**JavaScript**

**getElementbyId:-**

* <!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button" onclick='document.getElementById("demo").innerHTML = "Hello JavaScript!"'>Click Me!</button>

</body>

</html>

* <button onclick="document.getElementById('myImage').src='pic\_bulbon.gif'">Turn on the light</button>

<img id="myImage" src="pic\_bulboff.gif" style="width:100px">

<button onclick="document.getElementById('myImage').src='pic\_bulboff.gif'">Turn off the light</button>

* <button type="button" onclick="document.getElementById('demo').style.fontSize='35px'">Click Me!</button>
* <button type="button" onclick="document.getElementById('demo').style.display='none'">Click Me!</button>

**JavaScript can "display" data in different ways:**

* Writing into an HTML element, using innerHTML.
* Writing into the HTML output using document.write().
* Writing into an alert box, using window.alert().
* Writing into the browser console, using console.log().

1. <!DOCTYPE html>  
   <html>  
   <body>  
   <h1>My First Web Page</h1>  
   <p>My First Paragraph</p>  
   <p id="demo"></p>  
   <script>  
   document.getElementById("demo").innerHTML = 5 + 6;  
   </script>  
   </body>  
   </html>
2. <!DOCTYPE html>  
   <html>  
   <body>  
   <h1>My First Web Page</h1>  
   <p>My first paragraph.</p>  
   <script>  
   document.write(5 + 6);  
   </script>  
   </body>  
   </html>
3. <!DOCTYPE html>  
   <html>  
   <body>  
   <h1>My First Web Page</h1>  
   <p>My first paragraph.</p>  
   <script>  
   window.alert(5 + 6);  
   </script>  
   </body>  
   </html>
4. <!DOCTYPE html>  
   <html>  
   <body>  
   <script>  
   console.log(5 + 6);  
   </script>  
   </body>  
   </html>

**JavaScript Keywords**

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

Our [Reserved Words Reference](https://www.w3schools.com/js/js_reserved.asp) lists all JavaScript keywords.

Here is a list of some of the keywords you will learn about in this tutorial:

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| var | Declares a variable |
| let | Declares a block variable |
| const | Declares a block constant |
| if | Marks a block of statements to be executed on a condition |
| switch | Marks a block of statements to be executed in different cases |
| for | Marks a block of statements to be executed in a loop |
| function | Declares a function |
| return | Exits a function |
| try | Implements error handling to a block of statements |

## JavaScript Values

The JavaScript syntax defines two types of values:

Fixed values,Variable values

Fixed values are called **Literals**.

Variable values are called **Variables**.

**Strings** are text, written within double or single quotes

**Numbers** are written with or without decimals

## JavaScript Variables

In a programming language, **variables** are used to **store** data values.

JavaScript uses the keywords var, let and const to **declare** variables.

An **equal sign** is used to **assign values** to variables.

let x;  
x = 6;

## JavaScript Operators

JavaScript uses **arithmetic operators** ( + - \* / ) to **compute** values

(5 + 6) \* 10

JavaScript uses an **assignment operator** ( = ) to **assign** values to variables

let x, y;  
x = 5;  
y = 6;

## JavaScript Expressions

An expression is a combination of values, variables, and operators, which computes to a value.The computation is called an evaluation.

5 \* 10, x \* 10

he values can be of various types, such as numbers and strings.

For example, "John" + " " + "Doe", evaluates to "John Doe"

## JavaScript Keywords

JavaScript **keywords** are used to identify actions to be performed.

The let keyword tells the browser to create variables:

let x, y;  
x = 5 + 6;  
y = x \* 10;

The var keyword also tells the browser to create variables:

var x, y;  
x = 5 + 6;  
y = x \* 10;

## JavaScript Comments

Not all JavaScript statements are "executed".

Code after double slashes // or between /\* and \*/ is treated as a **comment**.

**JavaScript Identifiers / Names**

Identifiers are JavaScript names.

