JavaScript

Trainer: Muralidharan.R
Phone: 9894868015

Learning Speed

- As you have already completed HTML, CSS, Bootstrap. You are going to learn Javascript which is a high-level language.
- Everything is up to you.
- If you are struggling, take a break, or re-read the material.
- The only way to become a clever programmer is to: Practice. Practice. Practice. Code. Code. Code !

Introduction to Procedural Programming

- Procedural programming is derived from imperative programming.
- Its concept is based on procedure calls. Procedures are nothing but a series of computational steps to be carried out.
- In Procedural programming, the execution of the instructions takes place step by step.
- When you use Procedural language, you give instructions directly to your computer and tell it how to reach its goal through processes.
- Procedural programming focuses on the process rather than data (Object-oriented Programming) and function (Functional Programming).
- The first major procedural programming languages around 1957-1964 were FORTRAN, ALGOL, COBOL, PL/I and BASIC. The procedural Programming paradigm is one of the first paradigms to emerge in the computational world.

Types of Procedural Languages

- FORTRAN(Formula Translation Language)
- ALGOL(Algorithmic Language)
- COBOL(Common Business Oriented Language)
- BASIC(Beginner’s All Purpose Symbolic Instruction Code)
- PASCAL
- C

Advantages of Procedural Programming

- The programs written are straight forward with precise usage of accumulators and interpreters.
- The code is compact and reusable.
- It breaks problems into smaller subproblems, making them accessible and faster to solve.
- It expands the renewable energy of the program.
- It utilises CPU storage effectively.
- It is more flexible than other alternatives.

Disadvantages of Procedural Programming

- The procedural program is not recyclable.
- Information is defenceless.
- The information is accessible from the whole code resulting in safety issues.
- It is event-driven programming, not portable to other operating systems.
- Programmers need to specialise as each language is suitable for a specific type of application.

Features of Procedural Programming

1. **Pre-defined Functions**: It is an instruction identified by name. Such as “charAt()” is a pre-defined function that searches for a character in a string at a specific position. There are more pre-defined functions that make competitive programming a little easier.
2. **Local Variables**: Local variables are declared in the main structure of a method. You will only be able to access the local variable within the method.
3. **Global Variables**: They are declared outside all methods to be accessible from anywhere in the code.
4. **Programming libraries**: A programming library is a collection of code written previously to utilise whenever a programmer requires it.
5. **Modularity:** It is a general term that relates to the creation of software in a way that allows individual modules to be developed, often with a standard interface to let modules communicate with each other.

HTML,CSS and Javascript

HTML:

- HTML is at the core of every web page, regardless the complexity of a site or number of technologies involved. It's an essential skill for any web professional.
- It's the starting point for anyone learning how to create content for the web. And, luckily for us, it's surprisingly easy to learn.
- Once a tag has been opened, all of the content that follows is assumed to be part of that tag until you "close" the tag.
- When the paragraph ends, I'd put a closing paragraph tag: </p>. Notice that closing tags look exactly the same as opening tags, except there is a forward slash after the left angle bracket.

Here's an example: <p>This is a paragraph.</p>

- Using HTML, you can add headings, format paragraphs, control line breaks, make lists, emphasize text, create special characters, insert images, create links, build tables, control some styling, and much more.

HTML,CSS and Javascript

CSS

- CSS stands for Cascading Style Sheets.
- This programming language dictates how the HTML elements of a website should actually appear on the frontend of the page.
- If HTML is the drywall, CSS is the paint.

Javascript

- JavaScript is a more complicated language than HTML or CSS, and it wasn't released in beta form until 1995. Nowadays, JavaScript is supported by all modern web browsers and is used on almost every site on the web for more powerful and complex functionality.
- In short, JavaScript is a programming language that lets web developers design interactive sites. Most of the dynamic behavior you'll see on a web page is thanks to JavaScript, which augments a browser's default controls and behaviors.

Best Approaches To Learn Javascript Editors

1. Visual Studio: an industry-standard software
2. Visual Studio Code:A free version: lighter on the features; high in value
3. Webstorm:A premium JavaScript development environment
4. Atom:A strong contender for the best free JavaScript IDE
5. Brackets: An Adobe JavaScript IDE for free. Impossible!
6. Komodo IDE:Another one of the top JavaScript IDEs
7. Komodo Edit:Free text editor based on Komodo IDE

IDE - Integrated Development Environment

An integrated development environment is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of at least a source code editor, build automation tools and a debugger. Wikipedia

Commonly Asked Questions

- How do I get JavaScript? Where can I download JavaScript?
- Is JavaScript Free?
- **You don't have to get or download JavaScript.**
- **JavaScript is already running in your browser on your computer, on your tablet, and on your smart-phone. JavaScript is free to use for everyone.**
- Did You Know that JavaScript and Java are completely different languages, both in concept and design.
- JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.
- ECMA-262 is the official name of the standard. ECMAScript is the official name of the language.
-

Introduction to JavaScript

JavaScript History and Features

History

- JavaScript was created in 1995 by Brendan Eich at Netscape. Here's a brief timeline:
- 1995: Initially named Mocha.
- 1996: Renamed to LiveScript and then JavaScript.
- 1997: Standardized as ECMAScript (ES).
- Present: Continuously evolving with newer features (e.g., ES6 in 2015).

Features

- Lightweight and Interpreted: Runs directly in the browser without requiring compilation.
- Dynamic Typing: Variable types are determined at runtime.
- First-Class Functions: Functions can be assigned to variables, passed as arguments, or returned from other functions.
- Cross-Platform: Runs in browsers and on servers (e.g., Node.js).

Javascript Versions

Till date, ES has published nine versions and the latest one (12th version) was published in the year 2020.

1. ES1 1997
2. ES3 1999
3. ES5 2009
4. ES7 2016
5. ES9 2018
6. ES2 1998
7. ES4 Abandoned
8. ES6 2015
9. ES8 2017
10. ES8 2018
11. ES8 2019
12. ES8 2020

ECMA (European Computer Manufacturers Association) Script's first three versions- ES1, ES2, ES3 were yearly updates whereas, ES4 was never released due to political disagreements. After a decade, ES5 was eventually released with several additions.

Types of JavaScript

JavaScript is categorized into two main types:

1. Client-Side JavaScript (CSJS)

Executes in the browser.

Handles user interaction, DOM manipulation, and events.

Example:

```
document.getElementById("demo").innerHTML = "Hello, Client-Side JS!";
```

2. Server-Side JavaScript (SSJS)

Executes on a server, commonly using Node.js.

Handles backend logic, API creation, and database interaction.

Example:

```
const http = require("http");
http.createServer((req, res) => {
  res.write("Hello, Server-Side JS!");
  res.end();
}).listen(8080);
```

JavaScript Introduction

The <script> Tag

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

Example

```
<script>

document.getElementById("demo").innerHTML = "My First JavaScript";

</script>
```

Old JavaScript examples may use a type attribute: <script type="text/javascript">.

The type attribute is not required. JavaScript is the default scripting language in HTML.

JavaScript Functions and Events

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

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

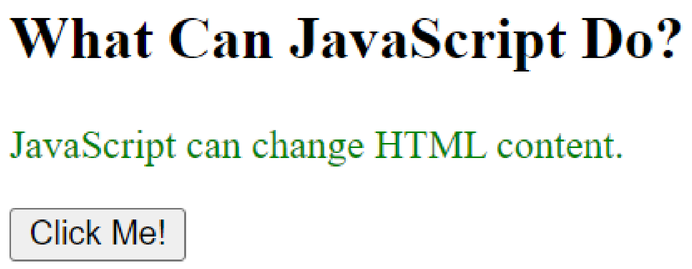
You can place any number of scripts in an HTML document.

Placement of Javascript Code

JavaScript in body or head: Scripts can be placed inside the body or the head section of an HTML page or inside both head and body.

JavaScript in head: A JavaScript function is placed inside the head section of an HTML page and the function is invoked when a button is clicked.

```
<!DOCTYPE html>
<html>
<head> <script>
function hello()
{  document.getElementById("demo").innerHTML = "Hello World!";  }
</script> </head>
<body>
<h2>What Can JavaScript Do?</h2>
<p id="demo" style="color:green;">JavaScript can change HTML content.</p>
<button type="button" onclick="hello()">Click Me!</button>
</body> </html>
```



What Can JavaScript Do?

JavaScript can change HTML content.

Javascript - External

```
<!DOCTYPE html>
<html>
<head>
<script src = "sample.js">
</script>
</head>
<body>
<h2>Use Javascript in the Body</h2>
<p id="demo">JavaScript can change HTML content.</p>
<button type="button" onclick="hello()">Click Me!</button>
</body>
</html>
```

sample.js

```
function hello() {
document.getElementById("demo").innerHTML = "Hello JavaScript!";
}
```

External JavaScript Advantages

Placing scripts in external files has some advantages:

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

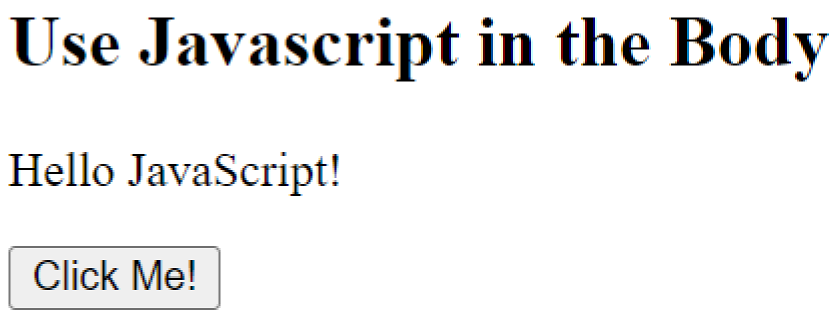
External References

An external script can be referenced in 3 different ways:

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

Javascript in Body

```
<!DOCTYPE html>
<html>
<body>
<h2>Use Javascript in the Body</h2>
<p id="demo">JavaScript can change HTML content.</p>
<button type="button" onclick="document.getElementById('demo').innerHTML = 'Hello
World!'">Click Me!</button>
</body>
</html>
```


```
Use Javascript in the Body

Hello JavaScript!
```

JavaScript Can Change

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is getElementById().

The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript":

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

JavaScript accepts both double and single quotes:

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

JavaScript Can Change HTML Attributes

In this example JavaScript changes the value of the src (source) attribute of an <img> tag:

JavaScript Can Change HTML Attribute Values

In this example JavaScript changes the value of the src (source) attribute of an <img> tag:

```
<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">Turn on the light</button>
<img id="myImage" src="pic_bulboff.gif" style="width:100px">
<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Turn off the light</button>
```

JavaScript Can Change

JavaScript Can Hide HTML Elements

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

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

JavaScript Can Change HTML Styles (CSS)

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

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

JavaScript Can Show HTML Elements

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

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

MCQ

MULTIPLE CHOICE QUESTION

JavaScript

Procedural language contains systematic order of

A. Statements
B. Object
C. classes
D. operations

JavaScript

A type of computer programming language that specifies a series of well-structured steps and procedures within its programming context to compose a program is known as.

A. Procedural Language

B. Structural Language

C. Assembly Language

D. Machine Language

JavaScript

Choose the correct JavaScript syntax to change the content of the following HTML code.

A. document.getElement ("letsfindcourse").innerHTML = "I am a letsfindcourse";

B. document.getElementById ("letsfindcourse").innerHTML = "I am a letsfindcourse";

C. document.getId ("letsfindcourse") = "I am a letsfindcourse";

D. document.getElementById ("letsfindcourse").innerHTML = I am a letsfindcourse;

JavaScript

Which of the following are advantages of JavaScript?

