**Everything you wanted to know about JavaScript scope**

[**What is Scope?**](https://toddmotto.com/everything-you-wanted-to-know-about-javascript-scope/#what-is-scope)

In JavaScript, scope refers to the current context of your code. Scopes can be *globally* or *locally* defined. Understanding JavaScript scope is key to writing bulletproof code and being a better developer. You’ll understand where variables/functions are accessible, be able to change the scope of your code’s context and be able to write faster and more maintainable code, as well as debug much faster.

Thinking about scope is easy, are we inside Scope A or Scope B?

[**What is Global Scope?**](https://toddmotto.com/everything-you-wanted-to-know-about-javascript-scope/#what-is-global-scope)

Before you write a line of JavaScript, you’re in what we call the Global Scope. If we declare a variable, it’s defined globally:

// global scope

var name = 'Todd';

Global scope is your best friend and your worst nightmare, learning to control your scopes is easy and in doing so, you’‘ll run into no issues with global scope problems (usually namespace clashes). You’ll often hear people saying “Global Scope is *bad*”, but never really justifying as to *why*. Global scope isn’t bad, you need it to create Modules/APIs that are accessible across scopes, you must use it to your advantage and not cause issues.

### [**What is Local Scope?**](https://toddmotto.com/everything-you-wanted-to-know-about-javascript-scope/#what-is-local-scope)

A local scope refers to any scope defined past the global scope. There is typically one global scope, and each function defined has its own (nested) local scope. Any function defined within another function has a local scope which is linked to the outer function.

If I define a function and create variables inside it, those variables becomes locally scoped. Take this example:

// Scope A: Global scope out here

var myFunction = function () {

// Scope B: Local scope in here

};

Any locally scoped items are not visible in the global scope - unless exposed, meaning if I define functions or variables within a new scope, it’s inaccessible outside of that current scope. A simple example of this is the following:

var myFunction = function () {

var name = 'Todd';

console.log(name); // Todd

};

// Uncaught ReferenceError: name is not defined

console.log(name);

### [**Lexical Scope**](https://toddmotto.com/everything-you-wanted-to-know-about-javascript-scope/#lexical-scope)

Whenever you see a function within another function, the inner function has access to the scope in the outer function, this is called Lexical Scope or Closure - also referred to as Static Scope. **The easiest way to demonstrate that again:**

**Example 1:**

**// Scope A**

**var myFunction = function () {**

**// Scope B**

**var name = 'Todd'; // defined in Scope B**

**var myOtherFunction = function () {**

**// Scope C: `name` is accessible here!**

**};**

**};**

**Example 2:**

**var name = 'Todd';**

**var scope1 = function () {**

**// name is available here**

**var scope2 = function () {**

**// name is available here too**

**var scope3 = function () {**

**// name is also available here!**

**};**

**};**

**};**

**Example 3:**

**// name = undefined**

**var scope1 = function () {**

**// name = undefined**

**var scope2 = function () {**

**// name = undefined**

**var scope3 = function () {**

**var name = 'Todd'; // locally scoped**

**};**

**};**

**};**

**Questions:**

1.   In the following program skeleton, fill in the blank line where it would be correct to have the statement: target = 25;

Top of Form

function a

{

let target;

function cMethod()

{

target = 25;

}

function aMethod()

{

target = 25;

}

function bMethod()

{

target = 25;

}

}

function b

{

let sum;

function aMethod() ()

{

}

function anotherMethod()

{

}

function someMethod()

{

}

}

function c()

{

function aMethod()

{

}

}

2. In the following program skeleton fill in the blank line where it would be OK to have the statements target = 25 and/or sum=25

Top of Form

function a()

{

let target;

function cMethod() ()

{

target = 25;

}

function aMethod()

{

target = 25;

}

function bMethod()

{

target = 25;

}

}

function b()

{

let sum;

function aMethod()

{

sum=25

}

function someMethod()

{

sum=25

}

}

function c()

{

function aMethod() {

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

3. Fill in the line where it would be OK to have the statement alert( sum + ””);

Top of Form

function a()

{

let sum;

function aMethod()

{

alert( sum + ””);

}

function bMethod()

{

alert( sum + ””);

}

}

function b()

{

function aMethod()

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

4. 4. Fill in the Fissss Fill in the line where it would be OK to have the statement value = 5;

Top of Form

function a()

{

let sum;

function aMethod()

{

let value;

value = 5;

}

function bMethod()

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

function cMethod( )

{

­­­­\_\_\_\_\_\_\_\_\_\_

let value;

}

function dMethod( )

{

let value;

value = 5;

}

}

5. 4. Fill in the Fissss Fill in the line where it would be OK to have the statement value = 5;

let value;

Top of Form

function a()

{

let sum;

function aMethod()

{

value = 5;

}

function bMethod()

{

value = 5;

}

function cMethod( )

{

value = 5;

}

function dMethod( )

{

value = 5;

}

}

6. 5. Decide if each statements sets the **global, lexical/static** or the **local variable** sum.

Top of Form

function a()

{

let sum;

function aMethod()

{

sum = 2 ; // lexical

}

function bMethod()

{

let sum; // local

sum = 3 ; // local

}

function cMethod( )

{

sum = 23 ; // lexical

let sum;// local

}

}

7. 6. In the following program skeleton, decide if each statements sets the **global lexical or**  **local variable** sum.

let sum; // global

Top of Form

function a()

{

function aMethod()

{

let sum; // local

sum = 5; // local

}

function bMethod()

{

sum = 6 ; // global

}

}

Bottom of Form

Bottom of Form

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Bottom of Form