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

The rules for legal names are the same in most programming languages.

A JavaScript name must begin with:

* A letter (A-Z or a-z)
* A dollar sign ($)
* Or an underscore (\_)
* Names can contain letters, digits, underscores, and dollar signs.
* Names must begin with a letter.
* Names can also begin with $ and \_ (but we will not use it in this tutorial).
* Names are case sensitive (y and Y are different variables).
* Reserved words (like JavaScript keywords) cannot be used as names.

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

## JavaScript is Case Sensitive

All JavaScript identifiers are **case sensitive**.

The variables lastName and lastname, are two different variables

## JavaScript and Camel Case

**Underscore:**

first\_name, last\_name, master\_card, inter\_city.

**Upper Camel Case (Pascal Case):**

FirstName, LastName, MasterCard, InterCity.

**Lower Camel Case:**

JavaScript programmers tend to use camel case that starts with a lowercase letter:

firstName, lastName, masterCard, interCity.

## JavaScript Character Set

JavaScript uses the **Unicode** character set.

Unicode covers (almost) all the characters, punctuations, and symbols in the world.

**JavaScript Comments**

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

JavaScript comments can also be used to prevent execution, when testing alternative code.

## Single Line Comments

Single line comments start with //.

Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

// Change heading:  
document.getElementById("myH").innerHTML = "My First Page";  
  
// Change paragraph:  
document.getElementById("myP").innerHTML = "My first paragraph.";

## Multi-line Comments

Multi-line comments start with /\* and end with \*/.

Any text between /\* and \*/ will be ignored by JavaScript.

/\*  
The code below will change  
the heading with id = "myH"  
and the paragraph with id = "myP"  
in my web page:  
\*/  
document.getElementById("myH").innerHTML = "My First Page";  
document.getElementById("myP").innerHTML = "My first paragraph.";

## Using Comments to Prevent Execution

Using comments to prevent execution of code is suitable for code testing.

Adding // in front of a code line changes the code lines from an executable line to a comment.

//document.getElementById("myH").innerHTML = "My First Page";  
document.getElementById("myP").innerHTML = "My first paragraph.";

# **javaScript Variables**

### **Variables are Containers for Storing Data**

JavaScript Variables can be declared in 4 ways:

* Automatically
* Using var
* Using let
* Using const

1. x = 5;  
   y = 6;  
   z = x + y;
2. var x = 5;  
   var y = 6;  
   var z = x + y;
3. let x = 5;  
   let y = 6;  
   let z = x + y;
4. const x = 5;  
   const y = 6;  
   const z = x + y;
5. const price1 = 5;  
   const price2 = 6;  
   let total = price1 + price2;

## The Assignment Operator

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.

## Declaring a JavaScript Variable

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

## let carName; var carName;

To **assign** a value to the variable, use the equal sign:

carName = "Volvo";

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

let carName = "Volvo";

## One Statement, Many Variables

You can declare many variables in one statement.

let person = "John Doe", carName = "Volvo", price = 200;

let person = "John Doe",  
carName = "Volvo",  
price = 200;

## Re-Declaring JavaScript Variables

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

let carName;

var carName = "Volvo";  
var carName;

using arithmetic operator (If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.)

## JavaScript Dollar Sign $

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

let $ = "Hello World";  
let $$$ = 2;  
let $myMoney = 5;

## JavaScript Underscore (\_)

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

let \_lastName = "Johnson";  
let \_x = 2;  
let \_100 = 5;

# **JavaScript Let**

Variables declared with let have **Block Scope.**Variables declared with let must be **Declared** before use.Variables declared with let cannot be **Redeclared** in the same scope.

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

## Global Scope

Variables declared with the var always have **Global Scope**.

Variables declared with varinside a { } block can be accessed from outside the block:

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

Variables defined with let **can not** be redeclared.

let x = "John Doe";  
  
let x = 0;

You can not accidentally redeclare a variable declared with let.