A. Less server interaction

B. Increased interactivity

C. Richer interfaces

D. All of the above

JavaScript

Which of the following true about Javascript?

A. Client-side JavaScript does not allow the reading or writing of files
B. JavaScript cannot be used for networking applications
C. JavaScript doesn't have any multi-threading or multiprocessor capabilities
D. All of the above

Javascript Output Methods

JavaScript can "display" data in different ways:

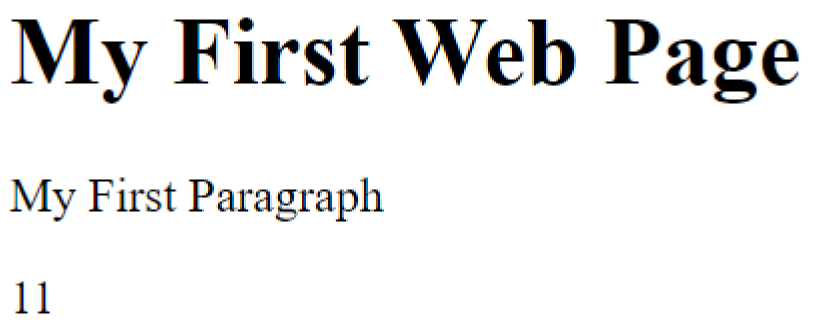
- Writing into an HTML element, using innerHTML.
- Writing into the HTML output using document.write().
- Writing into an alert box, using window.alert().
- Writing into the browser console, using console.log().

Using innerHTML

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

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

```
<html>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

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

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```



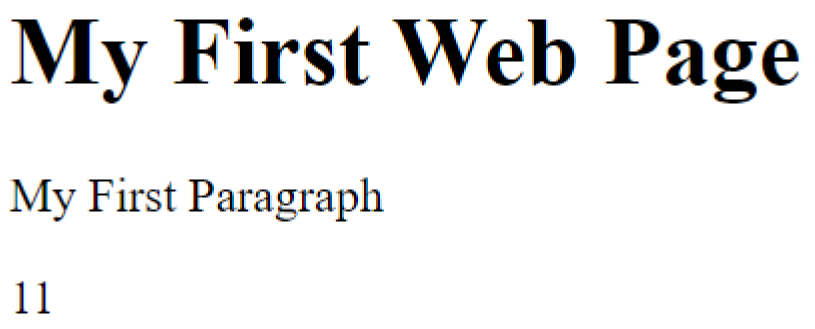
My First Web Page

My First Paragraph

11

Using document.write()

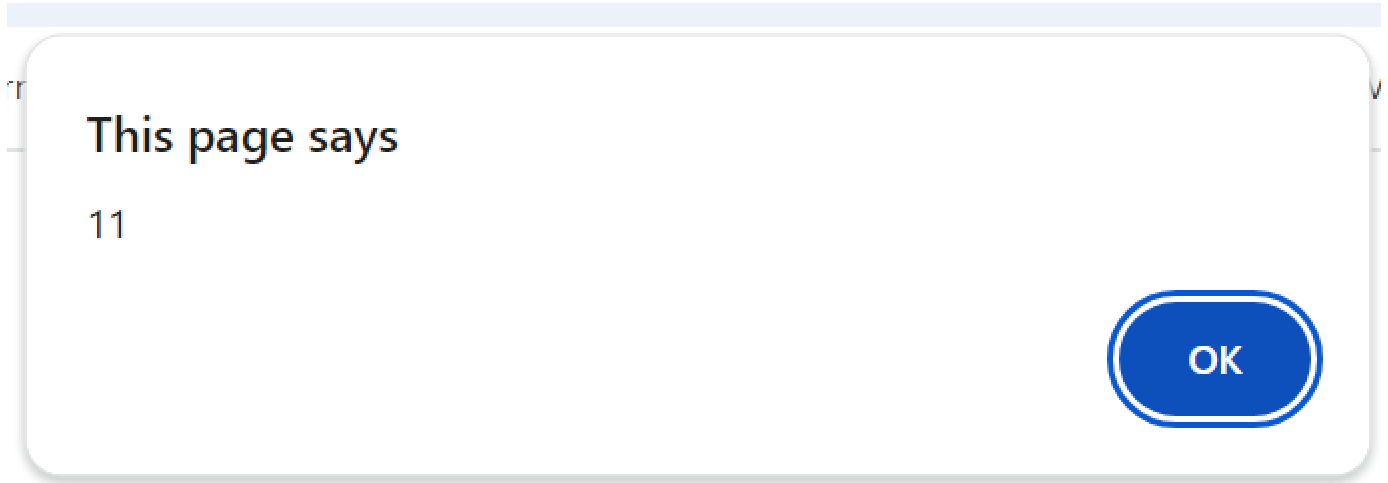
```
<html>
<body>

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

<script>
document.write(5 + 6);
</script>

</body>
</html>
```



```
My First Web Page

My First Paragraph

11
```

Using window.alert()

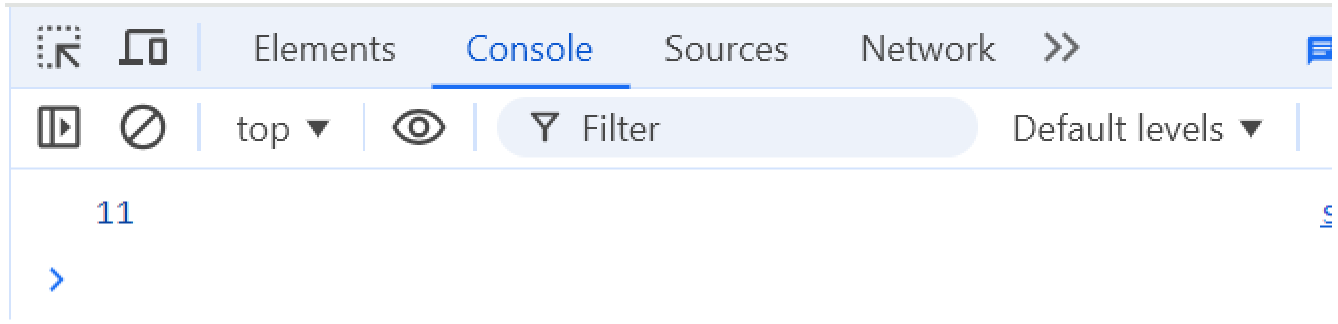
```
<html>
<body>

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

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```


This page says

11

Using console.log()

```
<html>
<body>

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

</body>
</html>
```



```
11
```

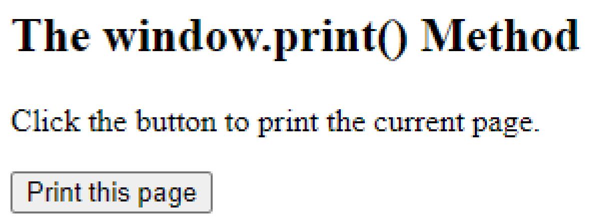
JavaScript Print

JavaScript does not have any print object or print methods.

You cannot access output devices from JavaScript.

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

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

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

</body>
</html>
```


The window.print() Method

Click the button to print the current page.

JavaScript Statements

JavaScript Programs
A computer program is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called **statements.**

A JavaScript program is a list of programming statements.

In HTML, JavaScript programs are executed by the web browser.

JavaScript Statements
JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

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

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

Most JavaScript programs contain many JavaScript statements.

The statements are executed, one by one, in the same order as they are written.

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

JavaScript Statements

Semicolons ;

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement:

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

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

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

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

JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

The following lines are equivalent:

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

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

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

JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

Example

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

JavaScript Statements

JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.
The purpose of code blocks is to define statements to be executed together.
One place you will find statements grouped together in blocks, is in JavaScript functions:

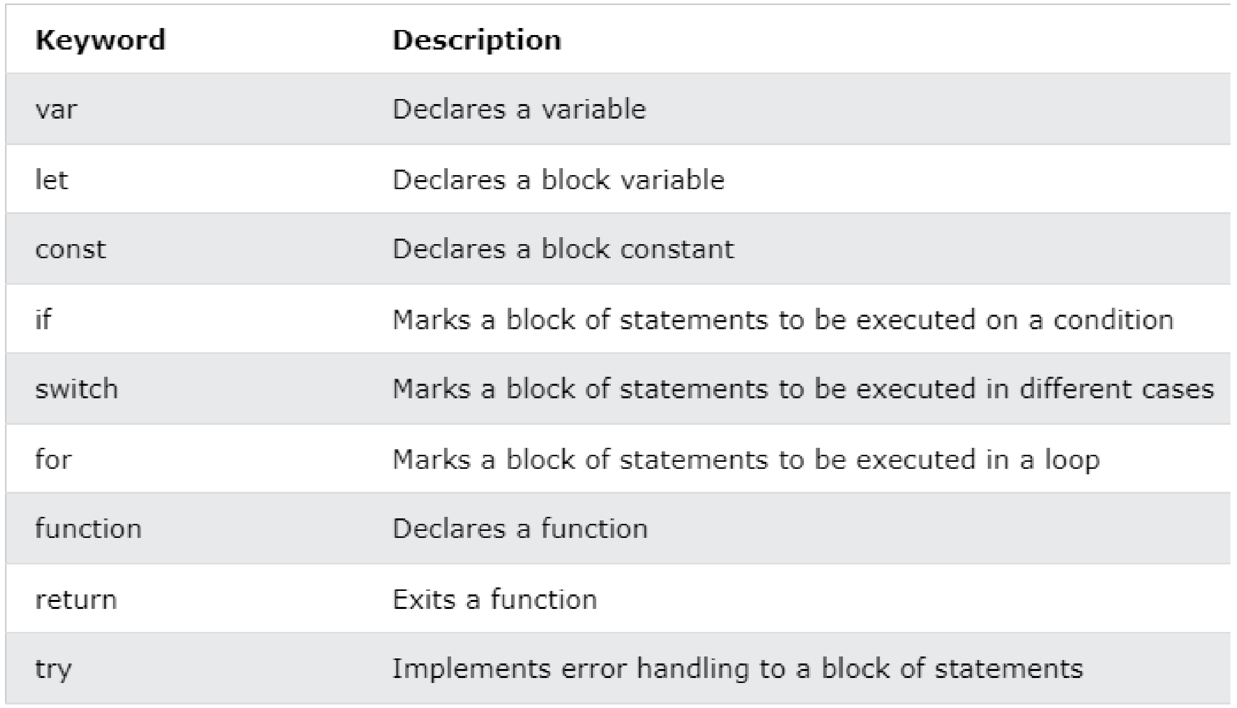
Example

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

JavaScript Keywords

JavaScript statements often start with a **keyword** to dentify the JavaScript action to be performed.
JavaScript keywords are reserved words.
Reserved words cannot be used as names for variables.

Here is a list of some of the keywords



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

Comments

Comments are used to make code more readable.

Single-Line Comments

```
// This is a single-line comment
console.log("Single-line comments are ignored by JavaScript.");
```

Multi-line Comments

Multi-line comments start with /* and end with */. Any text between /* and */ will be ignored by JavaScript.

This example uses a multi-line comment (a comment block) to explain the code:

Example

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
```

Using Comments to Prevent Execution

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

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

This example uses // to prevent execution of one of the code lines:

Example

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

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

JavaScript Variables

Variables are Containers for Storing Data. JavaScript Variables can be declared in 4 ways:

- Automatically
- Using var
- Using let
- Using const

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

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

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

Just like in algebra, variables hold values:

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

Just like in algebra, variables are used in expressions:

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

Variables are containers for storing values.

When to Use var, let, or const?

