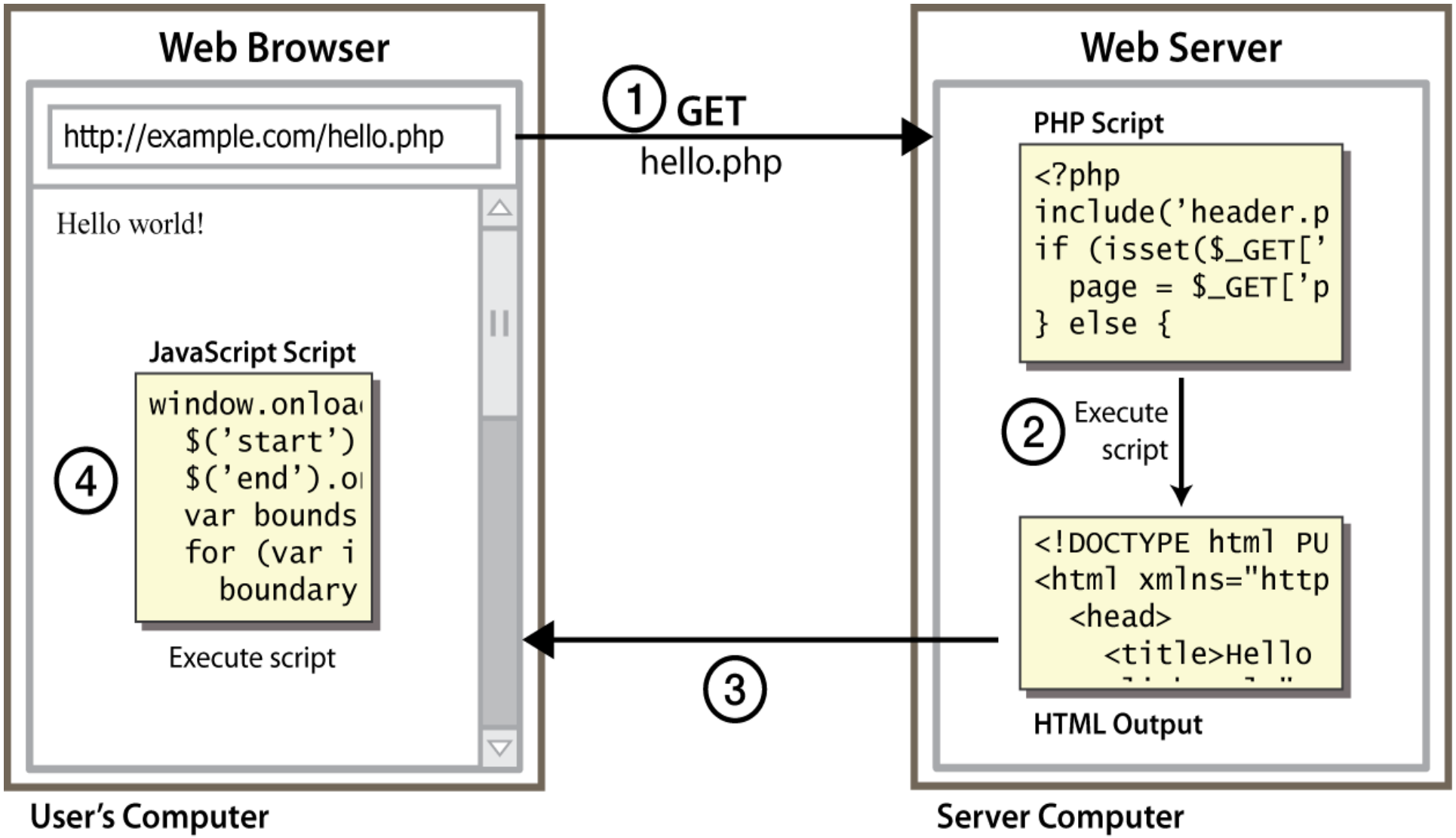
Chp:4 Javascript

Prepared by: Er. Simanta Kasaju

What's a Scripting Language?

- Language used to write programs that compute inputs to another language processor
  - One language embedded in another
    - Embedded JavaScript computes HTML input to the browser
    - Shell scripts compute commands executed by the shell
- Common characteristics of scripting languages
  - String processing – since commands often strings
  - Simple program structure, define things "on the fly"
  - Flexibility preferred over efficiency, safety
    - Is lack of safety a good thing? (Example: JavaScript used for Web applications…)

Client Side Scripting



Why use client-side programming?

PHP already allows us to create dynamic web pages. Why also use client-side scripting?

- client-side scripting (JavaScript) benefits:
  - **usability**: can modify a page without having to post back to the server (faster UI)
  - **efficiency**: can make small, quick changes to the page without waiting for the server
  - **event-driven**: can respond to user actions like clicks and key presses.
- server-side programming (PHP) benefits:
  - **security**: has access to server's private data; client can't see source code
  - **compatibility**: not subject to browser compatibility issues
  - **power**: can write files, open connections to servers, and connect to databases, ...

Introduction

- It is a scripting language most often used for client-side web development.
- It is an object-oriented scripting language that enables us to create interactive effects on web pages.
- JavaScript is the programming language of the Web.
- 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

Cont..

- a lightweight programming language ("scripting language")
  - used to make web pages interactive
  - insert dynamic text into HTML (ex: user name)
  - **react to events** (ex: page load user click)
  - get information about a user's computer (ex: browser type)
  - perform calculations on user's computer (ex: form validation)
- a web standard (but not supported identically by all browsers)
- NOT related to Java other than by name and some syntactic similarities

Why JavaScript is called client-side scripting language?

- JavaScript is called client-side scripting language because it runs in web browsers for interactions with users and web browsers usually run on the client.
- For example JavaScript can be used to make HTML/XHTML documents respond to user inputs such as mouse clicks, keyboard use, etc.
- Other examples of client-side scripting languages are: Microsoft VBScript, Java Applets, Adobe Flex, etc

Evolution

- JavaScript, which was developed by Netscape, was originally named Mocha but soon was renamed LiveScript.
- In late 1995 LiveScript became a joint venture of Netscape and Sun Microsystems, and its name again was changed, to JavaScript.
- JavaScript was invented by Brendan Eich in 1995, and became an ECMA (European Computer Manufacturers Association) standard in 1997.ECMA-262 is the official name of the standard.
- ECMAScript is the official name of the language.

Advantages of using client-side scripting language:

- Allow for more interactivity by immediately responding to users' actions
- Execute quickly because they don't require a trip to the server
- Quick validation of forms inputs
- May improve the usability of Web sites for users
- Can give developers more control over the look and behavior of their Web widgets
- Are easy to manipulate codes as well as the user interface
- Can be used to change HTML elements/ attributes, CSS styles, and so on.

Can JavaScript be used for server-side scripting?

- Yes, since the advent of Node.js and its frameworks we can use JavaScript on both the Client side and server side. Node.js is a JavaScript runtime built that uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.
- MEAN (MongoDB, ExpressJS, AngularJS, and Node.js) stack, used to develop web applications is also the collection of JavaScript-based technologies.

Javascript vs Java

- interpreted, not compiled
- more relaxed syntax and rules
  - fewer and "looser" data types
  - variables don't need to be declared
  - errors often silent (few exceptions)
- key construct is the function rather than the class
  - "first-class" functions are used in many situations
- contained within a web page and integrates with its HTML/CSS content


Java

+

=


Javascript

Although there is a similarity between Java and JavaScript in the syntax of their expressions, assignment statements, and control statements but there are key differences which are as follows:

- Java is an OOP programming language (mostly compiled) while Java Script is an OOP scripting language (mostly interpreted).
- Java creates applications that run in a virtual machine or browser while JavaScript code is run on a browser only.
- Java code needs to be compiled while JavaScript code need not be compiled and are all in text.
- Java is a strongly typed language. Types are all known at compile time, and operand types are checked for compatibility. Variables in JavaScript need not be declared and are dynamically typed, making compile-time type checking impossible.
- Objects in Java are static in the sense that their collection of data members and methods is fixed at compile time. JavaScript objects are dynamic: the number of data members and methods of an object can change during execution.
- They require different plug-ins.

Java	JavaScript
Java is a strongly typed language and variables must be declared first to use in the program. In Java, the type of a variable is checked at compile-time.	JavaScript is a loosely typed language and has a more relaxed syntax and rules.
Java is an object-oriented programming language.	JavaScript is an object-based scripting language.
Java applications can run in any virtual machine(JVM) or browser.	JavaScript code used to run only in the browser, but now it can run on the server via Node.js.
Objects of Java are class-based even we can't make any program in java without creating a class.	JavaScript Objects are prototype-based.
Java program has the file extension ".Java" and translates source code into bytecodes which are executed by JVM(Java Virtual Machine).	JavaScript file has the file extension ".js" and it is interpreted but not compiled, every browser has the Javascript interpreter to execute JS code.if compile time
Java is a Standalone language.	contained within a web page and integrates with its HTML content.
Java has a thread-based approach to concurrency.	Javascript has an event-based approach to concurrency.

Java supports multithreading.	Javascript doesn't support multi-threading.
Java is mainly used for backend	Javascript is used for the frontend and backend both.
Java uses more memory	Javascript uses less memory.
Java requires a Java Development Kit(JDK) to run the code	Javascript requires any text editor or browser console to run the code

JavaScript vs. PHP

- similarities:
  - both are interpreted, not compiled
  - both are relaxed about syntax, rules, and types
  - both are case-sensitive
  - both have built-in regular expressions for powerful text processing

cont..

- differences:
  - JS is more object-oriented: noun.verb(), less procedural: verb(noun)
  - JS focuses on user interfaces and interacting with a document; PHP is geared toward HTML output and file/form processing
  - JS code runs on the client's browser; PHP code runs on the web server

S.No.	Javascript	PHP
1.	Javascript does the job for Both Front-end and Back-end.	PHP is used mostly for Back-end purposes only.
2.	Javascript is synchronous but it has a lot of features like callbacks, promises, async/await which allows implementing asynchronous event handling	PHP is synchronous, It waits for IO operations to execute.
3.	Javascript can be run in browsers and after Node, we can also run it in the Command line.	PHP requires a server to run. Cannot run without a server.
4.	Javascript can be combined with HTML, AJAX, and XML.	PHP can be combined with HTML only.
5.	Javascript is a single-threaded language that is event-driven which means it never blocks and everything runs concurrently.	PHP is multi-threaded which means it blocks I/O to carry out multiple tasks concurrently.
6	Javascript code is less secure.	PHP code is highly secure.
7	Javascript requires an environment for accessing the database.	PHP allows easy and direct access to the database.
8	JavaScript is used to create real-time games and applications, mobile applications etc.	A PHP program is used to create dynamic pages, send cookies, receive cookies,

9	JavaScript is case sensitive in case of functions.	PHP is not case sensitive in case of functions.
10	Brendan Eich in 1995 developed JavaScript.	Rasmus Lerdorf in 1994 developed PHP.
11	JavaScript files are saved with ***.js*** extension.	PHP files are saved with an extension ***.PHP***

Parts of JavaScript:

- JavaScript can be divided into 3 parts:

i. The Core JavaScript: The core JavaScript includes its operators, expressions, statements and subprograms.

ii. **Client-side JavaScript:** It is the collection of objects that supports the control of browser and interactions with users. For example, XHTML/HTML document made responsive to user inputs such as mouse clicks, keyboard use, etc. with the help of JavaScript.

iii. Server-side JavaScript: It is a collection of objects that make the language useful on a Web server—for example, to support communication with a database management system.

Examples of Uses of JavaScript as client-side scripting language:

- Interactions with users through form elements, such as buttons and menus, can be conveniently described in JavaScript. Because button clicks and mouse movements are easily detected with JavaScript, they can be used to trigger computations and provide feedback to the user.
- Interactions with user without forms, which take place in dialog windows, include getting input from the user and allowing the user to make choices through buttons. It is also easy to generate new content in the browser display dynamically.
- JavaScript Document Object Model (DOM), allows JavaScript scripts to access and modify the CSS properties and content of the elements of a displayed XHTML document, making formally static documents highly dynamic.

Advantages of JS

- Cross Browser supporting: JavaScript codes are supported by various browsers like Internet Explorer, Netscape Navigator, Mozilla etc.
- Platform independent: JS codes can be executed in any platform like Linux, Microsoft Windows, and many other OS.
- Lightweight for fast downloading: JS codes run directly in the client machine. The code should be downloaded all the way from the server to the client and this time duration is very minimum and the execution of the codes of JS is also very fast.
- Validating Data: The data can be validated in 2 different ways:

-Validating in the server side or in the server machine

- Validating in client side or in client machine

cont..

- Sophisticated UI: By using JS you can create a UI that can communicate with the user and the Browser.
- In Built software: You don’t need extra tools to write JS any plain text or HTML editor will do so there is no expensive development software to buy.
- Easy to learn: The JS programmer should know the minimal syntax of JS since it supports many syntax and data types as C and C++.
- Designed for programming User Events: JS supports object/Event-based Programming. So the code written in JavaScript can easily be broken down into sub-modules.

Disadvantages

- **Launching an application on the client's computer:** JS is not used to create stand-alone applications, it is only used to add some functionality in any web page.
- **Reading or writing files:** JavaScript cant read and write files into the client machines. It can only be used as a utility language to develop any web site.Retrieving the text contents of HTML pages or files from the server.
- Reading and writing files to the server: JavaScript can read and write to any file in the server as well.
- **Sending secret e-mails from website visitors to you:** JS cant be used to send emails to the visitors or users of the website. This can be done only with server-side scripting.
- **Cannot create database application:** By using you cant connect the website to the database.For this you need to use server side scripting

Cont..

- Browser Compatibility issues: Not all browsers support the JS codes. The browser may support JS as a whole but may not support the codes or line of codes written in JS and may interpret differently.
- JS does not implement multiprocessing or multithreading.
- JS has limitations of writing in a client machine. It can only write the cookie in their machine that is also of a certain size i.e 4K

Application Of JS

- Used to create interactive(dynamic) websites.
- Client-side validation.
- Displaying date and time.
- Dynamic drop-down menus
- Navigation systems
- Basic math calculations
- Displaying pop-up windows and dialog boxes(like an alert dialog box, confirm dialog box).
- Displaying clocks.
- JS can update and change both HTML and CSS.
- JS can calculate, manipulate and validate data.
- Provide events based on user action to design CSS and HTML.

Ways to embed JavaScript in XHTML/HTML document:

- i. Explicit embedding:
- JavaScript code physically residing in concerned XHTML document- may be on head or body depending on the purpose of script within desired mime type of <script> tag.
- e.g <script type “text/javascript”>
- ……js code…..
- </script>)

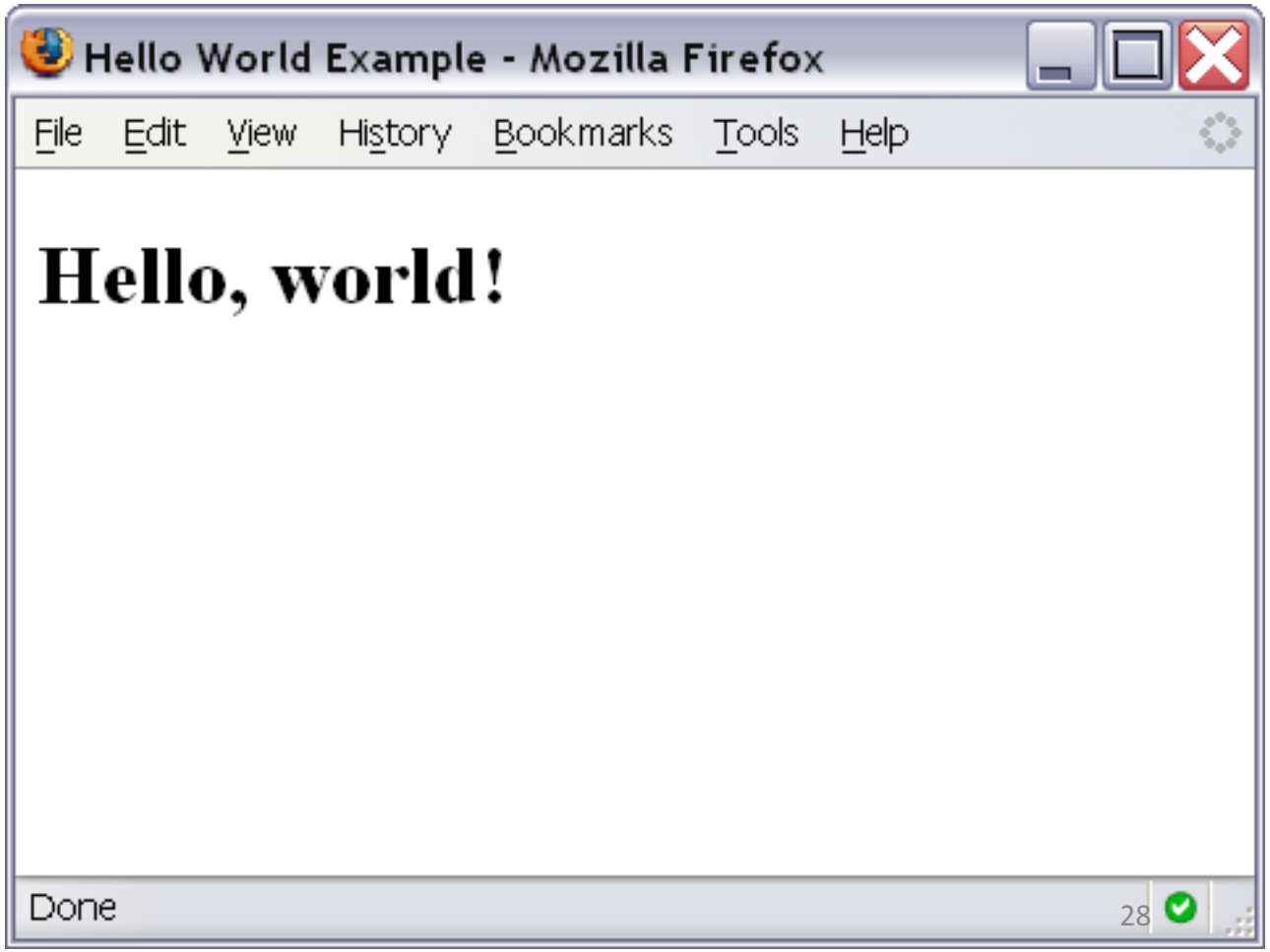
ii. Implicit embedding:
JavaScript code residing in separate .js file and that file included in the desired XHTML document- may be on head or body.
e.g. XHTML document:

```
<html>
 <head>
<script type=”text/javascript” src=”jquery.js”></script>
 </head>
 <body>
……XHTML code…..
 </body>
</html>
```

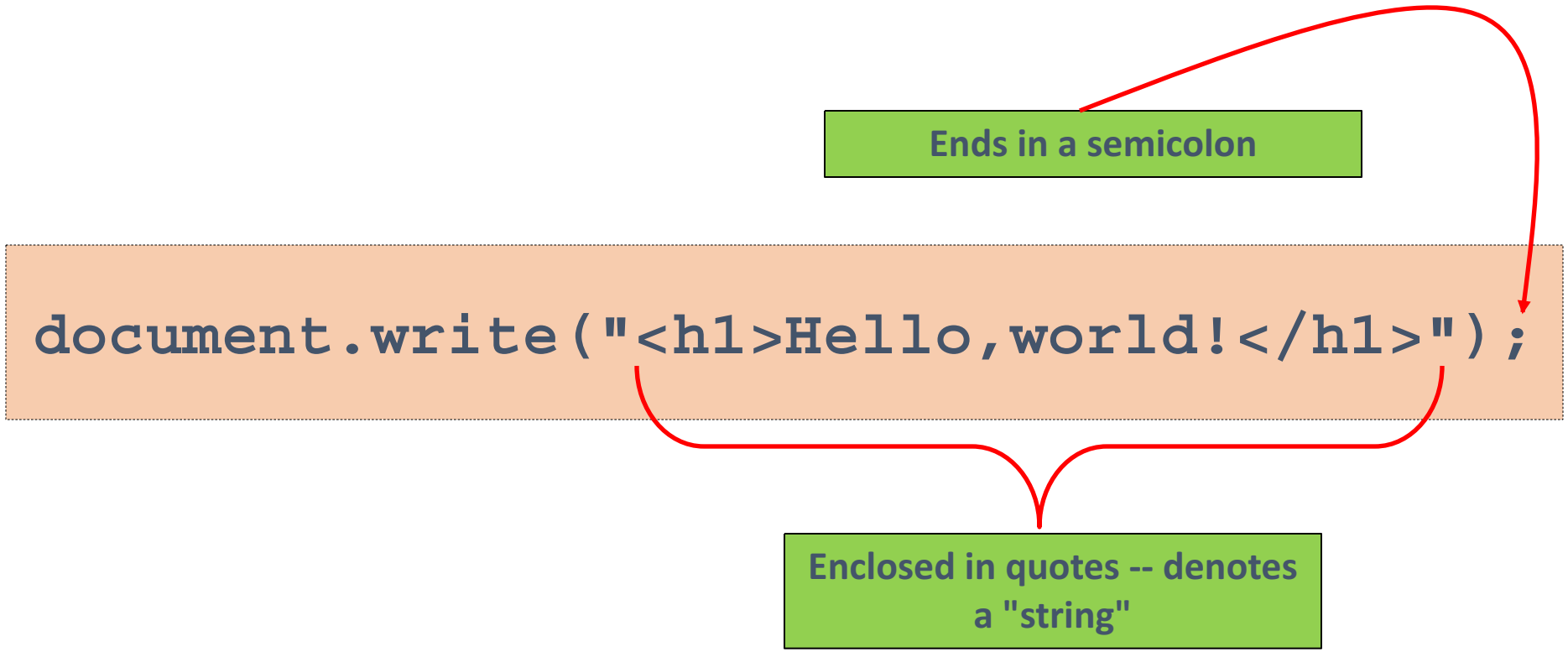
1st JS example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World Example</title>
  </head>
  <body>
     <script type="text/javascript">
       <!--
         document.write("<h1>Hello,
world!</h1>");
       //-->
     </script>
  </body>
</html>
```

```
<script>
        document.write("simanta")
    </script>
```



document.write()



Types of JavaScript Comments

- There are two types of comments are in JavaScript
- Single-line Comment
- Multi-line Comment
- **Single-line Comment**

<script>

// It is single line comment document.write("Hello Javascript");

</script>

Multi-line Comment

- **<script>**
- /* It is multi line comment. It will not be displayed */
- document.write("Javascript multiline comment");
- **</script>**

JavaScript Code:

- JavaScript code (or just JavaScript) is a sequence of JavaScript statements. Each statement is executed by the browser in the sequence they are written. Following example will write a heading and two paragraphs to a web page: Example
- <script type="text/javascript">
- document.write("<h1>This is a heading</h1>");
- document.write(" This is a paragraph. ");
- document.write(" This is another paragraph. ");
- </script>

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, are in JavaScript functions:
- Example
- function myFunction() {
- document.write("hello");
- document.write("Dev");
- }


```
<body>
    <div>
        <p id="A">This is paragraph</p>
    </div>
    <button type="button" onclick=AB();>Click
Me!</button>
    <script src="a.js">

    </script>
```

```
function AB(){
document.getElementById("A").in
nerHTML = "Hello JavaScript!";
}
```

```
OR
<button type="button"
onclick='document.getElementById("A").innerHTML
="Hello Javascript"'>
Click here</button>
```

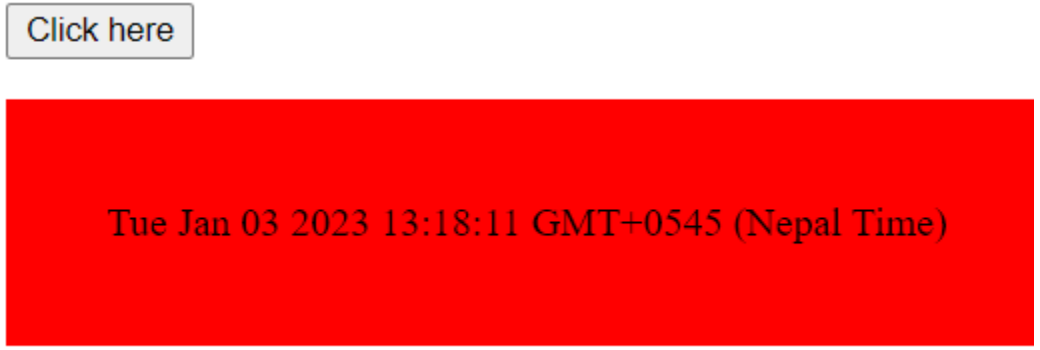

