Strict Mode in JavaScript

What It Is

- Introduced in **ECMAScript 5 (ES5)**.
- Activated with the directive:

```
"use strict";
```

- Applies stricter parsing and error handling to your JavaScript code.

Key Effects

1. Variable Declarations

- Disallows using undeclared variables.

```
"use strict";
x = 10; // ReferenceError
```

2. Silent Errors Become Exceptions

- Assignments that normally fail silently will throw errors.

```
"use strict";
const obj = {};
Object.freeze(obj);
obj.prop = 123; // TypeError
```

3. `this` Behavior

- `this` is `undefined` in functions that are not methods.

```
"use strict";
function test() { console.log(this); }
test(); // undefined
```

4. Duplicate Parameters

- Duplicate function parameter names are disallowed.

```
"use strict";
function sum(a, a) { return a + a; } // SyntaxError
```

5. Reserved Keywords

- Protects future ECMAScript keywords (e.g., `implements`, `interface`).

Benefits

- Catches common coding mistakes early.
- Improves security by preventing accidental globals.
- Enables optimizations in JavaScript engines.

How to Use

- At the beginning of a file:

```
"use strict";
```

- Or inside a specific function:

```
function example() {
  "use strict";
  // strict mode only applies here
}
```

Strict vs Non-Strict Mode

Feature / Behavior	Non-Strict Mode	Strict Mode
Undeclared Variables	Creates global variable implicitly	Throws ReferenceError
Duplicate Parameters	Allowed	SyntaxError
`this` in Functions	Defaults to global object (`window`)	`undefined`
Assign to Read-Only	Fails silently	Throws TypeError
Delete Vars/Funcs	Allowed (fails silently)	SyntaxError
Octal Literals	Allowed (e.g., `010`)	SyntaxError
Reserved Keywords	Usable as identifiers	SyntaxError
Eval/Arguments	Can be reassigned/overwritten	Restricted

Best Practice

Always enable strict mode. (Modern tools like Babel, ES modules use it by default.)