Introduction to Java Script

- Introduction to JavaScript.
- Java Script Syntax, Variables, Operators and Expression.
- Control Flow.
- Functions
- Concept of Object oriented Development
- Concept of DOM.
- Forms and JavaScript.

Introduction to JavaScript

What is JavaScript ?

JavaScript is a *lightweight, cross-platform, single-threaded,* and *interpreted compiled* programming language. It is also known as the scripting language for webpages. It is well-known for the development of web pages, and many non-browser environments also use it.
JavaScript is a **weakly typed language (dynamically typed)**. JavaScript can be used for **Client-side** developments as well as **Server-side** developments. JavaScript is both an imperative and declarative type of language. JavaScript contains a standard library of objects, like **Array**, **Date**, and **Math**, and a core set of language elements like **operators**, **control structures**, and **statements**.

JavaScript

- **Client-side:** It supplies objects to control a browser and its Document Object Model (DOM). Like if client-side extensions allow an application to place elements on an HTML form and respond to user events such as **mouse clicks**, **form input**, and **page navigation**. Useful libraries for the client side are **AngularJS**, **ReactJS**, **VueJS,** and so many others.
- **Server-side:** It supplies objects relevant to running JavaScript on a server. For if the server-side extensions allow an application to communicate with a database, and provide continuity of information from one invocation to another of the application, or perform file manipulations on a server. The useful framework which is the most famous these days is **node.js**.
- **Imperative language –** In this type of language we are mostly concerned about how it is to be done. It simply controls the flow of computation. The procedural programming approach, object, oriented approach comes under this as async await we are thinking about what is to be done further after the async call.
- **Declarative programming –** In this type of language we are concerned about how it is to be done, basically here logical computation requires. Her main goal is to describe the desired result without direct dictation on how to get it as the arrow function does.

How to Link JavaScript File in HTML ?

JavaScript can be added to HTML file in two ways:

- **Internal JS:** We can add JavaScript directly to our HTML file by writing the code inside the <script> tag. The <script> tag can either be placed inside the <head> or the <body> tag according to the requirement.
- **External JS:** We can write JavaScript code in another files having an extension.js and then link this file inside the <head> tag of the HTML file in which we want to add this code.

Syntax:

0 seconds of 17 secondsVolume 0%

```
<script>
   // JavaScript Code
</script>
```

Example:

- HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>
      Basic Example to Describe JavaScript
   </title>
</head>
 <body>
   <!-- JavaScript code can be embedded inside
      head section or body section -->
   <script>
      console.log("Welcome to GeeksforGeeks");
   </script>
</body>
</html>
```

Output: The output will display on the console.

Welcome to GeeksforGeeks

History of JavaScript

It was created in 1995 by Brendan Eich while he was an engineer at Netscape. It was originally going to be named LiveScript but was renamed. Unlike most programming languages,

JavaScript language has no concept of input or output. It is designed to run as a scripting language in a host environment, and it is up to the host environment to provide mechanisms for communicating with the outside world. The most common host environment is the browser.

Features of JavaScript

According to a recent survey conducted by **Stack Overflow**, JavaScript is the most popular language on earth.
With advances in browser technology and JavaScript having moved into the server with Node.js and other frameworks, JavaScript is capable of so much more. Here are a few things that we can do with JavaScript:

- JavaScript was created in the first place for DOM manipulation. Earlier websites were mostly static, after JS was created dynamic Web sites were made.
- Functions in JS are objects. They may have properties and methods just like other objects. They can be passed as arguments in other functions.
- Can handle date and time.
- Performs Form Validation although the forms are created using HTML.
- No compiler is needed.

Applications of JavaScript

- **Web Development:** Adding interactivity and behavior to static sites JavaScript was invented to do this in 1995. By using AngularJS that can be achieved so easily.
- **Web Applications:** With technology, browsers have improved to the extent that a language was required to create robust web applications. When we explore a map in Google Maps then we only need to click and drag the mouse. All detailed view is just a click away, and this is possible only because of JavaScript. It uses Application Programming Interfaces(APIs) that provide extra power to the code. The Electron and React are helpful in this department.
- **Server Applications:** With the help of Node.js, JavaScript made its way from client to server and Node.js is the most powerful on the server side.
- **Games:** Not only in websites, but JavaScript also helps in creating games for leisure. The combination of JavaScript and HTML 5 makes JavaScript popular in game development as well. It provides the EaseJS library which provides solutions for working with rich graphics.
- **Smartwatches:** JavaScript is being used in all possible devices and applications. It provides a library PebbleJS which is used in smartwatch applications. This framework works for applications that require the Internet for their functioning.
- **Art:** Artists and designers can create whatever they want using JavaScript to draw on HTML 5 canvas, and make the sound more effective also can be used **p5.js** library.
- **Machine Learning:** This JavaScript ml5.js library can be used in web development by using machine learning.

- **Mobile Applications:** JavaScript can also be used to build an application for non-web contexts. The features and uses of JavaScript make it a powerful tool for creating mobile applications. This is a Framework for building web and mobile apps using JavaScript. Using React Native, we can build mobile applications for different operating systems. We do not require to write code for different systems. Write once use it anywhere!

Limitations of JavaScript

- **Security risks:** JavaScript can be used to fetch data using AJAX or by manipulating tags that load data such as <img>, <object>, <script>. These attacks are called cross-site script attacks. They inject JS that is not part of the site into the visitor's browser thus fetching the details.
- **Performance:** JavaScript does not provide the same level of performance as offered by many traditional languages as a complex program written in JavaScript would be comparatively slow. But as JavaScript is used to perform simple tasks in a browser, so performance is not considered a big restriction in its use.
- **Complexity:** To master a scripting language, programmers must have a thorough knowledge of all the programming concepts, core language objects, and client and server-side objects otherwise it would be difficult for them to write advanced scripts using JavaScript.
- **Weak error handling and type checking facilities:** It is a weakly typed language as there is no need to specify the data type of the variable. So wrong type checking is not performed by compile.

Why JavaScript is known as a lightweight programming language ?

JavaScript is considered lightweight due to the fact that it has low CPU usage, is easy to implement, and has a minimalist syntax. Minimalist syntax as in, has no data types. Everything is treated here as an object. It is very easy to learn because of its syntax similar to C++ and Java.
A lightweight language does not consume much of your CPU's resources. It doesn't put excess strain on your CPU or RAM. JavaScript runs in the browser even though it has complex paradigms and logic which means it uses fewer resources than other languages. For example, NodeJs, a variation of JavaScript not only performs faster computations but also uses fewer resources than its counterparts such as Dart or Java.
Additionally, when compared with other programming languages, it has fewer in-built libraries or frameworks, contributing as another reason for it being lightweight. However, this brings a drawback in that we need to incorporate external libraries and frameworks.

Is JavaScript Compiled or Interpreted or both ?

JavaScript is both compiled and interpreted. In the earlier versions of JavaScript, it used only the interpreter that executed code line by line and shows the result immediately. But with time the performance became an issue as interpretation is quite slow. Therefore, in the newer

versions of JS, probably after the V8, the JIT compiler was also incorporated to optimize the execution and display the result more quickly. This JIT compiler generates a bytecode that is relatively easier to code. This bytecode is a set of highly optimized instructions.
The V8 engine initially uses an interpreter, to interpret the code. On further executions, the V8 engine finds patterns such as frequently executed functions, and frequently used variables, and compiles them to improve performance.

Java Script Syntax, Variables, Operators and Expression

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

