# **Variable Shadowing in JavaScript**

* Last Updated : 08 Mar, 2021

**Block Scoping:** To understand shadowing in JavaScript, we need to be clear with the scope first. In computer programming languages, **Scope** is a certain section/region of the program where a defined variable can have its existence and can be recognized, beyond that it can’t be accessed. In JavaScript, a **Block** is a compound statement that is defined by curly braces {} and used to combine multiple statements into one statement where JavaScript expects only one statement. And all the variables and functions that can be accessed inside a block are said to be inside that block scope, hence called **Block scoped.**

For example, *let* and *const* variables are stored in separate memory space, so it is called block-scoped but *var* variables can be accessed outside the block as it is stored in the Global object memory space, hence it is called **Global scoped.**

**Shadowing:** Now, when a variable is declared in a certain scope having the same name defined on its outer scope and when we call the variable from the inner scope, the value assigned to the variable in the inner scope is the value that will be stored in the variable in the memory space. This is known as **Shadowing or Variable Shadowing**. In JavaScript, the introduction of *let* and *const* in ECMAScript 6 along with block scoping allows variable shadowing.

**Example:**

* Javascript

| function func() {  let a = 'Geeks';    if (true) {  let a = 'GeeksforGeeks'; // New value assigned  console.log(a);  }    console.log(a);  func(); |
| --- |

**Output:**

GeeksforGeeks

Geeks

**Illegal Shadowing:** Now, while shadowing a variable, it should not cross the boundary of the scope, i.e. we can shadow *var* variable by *let* variable but cannot do the opposite. So, if we try to shadow *let* variable by *var* variable, it is known as **Illegal Shadowing** and it will give the error as *“variable is already defined.”*

**Example:**

* Javascript

| function func() {  var a = 'Geeks';  let b = 'Geeks;    if (true) {  let a = 'GeeksforGeeks'; // Legal Shadowing  var b = 'Geeks'; // Illegal Shadowing  console.log(a); // It will print 'GeeksforGeeks'  console.log(b); // It will print error  }  }  func(); |
| --- |

**Output:**

Identifier 'b' has already been declared

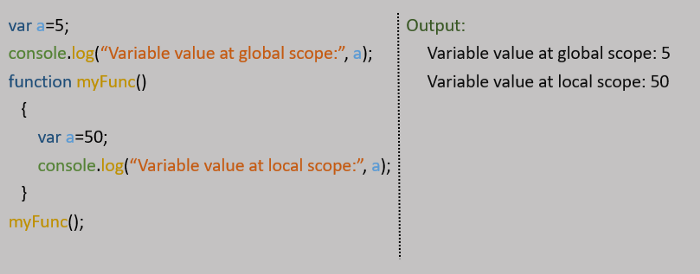
**Note:** Arrow functions also follow the same scope and variable shadowing rule.

# Variable Shadowing:

In programming languages, variable shadowing occurs when a variable is declared in certain scope has the same name defined on it’s outer scope. And when we call the variable from the inner scope, the value mentioned in inner scope will be taken into account.

Here is an example for the variable shadowing.





In the above code, we’ve declared two different values for a same variable **“a”**. Since, variable **“a”** was declared with a new value inside **myFunc()**, system will fetch that value for any operations performed inside that particular scope irrespective of global value.

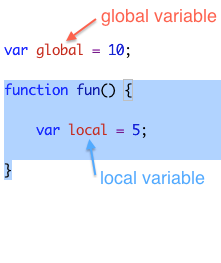
*This is called as* ***“Variable Shadowing”****.*

# Scope:

In JS there are two types of scope:

* Local scope
* Global scope





Variables which are declared inside a function can’t be used outside of the function. Sometimes system may return some garbage, but the exact defined value can’t be obtained.

-Variables defined **outside** a function are called **global variables**.

-Variables defined **within** a function are **local variables**.