```
<body>
    <div>
        <p id="A">This is paragraph</p>
    </div>
    <button type="button" onclick=AB();>Click
Me!</button>
    <script src="a.js">

    </script>
```

```
function AB(){
document.getElementById("A").style.fontSize =
"40px";
}
```

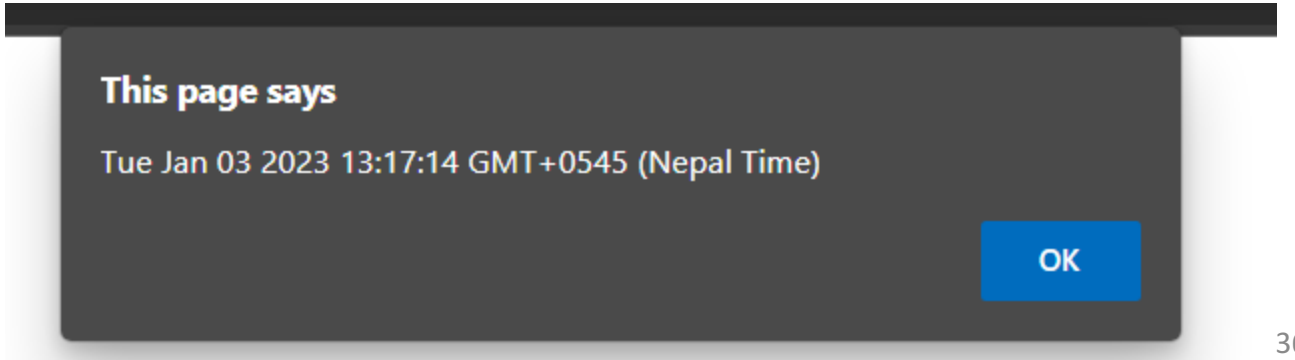
```
OR
 <button type="button"
onclick='document.getElementById("A").style.fontSize="35px"'>
Click here</button>
```

```
function AB(){
document.getElementById("A").innerHTML = Date();
}
```

```
 <body>

    <button type="button" onclick=AB();>
 Click here</button>
 <p id="A"></p>
    <script src="a.js">
     </script>
 </body>
```

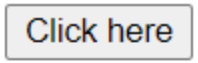



```
var my_time=new Date();
alert (my_time);
```



```
p#A{
            background-color: red;
            padding: 40px;
        }
    </style>

</head>
<body>

   <button type="button" onclick=AB();>
Click here</button>
<p id="A">This is paragraph</p>
   <script src="a.js">
    </script>
```

```
function AB(){
document.getElementById("A").style.display
="none";
}
```



```
 <style>
        p#A{
            background-color: red;
            padding: 40px;
            display: none;
        }
    </style>
</head>
<body>

   <button type="button" onclick=AB();>
Click here</button>
<p id="A" >This is paragraph</p>
   <script src="a.js">
    </script>
</body>
```

```
function AB(){
document.getElementById("A").style.display
="block";
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device
-width, initial-scale=1.0">
    <title>Document</title>
    <script type="text/javascript"
src="java.js"></script>
    <style>
    </style>
</head>
<body>
    <img src="car.jpg" alt="" height="100px"
id="AB">
<button onclick="A()">Yellow</button>
<button onclick="B()">red</button>
<button onclick="C()">Orange</button>
<button onclick="D()">green</button>
</body>
</html>
```

```
function A()
{

 document.getElementById("AB").sr
c="car1.png";
}

function B()
{

 document.getElementById("AB").sr
c="car2.png";
}
function C()
{

 document.getElementById("AB").sr
c="car3.png";
}

function D()
{

 document.getElementById("AB").sr
```

Yellow red Orange green

Yellow red Orange green

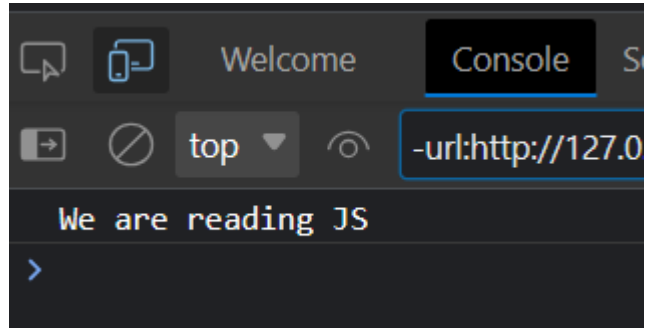
Yellow red Orange green

Screen o/p and keyboard input.

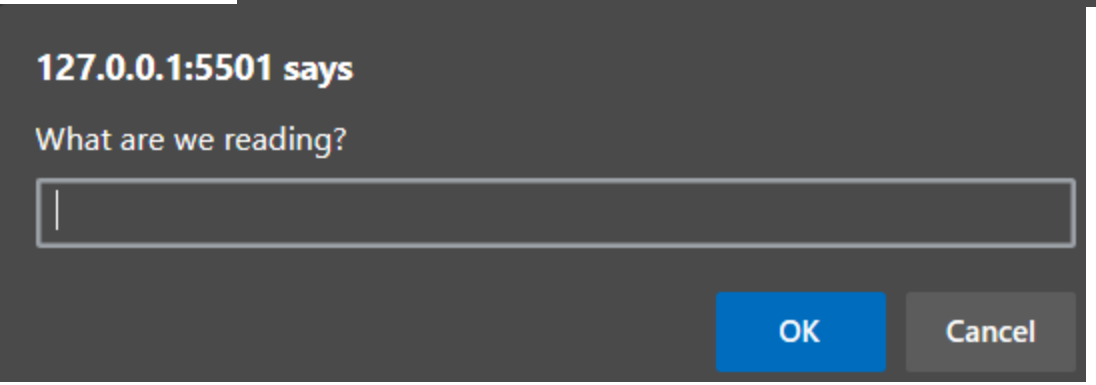
JavaScript Display Possibilities
JavaScript can "display" data in different ways:

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

window.alert().



console.log().



Prompt()

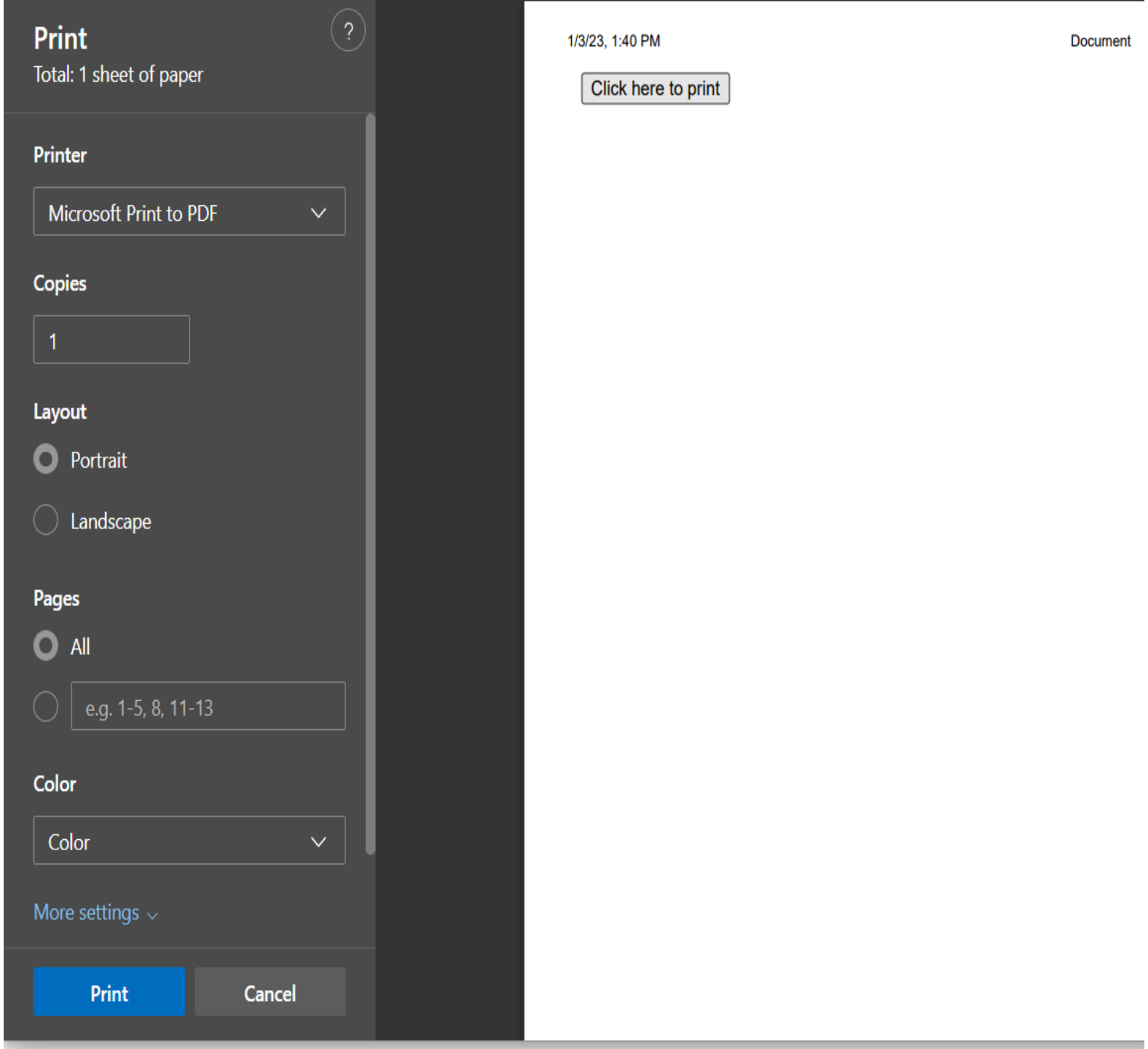
```
<script>
    prompt('What are we
reading?','js');
</script>
```

```
<script>
    document.write('We r reading JS');
</script>
```

We r reading JS

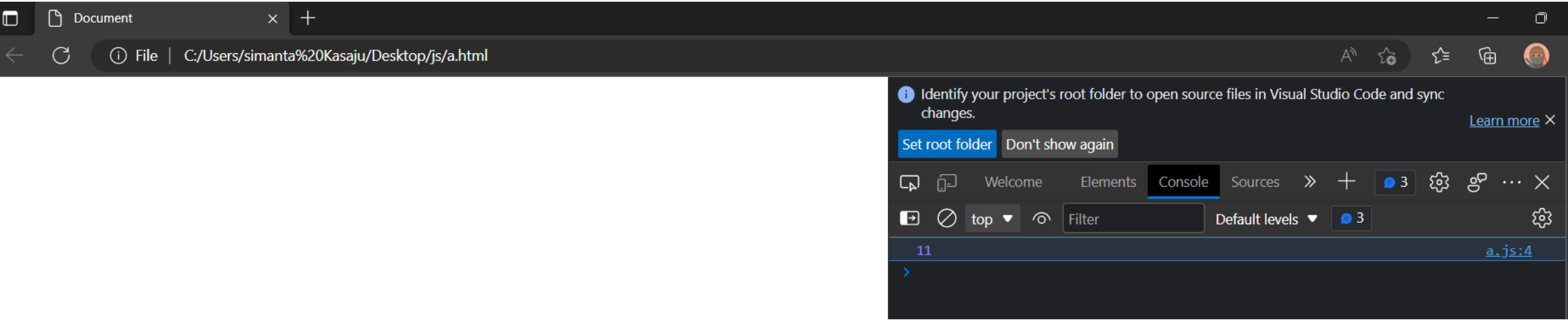
Confirm()

```
 <button
onclick='window.pri
nt();'>Click here
to print</button>
```



```
var a=5;
var b=6;
var c=a+b;
console.log(c);
```

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



JavaScript Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.

```
<script type="text/javascript">
<!--
var money;
var name;
//-->
</script>
```

You can also declare multiple variables with the same **var** keyword as follows:

```
<script type="text/javascript">
<!--
var money, name;
//-->
</script>
```

Storing a value in a variable is called **variable initialization**. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named **money** and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

```
<script type="text/javascript">
<!--
var name = "Ali";
var money;
money = 2000.50;
//-->
</script>
```

Note: Use the **var** keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

JavaScript is **untyped** language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- **Global Variables:** A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

```
<script type="text/javascript">
<!--
var myVar = "global"; // Declare a global variable
function checkscope( ) {
   var myVar = "local";  // Declare a local variable
   document.write(myVar);
}
//-->
</script>
```

It will produce the following result:

```
Local
```

JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, **break** or **boolean** variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, **123test** is an invalid variable name but **_123test** is a valid one.
- JavaScript variable names are case-sensitive. For example, **Name** and **name** are two different variables.

JavaScript Reserved Words

A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

abstract	else	Instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

JavaScript Variables

4 Ways to Declare a JavaScript Variable:
Using var
Using let
Using const
Using nothing

What are Variables?
Variables are containers for storing data (storing data values).

In this example, x, y, and z, are variables, declared with the var keyword:

```
<script>
var x = 5;
var y = 6;
var z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>
```

Why JS called as Dynamically typed because we can change the datatype during run time. like we can store number in string

Var
Variables defined with var can be Redeclared and updated.

Variables defined with let must be Declared before use.

Variables defined with var have not Block Scope it is global scoped

```
<script>
let x = 5;
let y = 6;
let z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>
```

Let
Variables defined with let cannot be Redeclared.

Variables defined with let must be Declared before use.

Variables defined with let have Block Scope.

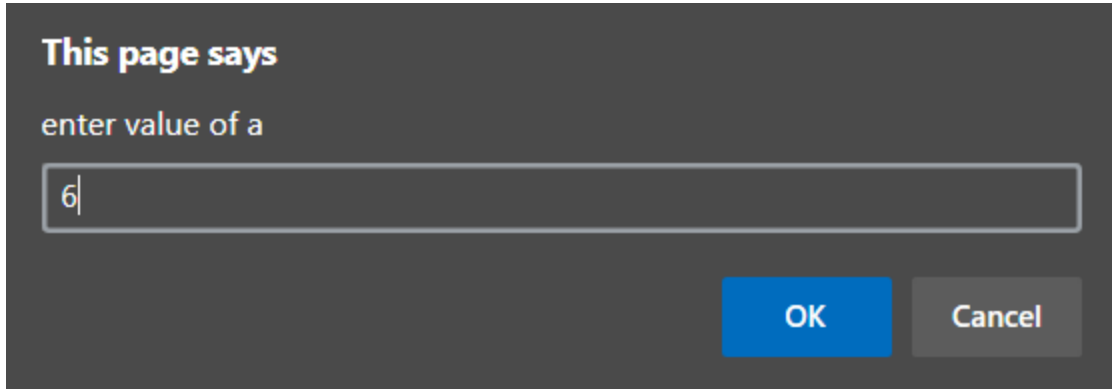
const
Variables defined with const cannot be Redeclared or updated

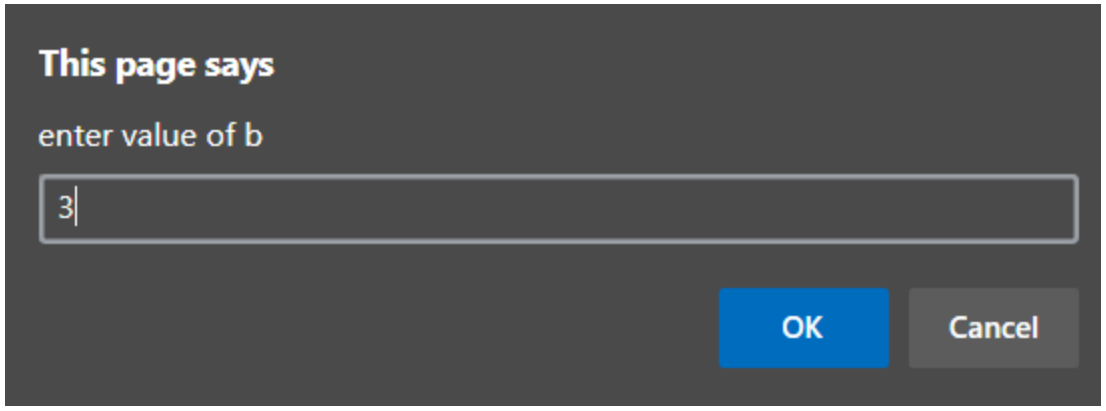
Variables defined with const cannot be Reassigned.

Variables defined with const have Block Scope.

```
const PI = 3.14159265359;
```

```
var a=Number(prompt("enter value of a"));
var b=Number(prompt("enter value of b"));
document.write(a + b);
document.write(a - b);
document.write(a * b);
document.write(a / b);
```





9

3

18

2

```
var bordersize;
bordersize =prompt("select a border table
size\n"+"1");
switch (bordersize)
{
    case "0": document.write("<table>");
    break;
    case "1": document.write("<table
border='1'>");
    break;
    case "4": document.write("<table
border='4'>");
    break;
    case "8": document.write("<table
border='8'>");
    break;
    default:document.write("error");
}
document.write("<caption>Personal Information
</caption>");
document.write("<tr>","<th></th>","<th>
Firstname </th>"
,
"<th>Lastname</th>",
"</tr>",
"<tr>",
    "<td>1</td>",
    "<td>Simanta</td>",
    "<td>Kasaju</td>",
"</tr>");
```

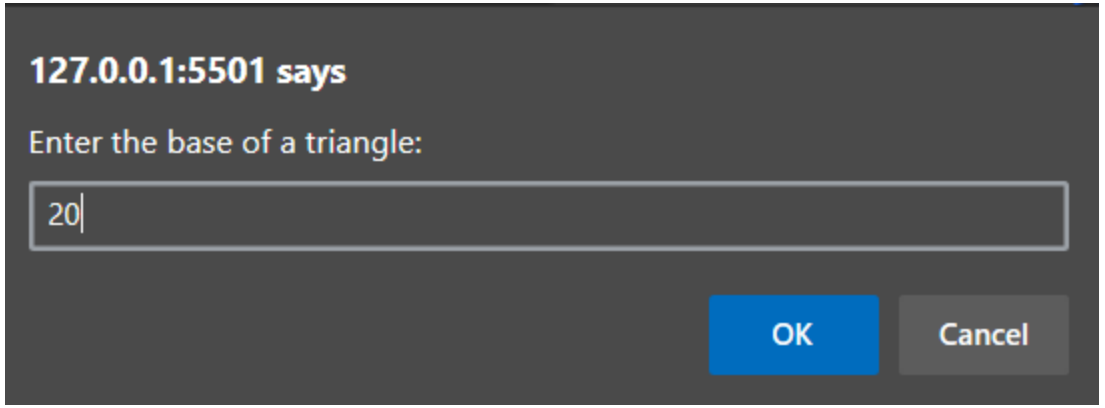
Personal Information

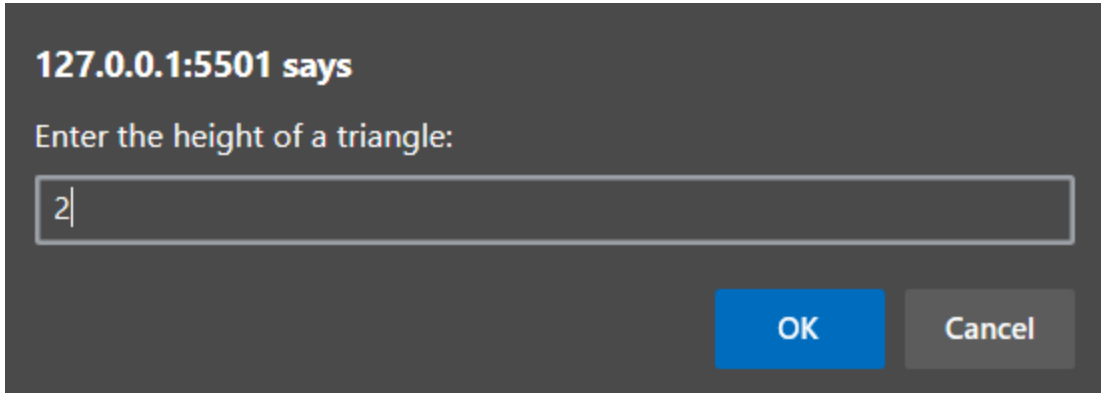
	Firstname	Lastname
1	Simanta	Kasaju

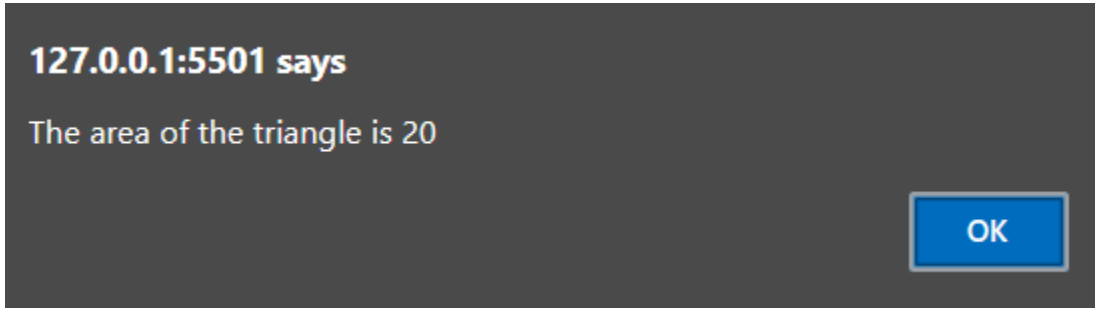
Quadratic Equation root

```
// Get the coefficients of the equation from the user
var a = prompt("What is the value of 'a'? \n", "");
var b = prompt("What is the value of 'b'? \n", "");
var c = prompt("What is the value of 'c'? \n", "");

// Compute the square root and denominator of the result
var root_part = Math.sqrt(b * b - 4.0 * a * c);
var denom = 2.0 * a;

// Compute and display the two roots
var root1 = (-b + root_part) / denom;
var root2 = (-b - root_part) / denom;
document.write("The first root is: ", root1, "<br />");
document.write("The second root is: ", root2, "<br />");
```

```
  const baseValue = prompt('Enter the base of a
triangle: ');
const heightValue = prompt('Enter the height of
a triangle: ');

// calculate the area
const areaValue = (baseValue * heightValue) /
2;

alert(`The area of the triangle is
${areaValue}`
);
```







JavaScript Datatypes

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types:

- **Numbers**, e.g., 123, 120.50 etc.
- **Strings** of text, e.g. "This text string" etc.
- **Boolean**, e.g. true or false.

JavaScript also defines two trivial data types, **null** and **undefined**, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as **object**. We will cover objects in detail in a separate chapter.

Note: Java does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format defined by the IEEE 754 standard.

```
<script>
 a=5;
 b='simanta';
 c=true;
 d=false;
 e=null;
 f=undefined;
 const h = {
  'simanta': 12,
  'kasaju':'hello',
  'hello' : 345
 }
 g=BigInt('1234')
 document.write(a,b,c,d,e,f)
 document.write('<br>');
 document.write(typeof h);
 document.write(h['kasaju']);

</script>
```

What is an Operator?

Let us take a simple expression **4 + 5 is equal to 9**. Here 4 and 5 are called **operands** and '+' is called the **operator**. JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Let's have a look at all the operators one by one.

Arithmetic Operators

JavaScript supports the following arithmetic operators:

Assume variable A holds 10 and variable B holds 20, then:

S. No.	Operator and Description
1	**+ (Addition)** Adds two operands **Ex:** A + B will give 30
2	**- (Subtraction)** Subtracts the second operand from the first **Ex:** A - B will give -10
3	*** (Multiplication)** Multiply both operands **Ex:** A * B will give 200
4	**/ (Division)** Divide the numerator by the denominator **Ex:** B / A will give 2
5	**% (Modulus)** Outputs the remainder of an integer division **Ex:** B % A will give 0
6	**++ (Increment)** Increases an integer value by one **Ex:** A++ will give 11
7	**-- (Decrement)** Decreases an integer value by one **Ex:** A-- will give 9

Note: Addition operator (**+**) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

Example

The following code shows how to use arithmetic operators in JavaScript.