1. Always declare variables

2. Always use const if the value should not be changed

3. Always use const if the type should not be changed (Arrays and Objects)

4. Only use let if you can't use const

5. Only use var if you MUST support old browsers.

Variable Hoisting

Hoisting refers to **JavaScript's behavior of moving declarations to the top of their scope.**

Example: Hoisting with var

```
console.log(a); // Output: undefined (declaration hoisted, not initialization)
var a = 5;
console.log(a); // Output: 5
```

Example: Hoisting with let and const

```
// console.log(b); // Error: Cannot access 'b' before initialization
let b = 10;
console.log(b); // Output: 10
```

Keywords & Scopes - Var

JavaScript provides three ways to declare variables: var, let, and const.

Var

Function-Scoped: Accessible throughout the function where it is declared.
Can be redeclared and reassigned.
Example: Function Scope

```
function exampleVar() {
  if (true) {
    var x = 10; // Declared inside a block
  }
  console.log(x); // Output: 10 (accessible outside the block)
}
exampleVar();
```

Example: Redeclaration

```
var a = 5;
var a = 10; // Allowed
console.log(a); // Output: 10
```

Keywords & Scopes - Let

Let

Block-Scoped: Accessible only within the {} block where it is defined.
Cannot be redeclared in the same block but can be reassigned.

Example: Block Scope

```
if (true) {
  let y = 20;
  console.log(y); // Output: 20
}
// console.log(y); // Error: y is not defined
```

Example: No Redeclaration

```
let z = 15;
// let z = 25; // Error: Cannot redeclare variable 'z'
z = 25; // Reassignment allowed
console.log(z); // Output: 25
```

Keywords & Scopes - Let

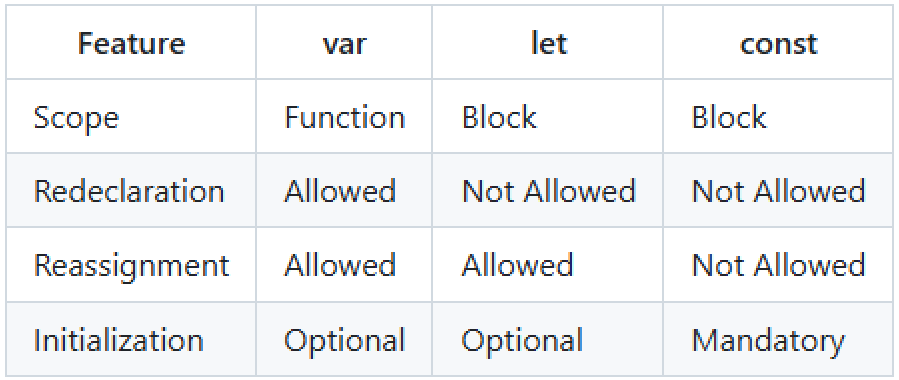
Const

Block-Scoped: Like let, but the value cannot be **reassigned.**
Must be initialized during declaration.

Example

```
const pi = 3.14;
// pi = 3.14159; // Error: Assignment to constant variable
console.log(pi); // Output: 3.14
```

Comparison of var, let, and const



Feature	var	let	const
Scope	Function	Block	Block
Redeclaration	Allowed	Not Allowed	Not Allowed
Reassignment	Allowed	Allowed	Not Allowed
Initialization	Optional	Optional	Mandatory

JavaScript Variables

var	let	const
The scope of a var variable is functional or global scope.	The scope of a let variable is block scope.	The scope of a const variable is block scope.
It can be updated and re-declared in the same scope.	It can be updated but cannot be re-declared in the same scope.	It can neither be updated or re-declared in any scope.
It can be declared without initialization.	It can be declared without initialization.	It cannot be declared without initialization.
It can be accessed without initialization as its default value is “undefined”.	It cannot be accessed without initialization otherwise it will give ‘referenceError’.	It cannot be accessed without initialization, as it cannot be declared without initialization.
These variables are hoisted.	These variables are hoisted but stay in the temporal dead zone untill the initialization.	These variables are hoisted but stays in the temporal dead zone until the initialization.

Javascript Datatypes

JavaScript provides different data types to hold different types of values. There are two types of data types in JavaScript.

- ➢ Primitive data type
- ➢ Non-primitive (reference) data type

JavaScript is a dynamic type language, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use var here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

```
var a=40;//holding number
var b="Rahul";//holding string
```

Javascript Primitive Datatypes

There are five types of primitive data types in JavaScript. They are as follows:

Datatype	Description	**Example**
String	Represents sequence of characters.Eg:"Hello"	let name = "Alice"; console.log(typeof name); // Output: string
Number	Represents numeric value.Eg:100	let age = 25; console.log(typeof age); // Output: number
Boolean	Represents Boolean value either "true" or "false"	let isActive = true; console.log(typeof isActive); // Output: boolean
Undefined	Represents undefined value	let value; console.log(typeof value); // Output: undefined
Null	Represents null.	let empty = null; console.log(typeof empty); // Output: object (a known quirk in JavaScript)
Symbol	Represents a unique and immutable value	let uniqueID = Symbol("id"); console.log(typeof uniqueID); // Output: symbol

Javascript Non Primitive Datatypes

Datatype	Description
Object	Represents instance through which we can access members.
Array	Represents group of similar values.
RegExp	Represents a regular expression.

Most programming languages have many number types:

Whole numbers (integers):
byte (8-bit), short (16-bit), int (32-bit), long (64-bit)
Real numbers (floating-point):
float (32-bit), double (64-bit).
Javascript numbers are always one type:
double (64-bit floating point).

JavaScript Syntax

JavaScript Literals
The two most important syntax rules for fixed values are:
1. **Numbers** are written with or without decimals:
10.50
1001
2. **Strings** are text, written within double or single quotes:
"John Doe"
'John Doe'
JavaScript Variables
In a programming language, variables are used to store data values.

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

An equal sign is used to assign values to variables.

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

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

JavaScript Syntax

JavaScript Expressions

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

For example, 5 * 10 evaluates to 50:

```
5 * 10
```

Expressions can also contain variable values:

```
x * 10
```

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

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

```
"John" + " " + "Doe"
```

JavaScript Keywords

JavaScript keywords are used to identify actions to be performed.

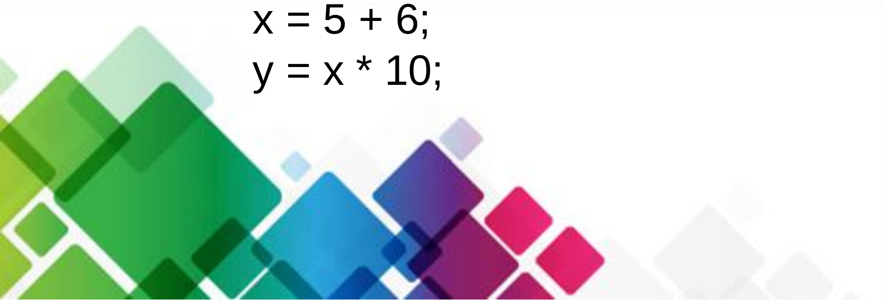
The let keyword tells the browser to create variables:

```
let x, y;
x = 5 + 6;
y = x * 10;
```

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

The var keyword also tells the browser to create variables:

```
var x, y;
x = 5 + 6;
y = x * 10;
```



JavaScript Comment

JavaScript Comments
Not all JavaScript statements are "executed".

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

Comments are ignored, and will not be executed:

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

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

JavaScript Identifiers / Names
Identifiers are JavaScript names.
Identifiers are used to name variables and keywords, and functions.
The rules for legal names are the same in most programming languages.

A JavaScript name must begin with:
A letter (A-Z or a-z)
A dollar sign ($)
Or an underscore (_)
Subsequent characters may be letters, digits, underscores, or dollar signs.

JavaScript Comment

JavaScript is Case Sensitive
All JavaScript identifiers are case sensitive.

The variables lastName and lastname, are two different variables:

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

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

Hyphens:

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

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

JavaScript Comment

Underscore:

first_name, last_name, master_card, inter_city.

Upper Camel Case (Pascal Case):

FirstName, LastName, MasterCard, InterCity.

Lower Camel Case:

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

firstName, lastName, masterCard, interCity.

JavaScript Character Set
JavaScript uses the Unicode character set.

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

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.
These unique names are called **identifiers**.
Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).
The general rules for constructing names for variables (unique identifiers) are:

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

Note: JavaScript identifiers are case-sensitive.

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

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

x = x + 5
In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".
In programming, text values are called **text strings**.
JavaScript can handle many types of data, but for now, just think of numbers and strings.
Strings are written inside double or single quotes. Numbers are written without quotes.
If you put a number in quotes, it will be treated as a text string.

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

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

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

```
var carName;
```
or:
```
let carName;
```
After the declaration, the variable has no value (technically it is undefined).

To assign a value to the variable, use the equal sign:
```
carName = "Volvo";
```

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

JavaScript Data Types

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

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

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

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

Example

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

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

MCQ

MULTIPLE CHOICE QUESTION

JavaScript

JavaScript allows you to work with _________ data types.

A. 1

B. 2

C. 3

D. 4

JavaScript

Variables are declared with the _______ keyword.

A. this

B. int

C. var

D. new

JavaScript

Storing a value in a variable is called ?

A. variable declaration

B. variable destroyed

C. variable store

D. variable initialization

JavaScript

A ----- has global scope which means it can be defined anywhere in your JavaScript code.

A. local variable

B. global variable

C. simple variable

D. complex variable

JavaScript

Which of the following true about JavaScript variable?

A. JavaScript variable names are case-sensitive

B. JavaScript variable names should not start with a numeral

C. You should not use any of the JavaScript reserved keywords as a variable name.

D. All of the above

JavaScript

123test is an valid variable name?

A. TRUE

B. FALSE

C. Can be true or false

D. Can not say

Arithmetic Operators

Following are the list of arithmetic operators:

- ✓ +(Addition):Used to add two operands.
- ✓ -(Subtraction):Used to subtract two operands
- ✓ *(Multiplication):Used to multiply two operands
- ✓ /(Division):Used to find the quotient
- ✓ %(Modulus):Used to find the remainder.
- ✓ ++(Increment):Used to perform increment operation
- ✓ --(Decrement):Used to perform decrement operation.

Example

```
<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);

         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>

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

```
a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
++a = 35
--b = 8
```

Comparison Operators

Following are the list of comparison/relational operators:

- >(greater than)
- <(less than)
- ==(equals)
- >=(greater than or equals)
- <=(less than or equals)
- !=(not equals)

Example

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

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

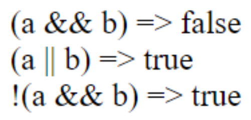
Logical Operators

Javascript supports the following logical operators:-

- ✓ &&(Logical AND):When both the operands are true ,it returns true
- ✓ ||(Logical OR):Even if one of the operands are true,it returns true
- ✓ !(Logical NOT):True becomes false,false becomes true.

Example

```
<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 then try...</p></body>
</html>
```



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

Assignment Operators

The different forms of assignment operators are:

- ✓ +=(Addition assignment)
- ✓ -=(Subtraction assignment)
- ✓ *=(Multiplication assignment)
- ✓ /=(Division assignment)
- ✓ %=(Modulus assignment)

Example

```
<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);
        </script>
</body>
</html>
```

```
Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
```

JavaScript

Modulus operator, %, can be applied to which of these?

a) Integers

b) Floating – point numbers

c) Both Integers and floating – point numbers

d) None of the mentioned

JavaScript

What will be the value of "z" after execution ?

```
let x = 2, y = 3 , z ;
 z = x++ + y-- ;
```

A. -2

B. -1

C. 5

D. 6

JavaScript

What will be the value of "x" after execution ?