```
let x;
x = 6;
```

Variables are Containers for Storing Data

JavaScript Variables can be declared in 4 ways:

- Automatically
- Using var
- Using let
- Using const

In this first example, x, y, and z are undeclared variables.
They are automatically declared when first used:

Example

```
x = 5;
y = 6;
z = x + y;
```

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

From the examples you can guess:

- x stores the value 5
- y stores the value 6
- z stores the value 11

Example using var

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

Note
The var keyword was used in all JavaScript code from 1995 to 2015.
The let and const keywords were added to JavaScript in 2015.
The var keyword should only be used in code written for older browsers.

Example using let

```
let x = 5;
let y = 6;
let z = x + y;
```

Example using const

```
const x = 5;
const y = 6;
const z = x + y;
```

Mixed Example

```
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
```

The two variables price1 and price2 are declared with the const keyword.

These are constant values and cannot be changed.

The variable total is declared with the let keyword.

The value total can be changed.

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.

Just Like Algebra

Just like in algebra, variables hold values:

```
let x = 5;
let y = 6;
```

Just like in algebra, variables are used in expressions:

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

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

Note

Variables are containers for storing values.

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

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

The general rules for constructing names for variables (unique identifiers) are:

- 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.

Note
JavaScript identifiers are case-sensitive.

JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put a number in quotes, it will be treated as a text string.

Example

```
const pi = 3.14;
let person = "John Doe";
let answer = 'Yes I am!';
```

Declaring a JavaScript Variable

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

You declare a JavaScript variable with the var or the let keyword:

```
var carName;
```

or:

```
let carName;
```

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

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";
```

In the example below, we create a variable called carName and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with id="demo":

Example

```
<p id="demo"></p>

<script>
let carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
```

Note

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

One Statement, Many Variables

You can declare many variables in one statement.

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

Example

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

A declaration can span multiple lines:

Example

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

Value = undefined

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

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

The variable carName will have the value undefined after the execution of this statement:

Example

```
let carName;
```

Re-Declaring JavaScript Variables

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

The variable carName will still have the value "Volvo" after the execution of these statements:

Example

```
var carName = "Volvo";
var carName;
```

Note

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

This will not work:

```
let carName = "Volvo";
let carName;
```

JavaScript Operators

Operators are the symbols between values that allow different operations like addition, substraction, multiplication, and more. JavaScript has dozens operators, so let's focus on the ones you're likely to see most often.

Operators is used to perform some operation on data.

There are many type of Operators:

Arithmetic Operator(+): JavaScript uses **arithmetic operators** (+ - * /) to **compute** values:

```
(5 + 6) * 10
```

Assignment Operator(%): JavaScript uses an **assignment operator** (=) to **assign** values to variables:

```
let x, y;
x = 5;
y = 6;
```

Exponentiation Operator(**): JavaScript uses an **exponentiation operator** (**) to **compute the power** values to variables:

```
let x, y;
x = 5;
y = 2;
d = x ** y
```

Modulus Operator(**): JavaScript uses an **modulus operator** (%) to **check the remainder** values to variables:

```
let x, y;
x = 8;
y = 2;
d = x % y
```

Unary Operators:

1. **Increment operators(++):** JavaScript uses an **increment operators (++)** to increasing the value of variable with (+ 1) .
   Increment Operators are two types:
   i. ***Pre- Increment operator(++a):*** It is use for increasing value before use of variable.
   ii. ***Post- Increment operator(a++):*** It is use for increasing value after use of variable.
2. **Decrement operators(--):** JavaScript uses an **decrement operators (--)** to decreasing the value of variable with (- 1) .
   i. ***Pre- decrement operator(++a):*** It is use for decreasing value before use of variable.
   ii. ***Post- decrement operator(a++):*** It is use for decreasing value after use of variable.

Comparison Operators: JavaScript uses an **comparison operators** to compare values between two variables.

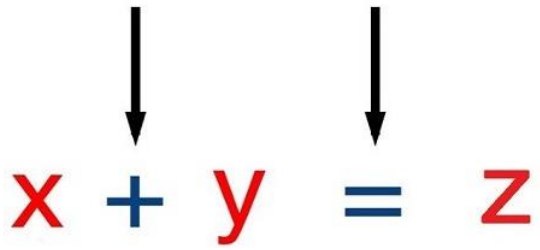
There are many type of comparison operators:

- Greater than(>)
- Greater than and Equal(>=)
- Smaller than(<)
- Smaller than and Equal(<=)
- Equal to(==)
- Equal to & Type(===)
- Not Equal to(!=)
- Not Equal to and Type(!==)

Logical Operators: JavaScript uses an **Logical operators** to compare and given a **boolean value.**

There are three types of Logical operators:

- Logical AND (&&)
- Logical OR (| |)
- Logical NOR (!)



Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- String Operators
- Logical Operators
- Bitwise Operators
- Ternary Operators
- Type Operators

JavaScript Arithmetic Operators

Arithmetic Operators are used to perform arithmetic on numbers:

Arithmetic Operators Example

```
let a = 3;
let x = (100 + 50) * a;
```

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Note

*Arithmetic operators are fully described in the **JS Arithmetic** chapter.*

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.
The **Addition Assignment Operator** (+=) adds a value to a variable.

Assignment

```
let x = 10;
x += 5;
```

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

Note

*Assignment operators are fully described in the **JS Assignment** chapter.*

JavaScript Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

?	ternary operator

Note

*Comparison operators are fully described in the **JS Comparisons** chapter.*

JavaScript String Comparison

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

Example

```
let text1 = "A";
let text2 = "B";
let result = text1 < text2;
```

Note that strings are compared alphabetically:

Example

```
let text1 = "20";
let text2 = "5";
let result = text1 < text2;
```

JavaScript String Addition

The + can also be used to add (concatenate) strings:

Example

```
let text1 = "John";
let text2 = "Doe";
let text3 = text1 + " " + text2;
```

The += assignment operator can also be used to add (concatenate) strings:

Example

```
let text1 = "What a very ";
text1 += "nice day";
```

The result of text1 will be:

What a very nice day

Note

When used on strings, the + operator is called the concatenation operator.

Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string:

Example

```
let x = 5 + 5;
let y = "5" + 5;
let z = "Hello" + 5;
```

The result of x, y, *and* z *will be:*

10

55

Hello5

Note

If you add a number and a string, the result will be a string!

JavaScript Logical Operators

Operator	Description
&&	logical and
||	logical or
!	logical not

Note
*Logical operators are fully described in the **JS Comparisons** chapter.*

JavaScript Type Operators

Operator	Description
Typeof	Returns the type of a variable
Instanceof	Returns true if an object is an instance of an object type

Note
*Type operators are fully described in the **JS Type Conversion** chapter.*

JavaScript Bitwise Operators
Bit operators work on 32 bits numbers.
Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	left shift	5 << 1	0101 << 1	1010	10
>>	right shift	5 >> 1	0101 >> 1	0010	2
>>>	unsigned right shift	5 >>> 1	0101 >>> 1	0010	2

The examples above uses 4 bits unsigned examples. But JavaScript uses 32-bit signed numbers. Because of this, in JavaScript, ~ 5 will not return 10. It will return -6.
~00000000000000000000000000000101 will return 11111111111111111111111111111010

*Bitwise operators are fully described in the **JS Bitwise** chapter.*

JavaScript's expression is a valid set of literals, variables, operators, and expressions that evaluate a single value that is an expression. This single value can be a number, a string, or a logical value depending on the expression.

