

Java Script



Hiruni Kahawalage

Introduction

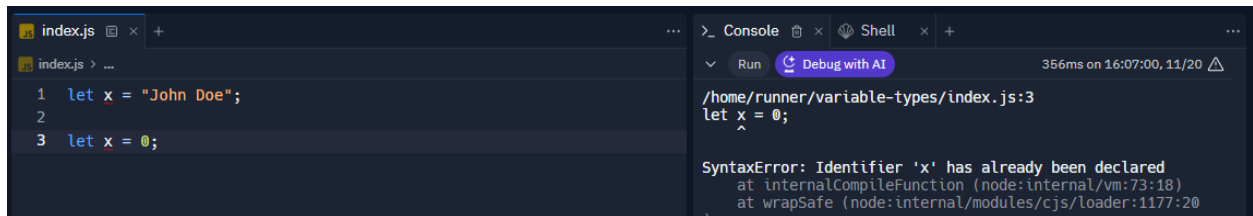
In JavaScript, variables are used to store and manage data in your programs. There are three keywords for declaring variables: `let`, `const`, and `var`. Each has its own purpose and rules. Let's explore them one by one.

1. let

- The let keyword was introduced in ES6(2015)
- Variables defined with let cannot be **Redeclared**.
- Variables defined with let must be **Declared** before use.
- Variables defined with let have **Block Scope**

➤ **let cannot be Redeclared.**

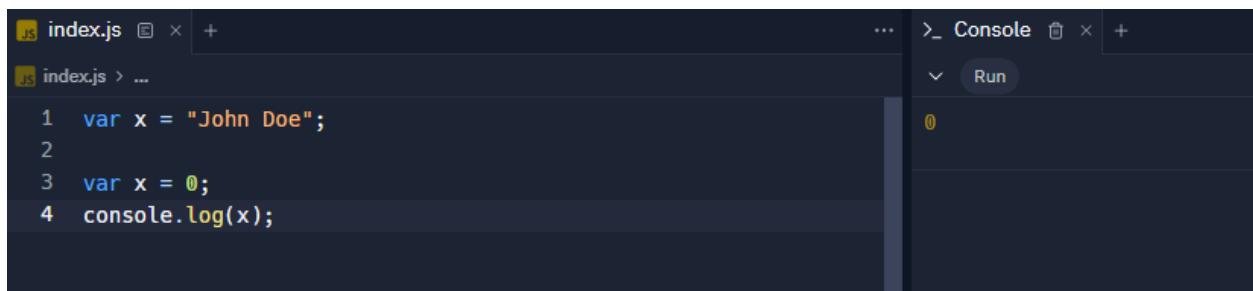
Variables defined with **let** cannot be redeclared.



```
index.js
1 let x = "John Doe";
2
3 let x = 0;
```

```
Console
Run Debug with AI 356ms on 16:07:00, 11/20
/home/runner/variable-types/index.js:3
let x = 0;
   ^
SyntaxError: Identifier 'x' has already been declared
    at internalCompileFunction (node:internal/vm:73:18)
    at wrapSafe (node:internal/modules/cjs/loader:1177:20)
```

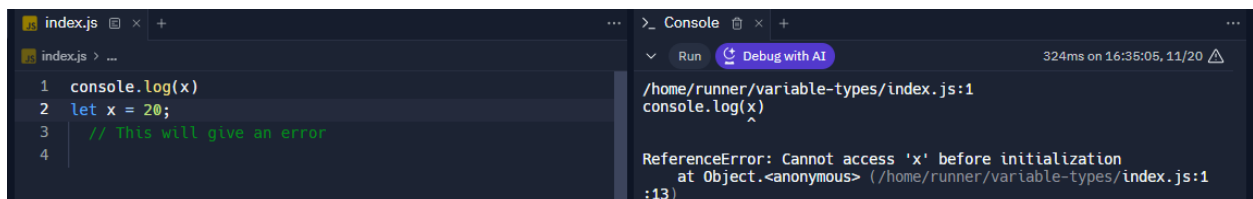
With var, we can be redeclared..



```
index.js
1 var x = "John Doe";
2
3 var x = 0;
4 console.log(x);
```

```
Console
Run 0
```

