JavaScript

Module 2 - JavaScript Basics

Chapter 3 - Variables

Javascript Identifiers

All JavaScript variables must be identified with unique names. These unique names are called identifiers.

Identifiers are used to name variables and keywords, and functions.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Note: JavaScript identifiers are case-sensitive.

ie, **lastName** and **lastname,** are two different identifiers **.**

Javascript Names

The general rules for constructing names for variables:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter, $ and _
- Names are case sensitive (y and Y are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.

A JavaScript name must begin with:

A letter (A-Z or a-z) (Ex: showmsg or Showmsg)

A dollar sign ($) (Ex: $showmsg)

Or an underscore (_) (Ex: _showmsg)

Subsequent characters may be letters, digits, underscores, or dollar signs.

Note:-

Numbers are not allowed as the first character in names.

Javascript Names

<u>Not Allowed:-</u>

- Hyphens are not allowed in JavaScript. They are reserved for subtractions. (Ex:show-msg)

<u>Allowed:-</u>

1. Underscore:

   first_name, last_name, master_card, inter_city.

1. Upper Camel Case (Pascal Case):

   FirstName, LastName, MasterCard, InterCity.

1. Lower Camel Case:

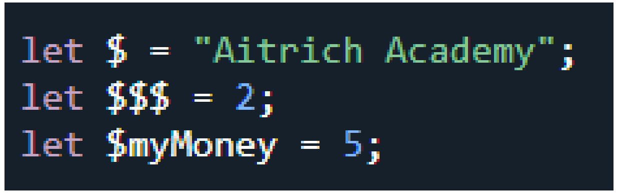
   firstName,lastName, masterCard, interCity.

Javascript Names

Dollar Sign $:-

Since JavaScript treats a dollar sign as a letter, identifiers containing $ are valid variable names.

Example:

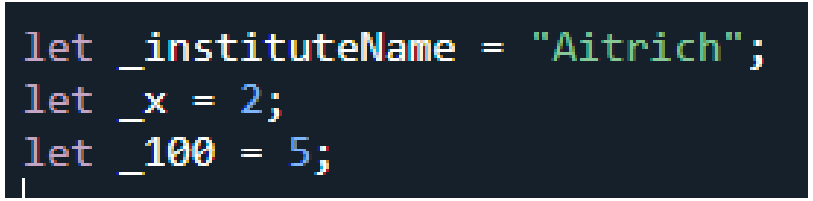


```
let $ = "Aitrich Academy";
let $$$ = 2;
let $myMoney = 5;
```

JavaScript Underscore (_)

Since JavaScript treats underscore as a letter, identifiers containing _ are valid variable names:

Example:



```
let _instituteName = "Aitrich";
let _x = 2;
let _100 = 5;
```

Javascript Keywords

JavaScript statements often start with a keyword, to identify the JavaScript action to be performed.

JavaScript keywords are reserved words.

Reserved words cannot be used as names for variables.

Example:-

var,let,const,if, for,switch,function,return,try

Javascript Values

The JavaScript syntax defines two types of values:

1. **Fixed values (Literals)**
2. **Variable values (Variables)**

Fixed values are called **Literals**.
Variable values are called **Variables**.

<u>JavaScript Literals:-</u>

1. **Numbers** (are written with or without decimals.)
   Ex:-10.50 or 1001
2. **Strings** (are text, written within double or single quotes.)
   Ex:-"Aitrich Academy" or **'Aitrich Academy'**

Javascript Variables

Variables are Containers for Storing Data

They serve as symbolic names for values and allow developers to work with data in a dynamic and flexible manner.

Javascript Variables

Declaring a JavaScript Variable

Creating a variable in JavaScript is called "declaring" a variable.

Ex:- `var jobName;` or `let jobName;`

Assigning a JavaScript Variable

After the declaration, the variable has no value (technically it is undefined).

To assign a value to the variable, use the equal sign: **Ex:** `jobName = "Manager";`

You can also assign a value to the variable when you declare it.

