



edureka!
a Veranda Enterprise



Full Stack Web Development Program



edureka!
a Veranda Enterprise



Day 24 - Strict Mode in JavaScript



Titles

- Strict Mode in JavaScript
- Advantages of Strict Mode
- Strict Mode for Entire Script
- Strict Mode for Functions
- Migrate JavaScript Codebase to Strict Mode
- Aspects of Strict Mode
- Default Parameters





Learning Objectives

By the end of this module, you will be able to:

- Describe Strict Mode in JavaScript
- Implement Default Parameters in JavaScript functions





Strict Mode in JavaScript

Strict Mode in JavaScript

- Introduced in ECMAScript version 5, strict mode modifies the semantics of JavaScript (JS) code for enhanced security.
- It causes JS code to behave differently, ensuring stricter parsing and error handling.
- Browsers lacking strict mode support execute strict mode code differently compared to those that support it.
- There are two ways to enable strict mode in JavaScript:
 - For an entire script
 - For a specific function



JS

< strict mode />

"use strict";

Advantages of Strict Mode

- **Error Detection and Prevention:** Mandates variable declarations using **var**, **let**, or **const** to avoid unintentional global variables.
- **Safer Code:** Prohibits problematic syntax, such as using **this** keyword to refer to the global object, for more reliable code.
- **Optimized Performance:** Enhances code execution speed and efficiency by reducing error handling overhead, with modern JavaScript engines like V8 and Chakra optimized for strict mode.
- **Enhanced Security:** Increases application security by limiting dangerous operations.
- **Improved Code Quality:** Promotes cleaner, more maintainable code, leading to higher quality projects.



Strict Mode for Entire Script

- Apply strict mode to a JavaScript script by placing `use strict` at the beginning.
- It directs the JavaScript engine to use a stricter rule set.
- Helps identify common coding errors and stop certain issues before they happen.

Syntax:

```
"use strict";  
/*  
    //Line of codes....  
*/
```

Example:

```
1 // Whole-script strict mode syntax  
2 "use strict";  
3 const v = "Hi! I'm a strict mode script!";
```


Strict Mode for Functions

- Strict mode in JavaScript can also be applied to individual functions by placing the **use strict** statement at the beginning of the function.
- It enforces strict mode rules within that function's scope only.
- Modules are in strict mode by default.
- **strict mode** does not apply to code written between braces (`{}`).

Example:

```
1 function myStrictFunction() {  
2     // Function-level strict mode syntax  
3     "use strict";  
4     function nested() {  
5         return "And so am I!";  
6     }  
7     return `Hi! I'm a strict mode function! ${nested()}`;  
8 }  
9 function myNotStrictFunction() {  
10     return "I'm not strict.";  
11 }
```

Migrate JavaScript Codebase to Strict Mode

- Gradually adopt strict mode by enabling it in smaller sections to pinpoint and resolve immediate problems.
- Utilize linting tools such as **ESLint** or **JSHint** for automatic detection of potential issues and to apply strict mode rules.
- Refactor code to eliminate `with` statements and make object references explicit to prevent problems.
- Exercise caution with the use of `this` as its behavior can be altered in strict mode.



Aspects of Strict Mode

- Here are the key aspects of using JavaScript's Strict Mode to enhance code reliability and prevent common errors:

Feature	Description
Errors for Undeclared Variables	Throws errors if variables are not properly declared before use.
Applies to Objects	Prevents initializing undeclared objects, as objects are considered variables.
Duplicate Function Parameters	Triggers an error for functions with parameters sharing the same name.
Deletion of Functions/Objects	Prohibits deleting functions or objects, enforcing stricter handling.
Restricted Keywords	Generates errors when reserved keywords are used as variable names, such as 'eval' or 'arguments'.