Example:

- Javascript

```
// Illustration of function* expression
// use of function* keyword
function* func() {
    yield 1;
    yield 2;
    yield 3;
    yield " - Geeks";
}

let obj = '';

// Function calling
for (const i of func()) {
    obj = obj + i;
}

// Output
console.log(obj);
```

Output

123 – Geeks

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;
```

In these examples, using var or let will produce the same result.

You will learn more about var and let later in this tutorial.

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
```

You will learn more about comments in a later chapter.

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**.
The 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.

Control Flow

JavaScript **control statement** is used to control the execution of a program based on a specific condition. If the condition meets then a particular block of action will be executed otherwise it will execute another block of action that satisfies that particular condition.

Types of Control Statements in JavaScript

- **Conditional Statement:** These statements are used for decision-making, a decision
- n is made by the conditional statement based on an expression that is passed. Either YES or NO.
- **Iterative Statement:** This is a statement that iterates repeatedly until a condition is met. Simply said, if we have an expression, the statement will keep repeating itself until and unless it is satisfied.

There are several methods that can be used to perform control statements in JavaScript:

Table of Content

Approach 1: If Statement

In this approach, we are using an if statement to check a specific condition, the code block gets executed when the given condition is satisfied.

Syntax:

```
if ( condition_is_given_here ) {
   // If the condition is met,
   //the code  will get executed.
}
```

Example: In this example, we are using an if statement to check our given condition.

- Javascript

```
const num = 5;

if (num > 0) {
   console.log("The number is positive.");
};
```

Output

The number is positive.

Approach 2: Using If-Else Statement

The if-else statement will perform some action for a specific condition. If the condition meets then a particular code of action will be executed otherwise it will execute another code of action that satisfies that particular condition.

Syntax:

```
if (condition1) {
    // Executes when condition1 is true
    if (condition2) {
        // Executes when condition2 is true
    }
}
```

Example: In this example, we are using the if..else statement to verify whether the given number is positive or negative.

- Javascript

```
let num = -10;

if (num > 0)
    console.log("The number is positive.");
else
    console.log("The number is negative");
```

Output

The number is negative

Approach 3: Using Switch Statement

The switch case statement in JavaScript is also used for decision-making purposes. In some cases, using the switch case statement is seen to be more convenient than if-else statements.

Syntax:

```
switch (expression) {
    case value1:
        statement1;
        break;
    case value2:
        statement2;
        break;
    .
    .
    case valueN:
        statementN;
        break;
    default:
        statementDefault;
}
```

Example: In this example, we are using the above-explained approach.

- Javascript

```
let num = 5;

switch (num) {
    case 0:
        console.log("Number is zero.");
        break;
    case 1:
        console.log("Nuber is one.");
        break;
    case 2:
        console.log("Number is two.");
        break;
    default:
        console.log("Number is greater than 2.");
};
```

Output

Number is greater than 2.

Approach 4: Using the Ternary Operator (Conditional Operator)

The conditional operator, also referred to as the ternary operator (?:), is a shortcut for expressing conditional statements in JavaScript.

Syntax:

condition ? value if true : value if false

Example: In this example, we are using the ternary operator to find whether the given number is positive or negative.

- Javascript

```
let num = 10;

let result = num >= 0 ? "Positive" : "Negative";

console.log(`The number is ${result}.`);
```

Output

The number is Positive.

Approach 5: Using For loop

In this approach, we are using for loop in which the execution of a set of instructions repeatedly until some condition evaluates and becomes false

Syntax:

```
for (statement 1; statement 2; statement 3) {
    // Code here . . .
}
```

Example: In this example, we are using Iterative Statement as a for loop, in which we find the even number between 0 to 10.

- Javascript

```
for (let i = 0; i <= 10; i++) {
  if (i % 2 === 0) {
    console.log(i);
  }
};
```

Output:

0

2

4

6

8

10

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).

Example

```
// 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)

You will learn a lot more about function invocation later in this tutorial.

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:

```
// 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;
}
```

Why Functions?

With functions you can reuse code

You can write code that can be used many times.

You can use the same code with different arguments, to produce different results.

The () Operator

The () operator invokes (calls) the function:

Example

Convert Fahrenheit to Celsius:

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}

let value = toCelsius(77);
```

Accessing a function with incorrect parameters can return an incorrect answer:

Example

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}

let value = toCelsius();
```

Accessing a function without () returns the function and not the function result:

Example

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}

let value = toCelsius;
```

Note

As you see from the examples above, toCelsius refers to the function object, and toCelsius() refers to the function result.

Functions Used as Variable Values

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

Example

Instead of using a variable to store the return value of a function:

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

You can use the function directly, as a variable value:

```
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.

Example

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.
Local variables are created when a function starts, and deleted when the function is completed.

Test Yourself With Exercises

Exercise:

Execute the function named myFunction.

```
function myFunction() {
  alert("Hello World!");
}
```

JavaScript **function** is a set of statements that take inputs, do some specific computation, and produce output.
A JavaScript function is executed when “something” invokes it (calls it).
Example 1: A basic javascript function, here we create a function that divides the 1st element by the second element.

- Javascript

```
function myFunction(g1, g2) {
    return g1 / g2;
}
const value = myFunction(8, 2); // Calling the function
console.log(value);
```

Output:
0 seconds of 17 secondsVolume 0%

4
You must already have seen some commonly used functions in JavaScript like alert(), which is a built-in function in JavaScript. But JavaScript allows us to create user-defined functions also. We can create functions in JavaScript using the keyword `function`.
Syntax*:*
The basic syntax to create a function in JavaScript is shown below.

```
function functionName(Parameter1, Parameter2, ...)
{
    // Function body
}
```

To create a function in JavaScript, we have to first use the keyword *function*, separated by the name of the function and parameters within parenthesis. The part of the function inside the curly braces {} is the body of the function.
In javascript, functions can be used in the same way as variables for assignments, or calculations.

Function Invocation:

- Triggered by an event (e.g., a button click by a user).
- When explicitly called from JavaScript code.
- Automatically executed, such as in self-invoking functions.

Function Definition:
Before, using a user-defined function in JavaScript we have to create one. We can use the above syntax to create a function in JavaScript. A function definition is sometimes also termed

a function declaration or function statement. Below are the rules for creating a function in JavaScript:

- Every function should begin with the keyword *function* followed by,
- A user-defined function name that should be unique,
- A list of parameters enclosed within parentheses and separated by commas,
- A list of statements composing the body of the function enclosed within curly braces {}.

Example 2: This example shows a basic declaration of a function in javascript.

- JavaScript

```
function calcAddition(number1, number2) {
   return number1 + number2;
}
console.log(calcAddition(6,9));
```

Output

15

In the above example, we have created a function named calcAddition,

- This function accepts two numbers as parameters and returns the addition of these two numbers.
- Accessing the function with just the function name without () will return the function object instead of the function result.

There are three ways of writing a function in JavaScript:

Function Declaration:

It declares a function with a function keyword. The function declaration must have a function name.

Syntax:

```
function geeksforGeeks(paramA, paramB) {
   // Set of statements
}
```

Function Expression:

It is similar to a function declaration without the function name. Function expressions can be stored in a variable assignment.

Syntax:

```
let geeksforGeeks= function(paramA, paramB) {
   // Set of statements
}
```

Example 3: This example explains the usage of the Function expression.

- Javascript

```
const square = function (number) {
     return number * number;
};
const x = square(4); // x gets the value 16
console.log(x);
```

Output
16

Functions as Variable Values:

Functions can be used the same way as you use variables.

Example:

```
// Function to convert Fahrenheit to Celsius
function toCelsius(fahrenheit) {
   return (fahrenheit - 32) * 5/9;
 }

