TypeScript is a superset of JavaScript that adds static typing and other useful features, which makes it better suited for large, maintainable codebases.

Here are 4 sensible reasons developers move toward TypeScript from JavaScript,

## 1. Static Typing Reduces Bugs and Increases Code Safety 🙌

**Explanation:** JavaScript is dynamically typed, meaning you don't need to define the type of a variable. This can lead to unexpected runtime errors. TypeScript introduces static typing, allowing developers to define and check variable types at compile time, which helps catch errors earlier.

Example in JavaScript:

```
function add(a, b) {  
  return a + b;  
}  
  
console.log(add(5, '10'));  
  
// Output: "510", unexpected behavior
```

Example in TypeScript:

```
function add(a: number, b: number): number {  
  return a + b;  
}  
  
// console.log(add(5, '10')); // ❌ Error at compile time
```

Why it's better: In TS, the above code will throw a compile-time error because `'10'` is a string, not a number. This prevents unexpected bugs in production and helps ensure correctness in logic.

## 2. Better Developer Experience with Autocomplete and IntelliSense

**Explanation:** TypeScript provides better editor support thanks to type definitions. Modern IDEs (like VS Code) offer autocomplete, type hints, and documentation as you type, making development faster and more efficient.

Example:

```
interface User {  
  id: number;  
  name: string;  
  email: string;  
}  
  
const user: User = {  
  id: 1,  
  name: "Alex",  
  email: "alex@example.com"  
};  
  
console.log(user.email);  
// As you type 'user.', IDE will suggest 'id', 'name', and 'email'
```

Why it's better: This reduces the chance of typos, speeds up development, and improves confidence in the code. In JS, you'd have to constantly refer to docs or guess field names.

### 3. Scalability and Maintainability in Large Codebases

**Explanation:** As projects grow, JS code becomes harder to manage. TypeScript's strong typing and interfaces allow you to define contracts for functions, objects, and components. This helps teams maintain large codebases and onboard new developers easily.

**Example:**

```
interface Product {  
  id: number;  
  title: string;  
  price: number;  
}  
  
function displayProduct(product: Product): void {  
  console.log(` Title: ${product.title}, Price: ${product.price}` );  
}
```

Why it's better: If the structure of **Product** changes, TypeScript will alert you everywhere that depends on it. In JavaScript, these issues would only appear during runtime and could easily go unnoticed.



## 4. Early Detection of Errors with Compile-Time Checking

**Explanation:** JavaScript only throws errors at runtime. TypeScript catches many common mistakes before the code even runs, thanks to compile-time checks.

Example:

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

```
greet(123); // ✗ Error: Argument of type 'number' is not  
assignable to parameter of type 'string'
```

Why it's better: In JS, passing a number might cause a runtime error like **TypeError: name.toUpperCase is not a function**. In TS, it won't even compile unless the correct type is passed, preventing production issues.