➤ **Initialization**

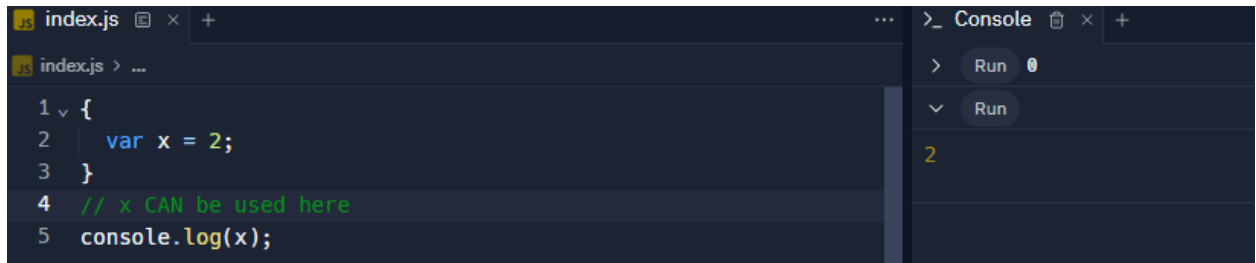


```
index.js
1 console.log(x)
2 let x = 20;
3 // This will give an error
4
```

```
Console
Run Debug with AI 324ms on 16:35:05, 11/20
/home/runner/variable-types/index.js:1
console.log(x)
         ^
ReferenceError: Cannot access 'x' before initialization
    at Object.<anonymous> (/home/runner/variable-types/index.js:1:13)
```

2. var

- Variables declared with the var keyword can NOT have block scope.
- Variables declared inside a { } block can be accessed from outside the block.

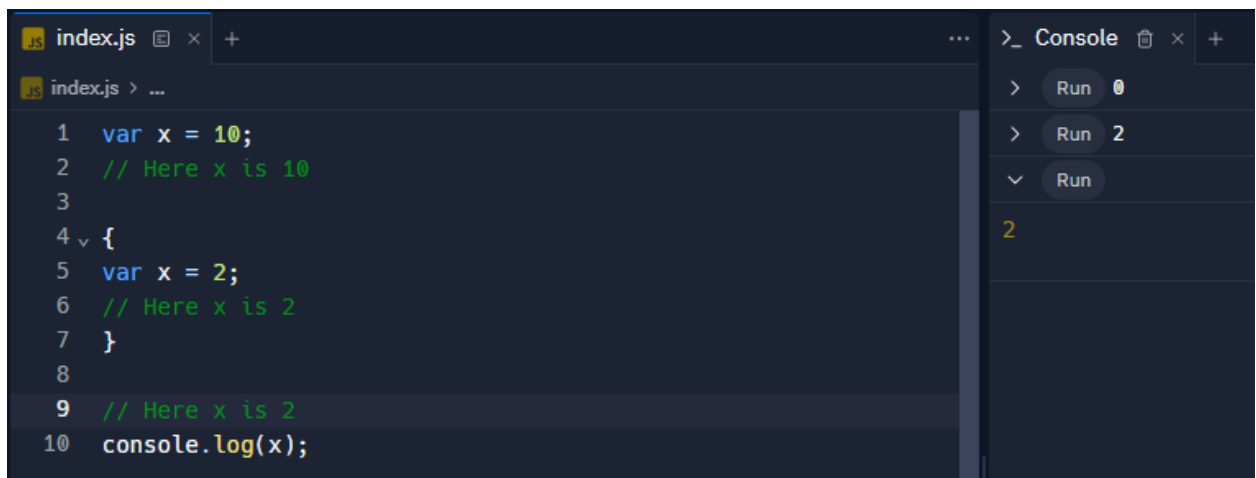


```
index.js 1 {  
2   var x = 2;  
3 }  
4 // x CAN be used here  
5 console.log(x);
```

The console shows the output: 2

➤ Redeclaring variables using var.