 // Using the function to convert temperature
 let temperatureInFahrenheit = 77;
 let temperatureInCelsius = toCelsius(temperatureInFahrenheit);
 let text = "The temperature is " + temperatureInCelsius + " Celsius";
```

Arrow Function:

It is one of the most used and efficient methods to create a function in JavaScript because of its comparatively easy implementation. It is a simplified as well as a more compact version of a regular or normal function expression or syntax.

Syntax:

```
let function_name = (argument1, argument2 ,..) => expression
```

Example 4: This example describes the usage of the Arrow function.

- Javascript

```
const a = ["Hydrogen", "Helium", "Lithium", "Beryllium"];

const a2 = a.map(function (s) {
   return s.length;
});

console.log("Normal way ", a2); // [8, 6, 7, 9]
```

```
const a3 = a.map((s) => s.length);

console.log("Using Arrow Function ", a3); // [8, 6, 7, 9]
```

Output

Normal way [8, 6, 7, 9]

Using Arrow Function [8, 6, 7, 9]

Function Parameters:

Till now, we have heard a lot about function parameters but haven't discussed them in detail. Parameters are additional information passed to a function. For example, in the above example, the task of the function *calcAddition* is to calculate the addition of two numbers. These two numbers on which we want to perform the addition operation are passed to this function as parameters. The parameters are passed to the function within parentheses after the function name and separated by commas. A function in JavaScript can have any number of parameters and also at the same time, a function in JavaScript can not have a single parameter.

Example 4: In this example, we pass the argument to the function.

- Javascript

```
function multiply(a, b) {
    b = typeof b !== "undefined" ? b : 1;
    return a * b;
}

console.log(multiply(69)); // 69
```

Output

69

Calling Functions:

After defining a function, the next step is to call them to make use of the function. We can call a function by using the function name separated by the value of parameters enclosed between the parenthesis and a semicolon at the end. The below syntax shows how to call functions in JavaScript:

Syntax:

functionName(Value1, Value2, ..);

Example 5: Below is a sample program that illustrates the working of functions in JavaScript:

- JavaScript

```
function welcomeMsg(name) {
    return ("Hello " + name + " welcome to GeeksforGeeks");

}

// creating a variable
let nameVal = "Admin";

// calling the function
console.log(welcomeMsg(nameVal));
```

Output:
Hello Admin welcome to GeeksforGeeks

Return Statement:

There are some situations when we want to return some values from a function after performing some operations. In such cases, we can make use of the return statement in JavaScript. This is an optional statement and most of the time the last statement in a JavaScript function. Look at our first example with the function named as *calcAddition*. This function is calculating two numbers and then returns the result.
Syntax: The most basic syntax for using the return statement is:
return value;

The return statement begins with the keyword *return* separated by the value which we want to return from it. We can use an expression also instead of directly returning the value.
Functions:

- Javascript | Arrow functions
- JavaScript | escape()
- JavaScript | unescape()
- JavaScript | Window print()
- Javascript | Window Blur() and Window Focus() Method
- JavaScript | console.log()
- JavaScript | parseFloat()
- JavaScript | uneval()
- JavaScript | parseInt()
- JavaScript | match()
- JavaScript | Date.parse()
- JavaScript | Replace() Method
- JavaScript | Map.get( )
- JavaScript | Map.entries( )
- JavaScript | Map.clear( )
- JavaScript | Map.delete()
- JavaScript | Map.has( )

Concept of Object oriented Development

In JavaScript, object-oriented development is based on a prototype-based model rather than a class-based model, as seen in languages like Java. Here are the key concepts of object-oriented development in JavaScript:

1. **Objects and Properties:**
   - In JavaScript, objects are collections of key-value pairs where keys are strings (or Symbols) and values can be of any data type.
   - Properties of an object can hold data or functions (methods).

```
var person = {
  firstName: "John",
  lastName: "Doe",
  getFullName: function() {
    return this.firstName + " " + this.lastName;
  }
};
```

2. **Prototypes and Inheritance:**
   - Every object in JavaScript has a prototype, which is another object. If a property or method is not found on the object itself, JavaScript looks for it in the object's prototype, forming a prototype chain.
   - Objects can inherit properties and methods from their prototypes.

```
var student = Object.create(person); // 'student' inherits from 'person'

student.school = "XYZ School";
```

3. **Constructor Functions:**
   - While JavaScript doesn't have classes in the traditional sense, constructor functions can be used to create objects with shared properties and methods.
   - The **new** keyword is used to instantiate objects from constructor functions.

```
function Person(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
}

Person.prototype.getFullName = function() {
  return this.firstName + " " + this.lastName;
};

var person1 = new Person("Alice", "Smith");
```

4. **Encapsulation:**
   - Encapsulation can be achieved through closures and private variables within functions.

```
function Counter() {
  var count = 0;

  this.increment = function() {
    count++;
  };

  this.getCount = function() {
    return count;
  };
}

var counter = new Counter();
counter.increment();
console.log(counter.getCount()); // Outputs: 1
```

5. **Polymorphism:**
   - JavaScript supports polymorphism through dynamic typing, allowing objects to be used in multiple contexts without explicit type definitions.

```
function displayInfo(obj) {
  console.log(obj.getDetails());
}

var student = {
  name: "Bob",
  getDetails: function() {
    return "Student: " + this.name;
  }
};

displayInfo(student); // Outputs: Student: Bob
```

In summary, JavaScript's object-oriented development is centered around objects, prototypes, and constructor functions. It offers a flexible and dynamic approach to building and extending objects, making it well-suited for web development where dynamic and responsive behaviors are essential.

Document Object Model

The **document object** represents the whole html document.

When html document is loaded in the browser, it becomes a document object. It is the **root element** that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

As mentioned earlier, it is the object of window. So

window.document

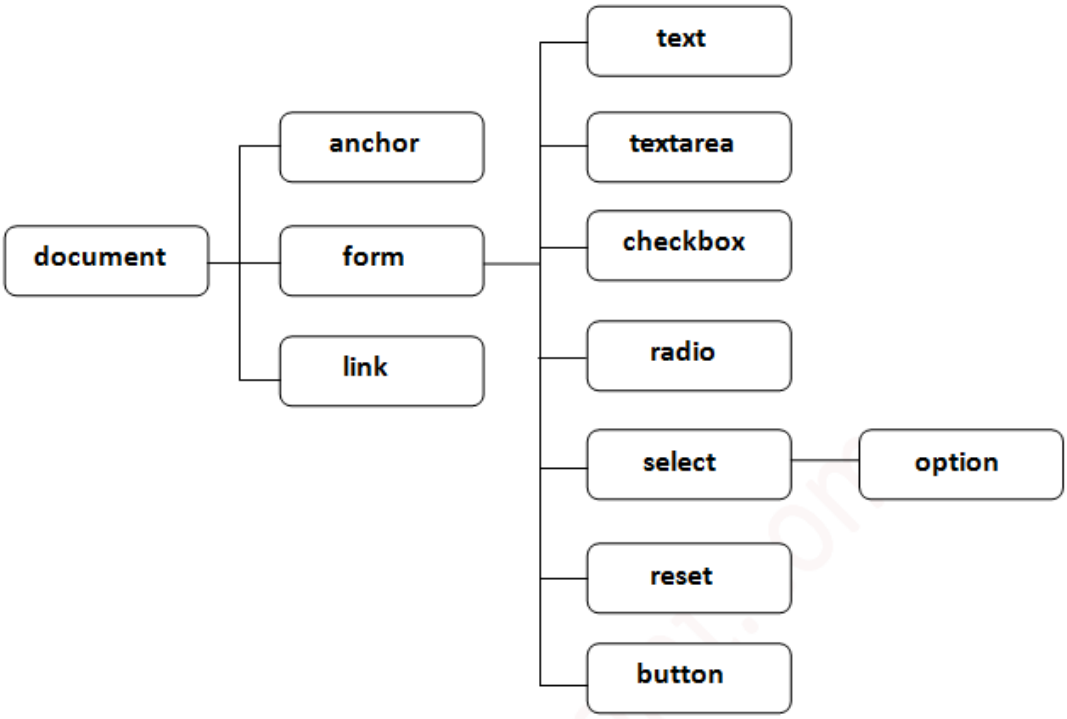
Is same as

document

Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

Properties of document object

Let's see the properties of document object that can be accessed and modified by the document object.



Methods of document object

We can access and change the contents of document by its methods.
The important methods of document object are as follows:

Method	Description
write("string")	writes the given string on the doucment.
writeln("string")	writes the given string on the doucment with newline character at the end.
getElementById()	returns the element having the given id value.
getElementsByName()	returns all the elements having the given name value.
getElementsByTagName()	returns all the elements having the given tag name.
getElementsByClassName()	returns all the elements having the given class name.

Accessing field value by document object

In this example, we are going to get the value of input text by user. Here, we are using **document.form1.name.value** to get the value of name field.
Here, **document** is the root element that represents the html document.
form1 is the name of the form.
name is the attribute name of the input text.
value is the property, that returns the value of the input text.
Let's see the simple example of document object that prints name with welcome message.