Ex: `let jobName = “Manager";`

Note: **It's a good programming practice to declare all variables at the beginning of a script.**

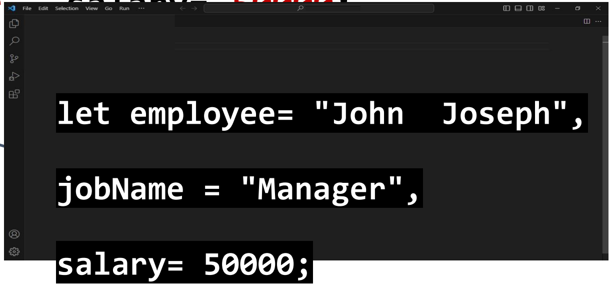
Declaring a Javascript Variable

You can declare many variables in one statement.

Start the statement with `let` and separate the variables by comma:

Ex: `let employee= "John Joseph", jobName = "Manager", salary= 50000;`

A declaration can span multiple lines:



```
let employee= "John  Joseph",
jobName = "Manager",
salary= 50000;
```

A variable declared without a value will have the value `undefined`.

Ex: `let jobName;`

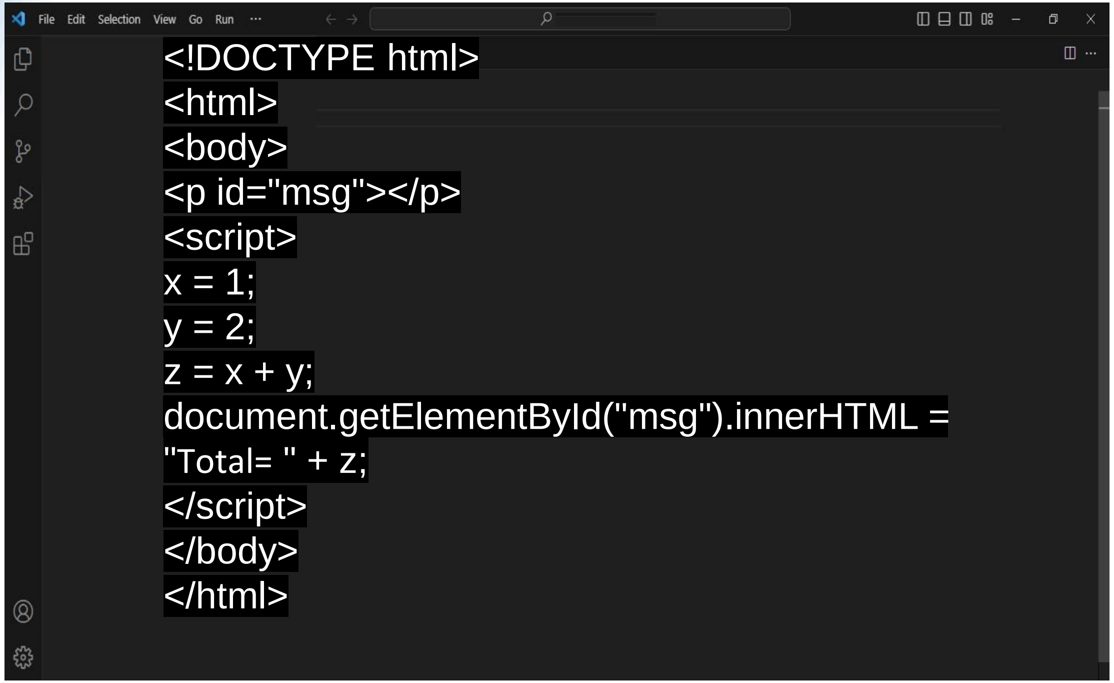
The variable `jobName` will have the value `undefined` after the execution of this statement.

Ways to Declare a Javascript Variable

JavaScript Variables can be declared in 4 ways:

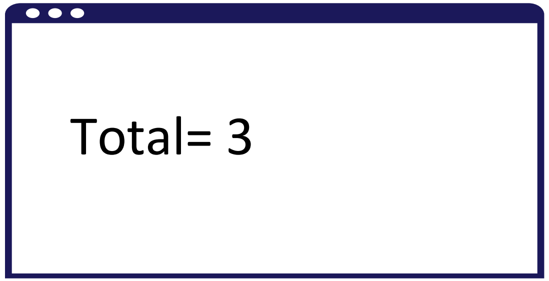
1. **Automatically declared when first used**
2. **Using** `var`
3. **Using** `let`
4. **Using** `const`

