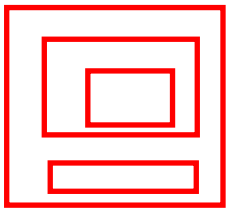
Cheatsheets / **Learn JavaScript**

Scope defines the positions in the program where a particular variable can be accessed.

codecademy

Scope



Global Scope -> Global variables
Block Scope -> Local variables

The Block Scope of JS is a powerful tool to precisely define the scope of variables

Scope

Scope is a concept that refers to where values and functions can be accessed.
Various scopes include:

- *Global* scope (a value/function in the global scope can be used anywhere in the entire program)
- *File* or *module* scope (the value/function can only be accessed from within the file)
- *Function* scope (only visible within the function),
- *Code block* scope (only visible within a `{ ... }` codeblock)

```
function myFunction() {

  var pizzaName = "Volvo";
  // Code here can use pizzaName

}

// Code here can't use pizzaName
```

Block Scoped Variables

`const` and `let` are *block scoped* variables, meaning they are only accessible in their block or nested blocks. In the given code block, trying to print the `statusMessage` using the `console.log()` method will result in a `ReferenceError`. It is accessible only inside that `if` block.

```
const isLoggedIn = true;

if (isLoggedIn == true) {
  const statusMessage = 'User is logged
in.';
}

console.log(statusMessage);

// Uncaught ReferenceError:
statusMessage is not defined
```

Global Variables

JavaScript variables that are declared outside of blocks or functions can exist in the *global scope*, which means they are accessible throughout a program. Variables declared outside of smaller block or function scopes are accessible inside those smaller scopes.

Note: It is best practice to keep global variables to a minimum.

Scope pollution: Too many global variables
=> name clashes, eg. between global and local variables
Global variables introduce the possibility of "side effects", functions that alters the value of a global variable, though it is not its main function.

Solution: TIGHT COUPLING (of variables and functions that deal with them

```
// Variable declared globally
const color = 'blue';

function printColor() {
  console.log(color);
}

printColor(); // Prints: blue
```

Search Results
Featured snippet from the web
In JavaScript, variables with the same name can be specified at multiple layers of nested scope. In such case local variables gain priority over global variables. If you declare a local variable and a global variable with the same name, the local variable will take precedence when you use it inside a function