```
let x = 0, y = 0 , z = 0 ;
 x = (++x + y-- ) * z++;
```

A. -2

B. -1

C. 0

D. 1

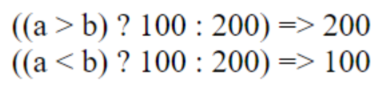
Conditional & Control Statements

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.

? : (Conditional):If Condition is true? Then value X : Otherwise value Y

```
<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) => ");
            result = (a < b) ? 100 : 200;
            document.write(result);
            document.write(linebreak);
      </script>
      <p>Set the variables to different values and then try...</p>
   </body>
</html>
```



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

Control Statements

Control Statements are those which can be used to control the sequence of execution of the program.

Different categories of control statements are:

- Branching Statements: if
- Looping Statements:for,while,do-while
- Execution Termination Statements:break,continue

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.

Example

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

            var age = 20;

            if( age > 18 ) {
               document.write("<b>Qualifies for driving</b>");
            }

      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

```
Qualifies for driving

Set the variable to different value and then try...
```

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:

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

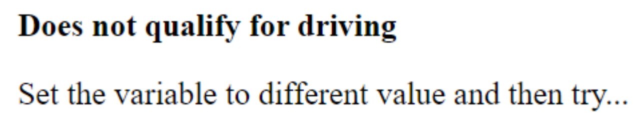
Here JavaScript expression is evaluated.

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

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

            var age = 15;

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

      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```



```
Does not qualify for driving

Set the variable to different value and then try...
```

If..else…if statement

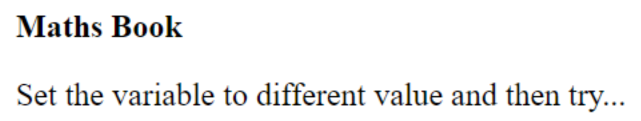
The if...else if... statement is an advanced form of if…else that allows JavaScript to make a correct decision out of several conditions.

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
}
```

Example

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

            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>
      <p>Set the variable to different value and then try...</p>
   </body>
<html>
```



Maths Book

Set the variable to different value and then try...

Switch Statement

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)
}
```

Example

```
<body>
      <script>

            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;

              default:  document.write("Unknown grade<br />")
            }
            document.write("Exiting switch block");

      </script> </body>
```

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

Looping Statements

Process of repeatedly executing set of statements is called Looping.There are 3 kinds of looping statements:

- ✓ for
- ✓ while
- ✓ for-in

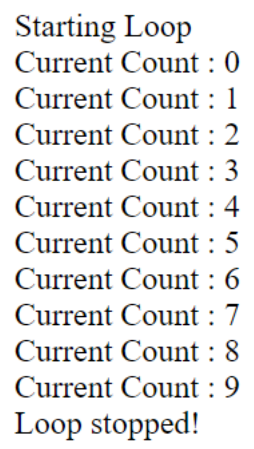
for

Syntax:

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

Example

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

            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>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```



```
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!
```

While Loop

Syntax:

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

The while loop executes set of statements as long as the condition is true.

Example

```
<html>
   <body>

      <script>

            var count = 0;
            document.write("Starting Loop ");

            while (count < 10) {
               document.write("Current Count : " + count + "<br />");
               count++;
            }

            document.write("Loop stopped!");

      </script>

      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

```
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!
```

For-In Loop

The for-in loop is used to loop through an object properties.

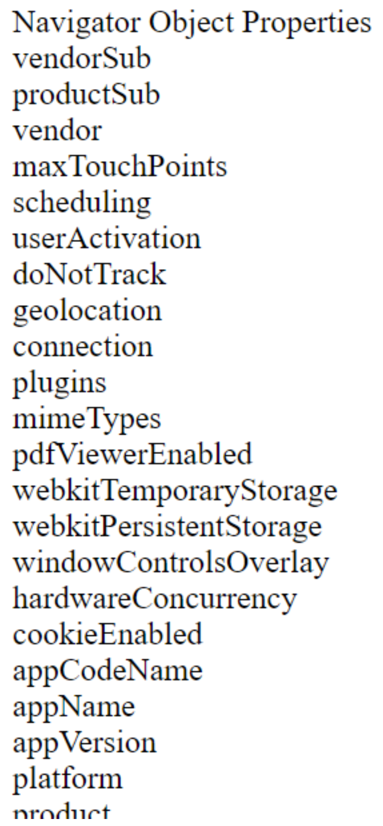
Syntax:
The syntax of ‘for..in’ loop is –

```
for (variablename in object) {
   statement or block to execute
}
```

In each iteration, one property from object is assigned to variablename and this loop continues till all the properties of the object are exhausted.

Example

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

            var aProperty;
            document.write("Navigator Object Properties<br /> ");
            for (aProperty in navigator) {
               document.write(aProperty);
               document.write("<br />");
            }
            document.write ("Exiting from the loop!");

      </script>
      <p>Set the variable to different object and then try...</p>
   </body>
</html>
```



```
Navigator Object Properties
vendorSub
productSub
vendor
maxTouchPoints
scheduling
userActivation
doNotTrack
geolocation
connection
plugins
mimeTypes
pdfViewerEnabled
webkitTemporaryStorage
webkitPersistentStorage
windowControlsOverlay
hardwareConcurrency
cookieEnabled
appCodeName
appName
appVersion
platform
product
```

Loop Control Statements

- There may be a situation when you need to come out of a loop without reaching 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 loop.

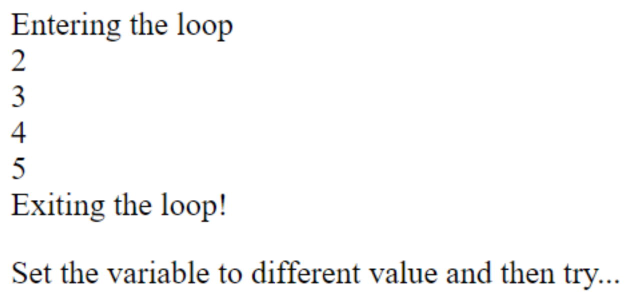
- To handle all such situations, JavaScript provides break and continue statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

Break Statement

Used to terminate the execution.

Example:

```
<body>
      <script>

         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>

      <p>Set the variable to different value and then try...</p>
   </body>
```



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

Set the variable to different value and then try...
```

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.

Example

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

            var x = 1;
            document.write("Entering the loop<br /> ");

            while (x < 10) {
               x = x + 1;

               if (x == 5) {
                  continue;   // skip rest of the loop body
               }
               document.write( x + "<br />");
            }
            document.write("Exiting the loop!<br /> ");

      </script>
      <p>Set the variable to different value and then try...</p>
   </body></html>
```

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

Set the variable to different value and then try...
```

MCQ

MULTIPLE CHOICE QUESTION

JavaScript

What will be the output of the following JavaScript code?

```
var string1 = ”123”;
var intvalue = 123;
alert( string1 + intvalue );
```

A. 123246

B. 246

C. 123123

D. Exception

JavaScript

What will be the output of the following JavaScript code?

```
function output(option)
{
        return (option ?  “yes” :  “no”);
}
        bool ans=true;
console.log(output(ans));
```

A. Yes

B. No

C. Runtime error

D. Compilation error

JavaScript

What will be the output of the following JavaScript code?

```
string  a = ”hi”;
string  b =”there”;
alert(a+b);
```

A. hi
B. there
C. hithere
D. undefined

JavaScript

The “var” and “function” are -------

A. Keywords

B. Declaration statements

C. Data types

D. Prototypes

JavaScript

What will be the output of the following JavaScript code?

```
var a=4;
var b=1;
var c=0;
if(a==b)
    document.write(a);
else if(a==c)
    document.write(a);
else
    document.write(c);
```

A. 4

B. 1

C. Error

D. 0

Javascript Date Object

- The JavaScript date object can be used to get year, month and day. You can display a timer on the webpage by the help of JavaScript date object.
- You can use different Date constructors to create date object. It provides methods to get and set day, month, year, hour, minute and seconds.

Constructor

You can use 4 variant of Date constructor to create date object.

Date()

Date(milliseconds)

Date(dateString)

Date(year, month, day, hours, minutes, seconds, milliseconds)

Javascript Date Methods

1) getDate() method

The JavaScript date getDate() method returns the day for the specified date on the basis of local time.

Syntax

The getDate() method is represented by the following syntax:

dateObj.getDate()

Return

An integer value between 1 and 31 that represents the day of the specified date.

```
<html>
<body>

<script>
var date=new Date();
document.writeln("Today's day: "+date.getDate());
</script>

</body>
</html>
```

getDay() method

The JavaScript date getDay() method returns the value of day of the week for the specified date on the basis of local time. The value of the day starts with 0 that represents Sunday.

Syntax

The getDay() method is represented by the following syntax:

dateObj.getDay()

Return

An integer value between 0 and 6 that represents the days of the week for the specified date.

```
<html>
<body>

<script>
var day=new Date();
document.writeln(day.getDay());
</script>

</body>
</html>
```

getHours() method

The JavaScript date getHours() method returns the hour for the specified date on the basis of local time.

Syntax:
The getHours() method is represented by the following syntax:

dateObj.getHours()
Return
An integer value between 0 and 23 that represents the hour.

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

<script>
var hour=new Date();
document.writeln(hour.getHours());
</script>

</body>
</html>
```

getMilliSeconds()

The JavaScript date getMilliseconds() method returns the value of milliseconds in specified date on the basis of local time.

Syntax
The getMilliseconds() method is represented by the following syntax:

dateObj.getMilliseconds()

Return
An integer value between 0 and 999 that represents milliseconds in specified date.

```
<html>
<body>

<script>
var milli =new Date();
document.writeln(milli.getMilliseconds());
</script>

</body>
</html>
```

getMinutes()

The JavaScript date getMinutes() method returns the minutes in the specified date on the basis of local time.

Syntax

The getMinutes() method is represented by the following syntax:

dateObj.getMinutes()

Return

An integer value between 0 and 59 that represents the minute.

```
<html>
<body>

<script>
var min=new Date();
document.writeln(min.getMinutes());
</script>

</body>
</html>
```

getMonth()

The JavaScript date getMonth() method returns the integer value that represents month in the specified date on the basis of local time. The value returned by getMonth() method starts with 0 that represents January.

Syntax
The getMonth() method is represented by the following syntax:

dateObj.getMonth()
Return
An integer value between 0 and 11 that represents the month in the specified date.

```
<html>
<body>

<script>
var date=new Date();
document.writeln(date.getMonth()+1);
</script>

</body>
</html>
```

getSeconds()

The JavaScript date getSeconds() method returns the seconds in the specified date on the basis of local time.

Syntax
The getSeconds() method is represented by the following syntax:

dateObj.getSeconds()
Return
An integer value between 0 and 59 that represents the second.

```
<html>
<body>

<script>
var sec=new Date();
document.writeln(sec.getSeconds());
</script>

</body>
</html>
```

getFullYear()

The JavaScript date getFullYear() method returns the year in the specified date on the basis of local time.