1. Automatically Declared when first used



```
<!DOCTYPE html>
<html>
<body>
<p id="msg"></p>
<script>
x = 1;
y = 2;
z = x + y;
document.getElementById("msg").innerHTML =
"Total= " + z;
</script>
</body>
</html>
```

Output

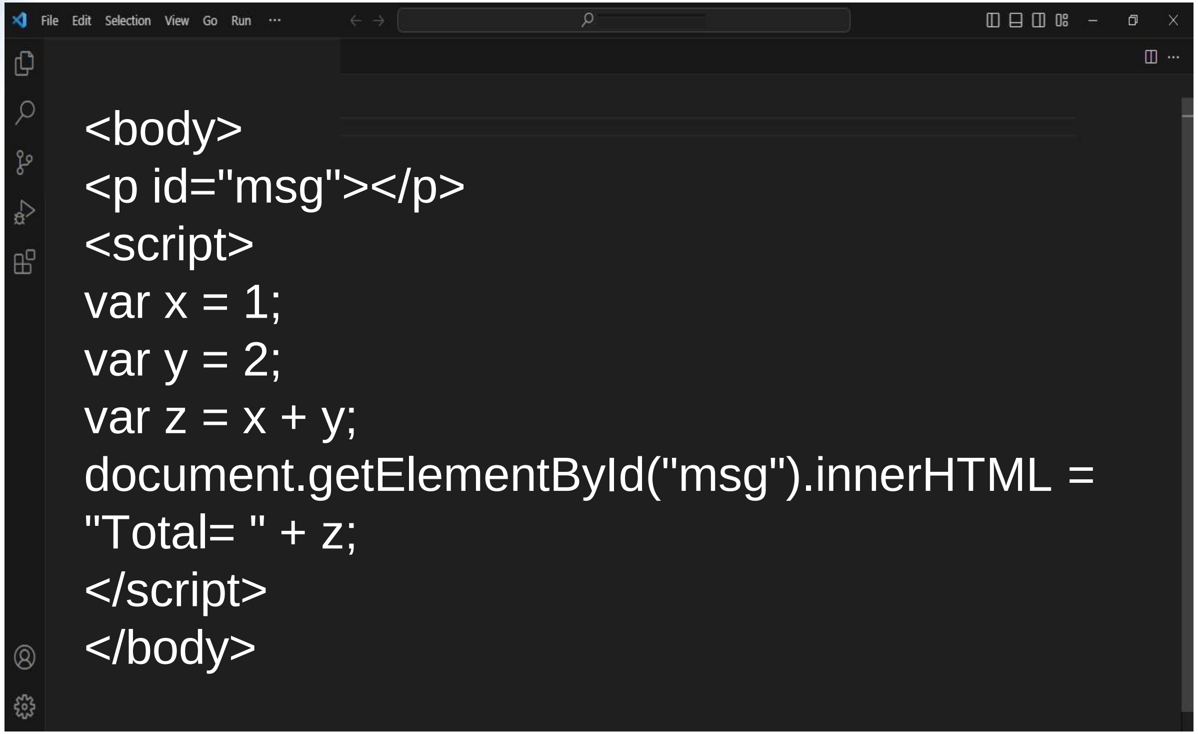


```
Total= 3
```

Always declare variables

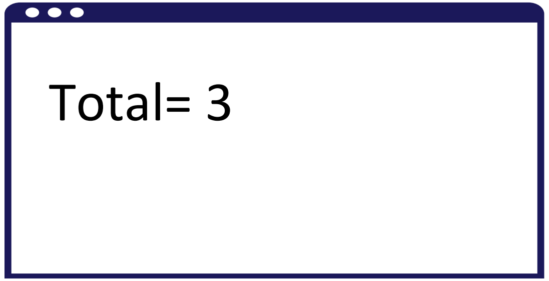
It is considered good programming practice to always declare variables before use.