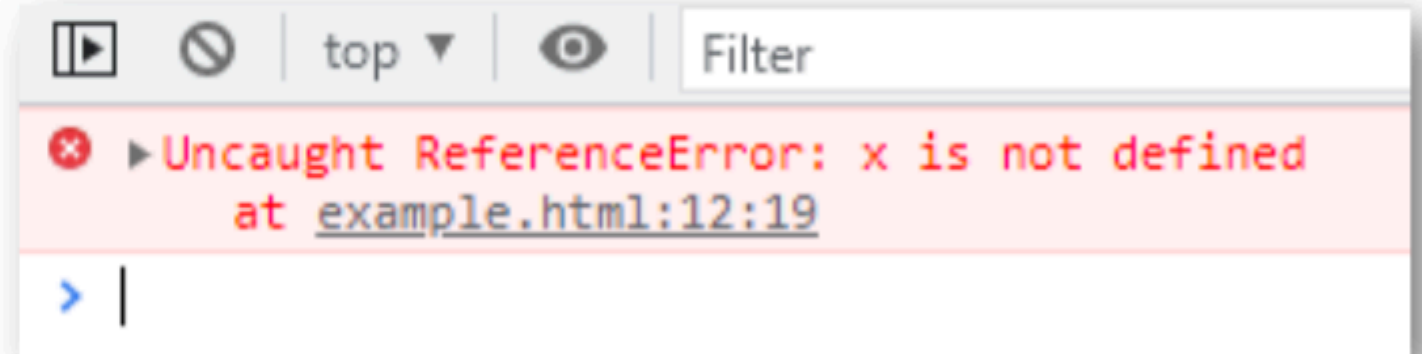
Example - I

Case - I: When using an undefined variable in JavaScript, it doesn't encounter any error. But after using use strict, it will encounter an error.

Code:

```
<script>  
    "use strict"  
    console.log(x);  
</script>
```

Output:



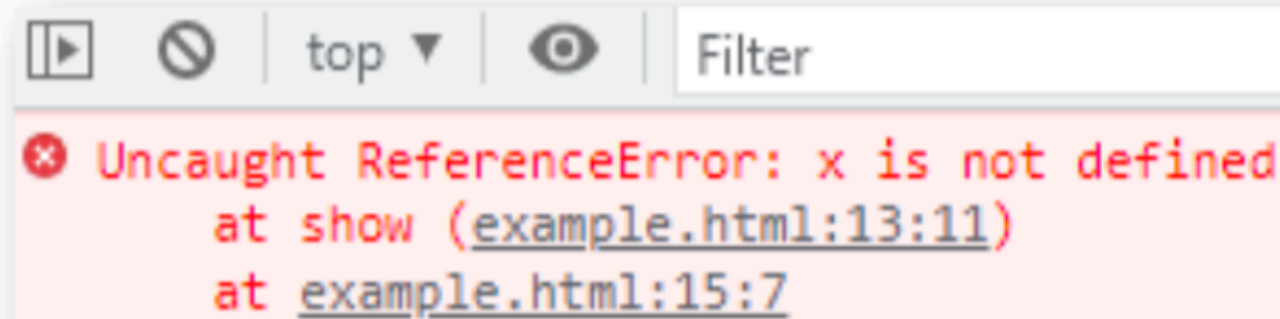
Example - 2

Case - 2: When using an undefined variable inside a function in JavaScript, it doesn't encounter any error. But after using **use strict**, it will encounter an error.

Code:

```
<script>
  "use strict"
  function show() {
    x = 10;    // This will also cause an error because x is not declared
  }
  show();
</script>
```

Output:



Uncaught ReferenceError: x is not defined
at show (example.html:13:11)
at example.html:15:7

Example - 3

Case - 3: Strict mode displays an error when using the same name for multiple parameters in function definitions.

Code:

```
1 "use strict";  
2  
3 function duplicateParam(x, x) {} // Error: Duplicate parameter name 'x'  
4  
5
```

Output:

Console

Uncaught SyntaxError: Duplicate parameter name not allowed in this context



Example - 4

Case - 4: Strict mode displays an error when using octal literals with a leading zero.

Code:

```
1  
2 "use strict";  
3  
4 var octalNumber = 0655; // Error: Octal literals are not allowed in strict mode  
5
```

Output:

Console

Uncaught SyntaxError: Octal literals are not allowed in strict mode.

Default Parameters

- Default parameters enable you to assign a predetermined value to a function's formal parameter when no value is provided during the function call.
- Introduced in ECMAScript 2015 (ES6) default parameters avoid undefined errors, enhance code functionality and user experience.

Syntax:

```
function fnName(param1 = defaultValue1, ..., paramN = defaultValueN)
{
    /*
    ...
    */
}
```

Example - I

Code:

```
<script>
  const sum = (a =5, b=2) => a + b;
  document.write("Sum using Default Params " +sum(7, 10)  + "</br>");
  document.write("Sum using Default Params " +sum(7)  + "</br>");
  document.write("Sum using Default Params " +sum()  + "</br>");
</script>
```

Output :

```
Sum using Default Params 17
Sum using Default Params 9
Sum using Default Params 7
```

Example - 2

Code:

```
2 function edureka_func(i = 33)
3 {
4     console.log(typeof i);
5     console.log("The value of i is: " + i);
6 }
7 edureka_func();    // outputs number and The value of a is: 5
8 edureka_func(undefined);    // output gives as 1
9 edureka_func('');    // outputs string and the value of a is:
10 edureka_func(null);    // outputs object and the value of a: null
```

Output :

Console

"number"

"The value of i is: 33"

"number"

"The value of i is: 33"

"string"

"The value of i is: "

"object"

"The value of i is: null"