Variables defined with var **can** be redeclared.

var x = "John Doe";  
  
var x = 0;

## Redeclaring Variables

Redeclaring a variable using the var keyword can impose problems.

var x = 10;  
// Here x is 10  
  
{  
var x = 2;  
// Here x is 2  
}  
  
// Here x is 2

var x = 2;   // Allowed  
let x = 3;   // Not allowed  
  
{  
let x = 2;   // Allowed  
let x = 3;   // Not allowed  
}

let x = 2;   // Allowed  
  
{  
let x = 3;   // Allowed  
}

## Let Hoisting

Variables defined with var are **hoisted** to the top and can be initialized at any time.

carName = "Volvo";  
var carName;

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

# **JavaScript Const**

It does not define a constant value. It defines a constant reference to a value.

Variables defined with const cannot be **Redeclared**

Variables defined with const cannot be **Reassigned**

Variables defined with const have **Block Scope**

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

const PI = 3.14159265359;

## Constant Arrays

You can change the elements of a constant array

// You can create a constant array:  
const cars = ["Saab", "Volvo", "BMW"];  
  
// You can change an element:  
cars[0] = "Toyota";  
  
// You can add an element:  
cars.push("Audi");

## Constant Objects

You can change the properties of a constant object:

// You can create a const object:  
const car = {type:"Fiat", model:"500", color:"white"};  
  
// You can change a property:  
car.color = "red";  
  
// You can add a property:  
car.owner = "Johnson";

## Block Scope

Declaring a variable with const is similar to let when it comes to **Block Scope**.

The x declared in the block, in this example, is not the same as the x declared outside the block

const x = 10;  
// Here x is 10  
  
{  
const x = 2;  
// Here x is 2  
}  
  
// Here x is 10

## Hoisting

Variables defined with var are **hoisted** to the top and can be initialized at any time.

carName = "Volvo";  
var carName;

# **JavaScript Operators**

The **Assignment Operator** **=** assigns values

The **Addition Operator** **+** adds values

The **Multiplication Operator** **\*** multiplies values

The **Comparison Operator** **>** compares values

## JavaScript Assignment

The **Assignment Operator** (=) assigns a value to a variable

// Assign the value 5 to x  
let x = 5;  
// Assign the value 2 to y  
let y = 2;  
// Assign the value x + y to z:  
let z = x + y;

**Operator Description**

+ Addition

- Subtraction

\* Multiplication

\*\* Exponentiation

/ Division

% Modulus

++ Increment

- - Decrement

## JavaScript Comparison Operators

## All the comparison operators above can also be used on strings

## === equal value and equal type

## !== not equal value and not equal type

## JavaScript Logical Operators

## && and

## || or

## ! not

## JavaScript Type Operator

## Instanceof Returns true if an object is an instance of an object type

## typeof Returns type of variable

## JavaScript Bitwise Operators

**Operator Description**

&(and), |(or), ~(not), ^(xor), <<(left shift), >>(right shift), >>>(unsigned right shift)

# **JavaScript Data Types**

### **JavaScript has 8 Datatypes**

String  
Number  
Bigint  
Boolean  
Undefined  
Null  
Symbol  
Object

### **The Object Datatype**

The object data type can contain both **built-in objects**, and **user defined objects**

Built-in object types can be:

objects, arrays, dates, maps, sets, intarrays, floatarrays, promises

// Numbers:  
let length = 16;  
let weight = 7.5;  
  
// Strings:  
let color = "Yellow";  
let lastName = "Johnson";  
  
// Booleans  
let x = true;  
let y = false;  
  
// Object:  
const person = {firstName:"John", lastName:"Doe"};  
  
// Array object:  
const cars = ["Saab", "Volvo", "BMW"];  
  
// Date object:  
const date = new Date("2022-03-25");

Concatenating String values with number values

let x = 16 + 4 + "Volvo"; 20Volvo

let x = "Volvo" + 16 + 4; Volvo164

## JavaScript Types are Dynamic

JavaScript has dynamic types. This means that the same variable can be used to hold different data types

let x;       // Now x is undefined  
x = 5;       // Now x is a Number  
x = "John";  // Now x is a String

## JavaScript Strings

A string (or a text string) is a series of characters like "John Doe".Strings are written with quotes.