2. Using **var** to declare variable



```
<body>
<p id="msg"></p>
<script>
var x = 1;
var y = 2;
var z = x + y;
document.getElementById("msg").innerHTML =
"Total= " + z;
</script>
</body>
```

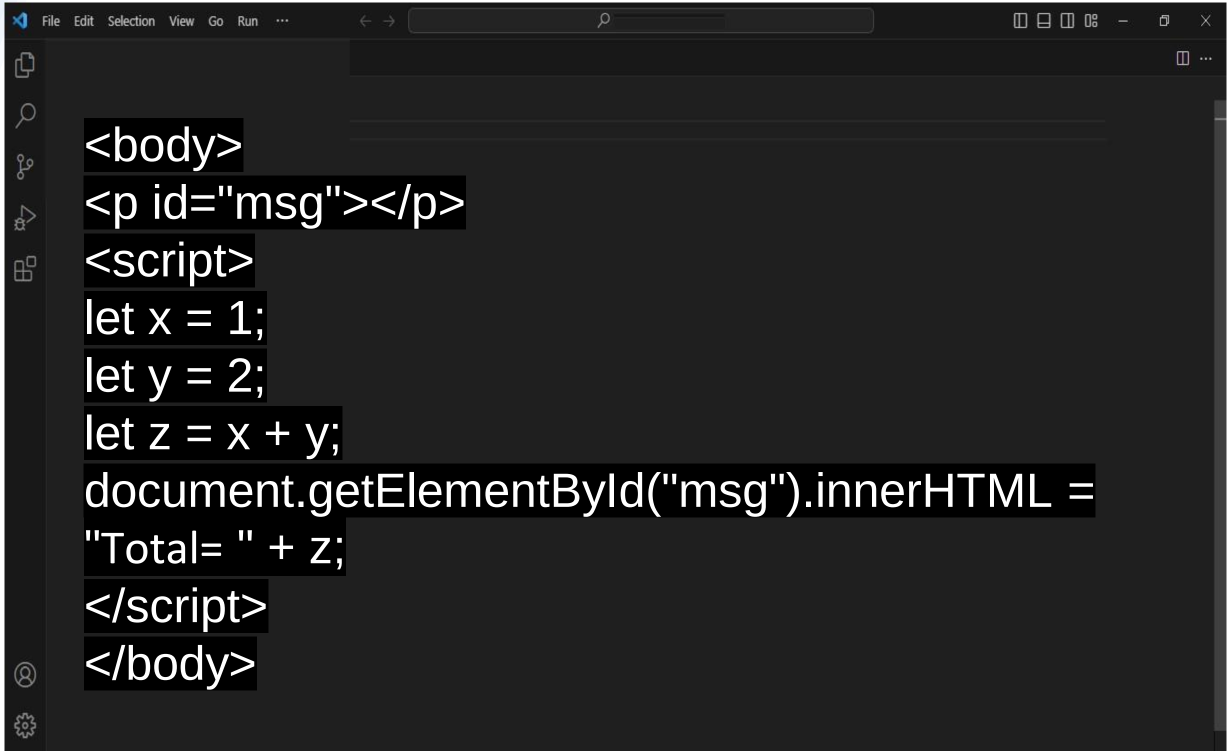
Output



```
Total= 3
```

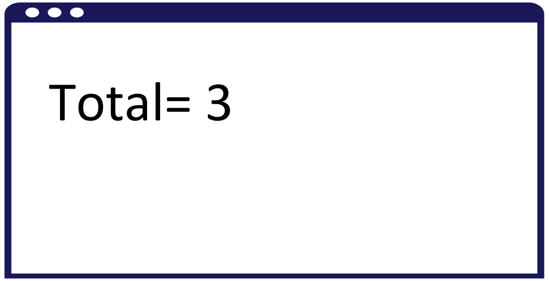
Aitrich

3. Using let to declare variable



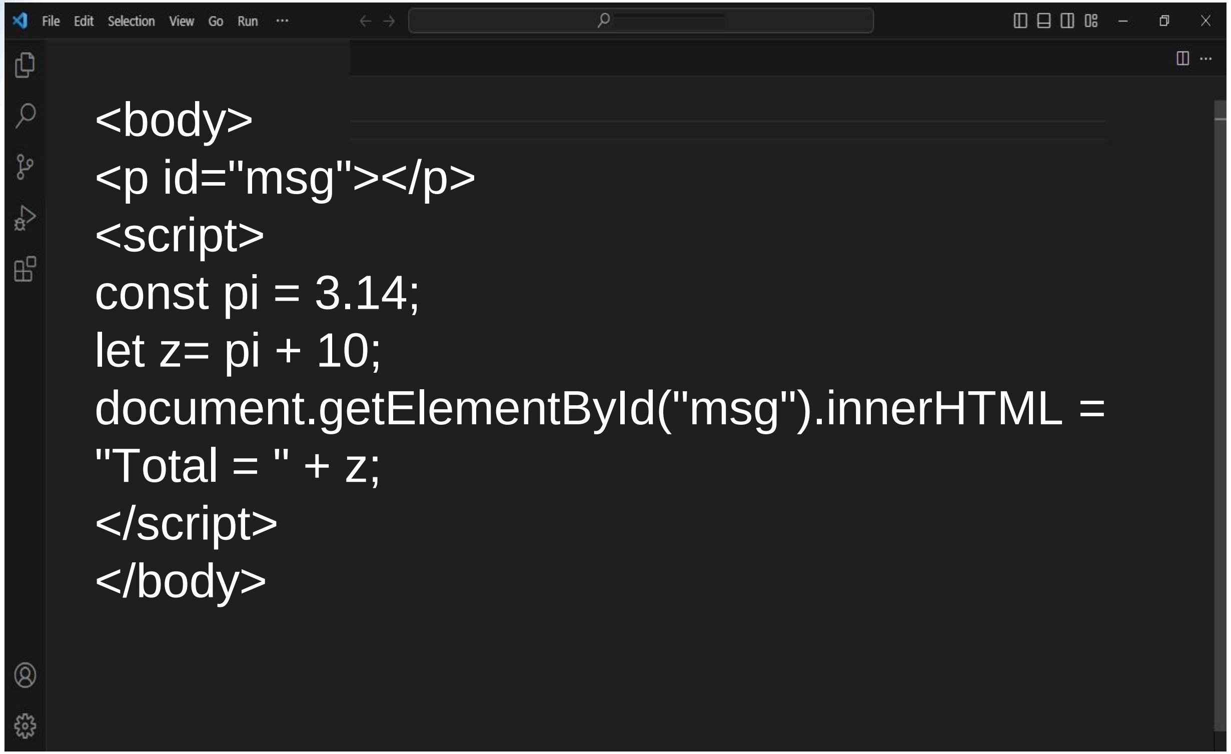
```
<body>
<p id="msg"></p>
<script>
let x = 1;
let y = 2;
let z = x + y;
document.getElementById("msg").innerHTML =
"Total= " + z;
</script>
</body>
```

Output



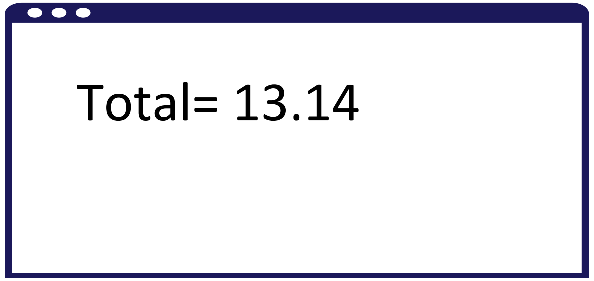
```
Total= 3
```

4. Using const to declare variable



```
<body>
<p id="msg"></p>
<script>
const pi = 3.14;
let z= pi + 10;
document.getElementById("msg").innerHTML =
"Total = " + z;
</script>
</body>
```

Output



```
Total= 13.14
```

When to use Var, let or const?

- If you never want the value of a variable to change, const is the keyword to use.
- Also, always use const if the type should not be changed (Arrays and Objects).
- If you want to reassign values,and you want the hoisting behavior, var is the keyword to use.
- If you don't want it, and if you can't use const, let is the keyword for you.

Use const when you declare:

