JavaScript

Contents

The <script> Tag

In HTML, JavaScript code is inserted between `<script>` and `</script>` tags.

Example

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

JavaScript Functions and Events

A JavaScript `function` is a block of JavaScript code, that can be executed when "called" for.

For example, a function can be called when an **event** occurs, like when the user clicks a button.

JavaScript in <head> or <body>

You can place any number of scripts in an HTML document.

Scripts can be placed in the `<body>`, or in the `<head>` section of an HTML page, or in both.

JavaScript in <head>

In this example, a JavaScript `function` is placed in the `<head>` section of an HTML page.

The function is invoked (called) when a button is clicked:

Example

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
```

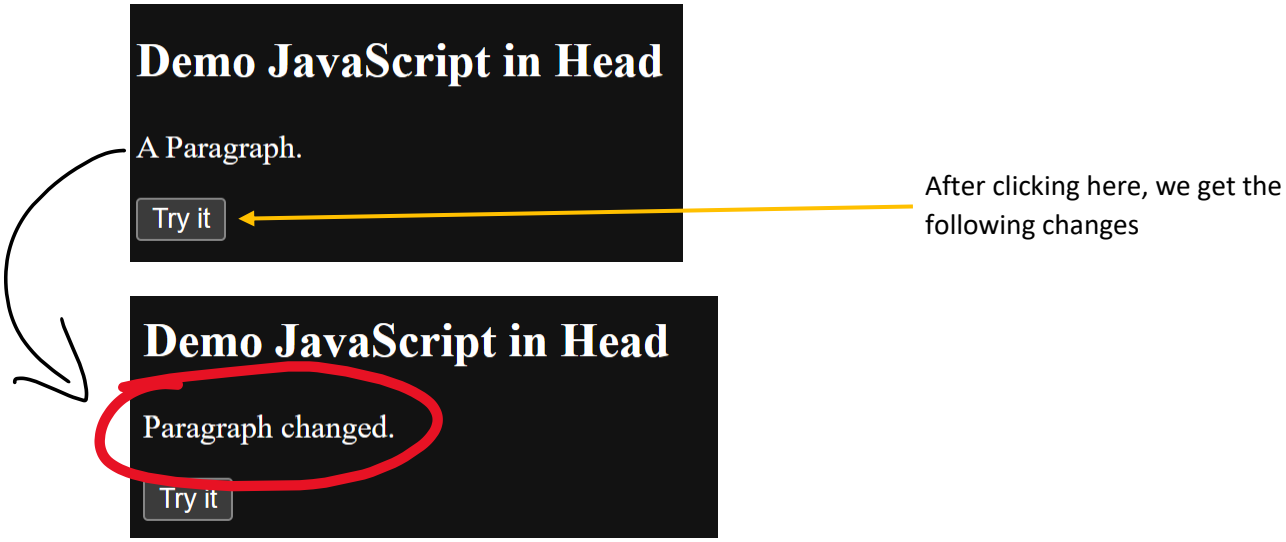
```
}
</script>
</head>
<body>

<h2>Demo JavaScript in Head</h2>

<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

Output:



We can get the exact same behavior by putting script in the Body tag.

IMPORTANT (for best practice):

- Whatever you see loaded on a webpage using script BEFORE ANY EVENT IS TAKEN PLACE, FOR THAT REASON THE SCRIPT FOR IT MUST BE IN HEADER TAG.
- Whatever behavior which takes place after an event has occurred, that script must be placed inside the body tag.

Using innerHTML

To access an HTML element, JavaScript can use the `document.getElementById(id)` method.

The `id` attribute defines the HTML element. The `innerHTML` property defines the HTML content:

Example

```
<!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>
```

Output:

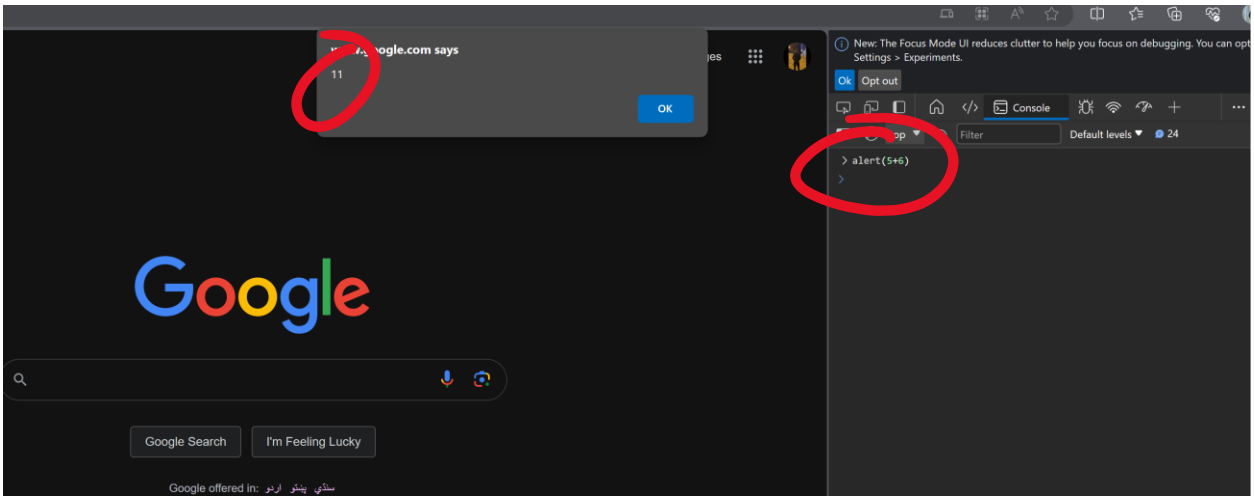
Using alert()