Syntax
The getFullYear() method is represented by the following syntax:

dateObj.getFullYear()
Return
It will return the year from the local time or from the given date.

```
<html>
<body>

<script>
var date=new Date();
document.writeln("Current Year: "+date.getFullYear());
</script>

</body>
</html>
```

Date Methods

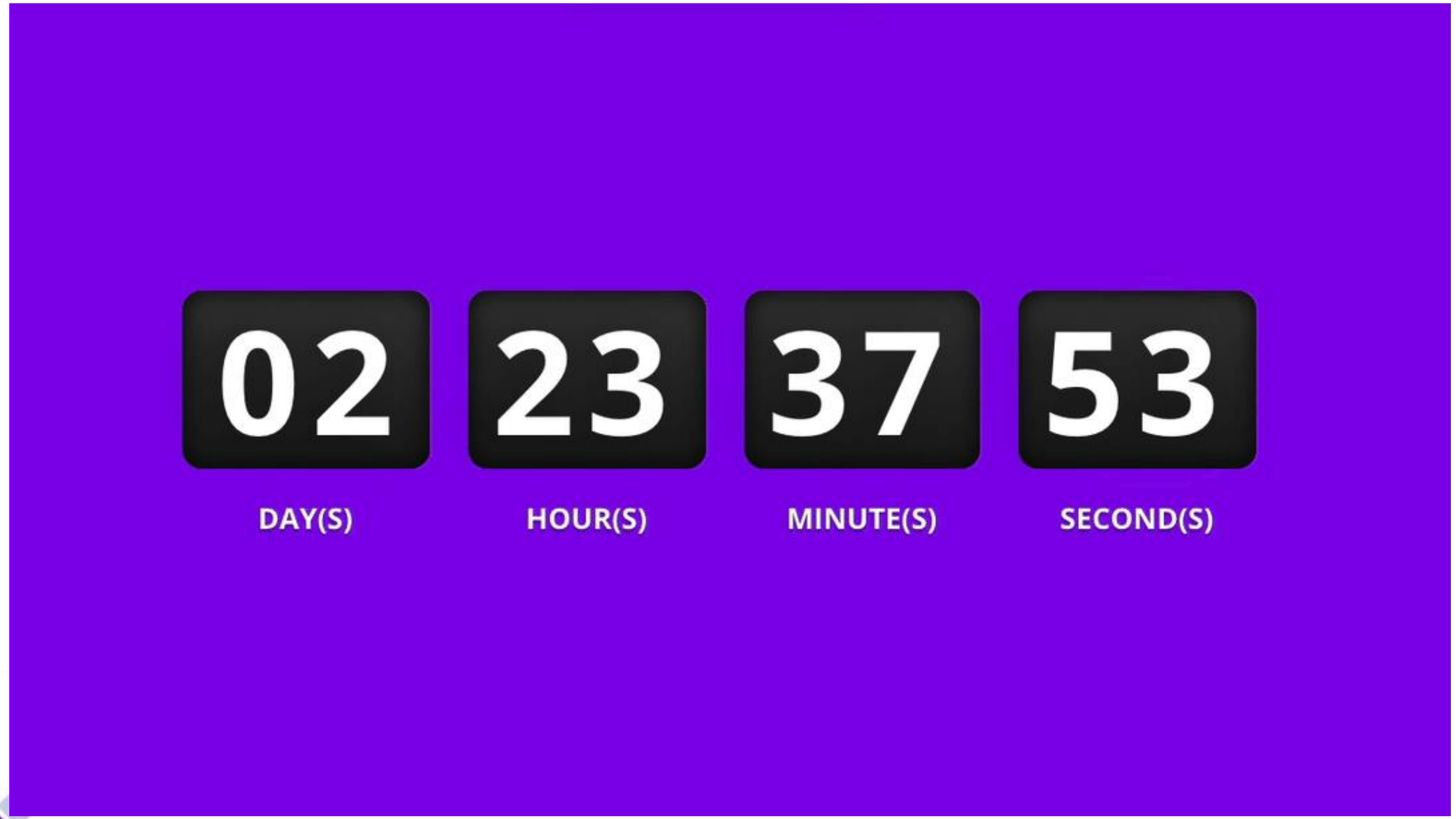
Method	**Description**
getFullYear()	Get **year** as a four digit number (yyyy)
getMonth()	Get **month** as a number (0-11)
getDate()	Get **day** as a number (1-31)
getDay()	Get **weekday** as a number (0-6)
getHours()	Get **hour** (0-23)
getMinutes()	Get **minute** (0-59)
getSeconds()	Get **second** (0-59)
getMilliseconds()	Get **millisecond** (0-999)
getTime()	Get **time** (milliseconds since January 1, 1970)

Methods to Set the Timer

```
<html>
<body>

<script>
var linebreak = "<br />";

var date=new Date();
document.writeln("Today's date: "+date.getDate());
document.write(linebreak);

var day=new Date();
document.writeln("Today's day: "+day.getDay());
document.write(linebreak);

var hour=new Date();
document.writeln("Current Hours: "+hour.getHours());
document.write(linebreak);

var milli =new Date();
document.writeln("Current Milliseconds: "+milli.getMilliseconds());
document.write(linebreak);

var min=new Date();
document.writeln("Current Minutes: "+min.getMinutes());
document.write(linebreak);

var date=new Date();
document.writeln("Current Month: "+date.getMonth()+1);
document.write(linebreak);

var sec=new Date();
document.writeln(sec.getSeconds());
document.write(linebreak);

var date=new Date();
document.writeln("Current Year: "+date.getFullYear());
document.write(linebreak);

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

Task

Create a Timer similar to the one below with Date, Month, Year, Day, Hours, Minutes, Milliseconds. Seconds



JavaScript Function

Javascript Functions

What are Functions in JavaScript?
Definition
A function in JavaScript is a reusable block of code designed to perform a specific task.
Functions help in:

Code Reusability: Write once, use multiple times.
Modularity: Divide the program into smaller, manageable parts.
Dynamic Behavior: Accept inputs (parameters) and return outputs.

Syntax

```
function functionName(parameters) {
  // Function body: code to execute
  return value; // Optional
}
```

Example

```
function greet(name) {
  return `Hello, ${name}!`;
}
document.write(greet("Alice")); // Output: Hello, Alice!
```

In JavaScript, the $ symbol inside backticks (``) is used for template literals (template strings). It allows string interpolation, meaning you can embed expressions directly within the string.

Why Functions Matter
Without functions, code would become repetitive and hard to maintain. Imagine having to write the same logic every time you need it. Functions make code concise and structured.

Javascript Functions

Built-in Functions

Definition
Built-in functions are pre-defined by JavaScript to perform specific tasks. They simplify common operations like string manipulation, math calculations, or date formatting.

Examples of Built-in Functions

1) console.log()
Used to print messages or outputs to the console.
Example:
```
console.log("Hello, World!");
// Output: Hello, World!
```

2) Math.max()
Finds the largest number from a list.
Example:
```
document.write(Math.max(10, 20, 30));
// Output: 30
```

3) parseInt()
Converts a string to an integer.
Example:
```
document.write(parseInt("42"));
// Output: 42
```

4) Date()
Returns the current date and time.
Example:
```
let now = new Date();
document.write(now);
```

5) Problem Example
Find the square root of a number using a built-in function:
```
let number = 64;
let result = Math.sqrt(number);
document.write(`Square root of ${number} is ${result}`);
// Output: Square root of 64 is 8
```

Javascript Functions

Regular Functions

Definition

A regular function is a named function declared using the function keyword. It can take parameters and return a value.

Syntax

```
function functionName(parameters) {
  // Function body
  return value; // Optional
}
```

Example

```
function add(a, b) {
  return a + b;
}
document.write(add(5, 3)); // Output: 8
```

Problem Example: Find the factorial of a number using a regular function.

```
function factorial(n) {
  let result = 1;
  for (let i = 1; i <= n; i++) {
    result *= i;
  }
  return result;
}
document.write(factorial(5)); // Output: 120
```

Javascript Functions

Passing Parameters to Functions

Definition

Parameters are variables passed to a function that allow the function to accept dynamic inputs.

Syntax

```
function functionName(parameter1, parameter2) {
  // Function body
}
```

Example

```
function multiply(a, b) {
  return a * b;
}
document.write(multiply(4, 5)); // Output: 20
```

Explanation

- **a and b are parameters that hold the values 4 and 5 during execution.**
- **Functions can accept multiple parameters or none at all.**

Problem Example: Write a function that greets a user in different languages.

```
function greetUser(name, language) {
  if (language === "English") {
    return `Hello, ${name}!`;
  } else if (language === "Spanish") {
    return `Hola, ${name}!`;
  } else {
    return `Hi, ${name}!`;
  }
}
document.write(greetUser("Alice", "Spanish")); // Output: Hola, Alice!
```

Javascript Functions

Return Value

Definition
A function can return a value using the return statement. The returned value can be stored in a variable or used directly.

Example
```
function square(number) {
  return number * number;
}

let result = square(4);
document.write(result); // Output: 16
```

Javascript Functions

Anonymous Functions

Definition
An anonymous function is a function without a name. It is usually assigned to a variable or passed as an argument.

Syntax
```
let variableName = function(parameters) {
  // Function body
};
```

Example: Assigning an anonymous function to a variable
```
let greet = function(name) {
  return `Hello, ${name}!`;
};
document.write(greet("Alice")); // Output: Hello, Alice!
```

Problem Example: Sort an array using an anonymous function.
```
let numbers = [10, 5, 8, 1];
numbers.sort(function(a, b) {
  return a - b;
});
document.write(numbers); // Output: [1, 5, 8, 10]
```

Javascript Functions

Arrow Functions

Definition
An arrow function is a shorter syntax for writing functions, introduced in ES6. It inherits the this context from its surrounding scope.

Syntax

```
(parameters) => {
  // Function body
  return value; // Optional
};
```

Example: Arrow function to calculate square

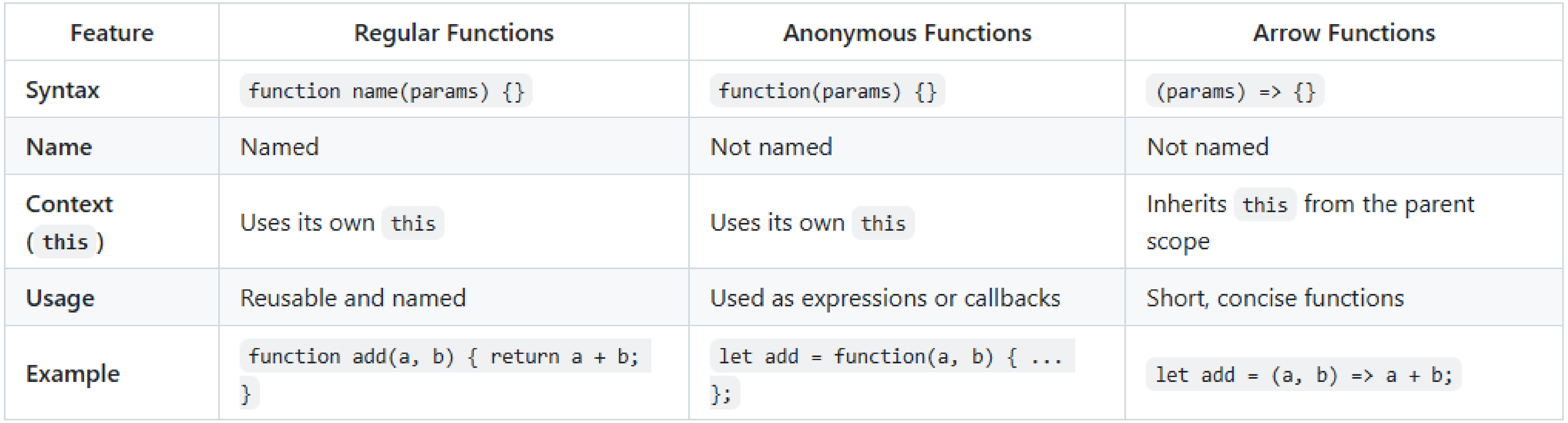
```
let square = (number) => number * number;
document.write(square(4)); // Output: 16
```

Problem Example: Double each number in an array using arrow functions.