- A new Array
- A new Object
- A new Function
- A new RegExp

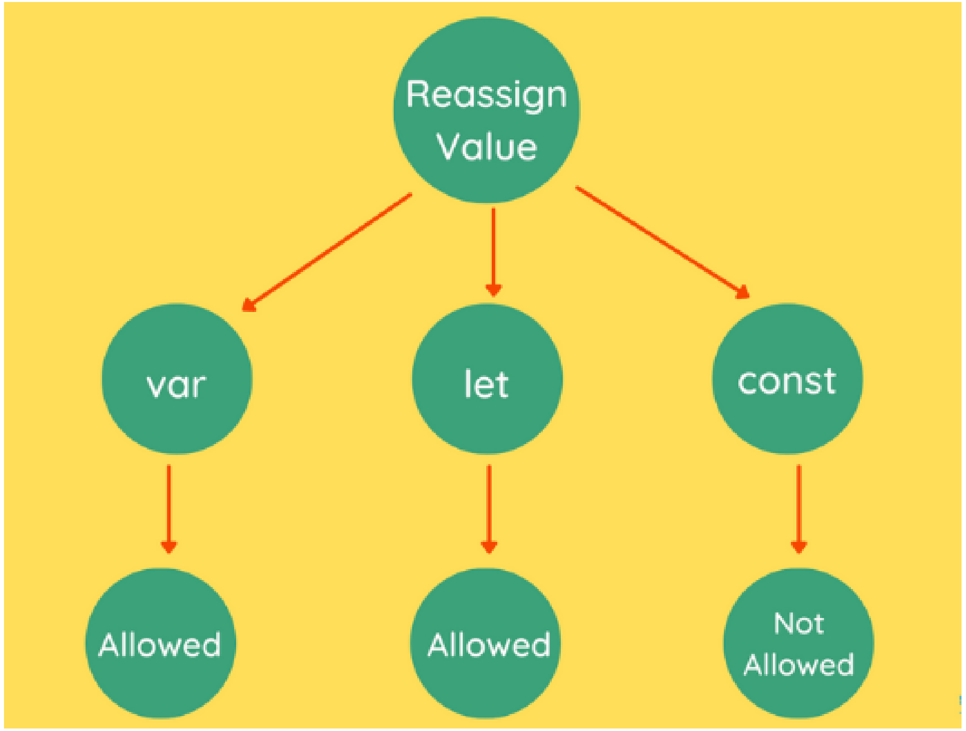
Reassigning variable value

- **var** and **let** create variables that can be reassigned to another value.
- **const** creates "constant" variables that cannot be reassigned to another value, in the same scope.

```
// Declare variables alongwith assigning initial values
let f_name = "Alex";
const ZIP = 560089;
var age = 25;
const PI = 3.141592653589793;

// Reassign values
f_name = "John"; // the f_name value is John.
ZIP = 65457; // Uncaught TypeError: Assignment to constant variable.
age = 78; // the age value is 78
PI = 3.14;      // This will give an error
PI = PI + 10;   // This will also give an error
```

Reassigning variable value



Re-Declaring Javascript Variables

You CAN re-declare a variable declared with var.

With var you CAN do this:

```
Var employee = "John Joseph";

Var employee = 0;
```

```
var jobName = "Manager";

var jobName;
```

If you re-declare a JavaScript variable declared with var, it will not lose its value.

The variable jobName will still have the value "Manager" after the execution of these statements.

Re-Declaring Javascript Variables

You cannot re-declare a variable declared with `let` or `const`.

These will not work:

```
let jobName = "Manager";

let jobName;
```

```
let employee = "John Joseph";
let employee = 0;
```

```
const PI = 3.14159265359;

const PI = 3.14;
```

Note: const variables **must be assigned** a value when they are declared

Correct

```
const PI = 3.14159265359;
```

Incorrect

```
const PI;
PI = 3.14159265359;
```

Javascript Scope

Scope determines the accessibility of variables, objects, and functions from different parts of the code.

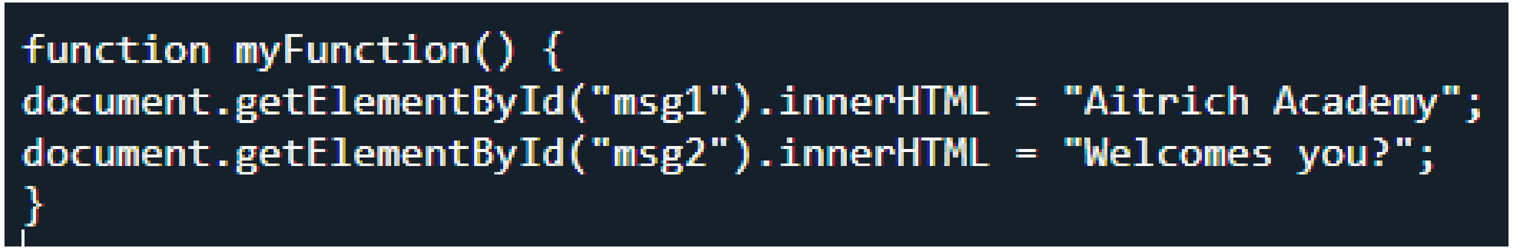
JavaScript has 3 types of scope:

- **Block scope**
- **Function scope**
- **Global scope**

Javascript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.