```
<html>
<body>

<script type="text/javascript">
<!--
var a = 33;
var b = 10;
var c = "Test";
var linebreak = "<br />";

document.write("a + b = ");
result = a + b;
document.write(result);
document.write(linebreak);

document.write("a - b = ");
result = a - b;
document.write(result);
document.write(linebreak);

document.write("a / b = ");
result = a / b;
document.write(result);
document.write(linebreak);

document.write("a % b = ");
result = a % b;
document.write(result);
document.write(linebreak);

document.write("a + b + c = ");
result = a + b + c;
document.write(result);
document.write(linebreak);

a = a++;
document.write("a++ = ");
result = a++;
document.write(result);
document.write(linebreak);

b = b--;
document.write("b-- = ");
result = b--;
document.write(result);
document.write(linebreak);
```

```
//-->
</script>


<p>Set the variables to different values and then try...</p>
</body>
</html>
```

Output

```
a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
a++ = 33
b-- = 10

Set the variables to different values and then try...
```

```
<script>
let x = 5 + 5;
let y = "5" + 5;
let z = "Hello" + 5;
document.getElementById("demo").innerHTML =
x + "<br>" + y + "<br>" + z;
</script>
```

JavaScript Operators

Adding a number and a string, returns a string.

10
55
Hello5

Comparison Operators

JavaScript supports the following comparison operators:

Assume variable A holds 10 and variable B holds 20, then:

S.No	Operator and Description
1	**== (Equal)** Checks if the value of two operands are equal or not, if yes, then the condition becomes true. **Ex:** (A == B) is not true.
2	**!= (Not Equal)** Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true. **Ex:** (A != B) is true.
3	**> (Greater than)** Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A > B) is not true.
4	**< (Less than)** Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A < B) is true.
5	**>= (Greater than or Equal to)** Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true. **Ex:** (A >= B) is not true.
6	**<= (Less than or Equal to)** Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true. **Ex:** (A <= B) is true.

===	equal value and equal type(Identical)
!==	not equal value or not equal type (non Identical)

== is Abstract equality just values are compared
=== is strict equality values and datatypes are compared.

```
let a = 10;
let b = '10';
let c = 10;

console.log(a===b);
//output: false;

console.log(a===c);
//output: true;
```

```
let a = 10;
let b = 10;
let c = -10;

console.log(a==b);
//output: true

console.log(a==c);
//output: false
```

Example

The following code shows how to use comparison operators in JavaScript

```
<html>
<body>

<script type="text/javascript">
<!--
var a = 10;
var b = 20;
var linebreak = "<br />";

document.write("(a == b) => ");
result = (a == b);
document.write(result);
document.write(linebreak);

document.write("(a < b) => ");
result = (a < b);
document.write(result);
document.write(linebreak);

document.write("(a > b) => ");
result = (a > b);
document.write(result);
document.write(linebreak);

document.write("(a != b) => ");
result = (a != b);
document.write(result);
document.write(linebreak);

document.write("(a >= b) => ");
result = (a >= b);
document.write(result);
document.write(linebreak);
```

```
document.write("(a <= b) => ");

result = (a <= b);

document.write(result);

document.write(linebreak);


//-->

</script>


<p>Set the variables to different values and different operators and
then try...</p>

</body>

</html>
```

Output

```
(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
(a <= b) => true


Set the variables to different values and different operators and then
try...
```

Logical Operators

JavaScript supports the following logical operators:

Assume variable A holds 10 and variable B holds 20, then:

S.No	Operator and Description
1	**&& (Logical AND)** If both the operands are non-zero, then the condition becomes true. **Ex:** (A && B) is true.
2	**|| (Logical OR)** If any of the two operands are non-zero, then the condition becomes true. **Ex:** (A || B) is true.
3	**! (Logical NOT)** Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. **Ex:** ! (A && B) is false.

Try the following code to learn how to implement Logical Operators in JavaScript.

```
<html>
<body>

<script type="text/javascript">
<!--
var a = true;
var b = false;
var linebreak = "<br />";

document.write("(a &&  b) => ");
result = (a && b);
document.write(result);
document.write(linebreak);

document.write("(a || b) => ");
result = (a || b);
document.write(result);
document.write(linebreak);

document.write("!(a && b) => ");
result = (!(a && b));
document.write(result);
document.write(linebreak);

//-->
</script>

<p>Set the variables to different values and different operators and
then try...</p>
```

```
</body>

</html>
```

Output

```
(a && b) => false
(a || b) => true
!(a && b) => true


Set the variables to different values and different operators and then
try...
```

Bitwise Operators

JavaScript supports the following bitwise operators:

Assume variable A holds 2 and variable B holds 3, then:

S.No	Operator and Description
1	**& (Bitwise AND)** It performs a Boolean AND operation on each bit of its integer arguments. **Ex:** (A & B) is 2.
2	**| (BitWise OR)** It performs a Boolean OR operation on each bit of its integer arguments. **Ex:** (A | B) is 3.
3	**^ (Bitwise XOR)** It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both. **Ex:** (A ^ B) is 1.
4	**~ (Bitwise Not)** It is a unary operator and operates by reversing all the bits in the operand. **Ex:** (~B) is -4.
5	**<< (Left Shift)** It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on. **Ex:** (A << 1) is 4.
6	**>> (Right Shift)** Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand. **Ex:** (A >> 1) is 1.
7	**>>> (Right shift with Zero)** This operator is just like the >> operator, except that the bits shifted in on the left are always zero. **Ex:** (A >>> 1) is 1.

Example

Try the following code to implement Bitwise operator in JavaScript.

```
<html>
<body>

<script type="text/javascript">
<!--
var a = 2;  // Bit presentation 10
var b = 3;  // Bit presentation 11
var linebreak = "<br />";

document.write("(a &  b) => ");
result = (a & b);
document.write(result);
document.write(linebreak);

document.write("(~b) => ");
result = (~b);
document.write(result);
document.write(linebreak);

document.write("(a << b) => ");
result = (a << b);
document.write(result);
document.write(linebreak);

document.write("(a >> b) => ");
result = (a >> b);
document.write(result);
document.write(linebreak);

//-->
</script>

<p>Set the variables to different values and different operators and
then try...</p>
</body>
</html>
```

```
(a & b) => 2
(a | b) => 3
(a ^ b) => 1
(~b) => -4
(a << b) => 16
(a >> b) => 0


Set the variables to different values and different operators and then
try...
```

Assignment Operators

JavaScript supports the following assignment operators:

S.No	Operator and Description
1	**= (Simple Assignment)** Assigns values from the right side operand to the left side operand **Ex:** C = A + B will assign the value of A + B into C
2	**+= (Add and Assignment)** It adds the right operand to the left operand and assigns the result to the left operand. **Ex:** C += A is equivalent to C = C + A
3	**-= (Subtract and Assignment)** It subtracts the right operand from the left operand and assigns the result to the left operand. **Ex:** C -= A is equivalent to C = C - A
4	***= (Multiply and Assignment)** It multiplies the right operand with the left operand and assigns the result to the left operand. **Ex:** C *= A is equivalent to C = C * A
5	**/= (Divide and Assignment)** It divides the left operand with the right operand and assigns the result to the left operand.
	Ex: C /= A is equivalent to C = C / A
6	**%= (Modules and Assignment)** It takes modulus using two operands and assigns the result to the left operand. **Ex:** C %= A is equivalent to C = C % A

Note: Same logic applies to Bitwise operators, so they will become <<=, >>=, >>=, &=, |= and ^=.

Example

Try the following code to implement assignment operator in JavaScript.

```
<html>
<body>

<script type="text/javascript">
<!--
var a = 33;
var b = 10;
var linebreak = "<br />";

document.write("Value of a => (a = b) => ");
result = (a = b);
document.write(result);
document.write(linebreak);

document.write("Value of a => (a += b) => ");
result = (a += b);
document.write(result);
document.write(linebreak);

document.write("Value of a => (a -= b) => ");
result = (a -= b);
document.write(result);
document.write(linebreak);

document.write("Value of a => (a *=  b) => ");
result = (a *= b);
document.write(result);
document.write(linebreak);

document.write("Value of a => (a /= b) => ");
result = (a /= b);
document.write(result);
document.write(linebreak);

document.write("Value of a => (a %= b) => ");
result = (a %= b);
document.write(result);
document.write(linebreak);

//-->
</script>

<p>Set the variables to different values and different operators and
then try...</p>
</body>
</html>
```

Miscellaneous Operators

We will discuss two operators here that are quite useful in JavaScript: the **conditional operator (? :)** and the **typeof operator**.

Conditional Operator (? :)

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

S.No	Operator and Description
1	**? : (Conditional)** If Condition is true? Then value X : Otherwise value Y

```
result = (a < b) ? 100 : 200;
document.write(result);
document.write(linebreak);

//-->
</script>

<p>Set the variables to different values and different operators and
then try...</p>
</body>
</html>
```

Example

Try the following code to understand how the Conditional Operator works in JavaScript.

```
<html>
<body>

<script type="text/javascript">
<!--
var a = 10;
var b = 20;
var linebreak = "<br />";

document.write ("((a > b) ? 100 : 200) => ");
result = (a > b) ? 100 : 200;
document.write(result);
document.write(linebreak);

document.write ("((a < b) ? 100 : 200) => ");
```

Output

```
((a > b) ? 100 : 200) => 200
((a < b) ? 100 : 200) => 100

Set the variables to different values and different operators and then
try...
```

typeof Operator

The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the **typeof** Operator.

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

Example

The following code shows how to implement **typeof** operator.

```
<html>
<body>

<script type="text/javascript">
<!--
var a = 10;
var b = "String";
var linebreak = "<br />";

result = (typeof b == "string" ? "B is String" : "B is Numeric");
document.write("Result => ");
document.write(result);
document.write(linebreak);

result = (typeof a == "string" ? "A is String" : "A is Numeric");
document.write("Result => ");
document.write(result);
document.write(linebreak);

//-->
</script>
```

Output

```
Result => B is String
Result => A is Numeric
```

```
var a=10;
alert(typeof a);
```

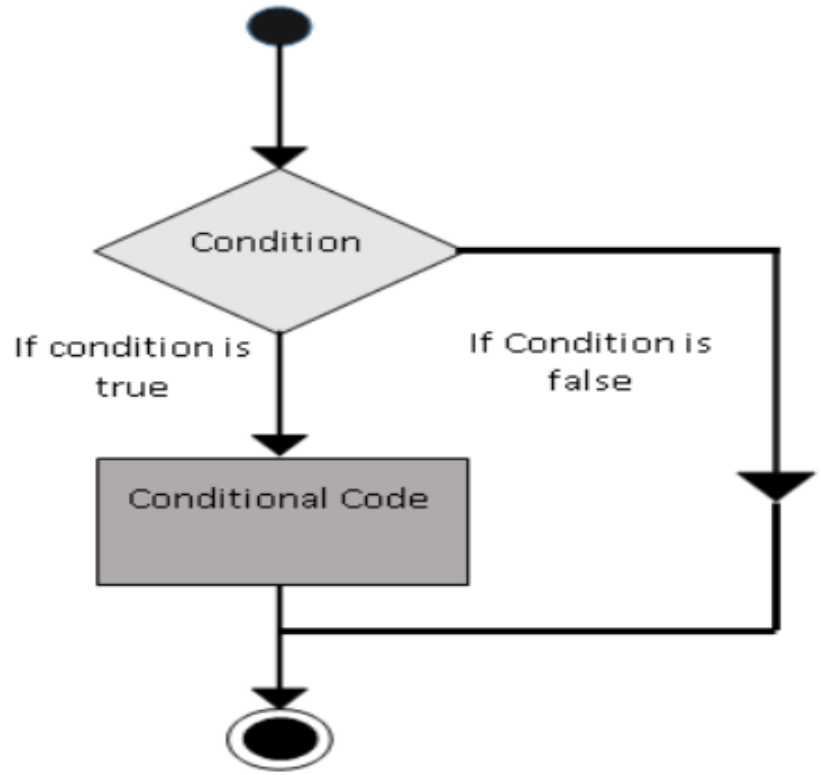

```
let a="simanta kasaju";
alert(typeof a);
```

Conditional Statement

While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform the right actions. JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain the if..else statement.

Flow Chart of if-else

The following flow chart shows how the if-else statement works.



JavaScript supports the following forms of **if..else** statement:

- if statement
- if...else statement
- if...else if... statement

if Statement

The '**if**' statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax

The syntax for a basic if statement is as follows:

```
if (expression){
   Statement(s) to be executed if expression is true
}
```

Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

```
<script type="text/javascript">
<!--
var age = 20;
if( age > 18 ){
   document.write("<b>Qualifies for driving</b>");
}
//-->
</script>
```

Output

```
Qualifies for driving
```

if...else Statement

The '**if...else**' statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax

The syntax of an **if-else** statement is as follows:

```
if (expression){
   Statement(s) to be executed if expression is true
}else{
   Statement(s) to be executed if expression is false
}
```

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the 'if' block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example

Try the following code to learn how to implement an if-else statement in JavaScript.

```
<html>
<body>

<script type="text/javascript">
<!--
var age = 15;

if( age > 18 ){
   document.write("<b>Qualifies for driving</b>");
}else{
   document.write("<b>Does not qualify for driving</b>");
}
```

Output

```
Does not qualify for driving
```

if...else if... Statement

The '**if...else if...**' statement is an advanced form of **if...else** that allows JavaScript to make a correct decision out of several conditions.

Syntax

The syntax of an if-else-if statement is as follows:

```
if (expression 1){
   Statement(s) to be executed if expression 1 is true
}else if (expression 2){
   Statement(s) to be executed if expression 2 is true
}else if (expression 3){
   Statement(s) to be executed if expression 3 is true
}else{
   Statement(s) to be executed if no expression is true
}
```

There is nothing special about this code. It is just a series of **if** statements, where each **if** is a part of the **else** clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the **else** block is executed.

```
<script type="text/javascript">
<!--
var book = "maths";
if( book == "history" ){
   document.write("<b>History Book</b>");
}else if( book == "maths" ){
   document.write("<b>Maths Book</b>");
}else if( book == "economics" ){
   document.write("<b>Economics Book</b>");
}else{
  document.write("<b>Unknown Book</b>");
}
//-->
</script>
```

Output

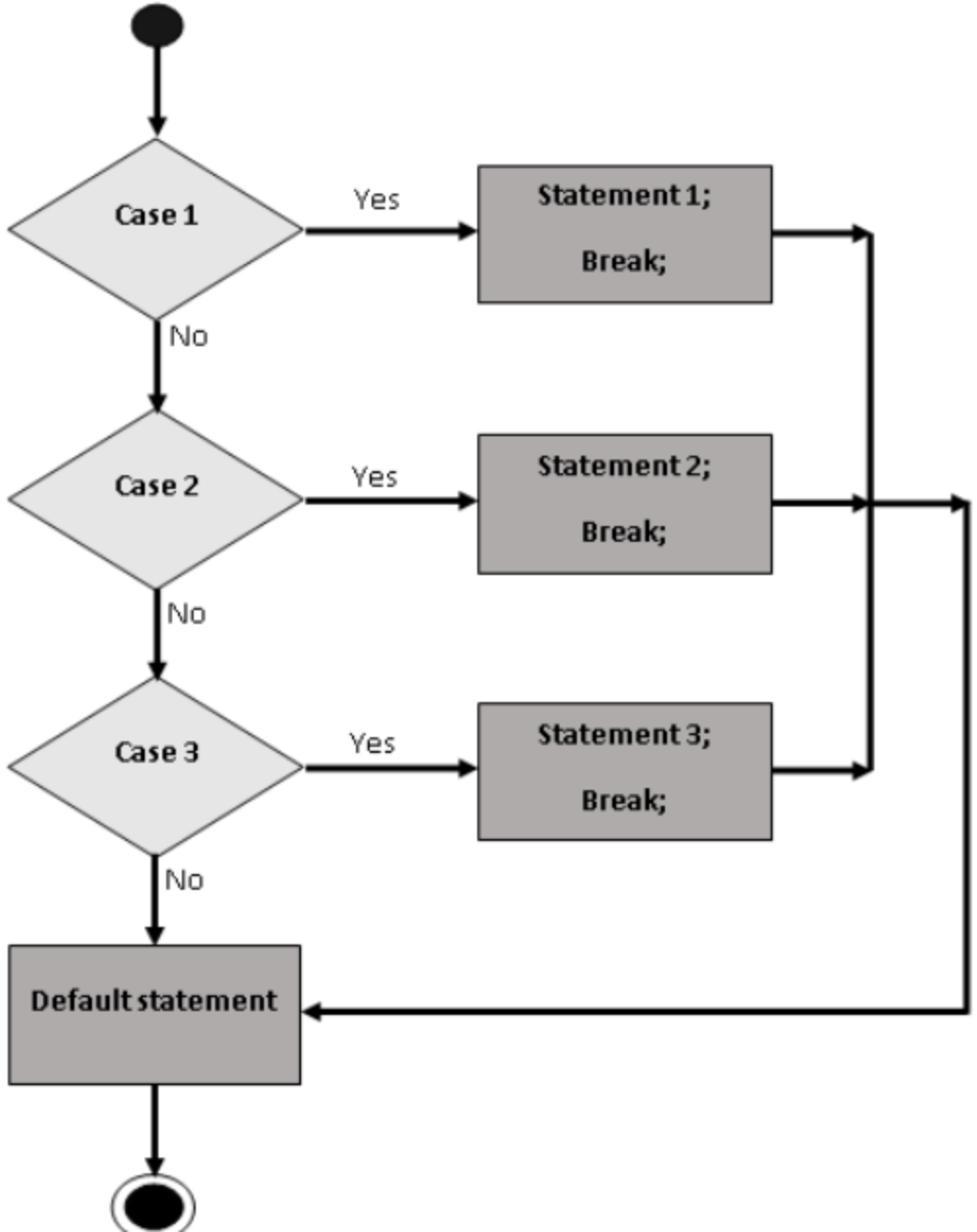
```
Maths Book
```

Switch case

You can use multiple if...else…if statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all branches depend on a single variable's value. Starting with JavaScript 1.2, you can use a switch statement that handles exactly this situation, and it does so more efficiently than repeated if...else if statements.

Flow Chart

The following flow chart explains a switch-case statement works.



Syntax

The objective of a **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each **case** against the value of the expression until a match is found. If nothing matches, a **default** condition will be used.

```
switch (expression)
{
  case condition 1: statement(s)
                    break;
  case condition 2: statement(s)
                    break;
   ...
  case condition n: statement(s)
                    break;
  default: statement(s)
}
```

The **break** statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

```
<script type="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br />");
switch (grade)
{
  case 'A': document.write("Good job<br />");
            break;
  case 'B': document.write("Pretty good<br />");
            break;
  case 'C': document.write("Passed<br />");
            break;
  case 'D': document.write("Not so good<br />");
            break;
  case 'F': document.write("Failed<br />");
            break;
  default:  document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>
```

Output

```
Entering switch block
Good job
Exiting switch block
```

Break statements play a major role in switch-case statements. Try the following code that uses switch-case statement without any break statement.

```
<script type="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br />");
 switch (grade)
 {
   case 'A': document.write("Good job<br />");
   case 'B': document.write("Pretty good<br />");
   case 'C': document.write("Passed<br />");
   case 'D': document.write("Not so good<br />");
   case 'F': document.write("Failed<br />");
   default:  document.write("Unknown grade<br />")
 }
 document.write("Exiting switch block");
 //-->
 </script>
```

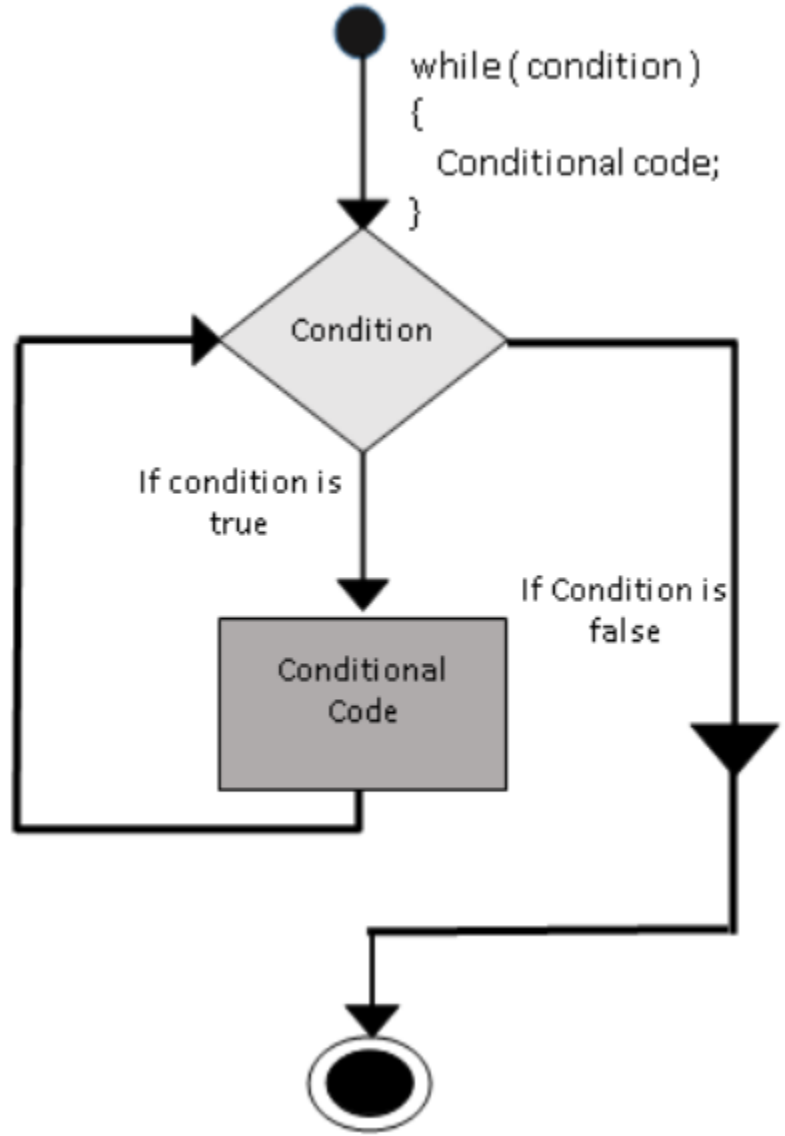
Output

```
Entering switch block
Good job
Pretty good
Passed
Not so good
Failed
Unknown grade
Exiting switch block
```

Loop

While writing a program, you may encounter a situation where you must perform an action repeatedly. In such situations, you would need to write loop statements to reduce the number of lines. JavaScript supports all the necessary loops to ease down the pressure of programming.

The **while Loop** The most basic loop in JavaScript is the while loop which would be discussed in this chapter. The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates. Flow Chart The flow chart of the while loop looks as follows:



Syntax

The syntax of **while loop** in JavaScript is as follows:

```
while (expression){
   Statement(s) to be executed if expression is true
}
```

```
<script type="text/javascript">
<!--
var count = 0;
document.write("Starting Loop ");
while (count < 10){
  document.write("Current Count : " + count + "<br />");
  count++;
}
document.write("Loop stopped!");
//-->
</script>
```

Output

```
Starting Loop Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
```

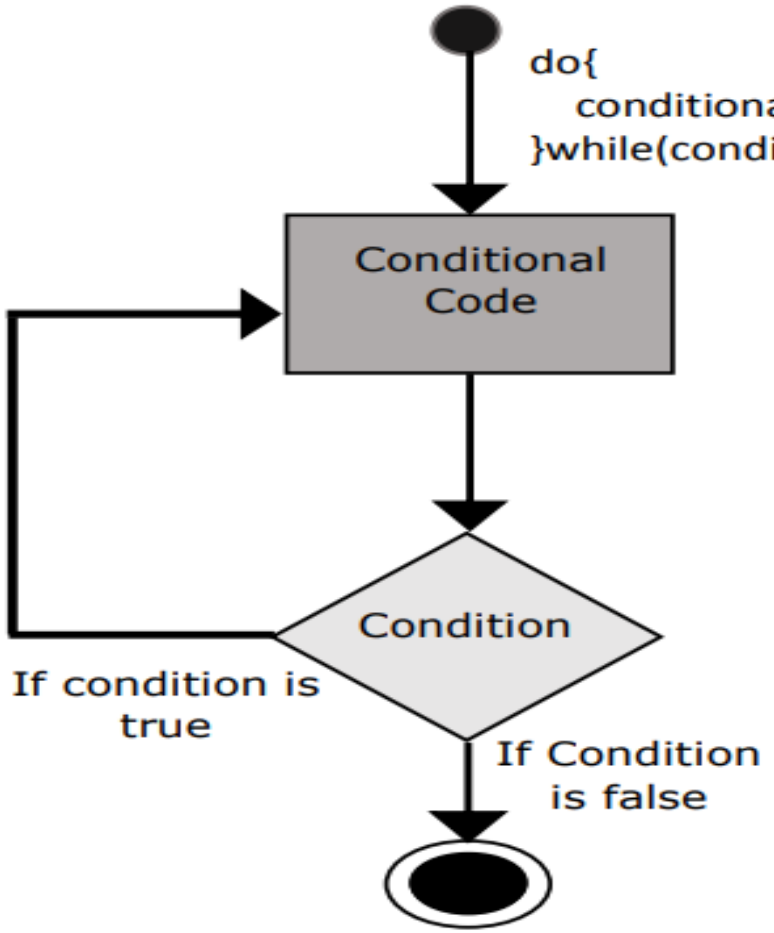
```
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

The do...while Loop

The **do...while** loop is similar to the **while** loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is **false**.