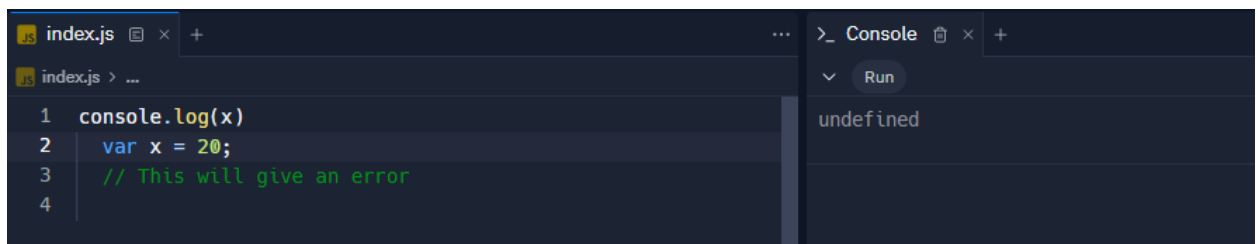
- Redeclaring a variable using the var keyword can impose problems.
- Redeclaring a variable inside a block will also redeclare the variable outside the block:



```
index.js 1 var x = 10;  
2 // Here x is 10  
3  
4 {  
5   var x = 2;  
6   // Here x is 2  
7 }  
8  
9 // Here x is 2  
10 console.log(x);
```

The console shows the output: 2

➤ Initialization



```
index.js 1 console.log(x)  
2   var x = 20;  
3   // This will give an error  
4
```

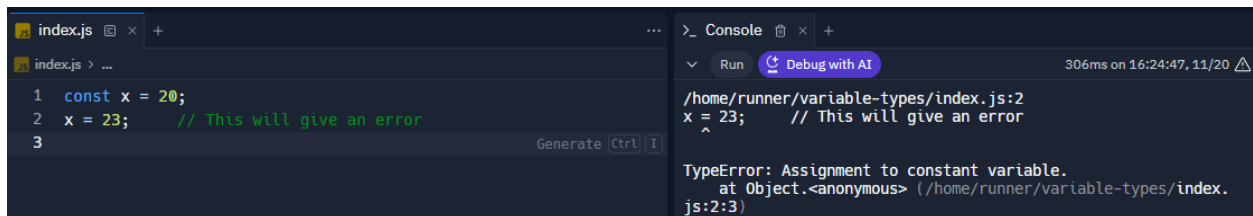
The console shows the output: undefined

3. Const

- The const keyword was introduced in ES6 (2015)
- Variables defined with const cannot be Redeclared.
- Variables defined with const cannot be Reassigned.
- Variables defined with const have Block Scope.

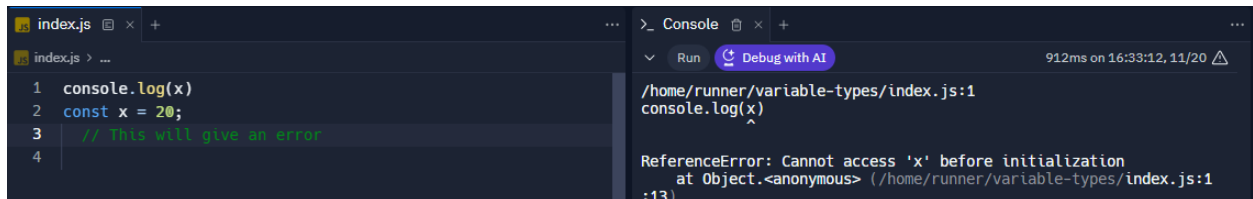
➤ Cannot be reassigned.

- A const variable cannot be reassigned.



The screenshot shows a code editor with a file named `index.js`. The code contains two lines: `const x = 20;` on line 1 and `x = 23; // This will give an error` on line 2. The console on the right shows the execution result: `/home/runner/variable-types/index.js:2`, `x = 23; // This will give an error`, followed by a red error message: `TypeError: Assignment to constant variable.` at `Object.<anonymous> (/home/runner/variable-types/index.js:2:3)`.

➤ Initialization



The screenshot shows a code editor with a file named `index.js`. The code contains three lines: `console.log(x)` on line 1, `const x = 20;` on line 2, and `// This will give an error` on line 3. The console on the right shows the execution result: `/home/runner/variable-types/index.js:1`, `console.log(x)`, followed by a red error message: `ReferenceError: Cannot access 'x' before initialization` at `Object.<anonymous> (/home/runner/variable-types/index.js:1:13)`.

Summarization:

Difference Between var, let and const

	Scope	Redeclare	Reassign
var	No	Yes	Yes
let	Yes	No	Yes
const	Yes	No	No