```
let numbers = [1, 2, 3, 4];
let doubled = numbers.map(num => num * 2);
document.write(doubled); // Output: [2, 4, 6, 8]
```

Javascript Functions

Differences Between Function Types



Feature	Regular Functions	Anonymous Functions	Arrow Functions
Syntax	`function name(params) {}`	`function(params) {}`	`(params) => {}`
Name	Named	Not named	Not named
Context (`this`)	Uses its own `this`	Uses its own `this`	Inherits `this` from the parent scope
Usage	Reusable and named	Used as expressions or callbacks	Short, concise functions
Example	`function add(a, b) { return a + b; }`	`let add = function(a, b) { ... };`	`let add = (a, b) => a + b;`

Key Takeaways

1.Functions are essential for reusability and modularity.
2.Use **built-in functions** for common tasks.
3.Regular functions are ideal for structured, named tasks.
4.Anonymous functions and **arrow functions** are useful for callbacks and concise logic.
5.Understanding function types and contexts is crucial for mastering JavaScript.

Built-in Functions

JavaScript provides a variety of built-in functions for different tasks. These are organized into categories based on their use cases:

- **Global Functions**
- **String Functions**
- **Array Functions**
- **Math Functions**
- **Date Functions**
- **Object Functions**
- **Utility Functions**

Built-in Functions

Global Functions

Global functions are available throughout the program and can be accessed directly without importing any module.

1) parseInt()

Description: Converts a string to an integer.
Syntax: parseInt(string, radix)
string: The string to parse.
radix: The base of the numeral system (optional).

Example:

```
console.log(parseInt("42"));       // Output: 42
console.log(parseInt("101", 2));  // Output: 5 (binary to decimal)
```

2) parseFloat()

Description: Converts a string to a floating-point number.

Example:

```
console.log(parseFloat("3.14"));  // Output: 3.14
console.log(parseFloat("10.5px"));// Output: 10.5
```

3) isNaN()

Description: Checks if a value is NaN (Not-a-Number).

Example:

```
console.log(isNaN("hello"));   // Output: true
console.log(isNaN(123));       // Output: false
```

4) Number()

Description: Converts a value to a number.

Example:

```
console.log(Number("42"));     // Output: 42
console.log(Number(true));     // Output: 1
```

Built-in Functions

String Functions
String functions are used to manipulate and analyze text data.

1) charAt()
Description: Returns the character at a specific index.
Example:

```
let str = "Hello";
console.log(str.charAt(1));  // Output: e
```

2) toUpperCase() / toLowerCase()
Description: Converts a string to uppercase or lowercase.
Example:

```
let text = "JavaScript";
console.log(text.toUpperCase()); // Output: JAVASCRIPT
console.log(text.toLowerCase()); // Output: javascript
```

3) indexOf()
Description: Returns the position of the first occurrence of a substring.
Example:

```
let str = "Hello, world!";
console.log(str.indexOf("world")); // Output: 7
```

4) substring()
Description: Extracts a portion of the string.
Syntax: string.substring(start, end)
Example:

```
let str = "JavaScript";
console.log(str.substring(0, 4)); // Output: Java
```

Built-in Functions

Array Functions
Array functions help in manipulating and analyzing arrays.

1) push() / pop()
Description: Adds or removes elements from the end of an array.
Example:

```
let arr = [1, 2, 3];
arr.push(4);
console.log(arr);  // Output: [1, 2, 3, 4]
arr.pop();
console.log(arr);  // Output: [1, 2, 3]
```

2) shift() / unshift()
Description: Adds or removes elements from the beginning of an array.
Example:

```
let arr = [1, 2, 3];
arr.unshift(0);
console.log(arr);  // Output: [0, 1, 2, 3]
arr.shift();
console.log(arr);  // Output: [1, 2, 3]
```

3) map()
Description: Creates a new array by applying a function to each element.
Example:

```
let nums = [1, 2, 3];
let squares = nums.map(num => num * num);
console.log(squares); // Output: [1, 4, 9]
```

4) filter()
Description: Filters an array based on a condition.
Example:

```
let nums = [1, 2, 3, 4];
let evenNums = nums.filter(num => num % 2
=== 0);
console.log(evenNums); // Output: [2, 4]
```

Built-in Functions

Math Functions
Math functions perform mathematical operations.

1) Math.round()
Description: Rounds a number to the nearest integer.
Example:
console.log(Math.round(4.5)); // Output: 5

2) Math.random()
Description: Generates a random number between 0 and 1.
Example:
console.log(Math.random()); // Output: 0.123456 (random)

3) Math.pow()
Description: Returns the base raised to the power of the exponent.
Example:
console.log(Math.pow(2, 3)); // Output: 8

Built-in Functions

Date Functions

Date functions handle date and time operations.

1) Date()

Description: Returns the current date and time.

Example:

```
let now = new Date();
console.log(now); // Output: Fri Jan 10 2025 14:00:00 GMT+0530 (IST)
```

2) getFullYear()

Description: Returns the year from a Date object.

Example:

```
let now = new Date();
console.log(now.getFullYear()); // Output: 2025
```

Built-in Functions

Object Functions
Object functions work with JavaScript objects.

1) Object.keys()
Description: Returns an array of an object's keys.
Example:
```
let obj = { a: 1, b: 2 };
console.log(Object.keys(obj)); // Output: ["a", "b"]
```

2) Object.values()
Description: Returns an array of an object's values.
Example:
```
let obj = { a: 1, b: 2 };
console.log(Object.values(obj)); // Output: [1, 2]
```

Built-in Functions

Utility Functions

1) setTimeout()

Description: Executes a function after a delay.

Example:

```
setTimeout(() => {
  console.log("Executed after 2 seconds");
}, 2000);
```

2) setInterval()

Description: Repeatedly executes a function at specified intervals.

Example:

```
let count = 0;
let interval = setInterval(() => {
  count++;
  console.log(`Count: ${count}`);
  if (count === 5) clearInterval(interval);
}, 1000);
```

What Are User-Defined Functions?

A user-defined function is a reusable block of code written by a programmer to perform a specific task. Unlike built-in functions (like Math.max()), these functions are explicitly defined by you.

Syntax

```
function functionName(parameters) {
    // Code to execute
    return value; // Optional, specifies the output of the function
}
```

<u>Understanding Scope in JavaScript</u>

Definition

Scope defines the accessibility of variables, functions, and objects in JavaScript. In simpler terms, scope determines where in your code a variable or function can be accessed.

Types of Scope in JavaScript

Global Scope: Variables declared outside any function or block are accessible everywhere in the code.
Local Scope: Variables declared within a function or block are only accessible within that specific function or block.
Block Scope: Variables declared with let and const inside a block ({}) are confined to that block.
Function Scope: Variables declared with var are accessible within the entire function they are defined in.

What Are User-Defined Functions?

Scope Examples

Global Scope

```
let globalVar = "I am global";

function showGlobal() {
    console.log(globalVar); // Accessible
}
showGlobal(); // Output: I am global
```

Local Scope

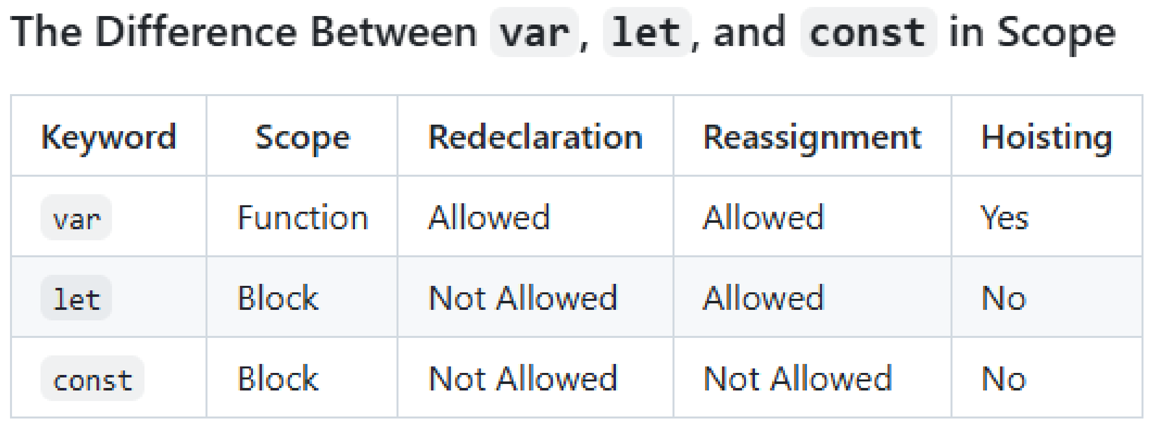
```
function localScope() {
    let localVar = "I am local";
    console.log(localVar); // Accessible
}
localScope();
console.log(localVar); // Error: localVar is not defined
```

Block Scope

```
{
    let blockScoped = "Only accessible in this block";
    console.log(blockScoped); // Accessible
}
console.log(blockScoped); // Error: blockScoped is not defined
```

Function Scope

```
function functionScope() {
    if (true) {
        var functionScopedVar = "I am function scoped";
    }
    console.log(functionScopedVar); // Accessible
}
functionScope();
```



The Difference Between `var`, `let`, and `const` in Scope

Keyword	Scope	Redeclaration	Reassignment	Hoisting
`var`	Function	Allowed	Allowed	Yes
`let`	Block	Not Allowed	Allowed	No
`const`	Block	Not Allowed	Not Allowed	No

JavaScript Objects

Javascript Objects

Definition:

- A JavaScript object is a collection of key-value pairs, where each key is a unique string, and the value can be any valid JavaScript data type. JavaScript objects allow you to group related data and functionality together. These objects can represent real-world entities or abstract concepts, like a person, car, or product.

- Objects are an essential part of JavaScript and are used extensively in everyday programming.

Example of an Object:

```
const person = {
name: "John Doe",
age: 30,
greet: function() {
console.log("Hello, " + this.name);
}
};
```

- **name** and **age** are properties of the object person.
- **greet** is a method (function inside an object) of the object person.

Why Use Objects?

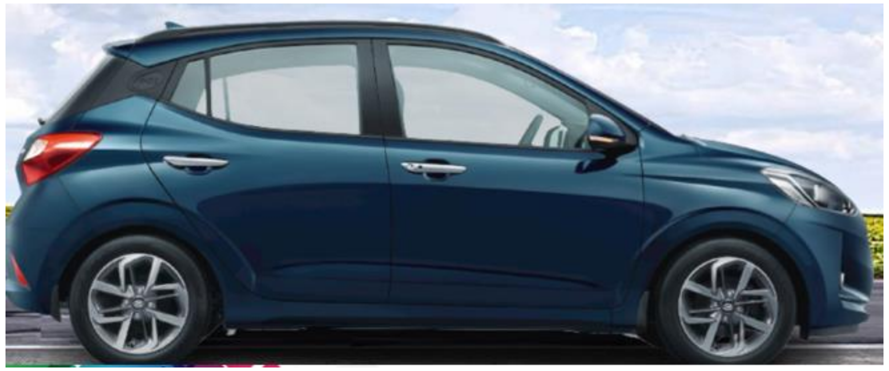
Objects provide a way to store and organize data. Instead of having multiple variables for different pieces of information (such as name, age, etc.), you can bundle all related information into one object, making your code more readable and maintainable.

Javascript Objects

Real Life Objects
In real life, **objects** are things like: houses, cars, people, animals, or any other subjects.

Here is a **car object** example:

Car Object	Properties	Methods

	car.name = Fiat
	car.model = 500
	car.weight = 850kg
	car.color = white
		car.start()
		car.drive()
		car.brake()
		car.stop()

Object Properties
A real life car has **properties** like weight and color:
car.name = Fiat, car.model = 500, car.weight = 850kg, car.color = white.

Car objects have the same **properties**, but the **values** differ from car to car.

Object Methods
A real life car has **methods** like start and stop:
car.start(), car.drive(), car.brake(), car.stop().
Car objects have the same **methods**, but the methods are performed **at different times**.

Javascript Variables & Objects

JavaScript Variables
JavaScript variables are containers for data values.
This code assigns a **simple value** (Fiat) to a **variable** named car:
let car = "Fiat";

JavaScript Objects
Objects are variables too. But objects can contain many values.
This code assigns **many values** (Fiat, 500, white) to an **object** named car:
const car = {type:"Fiat", model:"500", color:"white"};

It is a common practice to declare objects with the `const` keyword.

Creating objects in Javascript

There are 3 ways to create objects in Javascript:

- ➢ By object literal
- ➢ By creating instance of Object directly (using new keyword)
- ➢ By using an object constructor (using new keyword)

By object literal:

The syntax of creating object using object literal is given below:

object={property1:value1,property2:value2.....propertyN:valueN}

As you can see, property and value is separated by : (colon).

Example Object Literal

```
<html>
<body>
<script>
emp={id:102,name:"Shyam Kumar",salary:40000}
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
</body>
</html>
```

name:value pairs are also called **key:value pairs**.
object literals are also called **object initializers**.

Spaces and line breaks are not important. An object initializer can span multiple lines:

```
// Create an Object
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

```
102 Shyam Kumar 40000
```

Creating an Instance of Object

The syntax of creating object directly is given below:

var objectname=new Object();
Here, new keyword is used to create object.