```
<script type="text/javascript">
function printvalue(){
var name=document.form1.name.value;
alert("Welcome: "+name);
}
</script>

<form name="form1">
Enter Name:<input type="text" name="name"/>
<input type="button" onclick="printvalue()" value="print name"/>
</form>
```

the **document.getElementById()** method returns the element of specified id.In the previous page, we have used **document.form1.name.value** to get the value of the input value. Instead of this, we can use document.getElementById() method to get value of the input text. But we need to define id for the input field. Let's see the simple example of document.getElementById() method that prints cube of the given number.

```
<script type="text/javascript">
```

```
function getcube(){
var number=document.getElementById("number").value;
alert(number*number*number);
}
</script>
<form>
Enter No:<input type="text" id="number" name="number"/><br/>
<input type="button" value="cube" onclick="getcube()"/>
</form>
```

Example of document.getElementsByName() method

In this example, we going to count total number of genders. Here, we are using getElementsByName() method to get all the genders.

```
<script type="text/javascript">
function totalelements()
{
var allgenders=document.getElementsByName("gender");
alert("Total Genders:"+allgenders.length);
}
</script>
<form>
Male:<input type="radio" name="gender" value="male">
Female:<input type="radio" name="gender" value="female">

<input type="button" onclick="totalelements()" value="Total Genders">
</form>
```

Example of document.getElementsByTagName() method

In this example, we going to count total number of paragraphs used in the document. To do this, we have called the document.getElementsByTagName("p") method that returns the total paragraphs.

```
<script type="text/javascript">
function countpara(){
var totalpara=document.getElementsByTagName("p");
alert("total p tags are: "+totalpara.length);

}
</script>
<p>This is a pragraph</p>
<p>Here we are going to count total number of paragraphs by getElementByTagName() method.</p>
<p>Let's see the simple example</p>
<button onclick="countpara()">count paragraph</button>
```

In this example, we are dynamically writing the html form inside the div name having the id mylocation. We are identifing this position by calling the document.getElementById() method.

```
1. <script type="text/javascript" >
2. function showcommentform() {
3. var data="Name:<input type='text' name='name'><br>Comment:<br><textarea rows='5' cols=
   '80'></textarea>
4. <br><input type='submit' value='Post Comment'>";
5. document.getElementById('mylocation').innerHTML=data;
6. }
7. </script>
8. <form name="myForm">
9. <input type="button" value="comment" onclick="showcommentform()">
10. <div id="mylocation"></div>
11. </form>
```

JavaScript Form

Summary: in this tutorial, you will learn about JavaScript form API: accessing the form, getting values of the elements, validating form data, and submitting the form.

Form basics

To create a form in HTML, you use the <form> element:

<form action="/signup" method="post" id="signup">
</form>Code language: HTML, XML (xml)

The <form> element has two important attributes: action and method.

- The action attribute specifies a URL that will process the form submission. In this example, the action is the /signup URL.
- The method attribute specifies the HTTP method to submit the form with. Usually, the method is either post or get.

Generally, you use the get method when you want to retrieve data from the server and the post method when you want to change data on the server.

JavaScript uses the HTMLFormElement object to represent a form.
The HTMLFormElement has the following properties that correspond to the HTML attributes:

- action – is the URL that processes the form data. It is equivalent to the action attribute of the <form> element.
- method – is the HTTP method which is equivalent to the method attribute of the <form> element.

The HTMLFormElement element also provides the following useful methods:

- submit() – submit the form.
- reset() – reset the form.

Referencing forms

To reference the <form> element, you can use DOM selecting methods such as getElementById():

const form = document.getElementById('subscribe');Code language: JavaScript (javascript)

An HTML document can have multiple forms. The document.forms property returns a collection of forms (HTMLFormControlsCollection) on the document:

document.formsCode language: JavaScript (javascript)

To reference a form, you use an index. For example, the following statement returns the first form of the HTML document:

document.forms[0]Code language: CSS (css)

Submitting a form

Typically, a form has a submit button. When you click it, the browser sends the form data to the server. To create a submit button, you use <input> or <button> element with the type submit:

<input type="submit" value="Subscribe">Code language: HTML, XML (xml)

Or

<button type="submit">Subscribe**</button>**Code language: HTML, XML (xml)

If the submit button has focus and you press the Enter key, the browser also submits the form data.

When you submit the form, the submit event is fired before the request is sent to the server. This gives you a chance to validate the form data. If the form data is invalid, you can stop submitting the form.

To attach an event listener to the submit event, you use the addEventListener() method of the form element as follows:

```
const form  = document.getElementById('signup');

form.addEventListener('submit', (event) => {
   // handle the form data
});
```Code language: JavaScript (javascript)

To stop the form from being submitted, you call the preventDefault() method of the event object inside the submit event handler like this:

```
form.addEventListener('submit', (event) => {
   // stop form submission
   event.preventDefault();
});
```Code language: PHP (php)

Typically, you call the event.preventDefault() method if the form data is invalid. To submit the form in JavaScript, you call the submit() method of the form object:

form.submit();Code language: CSS (css)

Note that the form.submit() does not fire the submit event. Therefore, you should always validate the form before calling this method.

***Accessing form fields***

To access form fields, you can use DOM methods like getElementsByName(), getElementById(), querySelector(), etc.

Also, you can use the elements property of the form object. The form.elements property stores a collection of the form elements.

JavaScript allows you to access an element by index, id, or name. Suppose that you have the following signup form with two <input> elements:

```
<form action="signup.html" method="post" id="signup">
        <h1>Sign Up</h1>
        <div class="field">
                <label for="name">Name:</label>
                <input type="text" id="name" name="name" placeholder="Enter your fullname" />
                <small></small>
        </div>
        <div class="field">
                <label for="email">Email:</label>
                <input type="text" id="email" name="email" placeholder="Enter your email address" />
                <small></small>
        </div>
        <button type="submit">Subscribe</button>
</form>
```

Code language: HTML, XML (xml)

To access the elements of the form, you get the form element first:

const form = document.getElementById('signup');Code language: JavaScript (javascript)

And use index, id, or name to access the element. The following accesses the first form element:

form.elements[0]; *// by index*
form.elements['name']; *// by name*
form.elements['name']; *// by id (name & id are the same in this case)*Code language: JavaScript (javascript)

The following accesses the second input element:

```
form.elements[1]; // by index
form.elements['email']; //  by name
form.elements['email']; // by idCode language: JavaScript (javascript)
```

After accessing a form field, you can use the value property to access its value, for example:

```
const form = document.getElementById('signup');
const name = form.elements['name'];
const email = form.elements['email'];

// getting the element's value
let fullName = name.value;
let emailAddress = email.value;Code language: JavaScript (javascript)
```

Put it all together: signup form

The following illustrates the HTML document that has a signup form. See the live demo here.