Flow Chart

The flow chart of a **do-while** loop would be as follows:



Syntax

The syntax for **do-while** loop in JavaScript is as follows:

```
do{
   Statement(s) to be executed;
} while (expression);
```

Note: Don't miss the semicolon used at the end of the **do...while** loop.

```
<script type="text/javascript">
<!--
var count = 0;
document.write("Starting Loop" + "<br />");
do{
  document.write("Current Count : " + count + "<br />");
  count++;
}while (count < 5);
document.write ("Loop stopped!");
//-->
</script>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop Stopped!
```

The for Loop

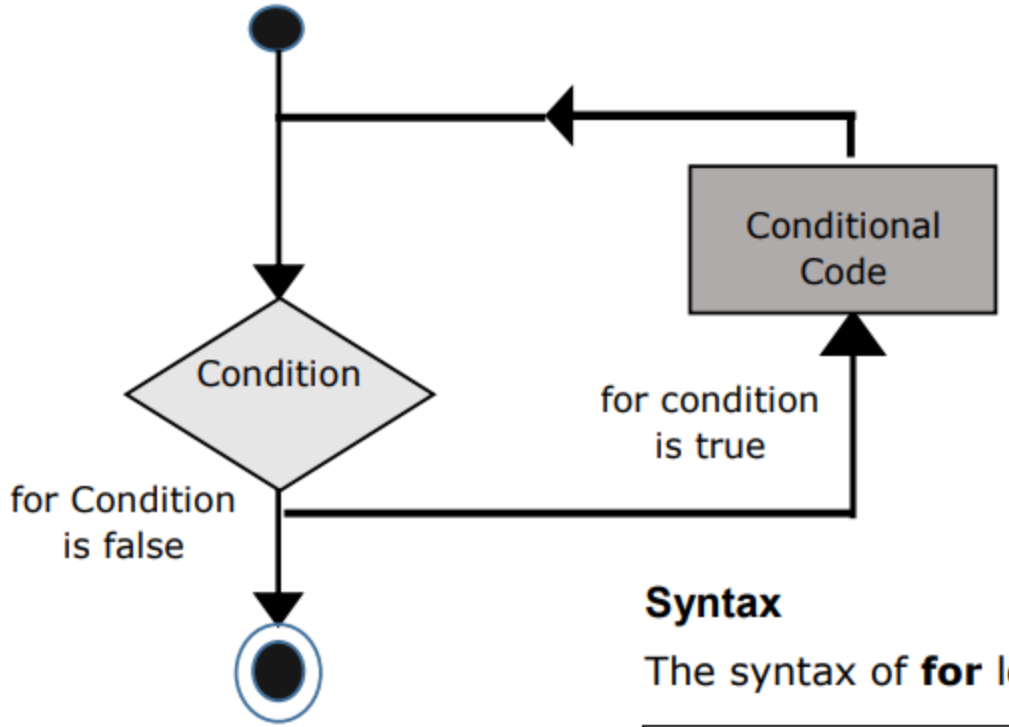
The '**for**' loop is the most compact form of looping. It includes the following three important parts:

- The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The **iteration statement** where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons.

Flow Chart

The flow chart of a **for** loop in JavaScript would be as follows:



Syntax

The syntax of **for** loop is JavaScript is as follows:

```
for (initialization; test condition; iteration statement){
     Statement(s) to be executed if test condition is true
}
```

```
<script type="text/javascript">
<!--
var count;
document.write("Starting Loop" + "<br />");
for(count = 0; count < 10; count++){
  document.write("Current Count : " + count );
  document.write("<br />");
}
document.write("Loop stopped!");
//-->
</script>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
```

```
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

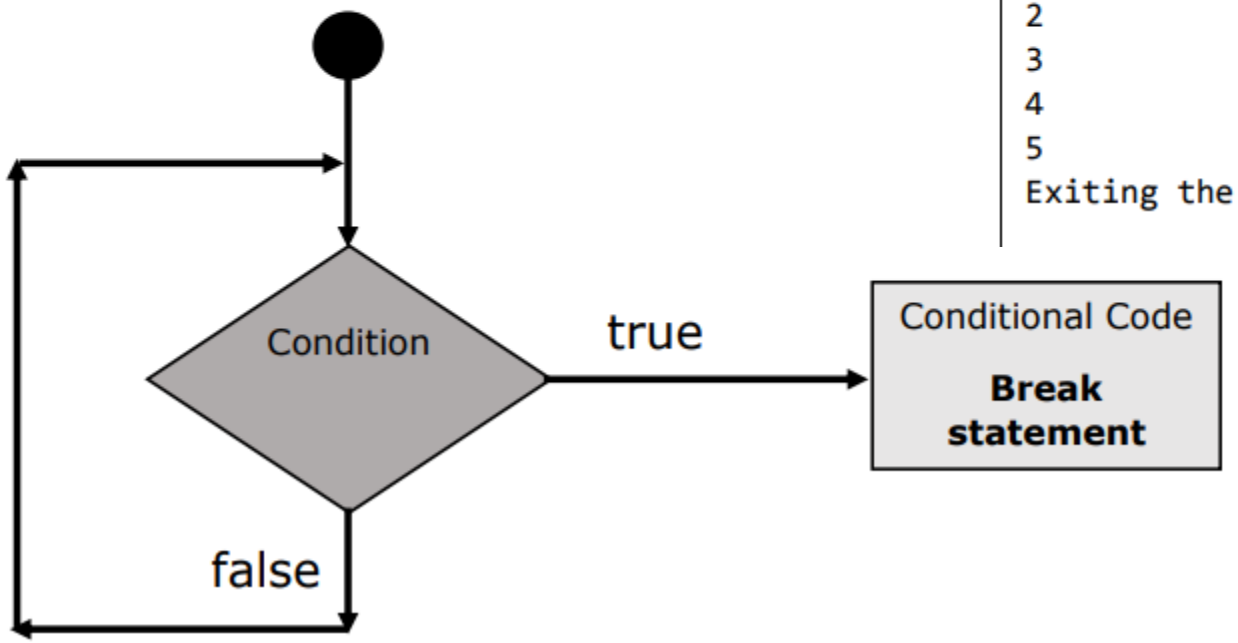
JavaScript provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching at its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the look. To handle all such situations, JavaScript provides break and continue statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

The break Statement

The **break** statement, which was briefly introduced with the *switch* statement, is used to exit a loop early, breaking out of the enclosing curly braces.

Flow Chart

The flow chart of a break statement would look as follows:



Output

```
Entering the loop
2
3
4
5
Exiting the loop!
```

```
<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop<br /> ");
while (x < 20)
{
  if (x == 5){
     break;  // breaks out of loop completely
  }
  x = x + 1;
  document.write( x + "<br />");
}
document.write("Exiting the loop!<br /> ");
//-->
</script>
```

The continue Statement

The **continue** statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a **continue** statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

```
<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop<br /> ");
while (x < 10)
{
  x = x + 1;
  if (x == 5){
     continue;  // skill rest of the loop body
  }
  document.write( x + "<br />");
}
document.write("Exiting the loop!<br /> ");
//-->
</script>
```

Output

```
Entering the loop
2
3
4
6
7
8
9
10
Exiting the loop!
```

Question

1. Write JS to display the sum of n natural number
2. Even and odd
3. Prime number
4. Multiplication table
5. Reverse
6. Leap year or not
7. Sum of digit
8. Swapping Two numbers
9. Find the largest among 3
10. Genderate multiplication table where number is provided by user
11. Fibonacci series
12. Count the no of digits

```
var a=Number(prompt("enter a number"));
if((a%2)==0)
{
    document.write("even");
}
else{
    document.write("odd");
}
```

```
date function in JS
var today=new Date();
var
date=today.toLocaleString();
var day=today.getDay();
var month=today.getMonth();
var year=
today.getFullYear();
var time=today.getTime();
var hour=today.getHours();
var min=today.getMinutes();
var sec=today.getSeconds();
var
mili=today.getMilliseconds();
```

```
document.write(today);
document.write("<br>");
document.write(date);
document.write("<br>");
document.write(day);
document.write("<br>");
document.write(month);
document.write("<br>");
document.write(year);
document.write("<br>");
document.write(time);
document.write("<br>");
document.write(hour);
document.write("<br>");
document.write(min);
document.write("<br>");
document.write(sec);
document.write("<br>");
document.write(mili);
```

```
Count the no of digits
  <script>
var a=Number(prompt('Enter a number'));
let count=0;
do{
a=Math.floor(a/10);
++count;
}while(a!=0);
document.write(count);
    </script>
```

JavaScript Objects and Primitives:

- In JavaScript, objects are collections of properties. Each property is either a data property or a function or method property.

In JavaScript, almost "everything" is an object.

- Booleans can be objects (if defined with the **new** keyword)
- Numbers can be objects (if defined with the **new** keyword)
- Strings can be objects (if defined with the **new** keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects

All JavaScript values, except primitives, are objects.

JavaScript Primitives:

A **primitive value** is a value that has no properties or methods.

A **primitive data type** is data that has a primitive value.

JavaScript defines 5 types of primitive data types:

JavaScript Primitives:

A **primitive value** is a value that has no properties or methods.

A **primitive data type** is data that has a primitive value.

JavaScript defines 5 types of primitive data types:

- string
- number
- boolean
- null
- undefined

Primitive values are immutable (they are hardcoded and therefore cannot be changed).

General Syntactic Characteristics:

- JavaScript can appear directly as the content of a <script> tag.
- The type attribute of <script> must be set to "text/javascript".
- The JavaScript script can be indirectly embedded in an XHTML document with the src attribute of a <script> tag, whose value is the name of a file that contains the script — for example,

```
<script type = "text/javascript" src = "tst_number.js" >
</script>
```

- The script element requires the closing tag, even though it has no content when the src attribute is included.

Objects

Real Life Objects, Properties, and Methods

In real life, a car is an **object**.
A car has **properties** like weight and color,
and **methods** like start and stop:

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

<h2>JavaScript Objects</h2>

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

<script>
// Create an object:
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};

// Display some data from the object:
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>

</body>
</html>
```

Accessing object
objectName.propertyName

objectName["propertyName"]

String Properties and Methods

- Because JavaScript coerces primitive string values to and from String objects when necessary, the differences between the String object and the String type have little effect on scripts.
- String methods can always be used through String primitive values, as if the values were objects. The String object includes one property, length, and a large collection of methods.
  The number of characters in a string is stored in the length property as follows:

var str = “George”;
var len = str.length;
In this code, len is set to the number of characters in str, namely, 6. In the expression str.length, str is a primitive variable, but we treated it as if it were an object (referencing one of its properties).
In fact, when str is used with the length property, JavaScript implicitly builds a temporary String object with a property whose value is that of the primitive variable.
After the second statement is executed, the temporary String object is discarded.
A few of the most commonly used `String` methods:

Method	Parameters	Result
`charAt`	A number	Returns the character in the `String` object that is at the specified position
`indexOf`	One-character string	Returns the position in the `String` object of the parameter
`substring`	Two numbers	Returns the substring of the `String` object from the first parameter position to the second
`toLowerCase`	None	Converts any uppercase letters in the string to lowercase
`toUpperCase`	None	Converts any lowercase letters in the string to uppercase

Implicit Type Conversion:

- The JavaScript interpreter performs several different implicit type conversions. Such conversions are called **coercions**.
- In general, when a value of one type is used in a position that requires a value of a different type, JavaScript attempts to convert the value to the type that is required.
- The most common examples of these conversions involve **primitive string** and **number values.**
- If either operand of a + operator is a string, the operator is interpreted as a string catenation operator. If the other operand is not a string, it is coerced to a string. For example, consider the following expression:
  "August " + 1977
  In this expression, because the left operand is a string, the operator is considered to be a catenation operator. This forces string context on the right operand, so the right operand is implicitly converted to a string. Therefore, the expression evaluates to
  "August 1997"
- Now consider the following expression:
  7 * "3"
  In this expression, the operator is one that is used only with numbers. This forces a numeric context on the right operand. Therefore, JavaScript attempts to convert it to a number. In this example, the conversion succeeds, and the value of the expression is 21.

Example 1: Implicit Conversion to String

```
// numeric string used with + gives string type
let result;

result = '3' + 2;
console.log(result) // "32"

result = '3' + true;
console.log(result); // "3true"

result = '3' + undefined;
console.log(result); // "3undefined"

result = '3' + null;
console.log(result); // "3null"
```

Note: When a number is added to a string, JavaScript converts the number to a string before concatenation.

Example 2: Implicit Conversion to Number

```
// numeric string used with - , / , * results number type

let result;

result = '4' - '2';
console.log(result); // 2

result = '4' - 2;
console.log(result); // 2

result = '4' * 2;
console.log(result); // 8

result = '4' / 2;
console.log(result); // 2
```

Example 3: Non-numeric String Results to NaN

```
// non-numeric string used with - , / , * results to NaN

let result;

result = 'hello' - 'world';
console.log(result); // NaN

result = '4' - 'hello';
console.log(result); // NaN
```

Example 4: Implicit Boolean Conversion to Number

```
// if boolean is used, true is 1, false is 0

let result;

result = '4' - true;
console.log(result); // 3

result = 4 + true;
console.log(result); // 5

result = 4 + false;
console.log(result); // 4
```

Note: JavaScript considers 0 as `false` and all non-zero number as `true`. And, if `true` is converted to a number, the result is always 1.

Example 5: null Conversion to Number

```
// null is 0 when used with number
let result;

result = 4 + null;
console.log(result);  // 4

result = 4 - null;
console.log(result);  // 4
```

Example 6: undefined used with number, boolean or null

```
// Arithmetic operation of undefined with number, boolean or null gives NaN

let result;

result = 4 + undefined;
console.log(result);  // NaN

result = 4 - undefined;
console.log(result);  // NaN

result = true + undefined;
console.log(result);  // NaN

result = null + undefined;
console.log(result);  // NaN
```

Explicit Type Conversions:

- There are several different ways to force type conversions, primarily between strings and numbers. Strings that contain numbers can be converted to numbers with the `String` constructor, as in the following code:
  var str_value = String(value);

- This conversion could also be done with the `toString` method, which has the advantage that it can be given a parameter to specify the base of the resulting number (although the base of the number to be converted is taken to be decimal). An example of such a conversion is:
  var num = 6;
  var str_value = num.toString();
  var str_value_binary = num.toString(2);

In the first conversion, the result is `"6"`; in the second, it is `"110"`.

- A number also can be converted to a string by catenating it with the empty string. Strings can be explicitly converted to numbers in several different ways. One way is with the `Number` constructor, as in the following statement:
  var number = Number(aString);
- The same conversion could be specified by subtracting zero from the string, as in the following statement:
  var number = aString - 0;
- Both of these conversions have the following restriction: The number in the string cannot be followed by any character except a space.
- For example, if the number happens to be followed by a comma, the conversion will not work. JavaScript has two predefined string functions that do not have this problem.

1. Convert to Number Explicitly

To convert numeric strings and boolean values to numbers, you can use `Number()`. For example,

```
let result;

// string to number
result = Number('324');
console.log(result); // 324

result = Number('324e-1')
console.log(result); // 32.4

// boolean to number
result = Number(true);
console.log(result); // 1

result = Number(false);
console.log(result); // 0
```

In JavaScript, empty strings and `null` values return **0**. For example,

```
let result;
result = Number(null);
console.log(result);  // 0

let result = Number(' ')
console.log(result);  // 0
```

If a string is an invalid number, the result will be `NaN`. For example,

```
let result;
result = Number('hello');
console.log(result); // NaN

result = Number(undefined);
console.log(result); // NaN

result = Number(NaN);
console.log(result); // NaN
```

Note: You can also generate numbers from strings using `parseInt()`, `parseFloat()`, unary operator `+` and `Math.floor()`. For example,

```
let result;
result = parseInt('20.01');
console.log(result); // 20

result = parseFloat('20.01');
console.log(result); // 20.01

result = +'20.01';
console.log(result); // 20.01

result = Math.floor('20.01');
console.log(result); // 20
```

2. Convert to String Explicitly

To convert other data types to strings, you can use either `String()` or `toString()`. For example,

```
//number to string
let result;
result = String(324);
console.log(result);  // "324"

result = String(2 + 4);
console.log(result); // "6"

//other data types to string
result = String(null);
console.log(result); // "null"

result = String(undefined);
console.log(result); // "undefined"

result = String(NaN);
console.log(result); // "NaN"

result = String(true);
console.log(result); // "true"

result = String(false);
console.log(result); // "false"
```

```
// using toString()
result = (324).toString();
console.log(result); // "324"

result = true.toString();
console.log(result); // "true"
```

3. Convert to Boolean Explicitly

To convert other data types to a boolean, you can use `Boolean()`.

In JavaScript, `undefined`, `null`, `0`, `NaN`, `''` converts to `false`. For example,

```
let result;
result = Boolean('');
console.log(result); // false

result = Boolean(0);
console.log(result); // false

result = Boolean(undefined);
console.log(result); // false

result = Boolean(null);
console.log(result); // false

result = Boolean(NaN);
console.log(result); // false
```

All other values give `true`. For example,

```
result = Boolean(324);
console.log(result); // true

result = Boolean('hello');
console.log(result); // true

result = Boolean(' ');
console.log(result); // true
```

JavaScript Type Conversion Table

The table shows the conversion of different values to String, Number, and Boolean in JavaScript.

Value	String Conversion	Number Conversion	Boolean Conversion
1	"1"	1	`true`
0	"0"	0	`false`
"1"	"1"	1	`true`
"0"	`"0"`	0	`true`
"ten"	`"ten"`	`NaN`	`true`
`true`	"true"	1	`true`
`false`	"false"	0	`false`
`null`	"null"	0	`false`
`undefined`	"undefined"	`NaN`	`false`
''	""	0	`false`
' '	" "	0	`true`

The two, `parseInt` and `parseFloat`, are not `String` methods, so they are not called through `String` objects; however, they operate on the strings given as parameters.

The `parseInt` function searches its string parameter for an integer literal. If one is found at the beginning of the string, it is converted to a number and returned.

If the string does not begin with a valid integer literal, `NaN` is returned. The `parseFloat` function is similar to `parseInt`, but it searches for a floating-point literal, which could have a decimal point, an exponent, or both.

In both `parseInt` and `parseFloat`, the numeric literal could be followed by any nondigit character without causing any problems.

The Date Object:

- A `Date` object is created with the `new` operator and the `Date` constructor, which has several forms. `Date` constructor, takes no parameters and builds an object with the current date and time for its properties. For example, we might have
  var today = new Date();

The date and time properties of a `Date` object are in two forms: local and Coordinated Universal Time (UTC, which was formerly named Greenwich Mean Time)

Methods of Date Object:

Method	Returns
`toLocaleString`	A string of the `Date` information
`getDate`	The day of the month
`getMonth`	The month of the year, as a number in the range from 0 to 11
`getDay`	The day of the week, as a number in the range from 0 to 6
`getFullYear`	The year
`getTime`	The number of milliseconds since January 1, 1970
`getHours`	The number of the hour, as a number in the range from 0 to 23
`getMinutes`	The number of the minute, as a number in the range from 0 to 59
`getSeconds`	The number of the second, as a number in the range from 0 to 59
`getMilliseconds`	The number of the millisecond, as a number in the range from 0 to 999

The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Syntax

Use the following syntax to create an **Array** Object.

```
var fruits = new Array( "apple", "orange", "mango" );
```

The **Array** parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

You can create array by simply assigning values as follows:

```
var fruits = [ "apple", "orange", "mango" ];
```

You will use ordinal numbers to access and to set values inside an array as follows.

```
fruits[0] is the first element
fruits[1] is the second element
fruits[2] is the third element
```

Arrays:

Array Object Creation

- `Array` objects, unlike most other JavaScript objects, can be created in two distinct ways:
- The usual way to create any object is with the `new` operator and a call to a constructor. In the case of arrays, the constructor is named `Array`:

```
var my_list = new Array(1, 2, "three", "four");
var your_list = new Array(100);
```

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

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fLen = fruits.length;

let text = "<ul>";
for (let i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";

document.getElementById("demo").innerHTML = text;
</script>
```

- Banana
- Orange
- Apple
- Mango

Accessing array

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

<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
</script>
```

Changing value of array

```
<script>
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars;
</script>
```

Array in newline

```
var a=[1,2,3,4]
var b=a.join('<br>');
document.write(b);
```

Array methods

Array length	Array join()
Array toString()	Array delete()
Array pop()	Array concat()
Array push()	Array flat()
Array shift()	Array splice()
Array unshift()	Array slice()

JavaScript Array length

The length property returns the length (size) of an array:

JavaScript Array toString()

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

Example

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

The join() method also joins all array elements into a string.

It behaves just like toString(), but in addition you can specify the separator:

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

JavaScript Array pop()
The pop() method removes the last element from an array:

Example
```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
```

The pop() method returns the value that was "popped out":

Example
```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.pop();
```

JavaScript Array push()
The push() method adds a new element to an array (at the end):

Example
```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");
```

The push() method returns the new array length:

Example
```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.push("Kiwi");
```

JavaScript Array unshift()
The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements:

Example
```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
```

Shifting Elements
Shifting is equivalent to popping, but working on the first element instead of the last.

JavaScript Array shift()
The shift() method removes the first array element and "shifts all other elements to a lower index.

Example
```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();
```

The shift() method returns the value that was "shifted out":

Example
```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.shift();
```

The unshift() method returns the new array length:

Example
```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
```

Changing Elements

Array elements are accessed using their **index number**:

Array **indexes** start with 0:

[0] is the first array element

[1] is the second

[2] is the third ...

Example
```
const fruits =
["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi";
```

JavaScript Array length

The length property provides an easy way to append a new element to an array:

Example
```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Kiwi";
```

JavaScript Array delete()
Warning !
Array elements can be deleted using the JavaScript operator delete.

Using delete leaves undefined holes in the array.

Use pop() or shift() instead.

Example
const fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];

```
<script>
const fruits = ["Banana", "Orange", "Apple",
"Mango"];
delete fruits[2];
document.write(fruits[2]);
</script>
```

Merging (Concatenating) Arrays
The concat() method creates a new array by merging (concatenating) existing arrays:

The concat() method does not change the existing arrays. It always returns a new array.

The concat() method can take any number of array arguments:

The concat() method can also take strings as arguments:

Example (Merging an Array with Values)
const arr1 = ["Emil", "Tobias", "Linus"];
const myChildren = arr1.concat("Peter");

Flattening an Array

Flattening an array is the process of reducing the dimensionality of an array.

The flat() method creates a new array with sub-array elements concatenated to a specified depth.

Example

```
const myArr = [[1,2],[3,4],[5,6]];
const newArr = myArr.flat();
```

Splicing and Slicing Arrays
The splice() method adds new items to an array.

The slice() method slices out a piece of an array.

JavaScript Array splice()
The splice() method can be used to add new items to an array:

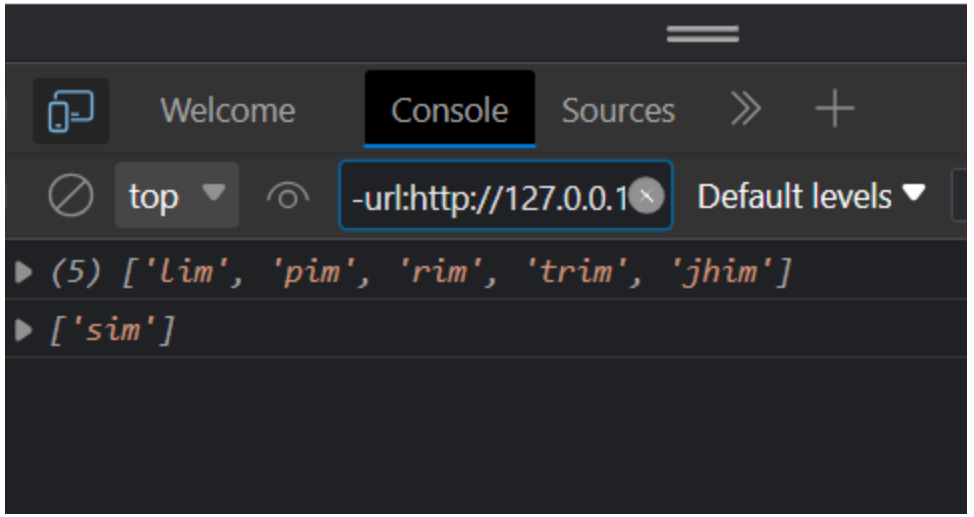
Example
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");

```
const fruits =
["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 2, "Lemon", "Kiwi");
```

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).
The second parameter (0) defines **how many** elements should be **removed**.
The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

```
<script>
var a=['sim','rim','trim','jhim'];
b=a.splice(0,1,'lim','pim');
document.write(a);
document.write('<br>')
document.write(b);
console.log(a);
console.log(b);
</script>
```

lim,pim,rim,trim,jhim
sim



JavaScript Array slice()
The slice() method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

```
const fruits =
["Banana", "Orange", "Lemon", "Apple", "Mango"]
;
const citrus = fruits.slice(3);
```

Example
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1);

Note
The slice() method creates a new array.