Example:

```
<script>
var emp=new Object();
emp.id=101;
emp.name="Vijay Adhiraj";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

But, there is no need to use new Object().
For readability, simplicity and execution speed, use the **object literal** method.

Object Properties
The **named values**, in JavaScript objects, are called **properties**.

Property	Value
firstName	John

```
101 Vijay Adhiraj 50000
```

By Using an Object Constructor

Constructor: In class-based, object-oriented programming, a constructor (abbreviation: ctor) is a special type of function called to create an object. It prepares the new object for use, often accepting arguments that the constructor uses to set required member variables.

The this keyword refers to the current object.

The example of creating object by object constructor is given below.

```
<script>
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;
}
e=new emp(103,"Vimal Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);
</script>
```

JavaScript Object Methods
Methods are **actions** that can be performed on objects.
Methods are **function definitions** stored as **property values**.

Property	Value
firstName	John

Accessing Object Properties
You can access object properties in two ways:
person.lastName;
person["lastName"];

```
103 Vijay Adhiraj 30000
```

Javascript Array

An array is a special variable, which can hold more than one value:

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

It is a common practice to declare arrays with the const keyword.

JavaScript array is an object that represents a collection of similar type of elements.

There are 3 ways to construct array in JavaScript

- **By array literal**
- **By creating instance of Array directly (using new keyword)**
- **By using an Array constructor (using new keyword)**

By Array Literal

The syntax of creating array using array literal is given below:

var arrayname=[value1,value2.....valueN];
As you can see, values are contained inside [ ] and separated by , (comma).

Example:

```
<html>
<body>
<script>
//single dimension array (1 for loop for iteration)
var stu =[10,11,12,13,14,15]; //array literal
for (i=0; i<stu.length; i++){
document.write(stu[i] + "<br/>");
}
</script>
</body>
</html>
```

```
10
11
12
13
14
15
```

Array Directly(By using new keyword)

The syntax of creating array directly is given below:

var arrayname=new Array();
Here, new keyword is used to create instance of array.

Example:

```
<html>
<body>
<script>
var i;
var emp = new Array();
emp[0] = "Arun";
emp[1] = "Varun";
emp[2] = "John";

for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
</body>
</html>
```

```
Arun
Varun
John
```

Array Constructor

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

Example:

```
<html>
<body>
<script>
//By using array constructor
var cus=new Array(2.3, 4.4,3.5,5.6);
for (k=0;k<cus.length;k++) {
document.write(cus[k] + "<br>");
}
</script>
</body>
</html>
```

```
2.3
4.4
3.5
5.6
```

Array Methods

1) concat() method:

- ➢ The JavaScript array concat() method combines two or more arrays and returns a new string.
- ➢ This method doesn't make any change in the original array.

Syntax:
The concat() method is represented by the following syntax:

array.concat(arr1,arr2,....,arrn)

```
<html>
<body>

<script>
var arr1=[1,2,3,4,5];
var arr2=[10,11,12,13];
var result= arr1.concat(arr2);
document.writeln(result);
</script>

</body>
</html>
```

```
1,2,3,4,5,10,11,12,13
```

copyWithin() method

The JavaScript array copyWithin() method copies the part of the given array with its own elements and returns the modified array. This method doesn't change the length of the modified array.

Syntax:
The copyWithin() method is represented by the following syntax:

array.copyWithin(target, start, end)

Parameters:

- ✓ target - The position where the copied element takes place.
- ✓ start - It is optional. It represents the index from where the method starts copying elements. By default, it is 0.
- ✓ end - It is optional. It represents the index at which elements stops copying. By default, it is array.length-1.

```
<html>
<body>
<script>
var i;
var emp = new Array();
emp[0] = "Arun";
emp[1] = "Varun";
emp[2] = "John";
result=emp.copyWithin(0,1,2);
document.writeln(result);
</script>
</body>
</html>
```

```
Varun,Varun,John
```

Every() Method

The JavaScript array every() method checks whether all the given elements in an array are satisfying the provided condition. It returns true when each given array element satisfying the condition otherwise false.

Syntax:
The every() method is represented by the following syntax:

array.every(callback(currentvalue,index,arr),thisArg)
Parameter:
- ✓ callback - It represents the function that test the condition.
- ✓ currentvalue - The current element of array.
- ✓ index - It is optional. The index of current element.
- ✓ arr - It is optional. The array on which every() operated.
- ✓ thisArg - It is optional. The value to use as this while executing callback.

```
<html>
<body>

<script>
var marks=[50,40,45,37,20];

function check(value)
{
  return value>30;   //return false, as
marks[4]=20
}

document.writeln(marks.every(check));
</script>

</body>
</html>
```

```
false
```

Flat() method

- The flat() method is an inbuilt array method that flattens a given array into a newly created one-dimensional array.
- It concatenates all the elements of the given multidimensional array, and flats upto the specified depth.
- We can specify the depth limit to where we need to flatten the array. By default, the depth limit is 1.

Syntax:
var newArr=arr.flat(<depth>);
Parameters
Depth: It is an optional parameter which specifies the depth to flatten an array. By default, its value is 1.

Return:
It returns a newly created array containing all the sub-array elements concatenated into it.

```
<html>
<head> <h5> Array Methods </h5> </head>
<body>
<script>
var arr=['a','b',['c','d']]; //given 2D array
var newArr=arr.flat(); //using flat() method
document.write("After flattening the array:
"+newArr);
</script>
</body>
</html>
```

```
Array Methods
After flattening the array: a,b,c,d
```

Fill() Method

The JavaScript array fill() method fills the elements of the given array with the specified static values. This method modifies the original array. It returns undefined, if no element satisfies the condition.

Syntax
The fill() method is represented by the following syntax:

arr.fill(value[, start[, end]])

Parameter:

- ✓ value - The static value to be filled.
- ✓ start - It is optional. It represents the index from where the value starts filling. By default, it is 0.
- ✓ end - It is optional. It represents the index where the value stops filling. By default, it is length-1

```
<html>
<body>

<script>
var arr=["AngularJS","Node.js","JQuery","HTML"];
var result=arr.fill("Bootstrap",0,2);
document.writeln(arr);
</script>

</body>
</html>
```

```
Bootstrap,Bootstrap,JQuery,HTML
```

Find()

The JavaScript array find() method returns the first element of the given array that satisfies the provided function condition.

Syntax:
The find() method is represented by the following syntax:

array.find(callback(value,index,arr),thisArg)
Parameter
- ✓ callback - It represents the function that executes each element.
- ✓ value - The current element of an array.
- ✓ index - It is optional. The index of current element.
- ✓ arr - It is optional. The array on which find() operated.
- ✓ thisArg - It is optional. The value to use as this while executing callback.

Return
The value of first element of the array that satisfies the function condition.

```
<html>
<body>
<script>
var arr=[5,22,19,25,34];
var result=arr.find(x=>x>20);
document.writeln(result)
</script>
</body>
</html>
```

isArray() method

The isArray() method is used to test whether the value passed is an array. If it finds the passed value is an array, it returns True. Otherwise, it returns False.

Syntax:
Array.isArray(obj_value);
Parameter:
obj_value: It is the value of the object which is passed for determining whether it is an array or not.

Return:
It returns either false or true, depending on the test.

```
<html>
<head> <h5> JavaScript Array Methods </h5>
</head>
<body>
<script>
document.write(Array.isArray(1,2,3,4));
//Testing the passed values.
</script>
</body>
</html>
```

```
JavaScript Array Methods
false
```

```
<html>
<head> <h5> JavaScript Array Methods </h5> </head>
<body>
<script>
var arr=new Array(1,2,3,4);
document.write(Array.isArray(arr)); //Testing the passed
values.
</script>
</body>
</html>
```

```
JavaScript Array Methods
true
```

pop() Method

The JavaScript array pop() method removes the last element from the given array and return that element. This method changes the length of the original array.

Syntax:
The pop() method is represented by the following syntax:

array.pop()

Return:
The last element of given array.

```
<html>
<body>
<script>
var arr=["AngularJS","Node.js","JQuery"];
document.writeln("Original array: "+arr+"<br>");
document.writeln("Extracted element:
"+arr.pop()+"<br>");
document.writeln("Remaining elements: "+ arr);
</script>
</body>
</html>
```

```
Original array: AngularJS,Node.js,JQuery
Extracted element: JQuery
Remaining elements: AngularJS,Node.js
```

Push()

The JavaScript array push() method adds one or more elements to the end of the given array. This method changes the length of the original array.

Syntax:
The push() method is represented by the following syntax:

array.push(element1,element2....elementn)

Parameter:
element1,element2....elementn - The elements to be added.

Return:
The original array with added elements.

```
<html>
<body>
<script>
var arr=["AngularJS","Node.js"];
arr.push("JQuery");
document.writeln(arr);
</script>
</body>
</html>
```

```
AngularJS,Node.js,JQuery
```

reverse() Method

- The JavaScript array reverse() method changes the sequence of elements of the given array and returns the reverse sequence.
- In other words, the arrays last element becomes first and the first element becomes the last.
- This method also made the changes in the original array.

Syntax:
The reverse() method is represented by the following syntax:

array.reverse()

Return:
The original array elements in reverse order.

```
<html>
<body>
<script>
var arr=["AngulaJS","Node.js","JQuery"];
var rev=arr.reverse();
document.writeln(rev);
</script>
</body>
</html>
```

```
JQuery,Node.js,AngulaJS
```

Slice()

The JavaScript array slice() method extracts the part of the given array and returns it. This method doesn't change the original array.

Syntax:
The slice() method is represented by the following syntax:

array.slice(start,end)

Parameter:

✓ start - It is optional. It represents the index from where the method starts to extract the elements.

✓ end - It is optional. It represents the index at where the method stops extracting elements.

Return:
A new array contains the extracted elements.