```
<!DOCTYPE html>
<html lang="en">
	<head>
		<title>JavaScript Form Demo</title>
		<meta name="viewport" content="width=device-width, initial-
scale=1.0" />
		<link rel="stylesheet" href="css/style.css" />
	</head>
	<body>
		<div class="container">
			<form action="signup.html" method="post" id="signup">
				<h1>Sign Up</h1>
				<div class="field">
					<label for="name">Name:</label>
					<input type="text" id="name"
name="name" placeholder="Enter your fullname" />
					<small></small>
				</div>
				<div class="field">
					<label for="email">Email:</label>
					<input type="text" id="email"
name="email" placeholder="Enter your email address" />
					<small></small>
				</div>
```

```
                                        <div class="field">
                                                <button type="submit"
class="full">Subscribe</button>
                                        </div>
                                </form>
                        </div>
                        <script src="js/app.js"></script>
        </body>
</html>
```
Code language: HTML, XML (xml)

The HTML document references the style.css and app.js files. It uses the <small> element to display an error message in case the <input> element has invalid data.

Submitting the form without providing any information will result in the following error:

:

The following shows the complete app.js file:

```
// show a message with a type of the input
function showMessage(input, message, type) {
        // get the small element and set the message
        const msg = input.parentNode.querySelector("small");
        msg.innerText = message;
        // update the class for the input
        input.className = type ? "success" : "error";
        return type;
}

function showError(input, message) {
        return showMessage(input, message, false);
}

function showSuccess(input) {
        return showMessage(input, "", true);
}

function hasValue(input, message) {
        if (input.value.trim() === "") {
                return showError(input, message);
        }
        return showSuccess(input);
}
```

```
function validateEmail(input, requiredMsg, invalidMsg) {
	// check if the value is not empty
	if (!hasValue(input, requiredMsg)) {
		return false;
	}
	// validate email format
	const emailRegex =
		/^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-
9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;

	const email = input.value.trim();
	if (!emailRegex.test(email)) {
		return showError(input, invalidMsg);
	}
	return true;
}

const form = document.querySelector("#signup");

const NAME_REQUIRED = "Please enter your name";
const EMAIL_REQUIRED = "Please enter your email";
const EMAIL_INVALID = "Please enter a correct email address format";

form.addEventListener("submit", function (event) {
	// stop form submission
	event.preventDefault();

	// validate the form
	let nameValid = hasValue(form.elements["name"], NAME_REQUIRED);
	let emailValid = validateEmail(form.elements["email"], EMAIL_REQUIRED,
EMAIL_INVALID);
	// if valid, submit the form.
	if (nameValid && emailValid) {
		alert("Demo only. No form was posted.");
	}
});
```

Code language: JavaScript (javascript)

How it works.

The showMessage() function

The showMessage() function accepts an input element, a message, and a type:

```
// show a message with a type of the input
function showMessage(input, message, type) {
        // get the <small> element and set the message
        const msg = input.parentNode.querySelector("small");
        msg.innerText = message;
        // update the class for the input
        input.className = type ? "success" : "error";
        return type;
}Code language: JavaScript (javascript)
```

The following shows the name input field on the form:

```
<div class="field">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" placeholder="Enter your fullname" />
        <small></small>
</div>Code language: HTML, XML (xml)
```

If the name's value is blank, you need to get its parent first which is the with the class "field".

```
input.parentNodeCode language: CSS (css)
```

Next, you need to select the <small> element:

```
const msg = input.parentNode.querySelector("small");Code language: JavaScript (javascript)
```

Then, update the message:

```
msg.innerText = message;
```

After that, we change the CSS class of the input field based on the value of the type parameter. If the type is true, we change the class of the input to success. Otherwise, we change the class to error.

```
input.className = type ? "success" : "error";Code language: JavaScript (javascript)
```

Finally, return the value of the type:

```
return type;Code language: JavaScript (javascript)
```

The showError() and showSuccess() functions

The the showError() and showSuccess() functions call the showMessage() function. The showError() function always returns false whereas the showSuccess() function always returns true. Also, the showSuccess() function sets the error message to an empty string.

```
function showError(input, message) {
	return showMessage(input, message, false);
}

function showSuccess(input) {
	return showMessage(input, "", true);
}Code language: JavaScript (javascript)
```

The hasValue() function

The hasValue() function checks if an input element has a value or not and shows an error message using the showError() or showSuccess() function accordingly:

```
function hasValue(input, message) {
	if (input.value.trim() === "") {
		return showError(input, message);
	}
	return showSuccess(input);
}Code language: JavaScript (javascript)
```

The validateEmail() function

The validateEmail() function validates if an email field contains a valid email address:

```
function validateEmail(input, requiredMsg, invalidMsg) {
	// check if the value is not empty
	if (!hasValue(input, requiredMsg)) {
		return false;
	}
	// validate email format
	const emailRegex =
		/^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-
9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;

	const email = input.value.trim();
	if (!emailRegex.test(email)) {
		return showError(input, invalidMsg);
	}
```

```
        return true;
}Code language: PHP (php)
```

The validateEmail() function calls the hasValue() function to check if the field value is empty. If the input field is empty, it shows the requiredMsg.

To validate the email, it uses a regular expression. If the email is invalid, the validateEmail() function shows the invalidMsg.

The submit event handler

First, select the signup form by its id using the querySelector() method:

```
const form = document.querySelector("#signup");Code language: JavaScript (javascript)
```

Second, define some constants to store the error messages:

```
const NAME_REQUIRED = "Please enter your name";
const EMAIL_REQUIRED = "Please enter your email";
const EMAIL_INVALID = "Please enter a correct email address format";Code language: JavaScript (javascript)
```

Third, add the submit event listener of the signup form using the addEventListener() method:

```
form.addEventListener("submit", function (event) {
        // stop form submission
        event.preventDefault();

        // validate the form
        let nameValid = hasValue(form.elements["name"], NAME_REQUIRED);
        let emailValid = validateEmail(form.elements["email"], EMAIL_REQUIRED, EMAIL_INVALID);
        // if valid, submit the form.
        if (nameValid && emailValid) {
                alert("Demo only. No form was posted.");
        }
});
Code language: JavaScript (javascript)
```

In the submit event handler:

1. Stop the form submission by calling the event.preventDefault() method.
2. Validate the name and email fields using the hasValue() and validateEmail() functions.
3. If both name and email are valid, show an alert. In a real-world application, you need to call the form.submit() method to submit the form.

Summary

- Use the <form> element to create an HTML form.

- Use DOM methods such as getElementById() and querySelector() to select a <form> element. The document.forms[index] also returns the form element by a numerical index.
- Use form.elements to access form elements.
- The submit event fires when users click the submit button on the form.

Java Script and dynamic web pages

- Concept of Cookies
- Cascaded Style Sheets
- Error Handling in JavaScript
- Concept of AJAX

Concept of Cookies

Cookies let you store user information in web pages.

What are Cookies?
Cookies are data, stored in small text files, on your computer.

When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Cookies were invented to solve the problem "how to remember information about the user":

- When a user visits a web page, his/her name can be stored in a cookie.
- Next time the user visits the page, the cookie "remembers" his/her name.

Cookies are saved in name-value pairs like:

username = John Doe

When a browser requests a web page from a server, cookies belonging to the page are added to the request. This way the server gets the necessary data to "remember" information about users.

None of the examples below will work if your browser has local cookies support turned off.

Create a Cookie with JavaScript
JavaScript can create, read, and delete cookies with the document.cookie property.

With JavaScript, a cookie can be created like this:

```
document.cookie = "username=John Doe";
```

You can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed:

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC";
```

With a path parameter, you can tell the browser what path the cookie belongs to. By default, the cookie belongs to the current page.