The slice() method does not remove any elements from the source array.

The slice() method can take two arguments like slice(1, 3).

The method then selects elements from the start argument, and up to (but not including) the end argument.

```
const fruits =
["Banana", "Orange", "Lemon", "Apple", "Mango"]
;
const citrus = fruits.slice(1, 3);
```

Question for you

a. create an array of the country name Nepal, India, Pakistan, America, Australia, Canada
b. Display the length of an array.
c. Display the 5th country in your array.
d. Convert your array to a string
e. Add a new country to the array and display size.
f. Pop 1st element from your array
g. Pop the last element from your array.
h. Separate the array element by *
i. Suppose china is 1st country in your array now recover it.
j. Change the 3rd country from your array and make it Bhutan.
k. Add 3 new countries to the 3rd position from the start but do not remove any country.
l. Same as k but remove 2 countries and display those countries which are removed
m. Remove 3rd country from the original array.
n. Create a new array that takes the value of Pakistan, America, and Australia from the original array.
o. Find the reverse of the array
p. Now create a second array with number number=[3,0,9,-5,-4,1,6,20,21,19,18,4,2]
q. Sort the above array by ascending and descending.
r. Find the reverse of the array.
s. Find the max and min value from given array.

```
<script>
var
a=['Nepal','india','Pakistan','America','Austra
lia','canada'];
b=a.splice(3,2,'hongkong','japan','dubai')
console.log(a);
console.log(b);
</script>
```

```
lowest
const a=[1,11,2,-4,5,6,0,9,7,8,15,12,13]
function AB(a,b)
{
    return a-b;
}
a.sort(AB);
console.log(a[0])
```

Array Methods

Here is a list of the methods of the Array object along with their description.

Method	Description
concat()	Returns a new array comprised of this array joined with other array(s) and/or value(s).
every()	Returns true if every element in this array satisfies the provided testing function.
filter()	Creates a new array with all of the elements of this array for which the provided filtering function returns true.
forEach()	Calls a function for each element in the array.
indexOf()	Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
join()	Joins all elements of an array into a string.
lastIndexOf()	Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
map()	Creates a new array with the results of calling a provided function on every element in this array.
pop()	Removes the last element from an array and returns that element.
push()	Adds one or more elements to the end of an array and returns the new length of the array.
reduce()	Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
reduceRight()	Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.
reverse()	Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
shift()	Removes the first element from an array and returns that element.
slice()	Extracts a section of an array and returns a new array.
some()	Returns true if at least one element in this array satisfies the provided testing function.
toSource()	Represents the source code of an object
sort()	Sorts the elements of an array.
splice()	Adds and/or removes elements from an array.
toString()	Returns a string representing the array and its elements.
unshift()	Adds one or more elements to the front of an array and returns the new length of the array.

```
var a=[1,2,3,4]
var b=a.toString();
document.write(a.join("-"));
```

```
var a=[1,2,3,4,"ABCD"]
var b=a.toString();
document.write(a.length);
delete a[0];
document.write(a);
document.write(a.length);
```

```
var a=[1,2,3,4]
var b=a.toString();
a[2]=10;
document.write(a)
```

Sorting in Descending order

```
var a=[8,2,20,21,4];
var b=[6,7,8,9];
let comare =(a,b)=>
{
    return b-a;
}
a.sort(comare)
document.write(a);
```

```
var a=[8,2,20,21,4];
var b=[6,7,8,9];

var c=a.slice(2,3);
document.write(c);
```

```
var a=[8,2,20,21,4];
var b=[6,7,8,9];

a.splice(3,2,11,12,13);
document.write(a);
```

Function Definition

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax

The basic syntax is shown here.

```
<script type="text/javascript">
<!--
function functionname(parameter-list)
{
  statements
}
//-->
</script>
```

Example

Try the following example. It defines a function called sayHello that takes no parameters:

The () Operator

The () operator invokes (calls) the function:

```
<script type="text/javascript">
<!--
function sayHello()
{
   alert("Hello there");
}
//-->
</script>
```

```
<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}

let value = toCelsius(77);
document.getElementById("demo").innerHTML = value;
</script>
```

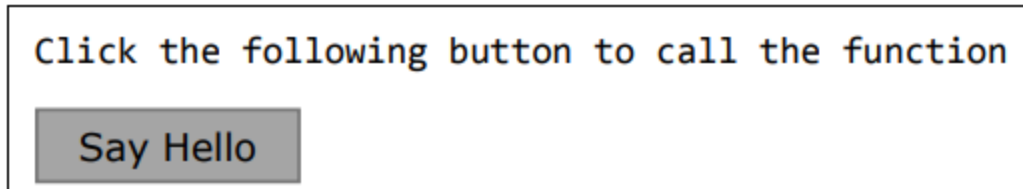
25

```
const a=[1,11,2,-4,5,6,0,9,7,8,15,12,13]
function AB(a,b)
{
    return a-b;
}
a.sort(AB);
console.log(a[0])
```

Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<html>
<head>
<script type="text/javascript">
function sayHello()
{
  document.write ("Hello there!");
}
</script>
</head>

<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello()" value="Say Hello">
</form>

<p>Use different text in write method and then try...</p>
</body>
</html>
```



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":

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Functions</h1>
<p>Call a function which performs a calculation and returns the result:</p>
<p id="demo"></p>
<script>
let x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;
function myFunction(a, b) {
  return a * b;
}
</script>
</body>
</html>
```

JavaScript Functions

Call a function which performs a calculation and returns the result:

12

Example

Try the following example. We have modified our **sayHello** function here. Now it takes two parameters.

```
<html>
<head>
<script type="text/javascript">
function sayHello(name, age)
{
   document.write (name + " is " + age + " years old.");
}
</script>
</head>

<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
</form>

<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible"
content="IE=edge">
    <meta name="viewport" content="width=device
-width, initial-scale=1.0">
    <title>Document</title>

</head>
<body>
    <button type="button"
onclick="A(4,5)">Click here</button>

    <script src="jsexample.js">

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

```javascript
function A(a,b)
{
 document.write(a*b)
;
}
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible"
content="IE=edge">
    <meta name="viewport" content="width=device
-width, initial-scale=1.0">
    <title>Document</title>

</head>
<body>
    <button type="button"
onclick="A('simanta',27)">Click here</button>

    <script src="jsexample.js">

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

```javascript
function A(name,age)
{
 document.write("hel
lo" + name +"You
are"+ age+"years
old" );
}
```

hellosimantaYou are27years old

```
Multiplication table
function A(a)
{
    for(var i=1;i<=5;i++)
    {
        document.write("<br>");
 document.write("3*"+ i+"=", a*i);
    }
}


     var a=Number(prompt("enter a number to find
     multiplication"));
     function A(){
     for(var i=1;i<=10;i++)
     {

         document.write(a*i);
         document.write("<br>");
     }
     }
```

```
<html>
<head>
<script type="text/javascript">
function concatenate(first, last)
{
 var full;
 full = first + last;
 return full;
}
function secondFunction()
{
 var result;
 result = concatenate('Simanta', 'Kasaju');
 document.write (result );
}
</script>
</head>
```

```
<body>
    <p>Click the following button to call the
function</p>
    <form>
    <input type="button"
onclick="secondFunction()" value="Call
Function">
    </form>

    </body>
    </html>
```

```
<html>
<head>
   <style>
      img{
         display: none;
         height: 50%;
         width: 50%;
      }
   </style>
<script src="jsexample.js">

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

```
<p >Image is hidden click below to show</p>
   <img src="2.jpg" alt="" id="AB">
   <form>
   <input type="button" onclick=A() value="Click to show
image">
   </form>

   </body>
   </html>
```

Image is hidden click below to show

Click to show image

Image is hidden click below to show

Click to show image

JS string

A JavaScript string is zero or more characters written inside quotes.

String Length

To find the length of a string, use the built-in length property:

```
let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
let length = text.length;
```

String Methods

String length	String trim()
String slice()	String trimStart()
String substring()	String trimEnd()
String substr()	String padStart()
String replace()	String padEnd()
String replaceAll()	String charAt()
String toUpperCase()	String charCodeAt()
String toLowerCase()	String split()
String concat()	

Extracting String Parts

There are 3 methods for extracting a part of a string:

slice(start, end)
substring(start, end)
substr(start, length)

If you omit the second parameter, the method will slice out the rest of the string:

```
let text = "Apple, Banana, Kiwi";
let part = text.slice(7);
```

JavaScript String slice()

slice() extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: start position, and end position (end not included).

Slice out a portion of a string from position 7 to position 13:

```
let text = "Apple, Banana, Kiwi";
let part = text.slice(7, 13);
```

```
let text = "Apple, Banana, Kiwi";
let part = text.slice(-12);
```

If a parameter is negative, the position is counted from the end of the string:

JavaScript String substring()
substring() is similar to slice().
The difference is that start and end values less than 0 are treated as 0 in the substring().

```
let str = "Apple, Banana, Kiwi";
let part = str.substring(7, 13);
```

If you omit the second parameter, substring() will slice out the rest of the string.

JavaScript String substr()
substr() is similar to slice().

The difference is that the second parameter specifies the length of the extracted part.

```
let str = "Apple, Banana, Kiwi";
let part = str.substr(7, 6);
```

If you omit the second parameter, substr() will slice out the rest of the string.

```
let str = "Apple, Banana, Kiwi";
let part = str.substr(7);
```

If the first parameter is negative, the position counts from the end of the string.

Example

```
let str = "Apple, Banana, Kiwi";
let part = str.substr(-4);
```

Replacing String Content
The replace() method replaces a specified value with another value in a string:

Example
let text = "Please visit Microsoft!";
let newText = text.replace("Microsoft", "W3Schools");

By default, the replace() method replaces only the first match:

```
<script>
function myFunction() {
  let text = document.getElementById("demo").innerHTML;
  document.getElementById("demo").innerHTML =
  text.replace("Microsoft","W3Schools");
}
</script>
```

By default, the replace() method is case sensitive. Writing MICROSOFT (with upper-case) will not work:

Example
let text = "Please visit Microsoft!";
let newText = text.replace("MICROSOFT", "W3Schools");

To replace case insensitive, use a regular expression with an /i flag (insensitive):

```
let text = "Please visit Microsoft!";
let newText =
text.replace(/MICROSOFT/i, "W3Schools");
```

To replace all matches, use a regular expression with a /g flag (global match):

Example
let text = "Please visit Microsoft and Microsoft!";
let newText = text.replace(/Microsoft/g, "W3Schools");

JavaScript String ReplaceAll()
In 2021, JavaScript introduced the string method replaceAll():

Example
```
text = text.replaceAll("Cats","Dogs");
text = text.replaceAll("cats","dogs");
```

The replaceAll() method allows you to specify a regular expression instead of a string to be replaced.
If the parameter is a regular expression, the global flag (g) must be set, otherwise a TypeError is thrown.

Example
```
text = text.replaceAll(/Cats/g,"Dogs");
text = text.replaceAll(/cats/g,"dogs");
```

Converting to Upper and Lower Case
A string is converted to upper case with toUpperCase():

A string is converted to lower case with toLowerCase():

JavaScript String toUpperCase()
Example

```
let text1 = "Hello World!";
let text2 = text1.toUpperCase();
```

```
let text1 = "Hello World!";       // String
let text2 = text1.toLowerCase();  // text2 is
text1 converted to lower
```

```
<script>
function myFunction() {
  let text = document.getElementById("demo").innerHTML;
  document.getElementById("demo").innerHTML =
  text.toUpperCase();
}
</script>
```

```
<script>
function myFunction() {
  let text = document.getElementById("demo").innerHTML;
  document.getElementById("demo").innerHTML =
  text.toLowerCase();
}
</script>
```

JavaScript String concat()
concat() joins two or more strings:

Example
```
let text1 = "Hello";
let text2 = "World";
let text3 = text1.concat(" ", text2);
```

JavaScript String trim()
The trim() method removes whitespace from both sides of a string:

Example
```
let text1 = "      Hello World!      ";
let text2 = text1.trim();
```

The concat() method can be used instead of the plus operator.
These two lines do the same:

Example
```
text = "Hello" + " " + "World!";
text = "Hello".concat(" ", "World!");
```

JavaScript String trimStart()
ECMAScript 2019 added the String method trimStart() to JavaScript.

The trimStart() method works like trim(), but removes whitespace only from the start of a string.

Example
```
let text1 = "     Hello World!     ";
let text2 = text1.trimStart();
```

```
<script>
let text1 = "     Hello World!     ";
let text2 = text1.trimStart();

document.getElementById("demo").innerHTML =
"Length text1 = " + text1.length + "<br>Length text2 = " +
text2.length;
</script>
```

JavaScript String trimEnd()
ECMAScript 2019 added the string method trimEnd() to JavaScript.

The trimEnd() method works like trim(), but removes whitespace only from the end of a string.

Example
```
let text1 = "     Hello World!     ";
let text2 = text1.trimEnd();
```

```
<script>
let text1 = "     Hello World!     ";
let text2 = text1.trimEnd();

document.getElementById("demo").innerHTML =
"Length text1 = " + text1.length + "<br>Length text2 = " +
text2.length;
</script>
```

JavaScript String Padding

ECMAScript 2017 added two new string methods to JavaScript: padStart() and padEnd() to support padding at the beginning and at the end of a string.

JavaScript String padStart()

The padStart() method pads a string from the start.

It pads a string with another string (multiple times) until it reaches a given length.

Examples

Pad a string with "0" until it reaches the length 4:

```
let text = "5";
let padded = text.padStart(4,"0");
```

```
<script>
let text = "5";
text = text.padStart(4,"0");

document.getElementById("demo").innerHTML = text;
</script>
```

Note

The padStart() method is a string method.

To pad a number, convert the number to a string first.

See the example below.

```
<script>
let numb = 5;
let text = numb.toString();
document.getElementById("demo").innerHTML =
text.padStart(4,0);
</script>
```

JavaScript String padEnd()
The padEnd() method pads a string from the end.

It pads a string with another string (multiple times) until it reaches a given length.

Examples

```
let text = "5";
let padded = text.padEnd(4,"0");
```

```
<script>
let text = "5";
text = text.padEnd(4,"0");

document.getElementById("demo").innerHTML = text;
</script>
```

Note
The padEnd() method is a string method.

To pad a number, convert the number to a string first.

See the example below.

Extracting String Characters
There are 3 methods for extracting string characters:

charAt(position)
charCodeAt(position)
Property access [ ]
JavaScript String charAt()
The charAt() method returns the character at a specified index (position) in a string:

Example
```
let text = "HELLO WORLD";
let char = text.charAt(0);
```

```
<script>
var text = "HELLO WORLD";
document.getElementById("demo").innerHTML =
text.charAt(0);
</script>
```

JavaScript String charCodeAt()
The charCodeAt() method returns the unicode of the character at a specified index in a string:

The method returns a UTF-16 code (an integer between 0 and 65535).

Example
```
let text = "HELLO WORLD";
let char = text.charCodeAt(0);
```

```
<script>
let text = "HELLO WORLD";
document.getElementById("demo").innerHTML =
text.charCodeAt(0);
</script>
```

Property Access

ECMAScript 5 (2009) allows property access [ ] on strings:

Example

```
let text = "HELLO WORLD";
let char = text[0];
```

```
<script>
let text = "HELLO WORLD";
document.getElementById("demo").innerHTML = text[7];
</script>
```

Note

Property access might be a little **unpredictable:**

- It makes strings look like arrays (but they are not)
- If no character is found, [ ] returns undefined, while charAt() returns an empty string.
- It is read only. str[0] = "A" gives no error (but does not work!)

```
let text = "HELLO WORLD";
text[0] = "A";    // Gives no error, but does
not work
```

String Search Methods

- String indexOf()
- String lastIndexOf()
- String search()
- String match()
- String matchAll()
- String includes()
- String startsWith()
- String endsWith()

JavaScript String indexOf()

The indexOf() method returns the index (position) the first occurrence of a string in a string:

Example

```
let text = "Please locate where 'locate' occurs!";
let index = text.indexOf("locate");
```

```
<script>
let text = "Please locate where 'locate' occurs!";
let index = text.indexOf("locate");
document.getElementById("demo").innerHTML = index;
</script>
```

JavaScript String lastIndexOf()

The lastIndexOf() method returns the index of the last occurrence of a specified text in a string:

Example

```
let text = "Please locate where 'locate' occurs!";
let index = text.lastIndexOf("locate");
```

```
<script>
let text = "Please locate where 'locate' occurs!";
let index = text.lastIndexOf("locate");
document.getElementById("demo").innerHTML = index;
</script>
```

Both indexOf(), and lastIndexOf() return -1 if the text is not found:

Both methods accept a second parameter as the starting position for the search:

```
let text = "Please locate where 'locate'
occurs!";
let index = text.indexOf("locate", 15);
```

The lastIndexOf() methods searches backwards (from the end to the beginning), meaning: if the second parameter is 15, the search starts at position 15, and searches to the beginning of the string.

```
let text = "Please locate where 'locate'
occurs!";
text.lastIndexOf("locate", 15);
```

JavaScript String search()
The search() method searches a string for a string (or a regular expression) and returns the position of the match:

Examples
let text = "Please locate where 'locate' occurs!";
text.search("locate");

```
<script>
let text = "Please locate where 'locate' occurs!";
let index = text.search("locate");
document.getElementById("demo").innerHTML = index;
</script>
```

RegExp Object

A regular expression is a **pattern** of characters.

The pattern is used to do pattern-matching **"search-and-replace"** functions on text.

In JavaScript, a **RegExp Object** is a pattern with **Properties** and **Methods**.

```
syntax
/pattern/modifier(s);
```

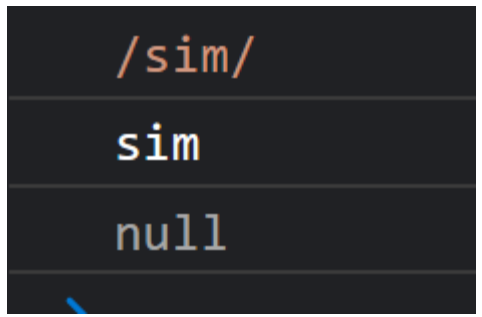
Modifiers

Modifiers are used to perform case-insensitive and global searches:

Modifier	Description
g	Perform a global match (find all matches rather than stopping after the first match)
i	Perform case-insensitive matching
m	Perform multiline matching

Some
example

```
var reg =/sim/;
var b="my name is siima kasaju";
console.log(reg);
console.log(reg.source);
result= reg.exec(b);
console.log(result);
```




```
var reg =/sim/;
var b="my name is sim kasaju";
console.log(reg);
console.log(reg.source);
result= reg.exec(b);
console.log(result);
```

Functions for regular expression

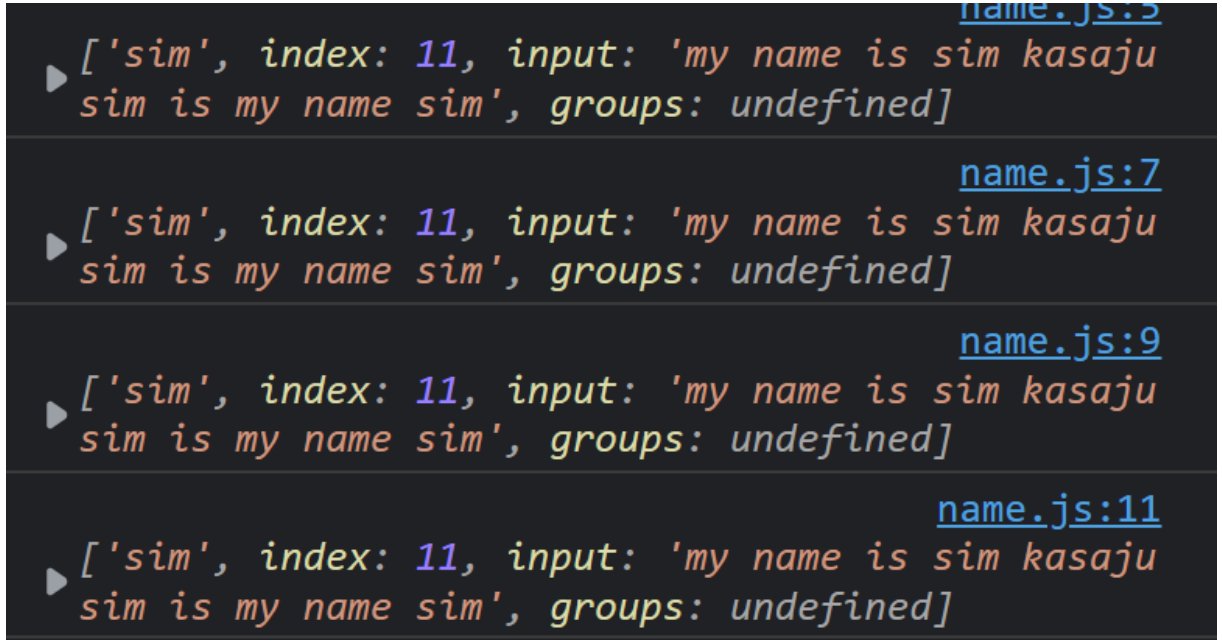
There are also some string methods that allow you to pass RegEx as its parameter. They are: `match()`, `replace()`, `search()`, and `split()`.

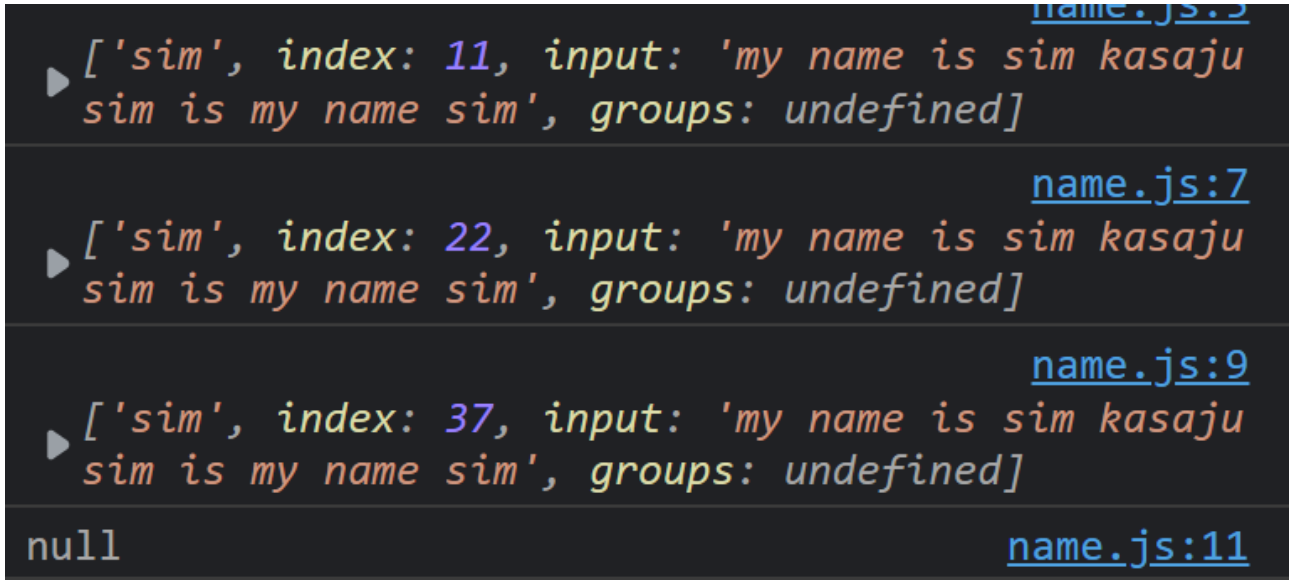
Method	Description
`exec()`	Executes a search for a match in a string and returns an array of information. It returns null on a mismatch.
`test()`	Tests for a match in a string and returns true or false.
`match()`	Returns an array containing all the matches. It returns null on a mismatch.
`matchAll()`	Returns an iterator containing all of the matches.
`search()`	Tests for a match in a string and returns the index of the match. It returns -1 if the search fails.
`replace()`	Searches for a match in a string and replaces the matched substring with a replacement substring.
`split()`	Break a string into an array of substrings.