The purpose of code blocks is to define statements to be executed together.



```
function myFunction() {
document.getElementById("msg1").innerHTML = "Aitrich Academy";
document.getElementById("msg2").innerHTML = "Welcomes you?";
}
```

Block Scope

A block in JavaScript involves opening and closing curly braces:

```
{
  // a block
}
```

- Block scope is for variables declared in a block. You can find blocks in if, loop, switch, and a couple of other statements.
- Variables declared with the `var` keyword can NOT have block scope.ie, Variables declared with var inside a { } block can be accessed from outside the block.
- Variables declared with `let` and `const` have block scope.ie, Variables declared using let or const inside a { } block cannot be accessed from outside the block.

```
{
  let x = 2;
}
// x can NOT be used here
```

```
{
  var x = 2;
}
// x CAN be used here
```

Example:

Local Scope

Variables declared within a JavaScript function, become LOCAL to the function.

```
// code here can NOT use jobName

function jobSearch() {
  let jobName= "Manager";
  // code here CAN use jobName
}

// code here can NOT use jobName
```

Local variables have Function Scope:

They can only be accessed from within the function.

Note: Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Function Scope

Variables defined inside a function are not accessible (visible) from outside the function.

Variables declared with var, let and const are quite similar when declared inside a function. They **all** have Function Scope:

Local variables are created when a function starts, and deleted when the function is completed.

```
function jobSearch() {
  var applicantName = "John";   // Function Scope
  let job = "Manager";   // Function Scope
  const birthday= "20-12-2023";   // Function Scope
}
```

Global Scope

A variable declared outside a function, becomes GLOBAL and has Global Scope.All scripts and functions on a web page can access it.

Variables declared Globally (outside any function) have Global Scope.

Global variables can be accessed from anywhere in a JavaScript program.

Variables declared with the var always have Global Scope.

Variables declared with var, let and const are quite similar when declared outside a block. They all have Global Scope:

```
  var applicantName = "John";   // Global scope
  let job = "Manager";   // Global Scope
  const birthday= "20-12-2023";   // Global scope
  // code here can use applicantName,job and birthday.

function jobSearch() {
// code here can also use applicantName,job and birthday.
}
```

Variable Scope

keyword	const	let	var
global scope	NO	NO	YES
function scope	YES	YES	YES
block scope	YES	YES	NO

Variable Hoisting

Hoisting is JavaScript's default behavior of moving declarations to the top.

Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope.

Javascript Hoisting (var)

- Variables defined with var are hoisted to the top and can be initialized at any time, with a **default initialization** of **undefined**. Ie, You can use the variable before it is declared.

- In JavaScript, a variable can be declared after it has been used.

```
<!DOCTYPE html>
<html>
<body>

<p id="msg"></p>

<script>
jobName = "Manager";
document.getElementById("msg").innerHTML = jobName;
var jobName;
</script>

</body>
</html>
```

Output

```
Manager
```

Javascript Hoisting (let)

let and const variables, on the other hand, are hoisted, without a default initialization. This makes them inaccessible.

Meaning: Using a let variable before it is declared will result in a ReferenceError.

```
<!DOCTYPE html>
<html>
<body>
<p id="msg"></p>
<script>
try {
  jobName = "Manager";
  let jobName = "Accountant";
}
catch(err) {
  document.getElementById("msg").innerHTML = err;
}
</script>
</body>
</html>
```

Output

```
ReferenceError: Cannot access 'jobName' before initialization
```

Javascript Hoisting (const)

With **const**, you cannot use a variable before it is declared. In this example, there will not be any output

```
<!DOCTYPE html>
<html>
<body>

<p id="msg"></p>

<script>
jobName = "Manager";
const jobName;
document.getElementById("msg").innerHTML =
jobName;
</script>

</body>
</html>
```

Output

‘var’ vs ‘let’ vs ‘const’

KEYWORD	SCOPE	REDECLARATION & REASSIGNMENT	HOISTING
var	Global, Local	yes & yes	yes, with default value
let	Global, Local, Block	no & yes	yes, without default value
const	Global, Local, Block	no & no	yes, without default value

Thanks