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
```

Read a Cookie with JavaScript
With JavaScript, cookies can be read like this:

```
let x = document.cookie;
```

document.cookie will return all cookies in one string much like: cookie1=value; cookie2=value; cookie3=value;

Change a Cookie with JavaScript

With JavaScript, you can change a cookie the same way as you create it:
document.cookie = "username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
The old cookie is overwritten.

Delete a Cookie with JavaScript

Deleting a cookie is very simple.
You don't have to specify a cookie value when you delete a cookie.
Just set the expires parameter to a past date:
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";
You should define the cookie path to ensure that you delete the right cookie.
Some browsers will not let you delete a cookie if you don't specify the path.

The Cookie String

The document.cookie property looks like a normal text string. But it is not.

Even if you write a whole cookie string to document.cookie, when you read it out again, you can only see the name-value pair of it.

If you set a new cookie, older cookies are not overwritten. The new cookie is added to document.cookie, so if you read document.cookie again you will get something like:

cookie1 = value; cookie2 = value;

Display All Cookies Create Cookie 1 Create Cookie 2 Delete Cookie 1 Delete Cookie 2

If you want to find the value of one specified cookie, you must write a JavaScript function that searches for the cookie value in the cookie string.

JavaScript Cookie Example

In the example to follow, we will create a cookie that stores the name of a visitor.

The first time a visitor arrives to the web page, he/she will be asked to fill in his/her name. The name is then stored in a cookie.

The next time the visitor arrives at the same page, he/she will get a welcome message.

For the example we will create 3 JavaScript functions:

1. A function to set a cookie value
2. A function to get a cookie value
3. A function to check a cookie value

A Function to Set a Cookie

First, we create a function that stores the name of the visitor in a cookie variable:

Example

```
function setCookie(cname, cvalue, exdays) {
  const d = new Date();
  d.setTime(d.getTime() + (exdays*24*60*60*1000));
  let expires = "expires="+ d.toUTCString();
  document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";
}
```

Example explained:

The parameters of the function above are the name of the cookie (cname), the value of the cookie (cvalue), and the number of days until the cookie should expire (exdays).

The function sets a cookie by adding together the cookiename, the cookie value, and the expires string.

A Function to Get a Cookie

Then, we create a function that returns the value of a specified cookie:

Example

```
function getCookie(cname) {
  let name = cname + "=";
  let decodedCookie = decodeURIComponent(document.cookie);
  let ca = decodedCookie.split(';');
  for(let i = 0; i <ca.length; i++) {
    let c = ca[i];
    while (c.charAt(0) == ' ') {
      c = c.substring(1);
    }
    if (c.indexOf(name) == 0) {
      return c.substring(name.length, c.length);
    }
  }
  return "";
}
```

Function explained:

Take the cookiename as parameter (cname).

Create a variable (name) with the text to search for (cname + "=").

Decode the cookie string, to handle cookies with special characters, e.g. '$'

Split document.cookie on semicolons into an array called ca (ca = decodedCookie.split(';')).

Loop through the ca array (i = 0; i < ca.length; i++), and read out each value c = ca[i]).

If the cookie is found (c.indexOf(name) == 0), return the value of the cookie (c.substring(name.length, c.length).

If the cookie is not found, return "".

A Function to Check a Cookie

Last, we create the function that checks if a cookie is set.

If the cookie is set it will display a greeting.

If the cookie is not set, it will display a prompt box, asking for the name of the user, and stores the username cookie for 365 days, by calling the setCookie function:

Example

```
function checkCookie() {
  let username = getCookie("username");
  if (username != "") {
   alert("Welcome again " + username);
  } else {
    username = prompt("Please enter your name:", "");
    if (username != "" && username != null) {
      setCookie("username", username, 365);
    }
  }
}
```

All Together Now

Example

```
function setCookie(cname, cvalue, exdays) {
  const d = new Date();
  d.setTime(d.getTime() + (exdays * 24 * 60 * 60 * 1000));
  let expires = "expires="+d.toUTCString();
  document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";
}

function getCookie(cname) {
  let name = cname + "=";
  let ca = document.cookie.split(';');
  for(let i = 0; i < ca.length; i++) {
    let c = ca[i];
    while (c.charAt(0) == ' ') {
      c = c.substring(1);
    }
    if (c.indexOf(name) == 0) {
      return c.substring(name.length, c.length);
    }
  }
```

```
  return "";
}

function checkCookie() {
  let user = getCookie("username");
  if (user != "") {
    alert("Welcome again " + user);
  } else {
    user = prompt("Please enter your name:", "");
    if (user != "" && user != null) {
      setCookie("username", user, 365);
    }
  }
}
```

The example above runs the checkCookie() function when the page loads

Cascaded Style Sheets

It seems there might be a slight confusion in your question. Cascading Style Sheets (CSS) and JavaScript are two distinct technologies used in web development, each with its own purpose. CSS is used for styling and layout, while JavaScript is a scripting language used for enhancing interactivity and manipulating the behavior of web pages. However, it's common for JavaScript to interact with and manipulate CSS to achieve dynamic effects on a webpage.

Here's a brief overview of how JavaScript can be used with CSS in web development:

1. **DOM Manipulation:**
   - JavaScript can manipulate the Document Object Model (DOM), which represents the structure of a webpage. Through the DOM, you can dynamically change the styles of HTML elements.

```
// Example: Change the color of a paragraph using JavaScript var paragraph = document.getElementById("myParagraph"); paragraph.style.color = "blue";
```

2. **Event Handling:**
   - JavaScript is often used to handle events such as clicks, mouseovers, or keyboard inputs. These events can trigger changes in CSS styles.

```
// Example: Change the background color on button click var button = document.getElementById("myButton"); button.addEventListener("click", function() { document.body.style.backgroundColor = "yellow"; });
```

3. **Animations and Transitions:**
   - JavaScript can be used to create animations and transitions by dynamically modifying CSS properties over time.

```
// Example: Create a simple fade-in effect var element = document.getElementById("fadeInElement"); element.style.opacity = 0; function fadeIn() { var opacity = 0; var interval = setInterval(function() { if (opacity >= 1) { clearInterval(interval); } else { opacity += 0.1; element.style.opacity = opacity; } }, 100); } fadeIn();
```

4. **CSS Class Manipulation:**
   - JavaScript can add, remove, or toggle CSS classes dynamically, allowing for more organized and modular styling.