// Using double quotes:  
let carName1 = "Volvo XC60";  
  
// Using single quotes:  
let carName2 = 'Volvo XC60';

## JavaScript Numbers

All JavaScript numbers are stored as decimal numbers (floating point).

// With decimals:  
let x1 = 34.00;  
  
// Without decimals:  
let x2 = 34;

## Exponential Notation

Extra large or extra small numbers can be written with scientific (exponential) notation

## JavaScript BigInt

All JavaScript numbers are stored in a 64-bit floating-point format.JavaScript BigInt is a new datatype ([ES2020](https://www.w3schools.com/js/js_2020.asp)) that can be used to store integer values that are too big to be represented by a normal JavaScript Number.

let x = BigInt("123456789012345678901234567890");

## JavaScript Booleans

## Booleans can only have two values: true or false.

## let x = 5; let y = 5; let z = 6; (x == y)       // Returns true (x == z)       // Returns false

## JavaScript Arrays

JavaScript arrays are written with square brackets.Array items are separated by commas.The following code declares (creates) an array called cars, containing three items

const cars = ["Saab", "Volvo", "BMW"];

## JavaScript Objects

JavaScript objects are written with curly braces {}.

Object properties are written as name:value pairs, separated by commas.

const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

## The typeof Operator

You can use the JavaScript typeof operator to find the type of a JavaScript variable.

The typeof operator returns the type of a variable or an expression

typeof ""             // Returns "string"  
typeof "John"         // Returns "string"  
typeof "John Doe"     // Returns "string"

typeof 0              // Returns "number"  
typeof 314            // Returns "number"  
typeof 3.14           // Returns "number"  
typeof (3)            // Returns "number"  
typeof (3 + 4)        // Returns "number"

## Undefined

In JavaScript, a variable without a value, has the value undefined. The type is also undefined.

let car;    // Value is undefined, type is undefined

## Empty Values

An empty value has nothing to do with undefined.

An empty string has both a legal value and a type.

let car = "";    // The value is "", the typeof is "string"

# **JavaScript Functions**

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

// Function to compute the product of p1 and p2  
function myFunction(p1, p2) {  
  return p1 \* p2;  
}

**JavaScript Function Syntax**

A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**.Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).The parentheses may include parameter names separated by commas:**(*parameter1, parameter2, ...*)**

The code to be executed, by the function, is placed inside curly brackets: **{}**

function *name*(*parameter1, parameter2, parameter3*) {  
  // *code to be executed*  
}

Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

## Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

* When an event occurs (when a user clicks a button)
* When it is invoked (called) from JavaScript code
* Automatically (self invoked)

## Function Return

When JavaScript reaches a return statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller"

// Function is called, the return value will end up in x  
let x = myFunction(4, 3);  
  
function myFunction(a, b) {  
// Function returns the product of a and b  
  return a \* b;  
}

## The () Operator

The () operator invokes (calls) the function.

function toCelsius(fahrenheit) {  
  return (5/9) \* (fahrenheit-32);  
}  
let value = toCelsius(77);

## Functions Used as Variable Values

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

let x = toCelsius(77);  
let text = "The temperature is " + x + " Celsius";

let text = "The temperature is " + toCelsius(77) + " Celsius";

## Local Variables

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

Local variables can only be accessed from within the function.

// code here can NOT use carName  
function myFunction() {  
  let carName = "Volvo";  
  // code here CAN use carName  
}  
// code here can NOT use carName

## 