```
//if g is not used
var reg =/sim/;
var b="my name is sim kasaju sim is my name
sim";
result= reg.exec(b);
console.log(result);
result= reg.exec(b);
console.log(result);
result= reg.exec(b);
console.log(result);
result= reg.exec(b);
console.log(result);
```

exec()

```
//if g is  used
var reg =/sim/g;
var b="my name is sim kasaju sim is my name
sim";
result= reg.exec(b);
console.log(result);
result= reg.exec(b);
console.log(result);
result= reg.exec(b);
console.log(result);
result= reg.exec(b);
console.log(result);
```





```
var reg =/sim/;
var b="my name is Sim kasaju Sim ";
result= reg.exec(b);
console.log(result);
result= reg.exec(b);
console.log(result);
```

```
null
null
```

```
var reg =/sim/i;
var b="my name is Sim kasaju Sim ";
result= reg.exec(b);
console.log(result);
result= reg.exec(b);
console.log(result);
```

```
name.js:5
['Sim', index: 11, input: 'my name is Sim kasaju Sim ', groups: undefined]
name.js:7
['Sim', index: 11, input: 'my name is Sim kasaju Sim ', groups: undefined]
```

```
var reg =/sim/gi;
var b="my name is Sim kasaju Sim ";
result= reg.exec(b);
console.log(result);
result= reg.exec(b);
console.log(result);
```

```
name.js:6
['Sim', index: 11, input: 'my name is Sim kasaju Sim ', groups: undefined]
name.js:8
['Sim', index: 22, input: 'my name is Sim kasaju Sim ', groups: undefined]
```

```
var reg =/sim/gi;
// var reg =/sim/i;
var b="my name is Sim kasaju Sim ";

result= reg.test(b);
console.log(result);
```

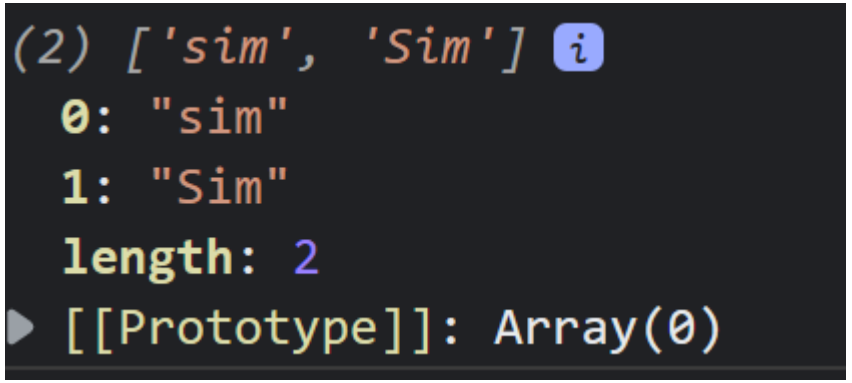
```
true
```

test()
Give true
if match
else false

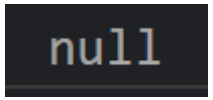
```
var reg =/sem/gi;
// var reg =/sim/i;
var b="my name is Sim kasaju Sim ";
result= reg.test(b);
console.log(result);
```

```
false
```

```
var reg =/sim/gi;
// var reg =/sim/i;
var b="my name is sim kasaju Sim ";
result= b.match(reg);
console.log(result);
```




```
var reg =/sem/gi;
// var reg =/sim/i;
var b="my name is sim kasaju Sim ";
result= b.match(reg);
console.log(result);
```



match(): Return array if match else null

```
var reg =/sim/gi;
// var reg =/sim/i;
var b="my name is sim kasaju Sim ";
result= b.search(reg);
console.log(result);
```

```
11
```

```
var reg =/sem/gi;
// var reg =/sim/i;
var b="my name is sim kasaju Sim ";
result= b.search(reg);
console.log(result);
```

```
-1
```

search() Give index if match else give -1

```
var reg =/sim/gi;
// var reg =/sim/i;
var b="my name is sim kasaju Sim ";
result= b.replace(reg,'Simanta');
console.log(result);
```

```
my name is Simanta kasaju Simanta
```

```
var reg =/sem/gi;
// var reg =/sim/i;
var b="my name is sim kasaju Sim ";
result= b.replace(reg,'Simanta');
console.log(result);
```

```
my name is sim kasaju Sim
```

replace()
replace
match
value
with
given
value

Pattern Matching Using Regular Expressions:

- JavaScript has powerful pattern-matching capabilities based on regular expressions.
- There are two approaches to pattern matching in JavaScript: one that is based on the `RegExp` object and one that is based on methods of the `String` object.
- Patterns, which are sent as parameters to the patternmatching methods, are delimited with slashes.
- The simplest pattern-matching method is `search`, which takes a pattern as a parameter. The `search` method returns the position in the `String` object (through which it is called) at which the pattern matched. If there is no match, `search` returns `-1`.

```
var a=/apple/gi;
var b="i love apple,banana,mango";
result=b.search(a);
if(result>=0)
{
document.write("apple is appeared at-",result,"<br>");
}
else{
   document.write("apple is not found <br>");
}
```

apple is appeared at-7

```
var a=/litchi/gi;
var b="i love apple,banana,mango";
result=b.search(a);
if(result>=0)
{
document.write("apple is appeared at-",result,"<br>");
}
else{
   document.write("apple is not found <br>");
}
```

apple is not found

```
var a=/hello/;
var b=" ";
result=b.search(a);
if(result>=0)
{
document.write("pattern is matched" );
}
else{
   document.write("not matched");
}
```

Character and Character-Class Patterns:

- The “normal” characters are those that are not metacharacters.
- Metacharacters are characters that have special meanings in some contexts in patterns. The following are
  the pattern metacharacters:
  \ | ( ) [ ] { } ^ $ * + ? .
- Metacharacters can themselves be matched by being immediately preceded by a backslash.
- For example, the following character class matches `'a'`, `'b'`, or `'c'`: [abc]
- The following character class matches any lowercase letter from `'a'` to `'h'`: [a-h]
- If a circumflex character (^) is the first character in a class, it inverts the specified set.
  For example, the following character class matches any character except the
  letters `'a'`, `'e'`, `'i'`, `'o'`, and `'u'`: [^aeiou]

Predefined Character- classes:

Name	Equivalent Pattern	Matches
\d	[0-9]	A digit
\D	[^0-9]	Not a digit
\w	[A-Za-z_0-9]	A word character (alphanumeric)
\W	[^A-Za-z_0-9]	Not a word character
\s	[\r\t\n\f]	A white-space character
\S	[^ \r\t\n\f]	Not a white-space character

Example of uses:

```
/\d\.\d\d/     // Matches a digit, followed by a period,
               // followed by two digits
/\D\d\D/       // Matches a single digit
/\w\w\w/       // Matches three adjacent word characters
```

In many cases, it is convenient to be able to repeat a part of a pattern, often a character or character class. To repeat a pattern, a numeric quantifier, delimited by braces, is attached. For example, the following pattern matches `xyyyyz`: /xy{4}z/ There are also three symbolic quantifiers: asterisk (`*`), plus (+), and question mark (`?`). An asterisk means zero or more repetitions, a plus sign means one or more repetitions, and a question mark means one or none. For example, the following pattern matches strings that begin with any number of `x`'s (including zero), followed by one or more `y`'s, possibly followed by `z`: /x*y+z?/

`.`	any character except newline
`\w \d \s`	word, digit, whitespace
`\W \D \S`	not word, digit, whitespace
`[abc]`	any of a, b, or c
`[^abc]`	not a, b, or c
`[a-g]`	character between a & g

Anchors

`^abc$`	start / end of the string
`\b \B`	word, not-word boundary

Escaped characters

`\. \* \\`	escaped special characters
`\t \n \r`	tab, linefeed, carriage return

Groups & Lookaround

`(abc)`	capture group
`\1`	backreference to group #1
`(?:abc)`	non-capturing group
`(?=abc)`	positive lookahead
`(?!abc)`	negative lookahead

https://regexr.com/

Quantifiers & Alternation

`a* a+ a?`	0 or more, 1 or more, 0 or 1
`a{5} a{2,}`	exactly five, two or more
`a{1,3}`	between one & three
`a+? a{2,}?`	match as few as possible
`ab|cd`	match ab or cd

Validating phone

```
// program to validate the phone number

function validatePhone(num) {

    // regex pattern for phone number
    const re = /^\(?([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]{4})$/g;

    // check if the phone number is valid
    let result = num.match(re);
    if (result) {
        console.log('The number is valid.');
    }
    else {
        let num = prompt('Enter number in XXX-XXX-XXXX format:');
        validatePhone(num);
    }
}

// take input
let number = prompt('Enter a number XXX-XXX-XXXX');

validatePhone(number);
```

```
Enter a number XXX-XXX-XXXXabcd
Enter number in -XXX-XXXX format:0123456789
The number is valid.
```

Validating email

```
// program to validate the email address

function validateEmail(email) {

    // regex pattern for email
    const re = /\S+@\S+\.\S+/g;

    // check if the email is valid
    let result = re.test(email);
    if (result) {
        console.log('The email is valid.');
    }
    else {
        let newEmail = prompt('Enter a valid email:');
        validateEmail(newEmail);
    }
}

// take input
let email = prompt('Enter an email: ');

validateEmail(email);
```

```
Enter an email: sim
Enter a valid email:sim.kasaju@gmail.com
The email is valid.
```

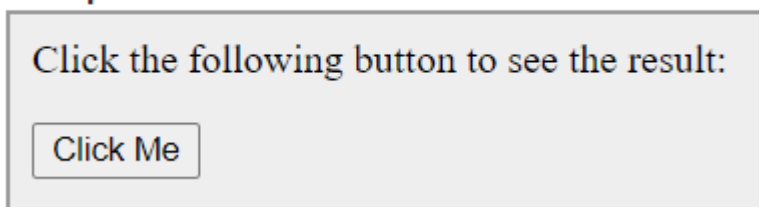
JavaScript - Dialog Boxes

Alert Dialog Box
An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed.

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function Warn() {
          alert ("This is a warning message!");
          document.write ("This is a warning
message!");
        }
      //-->
    </script>
  </head>

  <body>
    <p>Click the following button to see the
result: </p>
    <form>
      <input type = "button" value = "Click Me"
onclick = "Warn();" />
    </form>
  </body>
</html>
```

Output



Confirmation Dialog Box
A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: OK and Cancel.

If the user clicks on the OK button, the window method confirm() will return true. If the user clicks on the Cancel button, then confirm() returns false. You can use a confirmation dialog box as follows.

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function getConfirmation() {
          var retVal = confirm("Do you want to continue ?");
          if( retVal == true ) {
            document.write ("User wants to continue!");
            return true;
          } else {
            document.write ("User does not want to continue!");
            return false;
          }
        }
      //-->
    </script>
  </head>
```

```
<body>
    <p>Click the following button to see the result: </p>
    <form>
      <input type = "button" value = "Click Me" onclick =
"getConfirmation();" />
    </form>
  </body>
</html>
```

Prompt Dialog Box

The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.

This dialog box is displayed using a method called prompt() which takes two parameters: (i) a label which you want to display in the text box and (ii) a default string to display in the text box.

This dialog box has two buttons: OK and Cancel. If the user clicks the OK button, the window method prompt() will return the entered value from the text box. If the user clicks the Cancel button, the window method prompt() returns null.

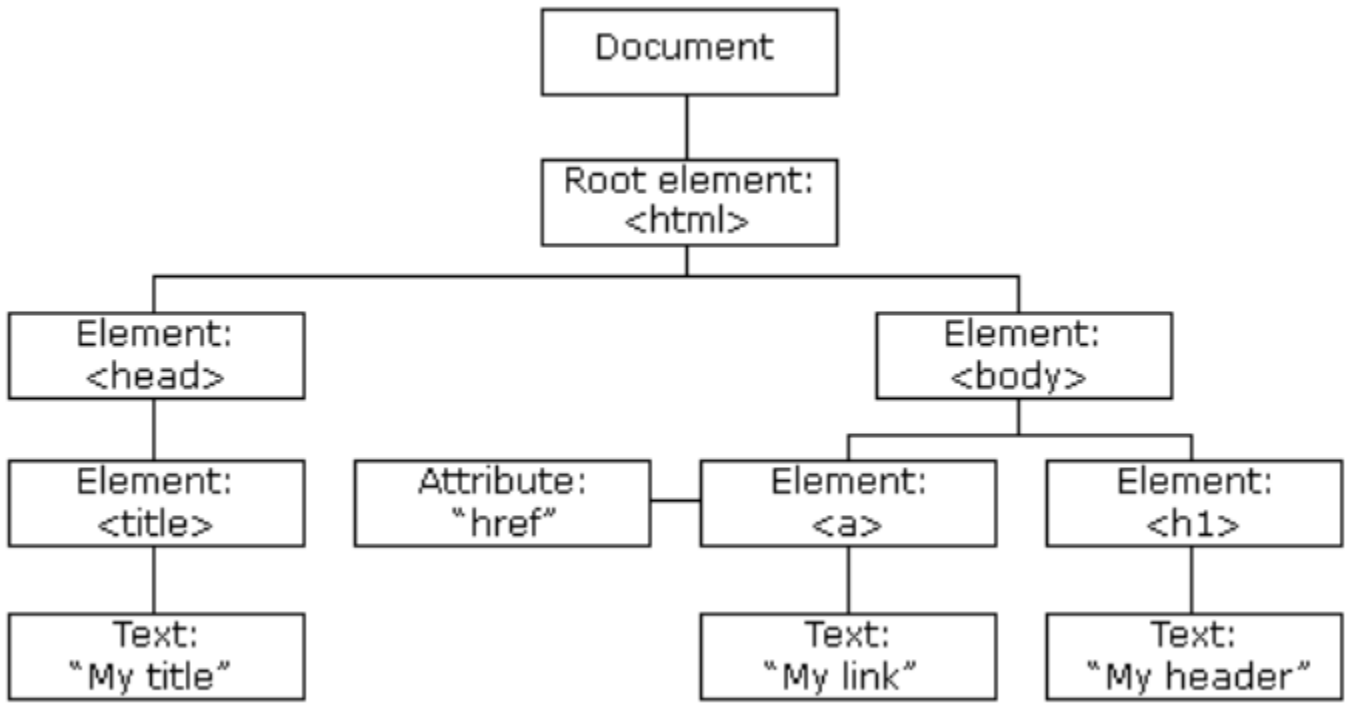
```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function getValue() {
          var retVal = prompt("Enter your name : ", "your
name here");
          document.write("You have entered : " +
retVal);
        }
      //-->
    </script>
  </head>

  <body>
    <p>Click the following button to see the result: </p>
    <form>
      <input type = "button" value = "Click Me" onclick =
"getValue();" />
    </form>
  </body>
</html>
```

Document Object Model (DOM):

- DOM is a W3C (World Wide Web Consortium) standard that defines a standard for accessing documents.
- *"The W3C 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"*
- The W3C DOM standard is separated into 3 different parts:
  - Core DOM - standard model for all document types
  - XML DOM - standard model for XML documents
  - HTML DOM - standard model for HTML documents
- Thus, The DOM is an application programming interface (API) that defines an interface between XHTML documents and application programs. It is an abstract model because it must apply to a variety of application programming languages.

The HTML DOM (Document Object Model)

- The HTML DOM is a standard for how to get, change, add, or delete HTML elements.
- It is a standard **object** model and **programming interface** for HTML that defines:
  - The HTML elements as **objects**
  - The **properties** of all HTML elements
  - The **methods** to access all HTML elements
  - The **events** for all HTML elements
- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
- The **HTML DOM** model is constructed as a tree of **Objects**:



Thus, with the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript **can change all the HTML elements** in the page
- JavaScript **can change all the HTML attributes** in the page
- JavaScript **can change all the CSS styles** in the page
- JavaScript **can remove existing HTML elements and attributes**
- JavaScript **can add new HTML elements and attributes**
- JavaScript **can react to all existing HTML events** in the page
- JavaScript can **create new HTML events** in the page

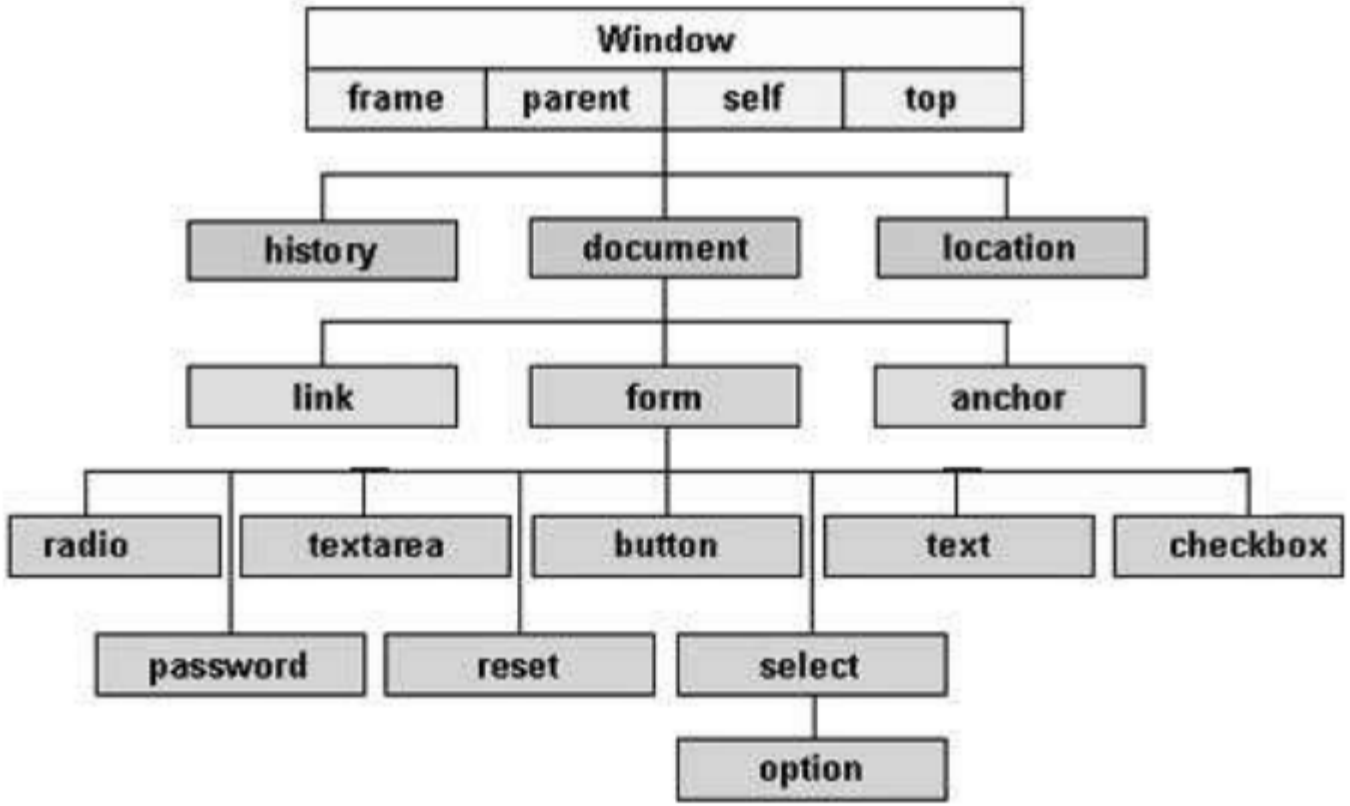
Every web page resides inside a browser window which can be considered as an object.

A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- **Window object:** Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object:** Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- **Form object:** Everything enclosed in the <form>...</form> tags sets the form object.
- **Form control elements:** The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of a few important objects:



There are several DOMs in existence. The following sections explain each of these DOMs in detail and describe how you can use them to access and modify document content.

- **The Legacy DOM**: This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but allows access only to certain key portions of documents, such as forms, form elements, and images.
- **The W3C DOM**: This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.
- **The IE4 DOM**: This document object model was introduced in Version 4 of Microsoft's Internet Explorer browser. IE 5 and later versions include support for most basic W3C DOM features.

```
console.log(window.body);
console.log(document.body);
document.body.style.background="red";
```

Methods and Properties in DOM:

- In the DOM, all HTML elements are defined as **objects**.
- The programming interface is the properties and methods of each object.
- A **property** is a value that you can get or set (like changing the content of an HTML element).
- A **method** is an action you can do (like add or deleting an HTML element).
- Thus, the data are called properties, and the operations are, naturally, called methods.
- For example, the following XHTML element would be represented as an object with two properties, type and name, with the values “text” and “address”, respectively:
  <input type = “text” name = “address”>
  In most cases, the property names in JavaScript are the same as their corresponding attribute names in XHTML.

Example

The following example changes the content (the innerHTML) of the <p> element with id="demo":

```
<!DOCTYPE html>
<html>
    <body>
        <h2>My First Page</h2>
        <p id="demo"></p>
        <script>
document.getElementById("demo").innerHTML = "Hello World!";
        </script>
    </body>
</html>
```

- In the example above, getElementById is a **method**, while innerHTML is a **property**.

Element Access in JavaScript:

- The elements of an XHTML document have corresponding objects that are visible to an embedded JavaScript script. The addresses of these objects are required, both by the event handling and by the code for making dynamic changes to documents.
- There are several ways the object associated with an XHTML form element can be addressed in JavaScript.
- **The original (DOM 0) way** is to use the forms and elements arrays of the Document object, which is referenced through the document property of the Window object. As an example, consider the following XHTML document:

```
<html>
        <head>Access to form elements</head>
 <body>
        <form name="myForm" action = "">
                <input type="button" name="turnItOn" id="turnItOn"/>
        </form>
 </body>
</html>
```

The DOM address of the button in this example, using the `forms` and `elements` arrays, is as follows:

```
var dom = document.forms[0].elements[0];
```

The problem with this approach to element addressing is that the DOM address is defined by address elements that could change—namely, the `forms` and `elements` arrays. For example, if a new button were added before the `turnItOn` button in the form, the DOM address shown would be wrong.

Another approach to DOM addressing is to use element names. For this, the element and its enclosing elements, up to but not including the body element, must include `name` attributes. For example, in above example:

Using the `name` attributes, the button's DOM address is as follows:
var dom = document.myForm.turnItOn;

Yet another approach to element addressing is to use the JavaScript method `getElementById`, which is defined in DOM 1. Because an element's identifier (`id`) is unique in the document, this approach works, regardless of how deeply the element is nested in other elements in the document.
For example, if the id attribute of our button is set to "`turnItOn`", the following could be used to get the DOM address of that button element:

var dom = document.getElementById("turnItOn");

The parameter of `getElementById` can be any expression that evaluates to a string.

Because ids are most useful for DOM addressing and `names` are required for form-processing code, form elements often have both ids and `names`, set to the same value. Buttons in a group of checkboxes and radio button often share the same name. In these cases, the names of the individual buttons obviously cannot be used in their DOM addresses. Of course, each radio button and checkbox can have an id, which would make them easy to address by using `getElementById`.

However, this approach does not provide a convenient way to search a group of radio buttons or checkboxes to determine which is checked.

An alternative to both `names` and ids is provided by the implicit arrays associated with each checkbox and radio button group.

Every such group has an array, which has the same name as the group name that stores the DOM addresses of the individual buttons in the group.

These arrays are properties of the form in which the buttons appear. To access the arrays, the DOM address of the form object must first be obtained, as in the following example:

```
<form id = "vehicleGroup">
  <input type = "checkbox"  name = "vehicles"
         value = "car" />  Car
  <input type = "checkbox"  name = "vehicles"
         value = "truck" />  Truck
  <input type = "checkbox"  name = "vehicles"
         value = "bike" />  Bike
</form>
```

The implicit array, `vehicles`, has three elements, which reference the three objects associated with the three checkbox elements in the group.
This array provides a convenient way to search the list of checkboxes in a group.
The `checked` property of a checkbox object is set to `true` if the button is checked.
For the preceding sample checkbox group, the following code would count the number of checkboxes that were checked:

```
var numChecked = 0;
var dom = document.getElementById("vehicleGroup");
for (index = 0; index < dom.vehicles.length; index++)
  if (dom.vehicles[index].checked)
    numChecked++;
```


Radio buttons can be addressed and handled exactly as are the checkboxes.

var AB=document.getElementById("A").attributes;

Finding HTML Elements

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name
`document.querySelectorAll("p.intro");`	Find HTML Elements by CSS Selectors
`var x = document.forms["frm1"];` `var text = "";` `var i;` `for (i = 0; i < x.length; i++) {` `  text += x.elements[i].value + " ";` `}` `document.getElementById("demo").innerHTML = text;`	Find HTML Element by HTML Object Collections

var AB=document.querySelector(".B");

Changing HTML Elements

Method	Description
element.innerHTML = *new html content*	Change the inner HTML of an element

```
<body>
  <p>This is paragraph 1</p>
  <p id="A" style="text-align: center;">This is paragraph.
     Lorem ipsum dolor, sit amet consectetur
     adipisicing elit. Enim ut tempore dolores
     provident cumque aut et, dolore nemo veniam.
      Autem molestias, dolore quasi inventore eum earum
quae. Est, rerum deserunt?
  </p>
        <h1 class="B">This is head</h1>
          <script>
            var AB=document.getElementById("A").innerText;
            var
        BC=document.getElementsByClassName("B").innerHTML;
            var
        CD=document.getElementsByTagName("p")[0].innerHTML;
        var element
        =document.getElementById("A").getAttribute("style");
        console.log(AB);
        console.log(BC);
        console.log(CD);

          </script>
```

element.attribute = new value	Change the attribute value of an HTML element
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element

Adding and Deleting Elements

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*element*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

```
var AB=document.getElementById("A").setAttribute("style","color:yellow");
```

```
var AB=document.getElementById("A").setAttribute("id","ABC")
```

```
<style>
    #ABC{
  background-color: red;
}
```

```
var AB=document.getElementById("A").attributes.value="ABC";
```

```
var AB=document.getElementById("A").getAttribute("style");
```

```
var AB=document.getElementById("A").removeAttribute("style");
```

```
const body=document.body;
body.append("hello");
const div=document.createElement("div");
body.append(div);
div.innerText="hello my";
body.append(div);

div=document.createElement("div");
console.log(a);
```

Difference between innertext and textcontent

```
var div=document.querySelector("div");
console.log(div.textContent);
console.log(div.innerText);
```

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-
scale=1.0">
   <title>Document</title>

   <style>
      .A{
       display: none;
      }
   </style>
</head>
<body>
<div>
   <h1>This is hidden</h1>
<p class="A">This is shown</p>
</div>
   <script src="name.js">
   </script>

</body>
</html>
```

Events and Events Handling:

Basic Concepts:

Event-driven Programming:

- One important use of JavaScript for Web programming is to detect certain activities of the browser and the browser user and provide computation when those activities occur.
- These computations are specified with a special form of programming called event-driven programming.
- In conventional (non-event-driven) programming, the code itself specifies the order in which it is executed, although the order is usually affected by the program's input data.
- In event-driven programming, parts of the program are executed at completely unpredictable times, often triggered by user interactions with the program that is executing.

