JavaScript Introduction

JavaScript is the programming language of the web. It can update and change both HTML and CSS. It can calculate, manipulate and validate data.

JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.

ECMA-262 is the official name of the standard. ECMAScript is the official name of the language.

Why JavaScript?

JavaScript is one of the **3 languages** all web developers must learn:

1. **HTML** to define the content of web pages
2. **CSS** to specify the layout of web pages
3. **JavaScript** to program the behavior of web pages

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is `getElementById()`. The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript":

Example

```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

JavaScript accepts both double and single quotes:

Example

```
document.getElementById('demo').innerHTML = 'Hello JavaScript';
```

JavaScript Can Change HTML Styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

Example

```
document.getElementById("demo").style.fontSize = "35px";
```

JavaScript Can Hide HTML Elements

Hiding HTML elements can be done by changing the `display` style:

Example

```
document.getElementById("demo").style.display = "none";
```

JavaScript Can Show HTML Elements

Showing hidden HTML elements can also be done by changing the `display` style:

Example

```
document.getElementById("demo").style.display = "block";
```

Where, How To Use JavaScript

The <script> Tag

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

Example

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

Old JavaScript examples may use a type attribute: <script type = "text / javascript">
The type attribute is not required. JavaScript is the default scripting language in HTML.

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>
```

JavaScript in <body>

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

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

Example

```
<!DOCTYPE html>
<html>
<body>
<h2>Demo JavaScript in Body</h2>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</body>
</html>
```

Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.

External JavaScript

Scripts can also be placed in external files:

External file: myScript.js

```
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

External scripts are practical when the same code is used in many different web pages. JavaScript files have the file extension **.js**. To use an external script, put the name of the script file in the `src` (source) attribute of a `<script>` tag:

Example

```
<script src="myScript.js"></script>
```

You can place an external script reference in `<head>` or `<body>` as you like.

The script will behave as if it was located exactly where the `<script>` tag is located.

External scripts cannot contain `<script>` tags.

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

To add several script files to one page – use several script tags:

Example

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

External References

An external script can be referenced in 3 different ways:

- With a full URL (a full web address)
- With a file path (like /js/)
- Without any path

Below example uses a **full URL** to link to myScript.js:

```
<script src="https://www.flipkart.com/js/myScript.js"></script>
```

Below example uses a **file path** to link to myScript.js:

```
<script src="/js/myScript.js"></script>
```

Below example uses no path to link to myScript.js:

```
<script src="myScript.js"></script>
```

JavaScript Display Output

JavaScript can "display" data in different ways:

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

Using innerHTML

To access an HTML element, we can use the method `document.getElementById(id).` Use the `id` attribute to identify the HTML element. Then use the `innerHTML` property to change content of the HTML element:

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 = "<h2>Hello World</h2>";
</script>
</body>
</html>
```

Note: Changing the innerHTML property of an HTML element is the most common way to display data in HTML.

Using innerText

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

Then use the `innerText` property to change the inner text of the HTML element:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My First Paragraph</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerText = "Hello World";
</script>
</body>
</html>
```

Note: Use innerHTML when you want to change an HTML element. Use innerText when you only want to change the plain text.

Using document.write()

For testing purposes, it is convenient to use `document.write()`:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
document.write(5 + 6);
</script>
</body>
</html>
```

Using document.write() after an HTML document is loaded, will **delete all existing HTML**:

Example

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

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

<button type="button" onclick="document.write(5 + 6)">Try it</button>

</body>
</html>
```

The document.write() method should only be used for testing.

Using window.alert()

You can use an alert box to display data:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
window.alert(5 + 6);
</script>
</body>
</html>
```

You can skip the `window` keyword. In JavaScript, the window object is the global scope object. This means that variables, properties, and methods by default belong to the window object. This also means that specifying the `window` keyword is optional:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
alert(5 + 6);
</script>
</body>
</html>
```

Using console.log()

For debugging purposes, you can call the `console.log()` method in the browser to display data.

Example

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

<script>
console.log(5 + 6);
</script>

</body>
</html>
```

JavaScript Print

JavaScript does not have any print object or print methods. You cannot access output devices from JavaScript.

The only exception is that you can call the `window.print()` method in the browser to print the content of the current window.

