What are the typescript features that is not available for javascript

TypeScript is a **superset of JavaScript**, which means every valid JavaScript program is also valid TypeScript. However, TypeScript introduces several powerful **features not available in JavaScript** that help with static typing, better tooling, and safer development.

✅ Key TypeScript Features Not Available in JavaScript

Feature Category	TypeScript Feature	Description
🔡 Static Typing	`Type Annotations`	Declare types for variables, function parameters, return values, etc. Example: `let count: number = 5;`
🧱 Interfaces & Types	`interface`, `type`	Define custom type shapes for objects, functions, etc. Example: `interface User { name: string; age: number }`
🧬 Generics	`Generics`	Create reusable components that work with multiple types. Example: `function identity<T>(arg: T): T { return arg; }`
🛡️ Type Safety	`Type Inference`, `Type Guards`	TypeScript can infer types or narrow them down using logic (like `typeof`, `instanceof`).
🧰 Enums	`enum`	Declare a set of named constants. Example: `enum Direction { Up, Down }`
⚙️ Advanced Types	`Union`, `Intersection`, `Literal`, `Tuple`, `Mapped`, `Conditional`, `Utility` Types	Complex type operations. Example: `type A = "yes"

📦 Modules & Namespaces	`namespace` (deprecated)	TypeScript supports `namespace` and more structured module systems.
🏗️ Decorators	`@decorator` syntax	Experimental feature to annotate and modify classes and properties. JavaScript has only recently added experimental support.
🧠 Abstract Classes	`abstract class`	Define base classes with abstract methods. JavaScript does not have built-in support for abstract methods.
🔄 Compile-Time Checking	`Compile-Time Errors`	TypeScript can catch many bugs at compile time, unlike JavaScript which catches at runtime.
📝 Declaration Files	`.d.ts` Files	Define types for existing JavaScript libraries, enabling IntelliSense and type checking.
🧪 Readonly and Access Modifiers	`private`, `protected`, `readonly`, `public`	Enforce access restrictions on class members. JavaScript now has `#private`, but TS has stronger compile-time enforcement.
🧭 Optional Chaining + Nullish Coalescing with Types	Strong type checks with `?.` and `??`	JavaScript has these operators, but TypeScript adds type safety with them.
🔍 Type Aliases	`type SomeType = ...`	Create shorthand names for complex types.
🏛️ Structural Typing	Duck typing with structure-based compatibility	More rigidly enforced in TypeScript than in JavaScript.
🎭 Discriminated Unions	Tagged unions to handle different shapes in type-safe ways	Not natively supported in JS.
🏷️ Type Assertions	`as Type`, `<>`	Manually tell the compiler what type something is.
🔌 Compiler Features	`tsconfig.json`, strict mode options, etc.	Configure how TypeScript checks and compiles code.

⚖️ Overloads	Function overloads based on input type	Not possible in plain JS.

🚫 Not Available in JavaScript (Summary List)

- ✅ Static types: `string`, `number`, `boolean`, etc.
- ✅ Interfaces and `type` aliases
- ✅ Generics
- ✅ Enums
- ✅ Function overloads
- ✅ Abstract classes
- ✅ Access modifiers (`private`, `protected`)
- ✅ Type assertions
- ✅ Type inference and compile-time checking
- ✅ Utility types (`Partial<T>`, `Pick<T>`, etc.)
- ✅ Declaration files (`.d.ts`)
- ✅ Mapped, conditional, and template literal types
- ✅ Decorators (only available as a proposal in JavaScript)

⚠️ Note:

TypeScript **transpiles** all these features to standard JavaScript using the TypeScript compiler (`tsc`), so the output will still be valid JS.

If you’d like code examples for each feature, I can show you a side-by-side comparison (TS vs JS). Would you like that?