Event

- An event is a notification that something specific has occurred, either with the browser, such as the completion of the loading of a document, or because of a browser user action, such as a mouse click on a form button.
- Strictly speaking, an event is an object that is implicitly created by the browser and the JavaScript system in response to something happening.

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

Event Handler:

- An event handler is a script that is implicitly executed in response to the appearance of an event.
- Event handlers enable a Web document to be responsive to browser and user activities.
- One of the most common uses of event handlers is to check for simple errors and omissions in user input to the elements of a form, either when they are changed or when the form is submitted. This kind of checking saves the time of sending incorrect form data to the server.
- Events are similar to exceptions in error handling.
    - Both events and exceptions occur at unpredictable times, and both often require some special program actions.

Registration:

- The process of connecting an event handler to an event is called registration.
- There are two distinct approaches to event handler registration, one that assigns tag attributes and one that assigns handler addresses to object properties

Why **write** method should never be used in an event handler?

- It is because a document is displayed as its XHTML code is parsed by the browser.
- **Events usually occur after the whole document is displayed**.
- If `write` appears in an event handler, the content produced by it might be placed over the top of the currently displayed document.

Events, Attributes and Tags:

Events and their tag attributes:

Event	Tag Attribute
blur	onblur
change	onchange
click	onclick
dblclick	ondblclick
focus	onfocus
keydown	onkeydown
keypress	onkeypress
keyup	onkeyup
load	onload
mousedown	onmousedown
mousemove	onmousemove
mouseout	onmouseout
mouseover	onmouseover
mouseup	onmouseup
reset	onreset
select	onselect
submit	onsubmit
unload	onunload

Event Attributes and their tags:

Attribute	Tag	Description
onblur	<a>	The link loses the input focus
	<button>	The button loses the input focus
	<input>	The input element loses the input focus
	<textarea>	The text area loses the input focus
	<select>	The selection element loses the input focus
onchange	<input>	The input element is changed and loses the input focus
	<textarea>	The text area is changed and loses the input focus
	<select>	The selection element is changed and loses the input focus
onclick	<a>	The user clicks on the link
	<input>	The input element is clicked
ondblclick	Most elements	The user double-clicks the left mouse button
onfocus	<a>	The link acquires the input focus
	<input>	The input element receives the input focus
	<textarea>	A text area receives the input focus
	<select>	A selection element receives the input focus
onkeydown	<body>, form elements	A key is pressed down

onkeypress	<body>, form elements	A key is pressed down and released
onkeyup	<body>, form elements	A key is released
onload	<body>	The document is finished loading
onmousedown	Most elements	The user clicks the left mouse button
onmousemove	Most elements	The user moves the mouse cursor within the element
onmouseout	Most elements	The mouse cursor is moved away from being over the element
onmouseover	Most elements	The mouse cursor is moved over the element
onmouseup	Most elements	The left mouse button is unclicked
onreset	<form>	The reset button is clicked

Handling events from body elements

```
<!DOCTYPE html>
<html>
<body onload="myFunction()">
<h1>Hello World!</h1>
<script>
function myFunction() {
 alert("Page is loaded");
}
</script>
</body>
</html>
```

- The above code loads alert window with alert message when the page is loaded or refreshed.
- Normally **onload** event can be used to check the visitor's browser type and version and load the proper version of webpage based on version, and also can be used to deal with cookies.

Similarly, the `unload` event is probably more useful than the `load` event. It is used to do some cleanup before a document is unloaded, as when the browser user goes on to some new document. For example, if the document opened a second browser window, that window could be closed by an `unload` event handler.

Handling Events from Button Elements

click.html

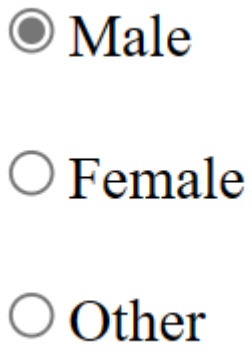
```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <title>Document</title>
   <script src="click.js"></script>
</head>
<body>
   <form id="myform">
      <input type="radio" name="gen" value="Male">Male
      <br><br>
      <input type="radio" name="gen" value="Female">Female
      <br><br>
      <input type="radio" name="gen" value="Other">Other
   </form>
   <script src="click1.js"></script>
</body>
</html>
```

```
function Gender(gen){
   var dom=document.getElementById("myform");
   for(var i=0;i<dom.gen.length;i++)
   {
      if(dom.gen[i].checked)
      {
         gen=dom.gen[i].value;
         break;
      }}
      switch(gen)
      {
         case "Male":
            alert("you are male");
            break;
            case "Female":
               alert("you are Female");
               break;
               case "Other":
                  alert("you are others");
                  break;
                  default:
                     alert("plz select appropriate gender");
      }}
```

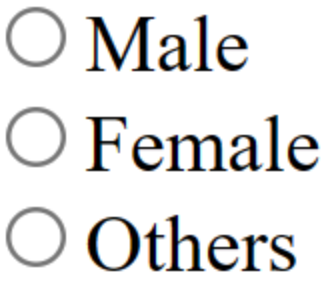
click.js

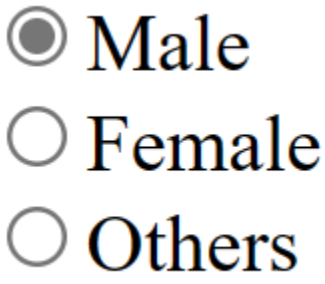
click1.js

```
var dom=document.getElementById("myform");
dom.elements[0].onclick=Gender;
dom.elements[1].onclick=Gender;
dom.elements[2].onclick=Gender;
```



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device
-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <input type="radio" name="gender" id=""
value="male" onclick="alert(this.value)">Male
    <br>
    <input type="radio" name="gender" id=""
value="female"
onclick="alert(this.value)">Female
    <br>
    <input type="radio" name="gender" id=""
value="other"
onclick="alert(this.value)">Others
</body>
</html>
```





```
<!DOCTYPE html>
<html lang="en">
<head>
    <title> Teacher </title>
    <meta charset="utf-8" />
    <!-- Script for the event handler -->
    <script type="text/javascript"
src="name.js">
    </script>
</head>
<body>
    <h4> You teacher name </h4>
    <form id="myForm" action="">
        <p>
            <label> <input type="radio"
name="tec" value="web" />
                Web Technology </label>
            <br />
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title> Teacher </title>
    <meta charset="utf-8" />
    <!-- Script for the event handler -->
    <script type="text/javascript"
src="name.js">
    </script>
</head>
<body>
    <h4> You teacher name </h4>
    <form id="myForm" action="">
        <p>
            <label> <input type="radio"
name="tec" value="web" />
                Web Technology </label>
            <br />
```

Name.Html

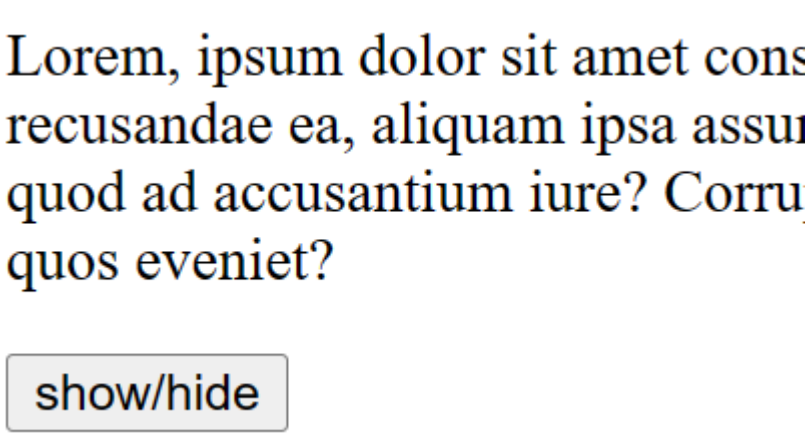
```
function Teacher(tec) {
    // Put the DOM address of the elements
array in a local variable
    var dom =
document.getElementById("myForm");
    // Determine which button was pressed
    for (var index = 0; index <
dom.tec.length;index++) {
        if (dom.tec[index].checked) {
            tec = dom.tec[index].value;
            break;
        }
    }
    // Produce an alert message about teacher
```

Name.js

```
var dom = document.getElementById("myForm");
dom.elements[0].onclick = Teacher;
dom.elements[1].onclick = Teacher;
dom.elements[2].onclick = Teacher;
dom.elements[3].onclick = Teacher;
```

Name1.
js

```
<p id="para">Lorem, ipsum dolor sit amet
    consectetur adipisicing elit. Blanditiis recusandae ea,
aliquam ipsa assumenda fugiat dolorem eaque in ex quod ad
accusantium iure? Corrupti, quidem.
    Incidunt autem quaerat quos eveniet?</p>
  <button id="btn" onclick=SH()>show/hide</button>
  <script>
    function SH(){
  var btn=document.getElementById("btn");
  var para=document.getElementById("para");
  if(para.style.display!='none')
  {
  para.style.display="none";
  }
  else{
    para.style.display="block";
  }}
  </script>
```



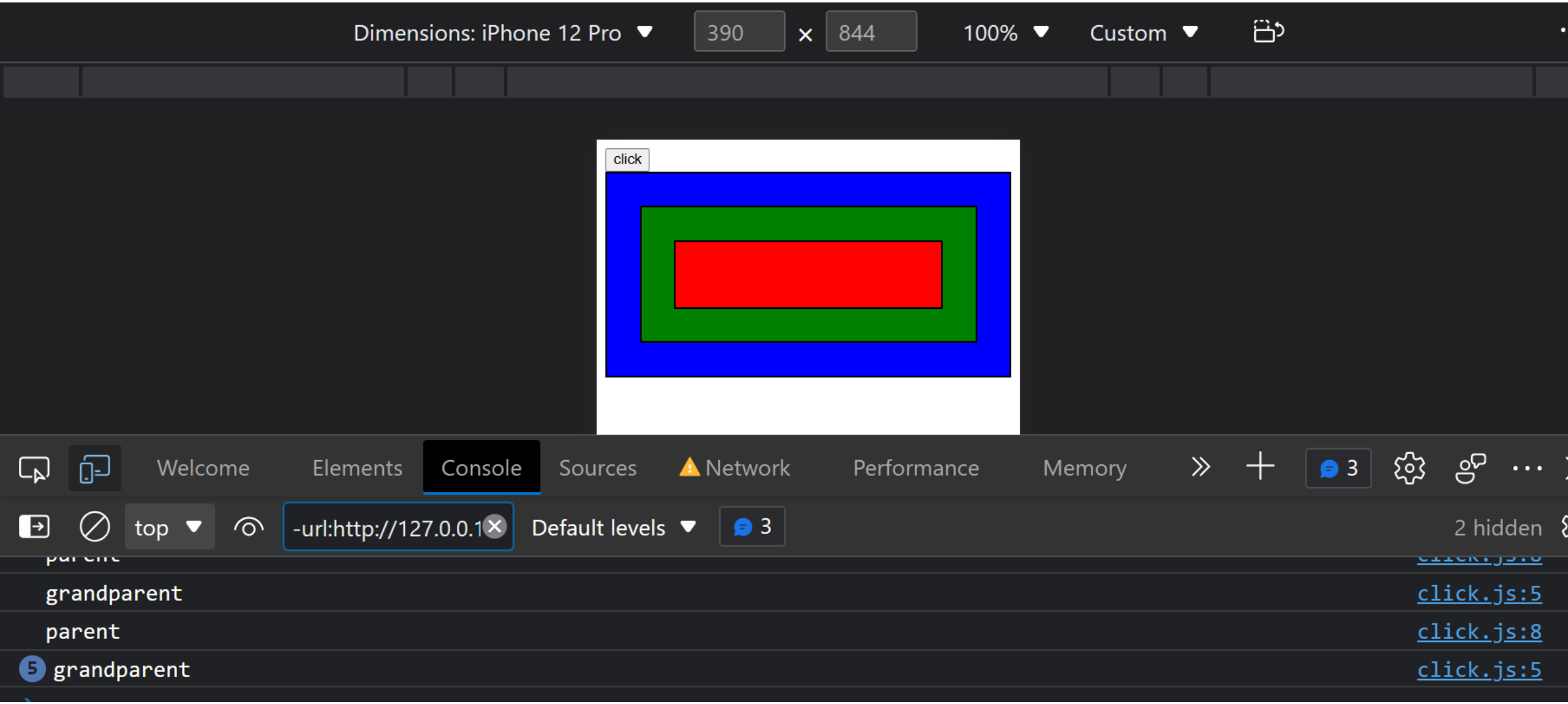
```
 <p id="para">Lorem ipsum, dolor sit amet
       consectetur adipisicing elit. Perspiciatis nostrum, rerum
       dolorum incidunt obcaecati laudantium fuga nulla
molestiae
        perferendis quos recusandae totam numquam quibusdam
beatae! Dolores veniam ratione maiores maxime.</p>

        <script>
          var para=document.getElementById("para");
           para.addEventListener('mouseover',function run()
           {

              console.log("mouse is here");
           })
           para.addEventListener('mouseout',function run()
           {

              console.log("mouse is outside");
           })
        </script>
```

Add and remove the listener
We have to use function reference not function on both side

```
let x= function(e){
  console.log(e);

    alert("hello 1");
}

btn.addEventListener('click',x);
btn.addEventListener('click',function(e){
    alert("hello 2")});
var a=Number(prompt("enter a number"));
{
    if(a==2){
        btn.removeEventListener('click',x)
    }
}
```

```
<style>
    #grandparent{
       margin:0;
       padding:30px;
       border:2px solid black;
       background-color: blue;

    }
    #parent{
       margin:0;
       padding:30px;
       border:2px solid black;
      background-color: green;

    }
    #child{
       margin:0;
       padding:30px;
       border:2px solid black;
       background-color: red;

    }
    </style>
</head>
<body>
    <button id="btn">click</button>


    <div id="grandparent">

        <div id="parent">
            <div id="child"></div>
        </div>
    </div>
         <script src="click.js">

         </script>
</body>
```

```
var grand=document.querySelector("#grandparent");
var parent=document.querySelector("#parent");
var child=document.querySelector("#child");
grand.addEventListener('click',e=>{
    console.log("grandparent")
})
parent.addEventListener('click',e=>{
    console.log("parent")
})
child.addEventListener('click',e=>{
    console.log("child")
})
```


Dimensions: iPhone 12 Pro
390
844
100%
Custom
click
Welcome
Elements
Console
Sources
Network
Performance
Memory
3
top
-url:http://127.0.0.1
Default levels
3
2 hidden
grandparent
click.js:5
parent
click.js:8
5
grandparent
click.js:5

Ondbclick when double click

```
<p ondblclick="he()">Hello</p>
        <script>
          function he(){
          alert("ohh");
         }
              </script>
```

Oncontextmenu when right key of touch pad is pressed

```
<p oncontextmenu="he()">Hello</p>
        <script>
          function he(){
          alert("ohh");
         }
              </script>
```

```
<p onmousedown="AB()">hello</p>
<p onmouseenter="AB()">Hy</p>
<p onmouseleave="AB()">Bye</p>
<p onmousemove="AB()">how are you</p>
<p onmouseout="AB()">good to see you</p>
<p onmouseover="AB()">Good day</p>
<p onmouseup="AB()">See ya</p>
<p onmousewheel="AB()"></p>
<script>
   function AB(){
      alert("we feel change")
   }
```

```
<body onkeypress="AB()">
    <button id="btn">click</button>

    <p >hello</p>

    <script>
        function AB(){
            alert("we feel change")
        }
```

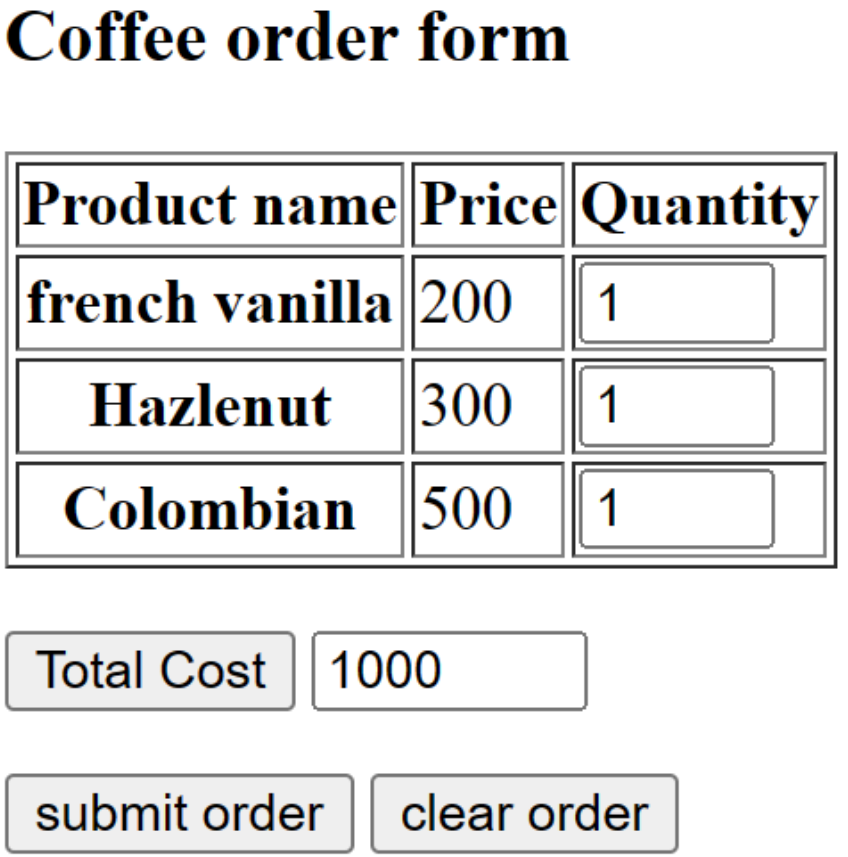
```
<body onresize="AB()">
<body onscroll="AB()">
```

Handling Events from Text Box and Password Elements:

- Text boxes and passwords can create four different events: `blur`, `focus`, `change`, and `select`.
- Suppose JavaScript is used to compute the total cost of an order and display it to the customer before the order is submitted to the server for processing.
- For example:

```
<body>
  <form action="">
    <h3>Coffee order form</h3>
    <table border="1">
      <tr>
        <th>Product name</th>
        <th>Price</th>
        <th>Quantity</th>
      </tr>

      <tr>
        <Th>french vanilla</Th>
        <td>200</td>
        <td>
          <input type="text" id="french" size="2">
        </td>
      </tr>
      <tr>
        <Th>Hazlenut</Th>
        <td>300</td>
        <td>
          <input type="text" id="Hazlenut" size="2">
        </td>
      </tr>
      <tr>
        <Th>Colombian</Th>
        <td>500</td>
        <td>
          <input type="text" id="Colombian" size="2">
        </td>
      </tr>
    </table>
    <p>
      <input type="button" id="total" value="Total Cost"
onclick="computecost()">
      <input type="text" size="5" id="cost“
onblur=“this.blur()”>
    </p>
    <p>
      <input type="submit" value="submit order">
      <input type="reset" value="clear order">
    </p>
  </form>
```

```
function computecost()
{
var french=document.getElementById("french").value;
var hazlenut=document.getElementById("Hazlenut").value;
var colombian=document.getElementById("Colombian").value;
document.getElementById("cost").value=
totalcost=french*200+hazlenut*300+ colombian*500;
}
```



In this example, the button labeled `Total Cost` allows the user to compute the total cost of the order before submitting the form.
The event handler for this button gets the values (input quantities) of the three kinds of coffee and computes the total cost.
The cost value is placed in the text box's value property, and it is then displayed for the user.
Whenever this text box acquires focus, it is forced to blur with the `blur` method, which prevents the user from changing the value.