Example

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

<button onclick="window.print()">Print this page</button>

</body>
</html>
```

JavaScript Statements

Example

```
let x, y, z;     // Statement 1
x = 5;           // Statement 2
y = 6;           // Statement 3
z = x + y;       // Statement 4
```

JavaScript Programs

A **computer program** is a list of "instructions" to be "executed" by a computer. In a programming language, these programming instructions are called **statements**.

A **JavaScript program** is a list of programming **statements**. In HTML, JavaScript programs are executed by the web browser.

JavaScript Statements

JavaScript statements are composed of: Values, Operators, Expressions, Keywords, and Comments.

Below statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":

Example

```
document.getElementById("demo").innerHTML = "Hello Dolly.";
```

Most JavaScript programs contain many JavaScript statements. The statements are executed, one by one, in the same order as they are written.

JavaScript programs (and JavaScript statements) are often called JavaScript code.

Semicolons ";"

Semicolons separate JavaScript statements. Add a semicolon at the end of each executable statement:

Examples

```
let a, b, c;  // Declare 3 variables
a = 5;        // Assign the value 5 to a
b = 6;        // Assign the value 6 to b
c = a + b;    // Assign the sum of a and b to c
```

When separated by semicolons, multiple statements on one line are allowed:

```
a = 5; b = 6; c = a + b;
```

On the web, you might see examples without semicolons. Ending statements with semicolon is not required, but highly recommended.

JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable. The following lines are equivalent:

```
let person = "Hege";
let person="Hege";
```

A good practice is to put spaces around operators (= + - * /):

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

JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters. If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

Example

```
document.getElementById("demo").innerHTML =
"Hello Dolly!";
```

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.

One place you will find statements grouped together in blocks, is in JavaScript functions:

Example

```
function myFunction() {
  document.getElementById("demo1").innerHTML = "Hello Dolly!";
  document.getElementById("demo2").innerHTML = "How are you?";
}
```

JavaScript Keywords

JavaScript statements often start with a **keyword** to identify the JavaScript action to be performed. Here is a list of some of the keywords

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 keywords are reserved words. Reserved words cannot be used as names for variables.

JavaScript Syntax

JavaScript syntax is the set of rules, how JavaScript programs are constructed:

```
// How to create variables:
var x;
let y;

// How to use variables:
x = 5;
y = 6;
let z = x + y;
```

JavaScript Values

The JavaScript syntax defines two types of values:

- Fixed values
- Variable values

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

JavaScript Literals

The two most important syntax rules for fixed values are:

1. **Numbers** are written with or without decimals:

```
10.50
```

```
1001
```

2. **Strings** are text, written within double or single quotes:

```
"John Doe"
```

```
'John Doe'
```

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. In below example, x is defined as a variable. Then, x is assigned (given) the value 6:

```
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.

For example, 5 * 10 evaluates to 50:

```
5 * 10
```

Expressions can also contain variable values:

```
x * 10
```

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

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

```
"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**.

Comments are ignored, and will not be executed:

```
let x = 5; // I will be executed

// x = 6; I will NOT be executed
```

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 (_)

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

Note: Numbers are not allowed as the first character in names. This way JavaScript can easily distinguish identifiers from numbers.

JavaScript is Case Sensitive

All JavaScript identifiers are **case sensitive**. In the below example, variables `lastName` and `lastname`, are two different variables:

```
let lastname, lastName;
lastName = "Doe";
lastname = "Peterson";
```

JavaScript does not interpret **LET** or **Let** as the keyword **let**.

JavaScript and Camel Case

Historically, programmers have used different ways of joining multiple words into one variable name:

Hyphens:

first-name, last-name, master-card, inter-city.

Hyphens are not allowed in JavaScript. They are reserved for subtractions.

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).

This example uses a single-line comment before each code line:

Example

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

// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
```

This example uses a single line comment at the end of each line to explain the code:

Example

```
let x = 5;       // Declare x, give it the value of 5
let y = x + 2;   // Declare y, give it the value of x + 2
```

Multi-line Comments

Multi-line comments start with /* and end with */. Any text between /* and */ will be ignored by JavaScript. Below example uses a multi-line comment (a comment block) to explain the code:

Example

```
/*
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.";
```

It is most common to use single line comments. Block comments are often used for formal documentation.

Using Comments to Prevent Execution

Adding `//` in front of a code line changes the code lines from an executable line to a comment. Below example uses // to prevent execution of one of the code lines:

Example

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

This example uses a comment block to prevent execution of multiple lines:

Example

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