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
alert(5 + 6);
</script>
```

```
</body>
</html>
```



Variables and Arithmetic operations

JavaScript Variables can be declared in 4 ways:

- Automatically
- Using `var`
- Using `let`
- Using `const`

`Example:`

Just like in algebra, variables hold values:

```
var x = 5;
var y = 6;
```

Just like in algebra, variables are used in expressions:

```
var z = x + y;
```

From the example above, you can guess that the total is calculated to be 11.

//Console.log(z) here works same as printf("%d",z);

When to Use var, let, or const?

1. Always declare variables
2. Always use `const` if the value should not be changed
3. Always use `const` if the type should not be changed (Arrays and Objects)
4. Only use `let` if you can't use `const`
5. Only use `var` if you MUST support old browsers.

The Assignment Operator

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

This is different from algebra. The following does not make sense in algebra:

```
x = x + 5
```

Note

The "equal to" operator is written like `==` in JavaScript, just like in C Language.

Adding

The **addition** operator (`+`) adds numbers:

Example

```
let x = 5;
let y = 2;
let z = x + y;
```

Subtracting

The **subtraction** operator (`-`) subtracts numbers.

Example

```
let x = 5;
let y = 2;
let z = x - y;
```

Multiplying

The **multiplication** operator (`*`) multiplies numbers.

Example

```
let x = 5;
let y = 2;
let z = x * y;
```

Dividing

The **division** operator (`/`) divides numbers.

Example

```
let x = 5;
let y = 2;
let z = x / y;
```

Remainder

The **modulus** operator (`%`) returns the division remainder.

Example

```
let x = 5;
let y = 2;
let z = x % y;
```

In arithmetic, the division of two integers produces a **quotient** and a **remainder**.

In mathematics, the result of a **modulo operation** is the **remainder** of an arithmetic division.

Incrementing

The **increment** operator (`++`) increments numbers.

Example

```
let x = 5;
x++;
let z = x;
```

Decrementing

The **decrement** operator (`--`) decrements numbers.

Example

```
let x = 5;
x--;
let z = x;
```

Exponentiation

The **exponentiation** operator (`**`) raises the first operand to the power of the second operand.

Example

```
let x = 5;
let z = x ** 2;
```

x ** y produces the same result as `Math.pow(x,y)`:

Example

```
let x = 5;
let z = Math.pow(x,2);
```

Operator Precedence

Operator precedence describes the order in which operations are performed in an arithmetic expression.

Example

```
let x = 100 + 50 * 3;
```

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y
=	x = y	x = y
+=	x += y	x = x + y

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 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":

Example

Calculate the product of two numbers, and return the result.

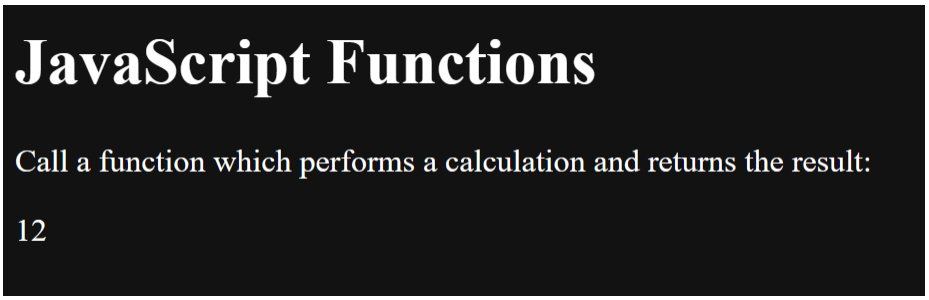
```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Functions</h1>
<p id=”demo”>Call a function which performs a calculation and returns the result: </p>

<script>
let x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;


function myFunction(a, b) {
  return a * b;
}
</script>
```

```
</body>
</html>
```



JavaScript Arrays

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

Why Use Arrays?

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
let car1 = "Saab";
let car2 = "Volvo";
let car3 = "BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

```
const array_name = [item1, item2, ...];
```

Example

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

Spaces and line breaks are not important. A declaration can span multiple lines:

Example

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

You can also create an array, and then provide the elements:

Example

```
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

Using the JavaScript Keyword new

The following example also creates an Array, and assigns values to it:

Example

```
const cars = new Array("Saab", "Volvo", "BMW");
```

Accessing Array Elements

You access an array element by referring to the **index number**:

```
const cars = ["Saab", "Volvo", "BMW"];
let car = cars[0];
```

Changing an Array Element

This statement changes the value of the first element in `cars`:

```
cars[0] = "Opel";
```

Converting an Array to a String

The JavaScript method `toString()` converts an array to a string of (comma separated) array values.

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- Use `if` to specify a block of code to be executed, if a specified condition is true
- Use `else` to specify a block of code to be executed, if the same condition is false
- Use `else if` to specify a new condition to test, if the first condition is false
- Use `switch` to specify many alternative blocks of code to be executed

The if Statement

Use the `if` statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax

```
if (condition) {
  //  block of code to be executed if the condition is true
}
```

Note that `if` is in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error.

Example

Make a "Good day" greeting if the hour is less than 18:00:

```
if (hour < 18) {
  greeting = "Good day";
}
else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Often this is the case when working with arrays:

Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

You can write:

```
for (var i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
```