```
<body>
   <h3>Electronic</h3>
   <form action="">
      <table border="1">
         <tr>
            <Th>Product Name</Th>
            <Th>Price </Th>
            <Th>Quantity</Th>
         </tr>
         <tr>
            <Th>Computer</Th>
            <td>10000</td>
            <td>
               <input type="number" size="2"
id="Computer">
            </td>
         </tr>
         <tr>
            <Th>Mobile</Th>
            <td>1000</td>
            <td>
<input type="number" size="2" id="Mobile">
            </td>
         </tr>
         <tr>
            <Th>Headphone</Th>
            <td>2000</td>
            <td>
               <input type="number" size="2" id="Headphone">
            </td>
         </tr>

      </table>
      <input type="button" value="Total"
onclick="computecost()">
      <input type="text" id="cost">
      <p>
         <input type="submit"  >
         <input type="reset">
      </p>
   </form>
   <script src="click.js"></script>
</body>
```

```
function computecost()
{
   var cmp=document.getElementById("Computer").value;
   var mbl=document.getElementById("Mobile").value;
   var hdphn=document.getElementById("Headphone").value;
   document.getElementById("cost").value=
   total=cmp*10000+mbl*1000+hdphn*2000;
}
```

Electronic

Product Name	Price	Quantity
Computer	10000	1
Mobile	1000	1
Headphone	2000	1

Total 13000

Submit Reset

Validating Form Input:

- JavaScript can be used to prevent the form being submitted by checking the empty form field or input data validation.
- Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

- has the user filled in all **required** fields?
- has the user entered a valid date?
- has the user entered text in a numeric field?
- Has the user entered correct email?,
- Match the two passwords etc.

- Validation can be defined by many different methods, and deployed in many different ways. The most common ways of data validation in HTML form are:
  - **Server side validation**: It is performed by a web server, after input has been sent to the server.
  - **Client side validation:** It is performed by a web browser, before input is sent to a web server. JavaScript is used for client-side validation.

Important Advantages and Disadvantages of Server and Client Side Validation:

Client-side Validation:

Advantages:

- Allow for more interactivity by immediately responding to users' actions at browser level.
- Execute quickly because they do not require a trip to the server that results in less network traffic.

Disadvantages:

- Client-side Validation can be easily bypassed by different techniques, so can never be most reliable technique.

Server-side Validation:

Advantages:

- Most often server side validation can **protect against the malicious user** from putting an invalid data which may corrupt database.
- Server-side validation is important for compatibility.

Note: Most often both client-side validation and server-side validation must be implemented for crucial data input. In some cases, Database validation can also be done if server-side validation fails.

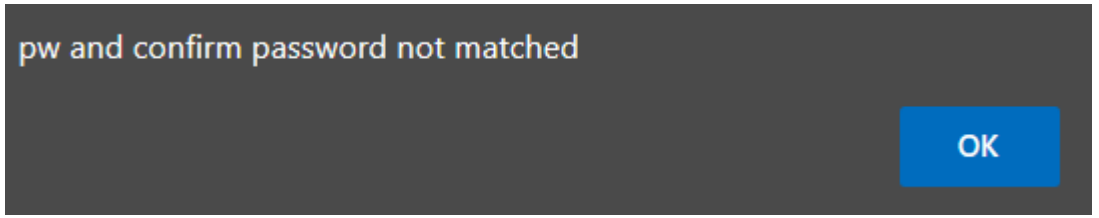
```
<h2>JavaScript Validation</h2>
<form name="myForm" action="" onsubmit="return
validateForm()" method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>
```


```
function validateForm() {
    let x = document.forms["myForm"]["fname"].value;
    if (x == "") {
      alert("Name must be filled out");
      return false;
    }
  }
```

```
 <script src="click.js"></script>
</head>
<body>

   <form  id="myform" onsubmit="return validateForm()">
      password:<input type="password" name="First">
      <br><br>
      Confirm password<input type="password"
name="Second">
      <br><br>
      <input type="submit" value="submit">

      <input type="reset">
   </form>
```

```
function validateForm(){
   let x=document.forms["myform"]["First"].value;
   let y=document.forms["myform"]["Second"].value;
    if(x == "")
    {
       alert("plz fill out");
       return false;
    }
    if(x!=y)
    {
       alert("pw and confirm password not matched");
    }
}
```

password: Confirm password submit Reset



Another Approach

```
<script src="click.js"></script>
</head>
<body>

  <form  id="myform">
     password:<input type="Password" id="First">
     <br><br>
     Confirm password<input type="password" id="Second">
     <br><br>
     <input type="submit" value="submit">

     <input type="reset">
  </form>
  <script src="click1.js"></script>
```

click1.js

```
document.getElementById("First").onblur=check;
document.getElementById("Second").onblur=check;
document.getElementById("myform").onsubmit=check;
```

Click.js

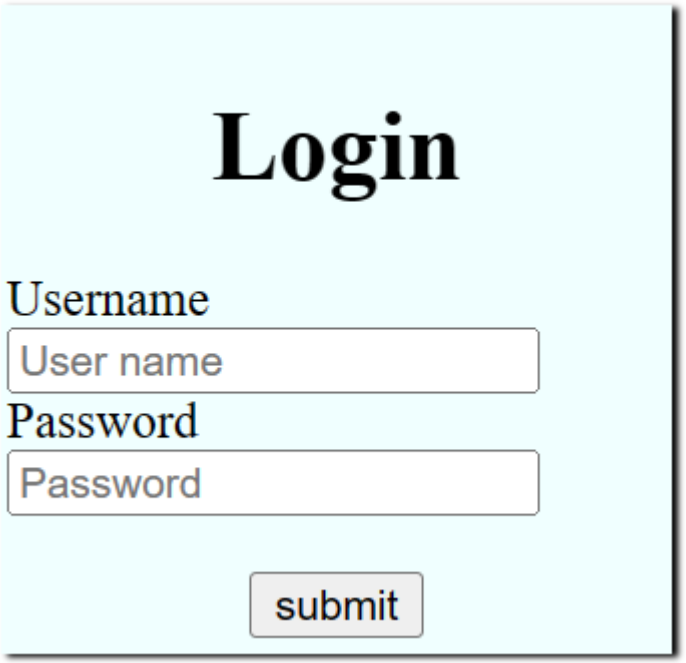
```
function check(){
var F=document.getElementById("First").value;
var S=document.getElementById("Second").value;
if(F == "")
{
   alert("plz fill out");
   return false;
}
if(F!=S){
   alert("PW and Confirm PW not match");
   return false;
}}
```

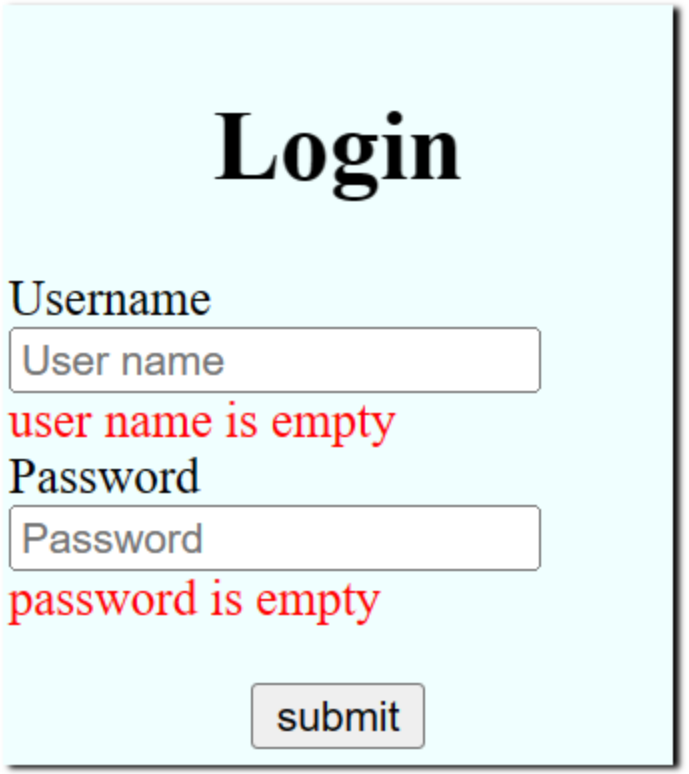
click.js

```
<style>
#myform{
   margin:50px;
   box-shadow: 2px 1px 2px;
   background-color: azure;
   padding: 5px 2px;
   width: 50%;
}
.error
{
   color: red;
   margin:0px;
}
   </style>
</head>
<body>
```

```
<form action="" id="myform" onsubmit="return validate()">
  <h1 style="text-align: center;"> Login</h1>
   <div>
     <label for="User">Username</label>
     <br>
      <input type="text" id="User" placeholder="User name">
      <p id="Usererror" class="error"></p>

   </div>
   <div>
                    <label for="User">Password</label>
                    <br>
                     <input type="password"
id="Password" placeholder="Password">
                     <p id="Passworderror"
class="error"></p>
            </div>
            <br>
          <center> <input type="submit"
id="submit" value="submit"></center>
    </form>
    <script src="click.js"></script>
</body>
</html>
```

```
var User=document.getElementById("User");
var
Password=document.getElementById("Password");
let flag=1
function validate()
{
    if(User.value == "" || User.value ==null)
    {

document.getElementById("Usererror").innerHTML=
"user name is empty";
       flag=0;
    }
    else if(User.value.length < 2)
    {

document.getElementById("Usererror").innerHTML=
"Length shouldnt match";
       flag=0;
    }
   else{

document.getElementById("Usererror").innerHTML=
"";
    flag=1;
   }
    if(Password.value == "")
    {

document.getElementById("Passworderror").innerHTML="
password is empty";
        flag=0;
    }
    else if(Password.value.length<=3){

document.getElementById("Passworderror").innerHTML="
Length shouldnt match";
        flag=0;
    }
    else{

document.getElementById("Passworderror").innerHTML="
";

    }
       if(flag)
       {
        return true;
       }
   else{
    return false;
   }
}
```


Login
Username
User name
Password
Password
submit


Login
Username
User name
user name is empty
Password
Password
password is empty
submit


Login
Username
1
Length shouldnt match
Password
•••
Length shouldnt match
submit

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible"
content="IE=edge">
    <meta name="viewport" content="width=device-
width, initial-scale=1.0">
    <title>Form Validation</title>

    <link rel="stylesheet" href="index.css">
    <script  src="index.js"></script>
</head>
<body>
    <div class="container">
        <form id="form" >
            <h1>Registration</h1>
            <div class="input-control">
                <label
for="username">Username</label>
                <input id="username" name="username"
type="text">
                <div class="error"></div>
            </div>
            <div class="input-control">
                <label
for="email">Email</label>
                <input id="email" name="email"
type="text">
                <div class="error"></div>
            </div>
            <div class="input-control">
                <label
for="password">Password</label>
                <input id="password"name="password"
type="password">
                <div class="error"></div>
            </div>
            <div class="input-control">
                <label for="password2">Password
again</label>
                <input
id="password2"name="password2" type="password">
                <div class="error"></div>
            </div>
            <button type="submit">Sign Up</button>
        </form>
    </div>
</body>
</html>
```

```
body {
    background: linear-gradient(to right, #0f2027, #203a43, #2c5364);
    font-family: 'Poppins', sans-serif;
}
#form {
    width: 300px;
    margin: 20vh auto 0 auto;
    padding: 20px;
    background-color: WhiteSmoke;
    border-radius: 4px;
    font-size: 12px;
}
#form h1 {
    color: #0f2027;
    text-align: center;
}
#form button {
    padding: 10px;
    margin-top: 10px;
    width: 100%;
    color: white;
    background-color: rgb(41, 57, 194);
    border: none;
    border-radius: 4px;
}
```

```
.input-control {
    display: flex;
    flex-direction: column;
}

.input-control input {
    border: 2px solid #f0f0f0;
            border-radius: 4px;
            display: block;
            font-size: 12px;
            padding: 10px;
            width: 100%;
}
```

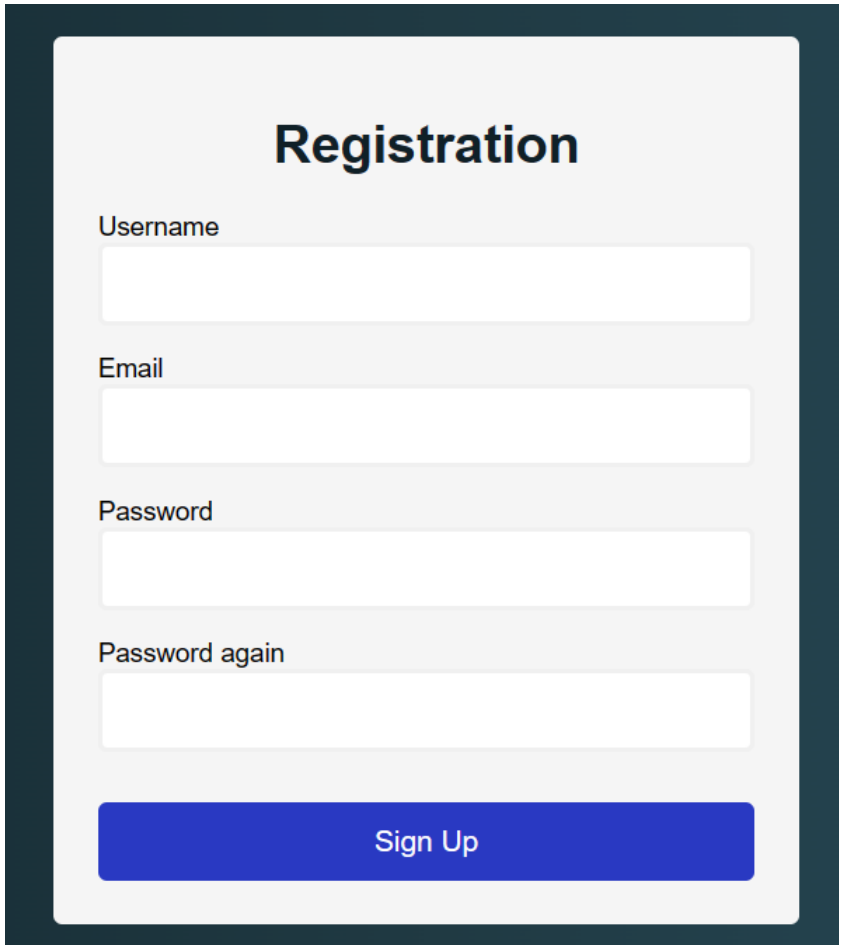
```
.input-control input:focus {
    outline: 0;
}
.input-control.success input {
    border-color: #09c372;
}
.input-control.error input {
    border-color: #ff3860;
}
.input-control .error {
    color: #ff3860;
    font-size: 9px;
    height: 13px;
}
```

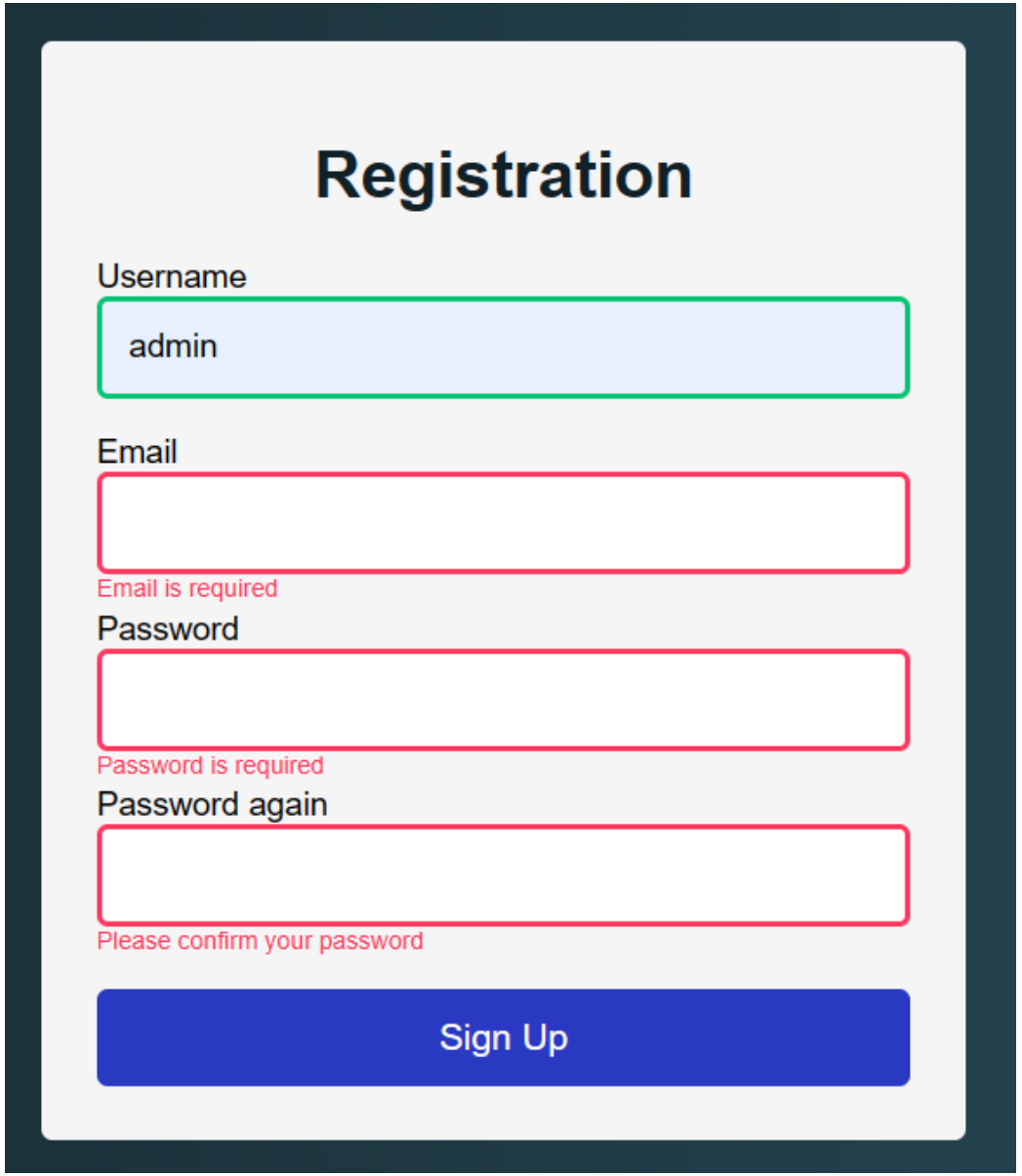
```
const form = document.getElementById('form');
const username = document.getElementById('username');
const email = document.getElementById('email');
const password = document.getElementById('password');
const password2 = document.getElementById('password2');

form.addEventListener('submit', e => {
    e.preventDefault();

    validateInputs();
});

const setError = (element, message) => {
    const inputControl = element.parentElement;
    const errorDisplay = inputControl.querySelector('.error');

    errorDisplay.innerText = message;
    inputControl.classList.add('error');
    inputControl.classList.remove('success')
}
```

```
const setSuccess = element => {
    const inputControl = element.parentElement;
    const errorDisplay = inputControl.querySelector('.error');

    errorDisplay.innerText = '';
    inputControl.classList.add('success');
    inputControl.classList.remove('error');
};

const isValidEmail = email => {
    const re =
/^(([^<>()[\]\\.,;:\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,}))$/;
    return re.test(String(email).toLowerCase());
}
```

```
const validateInputs = () => {
    const usernameValue = username.value.trim();
    const emailValue = email.value.trim();
    const passwordValue = password.value.trim();
    const password2Value = password2.value.trim();

    if(usernameValue === '') {
        setError(username, 'Username is required');
    } else {
        setSuccess(username);
    }

    if(emailValue === '') {
        setError(email, 'Email is required');
    } else if (!isValidEmail(emailValue)) {
        setError(email, 'Provide a valid email address');
    } else {
        setSuccess(email);
    }
```

```
    if(passwordValue === '') {
        setError(password, 'Password is required');
    } else if (passwordValue.length < 8 ) {
        setError(password, 'Password must be at least 8
character.')
    } else {
        setSuccess(password);
    }

    if(password2Value === '') {
        setError(password2, 'Please confirm your password');
    } else if (password2Value !== passwordValue) {
        setError(password2, "Passwords doesn't match");
    } else {
        setSuccess(password2);
    }

};
```


Registration
Username
Email
Password
Password again
Sign Up


Registration
Username
admin
Email
Email is required
Password
Password is required
Password again
Please confirm your password
Sign Up

Positioning Elements:

- Mostly, the elements found in the HTML file were simply placed in the document the way text is placed in a document with a word processor: Fill a row, start a new row, fill it, and so forth.
- HTML tables provide a framework of columns for arranging elements, but they lack flexibility and also take a considerable time to display.
- Cascading Style Sheets–Positioning (CSS-P) provides the means not only to position any element anywhere in the display of a document, but also to move an element to a new position in the display dynamically, using JavaScript to change the positioning style properties of the element.
- The style properties, `left` and `top`, dictate the distance from the left and top of some reference point to where the element is to appear.
- Another style property, `position`, interacts with `left` and `top` to provide a higher level of control of placement and movement of elements.
- The `position` property has three possible values: `absolute`, `relative`, and `static`.

Absolute Positioning:

- The `absolute` value is specified for `position` when the element is to be placed at a specific place in the document display without regard to the positions of other elements.
- For example, if a paragraph of text is to appear 100 pixels from the left edge and 200 pixels from the top of the display window, the following element could be used:

```
<p style = "position: absolute; left: 100px; top: 200px">
    -- text --
</p>
```

- One use of absolute positioning is to superimpose special text over a paragraph of ordinary text to create an effect similar to a watermark on paper.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible"
content="IE=edge">
    <meta name="viewport" content="width=device
-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .abstext{
            position:absolute;
            top: 25px;
            font-size: 100px;
        }
    </style>
</head>
<body>
```

```
<p class="regtext">
        Lorem ipsum dolor sit amet consectetur
adipisicing elit. Provident quam delectus
consequatur nesciunt facere, iusto velit
quisquam veritatis eius? Iure excepturi
doloremque
Harum consectetur nisi vitae nemo optio culpa
doloribus voluptas dicta asperiores commodi?
Consectetur fugit voluptates id perspiciatis,
distinctio impedit cum illum doloribus
aspernatur odit sed quod, assumenda facilis eum
ducimus.
         Quo quaerat corrupti ullam voluptate
cumque adipisci maxime eaque provident rem,
assumenda        </p>
        <p class="abstext">
            i m using absolute position
        </p>
</body>
</html>
```

i musng absolute positon

Lorem ipsum dolor sit amet consectetur adipisicing elit. Provident quam delectus consequatur nesciunt facere, iusto velit quisquam veritatis eius? Iure excepturi doloremque a enim nesciunt pariatur maiores, laudantium fugiat veniam. Lorem ipsum dolor sit, amet consectetur adipisicing elit. Hic provident at magnam nisi dolorem? Esse nemo nisi eligendi iste, voluptates facilis, suscipit distinctio soluta quaerat veniam quo neque rem animi! Eos harum voluptate blanditiis sunt alias illo distinctio consectetur, quis esse. Molestias hic ipsa corporis totam quod facere officia incidunt sequi iure quas? In officia corporis nihil, quisquam sed unde. Deleniti modi omnis tempore repellendus est inventore obcaecati voluptatibus dolorem eum vitae, quis, libero impedit, qui consectetur? Non quod suscipit commodi numquam architecto fuga amet. Eum corrupti enim ducimus itaque! Culpa tenetur aliquam, cum voluptatem modi a quisquam corrupti odio reiciendis voluptatibus totam ducimus unde repellendus nisi explicabo, debitis, consequuntur mollitia saepe assumenda et? Voluptatibus sit ab pariatur porro assumenda. Aliquam molestiae ipsum suscipit a est officiis placeat unde omnis, necessitatibus officia quo debitis dolore iure rem magnam consectetur at, accusamus eveniet nostrum voluptate minus voluptas enim? Repudiandae, placeat voluptatibus? Id unde veniam neque delectus consequuntur error temporibus, est veritatis tempora deserunt odit quam expedita adipisci blanditiis quos libero ut ullam labore mollitia! Nesciunt dolorum iure iusto nemo perspiciatis labore. Quas perspiciatis dolor cupiditate tempora tenetur blanditiis unde velit alias sapiente aut ea dignissimos, veniam est vitae architecto non voluptas iure? Modi, doloremque dicta sed placeat voluptatem totam nulla magnam. Harum consectetur nisi vitae nemo optio culpa doloribus voluptas dicta asperiores commodi? Consectetur fugit voluptates id perspiciatis, distinctio impedit cum illum doloribus aspernatur odit sed quod, assumenda facilis eum ducimus. Quo quaerat corrupti ullam voluptate cumque adipisci maxime eaque provident rem, assumenda reprehenderit laudantium, debitis hic, fugit dignissimos et. Fuga quae ducimus similique laudantium qui? Facere corrupti sunt id aliquam. Nihil architecto id quibusdam rerum, ducimus aut eaque. Necessitatibus saepe corrupti in nesciunt sint vero expedita nihil culpa, reiciendis delectus qui accusantium natus ab repellendus quis? Harum consequuntur culpa vero.

Relative Positioning:

- An element that has the `position` property set to `relative`, but does not specify `top` and `left` property values, is placed in the document as if the `position` attribute were not set at all.
- For example, suppose that two buttons are placed in a document and the `position` attribute has its default value, which is `static`. Then the buttons would appear next to each other in a row, assuming that the current row has sufficient horizontal space for them.
- If `position` has been set to `relative` and the second button has its `left` property set to `50px`, the effect would be to move the second button 50 pixels farther to the right than it otherwise would have appeared.
- Relative positioning can be used for a variety of special effects in placing elements.
- For example, it can be used to create superscripts and subscripts by placing the values to be raised or lowered in `<span>` tags and displacing them from their regular positions.
- In the next example, a line of text is set in a normal font style in 24-point size. Embedded in the line is one word that is set in italic, 48-point, red font.
- Normally, the bottom of the special word would align with the bottom of the rest of the line.
- In this case, the special word is to be vertically centered in the line, so its `position` property is set to `relative` and its `top` property is set to 10 pixels, which lowers it by that amount relative to the surrounding text. The XHTML document to specify this, which is named `relPos.html`, is as follows:

Static Positioning:

- The default value for the `position` property is `static`. With `position: static`, the document moves along with the page. A statically positioned element is placed in the document as if it had the `position` value of `relative` but no values for `top` or `left` were given.
- The difference is that a statically positioned element cannot have its `top` or `left` properties initially set or changed later.
- Therefore, a statically placed element cannot be displaced from its normal position and cannot be moved from that position later.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <title>Document</title>
   <style>

      .abstext{
         position:absolute;
         top: 0px;
         left:0px;
      }
```

change position to static(default),absolute and relative to check

```
      .parent{
          top: 100px;
          font-size: 100px;
          padding: 20px;

          background-color: red;
      }
      .A{
          padding: 20px;
          border:  2px solid black;
          background-color: aqua;
      }
    </style>
</head>
<body>
    <div class="parent">
        <div class="A A">1</div>
        <div class="A B">2</div>
        <div class=" A abstext">3</div>
        <div class="A C">4</div>
        <div class="A D">5</div>
    </div>
</body>
</html>
```