```
// Example: Toggle a class on button click var button = document.getElementById("myButton"); var element = document.getElementById("myElement"); button.addEventListener("click", function() { element.classList.toggle("highlight"); });
```

In summary, while CSS and JavaScript serve different purposes, JavaScript is commonly used to dynamically interact with and modify CSS styles to create more interactive and responsive web pages.

JavaScript Errors

Throw, and Try...Catch...Finally

The try statement defines a code block to run (to try).

The catch statement defines a code block to handle any error.

The finally statement defines a code block to run regardless of the result.

The throw statement defines a custom error.

Errors Will Happen!

When executing JavaScript code, different errors can occur.

Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.

Example

In this example we misspelled "alert" as "adddlert" to deliberately produce an error:

```
<p id="demo"></p>

<script>
try {
  adddlert("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
</script>
```

*JavaScript catches **adddlert** as an error, and executes the catch code to handle it.*

JavaScript try and catch

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The JavaScript statements try and catch come in pairs:

```
try {
  Block of code to try
}
catch(err) {
```

```
  Block of code to handle errors
}
```

JavaScript Throws Errors

When an error occurs, JavaScript will normally stop and generate an error message.

The technical term for this is: JavaScript will **throw an exception (throw an error)**.

*JavaScript will actually create an **Error object** with two properties: **name** and **message**.*

The throw Statement

The throw statement allows you to create a custom error.

Technically you can **throw an exception (throw an error)**.

The exception can be a JavaScript String, a Number, a Boolean or an Object:

```
throw "Too big";    // throw a text
throw 500;          // throw a number
```

If you use throw together with try and catch, you can control program flow and generate custom error messages.

Input Validation Example

This example examines input. If the value is wrong, an exception (err) is thrown.

The exception (err) is caught by the catch statement and a custom error message is displayed:

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

<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>

<script>
function myFunction() {
  const message = document.getElementById("p01");
  message.innerHTML = "";
  let x = document.getElementById("demo").value;
  try {
```

```
    if(x.trim() == "") throw "empty";
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x < 5) throw "too low";
    if(x > 10) throw "too high";
  }
  catch(err) {
    message.innerHTML = "Input is " + err;
  }
}
</script>

</body>
</html>
```

HTML Validation

The code above is just an example.

Modern browsers will often use a combination of JavaScript and built-in HTML validation, using predefined validation rules defined in HTML attributes:

```
<input id="demo" type="number" min="5" max="10" step="1">
```

You can read more about forms validation in a later chapter of this tutorial.

The finally Statement

The finally statement lets you execute code, after try and catch, regardless of the result:

Syntax

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
finally {
  Block of code to be executed regardless of the try / catch result
}
```

Example

```
function myFunction() {
  const message = document.getElementById("p01");
  message.innerHTML = "";
  let x = document.getElementById("demo").value;
  try {
```

```
    if(x.trim() == "") throw "is empty";
    if(isNaN(x)) throw "is not a number";
    x = Number(x);
    if(x > 10) throw "is too high";
    if(x < 5) throw "is too low";
  }
  catch(err) {
    message.innerHTML = "Error: " + err + ".";
  }
  finally {
    document.getElementById("demo").value = "";
  }
}
```

The Error Object

JavaScript has a built in error object that provides error information when an error occurs.

The error object provides two useful properties: name and message.

Error Object Properties

Property	Description
name	Sets or returns an error name
message	Sets or returns an error message (a string)

Error Name Values

Six different values can be returned by the error name property:

Error Name	Description
EvalError	An error has occurred in the eval() function
RangeError	A number "out of range" has occurred
ReferenceError	An illegal reference has occurred
SyntaxError	A syntax error has occurred
TypeError	A type error has occurred

URIError	An error in encodeURI() has occurred

The six different values are described below.

Eval Error

An EvalError indicates an error in the eval() function.

Newer versions of JavaScript do not throw EvalError. Use SyntaxError instead.

Range Error

A RangeError is thrown if you use a number that is outside the range of legal values.

For example: You cannot set the number of significant digits of a number to 500.

Example

```
let num = 1;
try {
  num.toPrecision(500);   // A number cannot have 500 significant digits
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name;
}
```

Reference Error

A ReferenceError is thrown if you use (reference) a variable that has not been declared:

Example

```
let x = 5;
try {
  x = y + 1;   // y cannot be used (referenced)
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name;
}
```

Syntax Error

A SyntaxError is thrown if you try to evaluate code with a syntax error.

Example

```
try {
  eval("alert('Hello)");   // Missing ' will produce an error
```

```
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name;
}
```

Type Error

A TypeError is thrown if an operand or argument is incompatible with the type expected by an operator or function.

Example

```
let num = 1;
try {
  num.toUpperCase();   // You cannot convert a number to upper case
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name;
}
```

URI (Uniform Resource Identifier) Error

A URIError is thrown if you use illegal characters in a URI function:

Example

```
try {
  decodeURI("%%%");   // You cannot URI decode percent signs
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name;
}
```

Non-Standard Error Object Properties

Mozilla and Microsoft define some non-standard error object properties:

fileName (Mozilla)
lineNumber (Mozilla)
columnNumber (Mozilla)
stack (Mozilla)
description (Microsoft)
number (Microsoft)

Do not use these properties in public web sites. They will not work in all browsers.

Concept of AJAX

AJAX is a developer's dream, because you can:

- Read data from a web server - after a web page has loaded
- Update a web page without reloading the page
- Send data to a web server - in the background

HTML Page

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

<div id="demo">
  <h2>Let AJAX change this text</h2>
  <button type="button" onclick="loadDoc()">Change Content</button>
</div>

</body>
</html>
```

The HTML page contains a section and a <button>.

The section is used to display information from a server.

The <button> calls a function (if it is clicked).

The function requests data from a web server and displays it:

Function loadDoc()

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
     document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

What is AJAX?

AJAX is an acronym for **Asynchronous JavaScript and XML**. It is a group of inter-related technologies like *JavaScript*, DOM, *XML*, *HTML/XHTML*, *CSS*, *XMLHttpRequest* etc.

AJAX allows you to send and receive data asynchronously without reloading the web page. So it is fast.

AJAX allows you to send only important information to the server not the entire page. So only valuable data from the client side is routed to the server side. It makes your application interactive and faster.

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX is not a programming language.

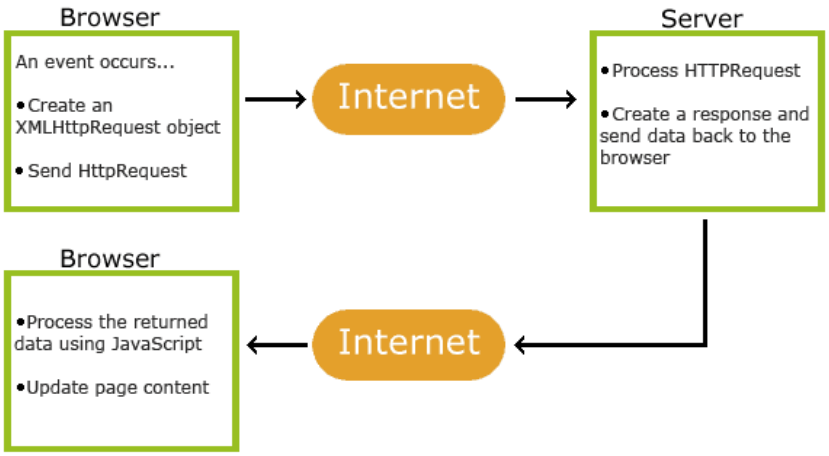
AJAX just uses a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

AJAX is a misleading name. AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text.

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

How AJAX Works



- 1. An event occurs in a web page (the page is loaded, a button is clicked)
- 2. An XMLHttpRequest object is created by JavaScript
- 3. The XMLHttpRequest object sends a request to a web server
- 4. The server processes the request
- 5. The server sends a response back to the web page
- 6. The response is read by JavaScript
- 7. Proper action (like page update) is performed by JavaScript