```
<html>
<body>
<script>
var
arr=["AngularJS","Node.js","JQuery","Bootstra
p"]
var result=arr.slice(1,3);
document.writeln(result);
</script>
</body>
</html>
```

Sort()

The JavaScript array sort() method is used to arrange the array elements in some order. By default, sort() method follows the ascending order.

Syntax:
The sort() method is represented by the following syntax:

array.sort(compareFunction)

Parameter:
compareFunction - It is optional. It represents a function that provides an alternative sort order.

Return:
An array of sorted elements

```
<html>
<body>
<script>
var arr=["AngularJS","Node.js","JQuery","Bootstrap"]
var result=arr.sort();
document.writeln(result);
</script>
</body>
</html>
```

```
AngularJS,Bootstrap,JQuery,Node.js
```

String

Strings are for **storing text**
Strings are written **with quotes**

A JavaScript string is zero or more characters written inside quotes.
Example
```
let text = "John Doe";
```

You can use single or double quotes:
Example
```
let carName1 = "Volvo XC60";  // Double quotes
let carName2 = 'Volvo XC60';  // Single quotes
```

The JavaScript string is an object that represents a sequence of characters.

There are 2 ways to create string in JavaScript

- **By string literal**
- **By string object (using new keyword)**

By string literal

The string literal is created using double quotes. The syntax of creating string using string literal is given below:

var stringname="string value";

Example:

```
<html>
<body>
<script>
var str="This is string literal";
document.write(str);
</script>
</body>
</html>
```

```
This is string literal
```

By string object

The syntax of creating string object using new keyword is given below:

var stringname=new String("string literal");
Here, new keyword is used to create instance of string.

Example:

```
<html>
<body>
<script>
var stringname=new String("hello javascript string");
document.write(stringname);
</script>
</body>
</html>
```

```
hello javascript string
```

String Methods

1) JavaScript String charAt(index) Method
The JavaScript String charAt() method returns the character at the given index.

```
<script>
var str="javascript";
document.write(str.charAt(2));
</script>
```

```
v
```

2) JavaScript String concat(str) Method
The JavaScript String concat(str) method concatenates or joins two strings.

```
<script>
var s1="javascript ";
var s2="concat example";
var s3=s1.concat(s2);
document.write(s3);
</script>
```

```
javascript concat example
```

String Methods

3) JavaScript String indexOf(str) Method

The JavaScript String indexOf(str) method returns the index position of the given string.

```
<script>
var s1="javascript from itvedant indexof";
var n=s1.indexOf("from");
document.write(n);
</script>
```

11

4) JavaScript String lastIndexOf(str) Method

The lastIndexOf() method returns the position of the last occurrence of specified character(s) in a string.
When is the letter occurred the last time – that starting index position will be displayed.

```
<script>
var s1="javascript from itvedant indexof java";
var n=s1.lastIndexOf("java");
document.write(n);
</script>
```

31

String Methods

5) JavaScript String toLowerCase() Method
The JavaScript String toLowerCase() method returns the given string in lowercase letters.

```
<script>
var s1="JavaScript toLowerCase Example";
var s2=s1.toLowerCase();
document.write(s2);
</script>
```

```
javascript tolowercase example
```

6) JavaScript String toUpperCase() Method
The JavaScript String toUpperCase() method returns the given string in uppercase letters.

```
<script>
var s1="JavaScript toUpperCase Example";
var s2=s1.toUpperCase();
document.write(s2);
</script>
```

```
JAVASCRIPT TOUPPERCASE EXAMPLE
```

String Methods

7) JavaScript String slice(beginIndex, endIndex) Method

The JavaScript String slice(beginIndex, endIndex) method returns the parts of string from given beginIndex to endIndex. In slice() method, beginIndex is inclusive and endIndex is exclusive.

```
<script>
var s1="abcdefgh";
var s2=s1.slice(2,5);
document.write(s2);
</script>
```

```
cde
```

8) JavaScript String trim() Method

The JavaScript String trim() method removes leading and trailing whitespaces from the string.

```
<script>
var s1="     javascript trim    ";
var s2=s1.trim();
document.write(s2);
</script>
```

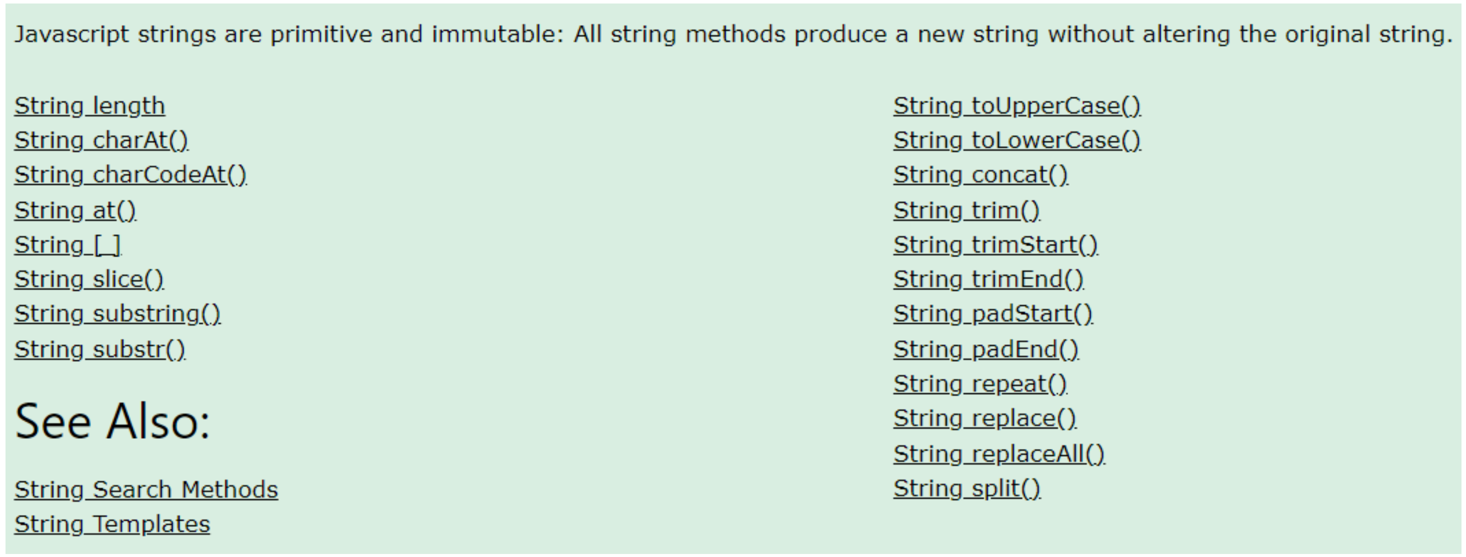
```
javascript trim
```

String Methods

9) JavaScript String split() Method:

```
<script>
var str="This is Itvedant website";
document.write(str.split(" ")); //splits the given string.
</script>
```

```
This,is,Itvedant,website
```



Javascript strings are primitive and immutable: All string methods produce a new string without altering the original string.

String length
String charAt()
String charCodeAt()
String at()
String [ ]
String slice()
String substring()
String substr()

See Also:

String Search Methods
String Templates

String toUpperCase()
String toLowerCase()
String concat()
String trim()
String trimStart()
String trimEnd()
String padStart()
String padEnd()
String repeat()
String replace()
String replaceAll()
String split()

Math Methods

Math Object:

The Math object provides useful mathematical constants and functions.

```
document.write(Math.PI);          // Output: 3.14159 (π constant)

document.write(Math.max(1, 2, 3)); // Output: 3

document.write(Math.random());    // Output: Random number between 0 and 1
```

MCQ

MULTIPLE CHOICE QUESTION

JavaScript

What will be the output of the following JavaScript code?

```
var a1 = [,,,];
var a2 = new Array(3);
document.write(a1[0]);
document.write(a2[0]);
```

A. true false

B. false true

C. undefined undefined

D. false true

JavaScript

The pop() method of the array does which of the following task?

A. decrements the total length by 1

B. increments the total length by 1

C. prints the first element but no effect on the length

D. updates the element

JavaScript

What will be the output of the following JavaScript code?

```
var a = [1,2,3,4,5];
a.slice(0,3);
```

A. Returns [1,2,3]

B. Returns [4,5]

C. Returns [1,2,3,4]

D. Returns [1,2,3,4,5]

JavaScript

What will be the output of the following JavaScript code?

```
var val1=[1,2,3];
var val2=[6,7,8];
var result=val1.concat(val2);
document.writeln(result);
```

A. 1, 2, 3

B. Error

C. 1, 2, 3, 6, 7, 8

D. 123

JavaScript

What will be the output of the following JavaScript code?

```
var values=[1,2,3,4]
var ans=values.slice(1);
document.writeln(ans);
```

A. 1, 2, 3, 4

B. 2, 3, 4

C. 1, 3, 4

D. error

Window Object - Console, Screen, Location, History

The window object in JavaScript represents the global context and provides access to various properties and methods related to the browser window. Below are key components of the window object:

Console Object:

- The console object provides methods to log information to the browser’s developer console. These methods are helpful for debugging.

- console.log("This is a log message");
- console.error("This is an error message");
- console.warn("This is a warning message");

- console.log() logs general messages.
- console.error() logs errors.
- console.warn() logs warnings.

History Object

The JavaScript history object represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.

The history object is the window property, so it can be accessed by: window.history or,history.

Property of Javascript History Object
1.**length:**Returns the length of the history URL's.

Methods of Javascript History Object

- ✓ **forward():** loads the next page.
- ✓ **back()** :loads the previous page.
- ✓ **go()**   :loads the given page number.

History Object

The JavaScript history object represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.

The history object is the window property, so it can be accessed by: window.history or,history.

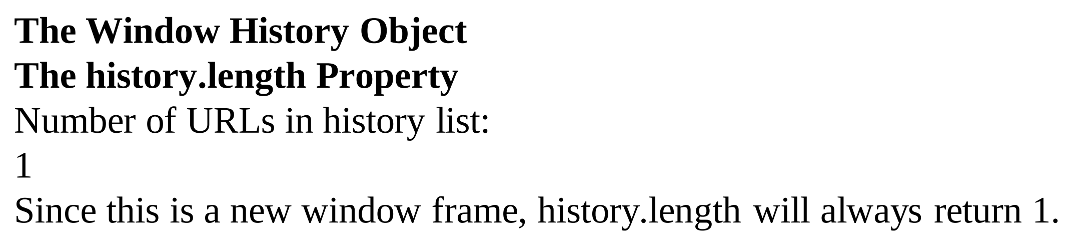
Property of Javascript History Object
1.**length:**Returns the length of the history URL's.

Methods of Javascript History Object

- ✓ **forward():** loads the next page.
- ✓ **back()** :loads the previous page.
- ✓ **go()**   :loads the given page number.

Example

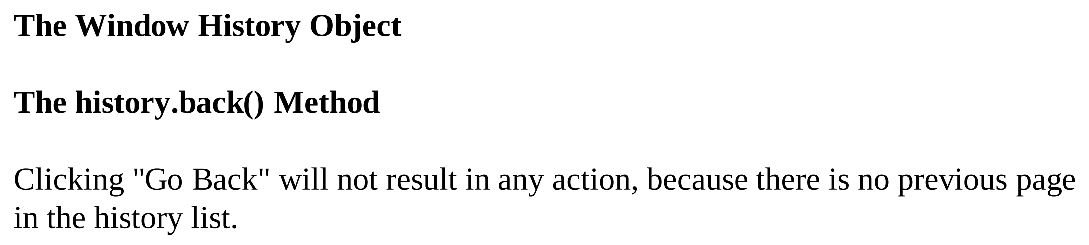
```
<html>
<body>

<h1>The Window History Object</h1>
<h2>The history.length Property</h2>

<p>Number of URLs in history list:</p>

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

<p>Since this is a new window frame, history.length will always return 1.</p>

<script>
let length = window.history.length;
document.getElementById("demo").innerHTML = length;
</script>

</body>
</html>
```



The Window History Object
The history.length Property
Number of URLs in history list:
1
Since this is a new window frame, history.length will always return 1.

back() method

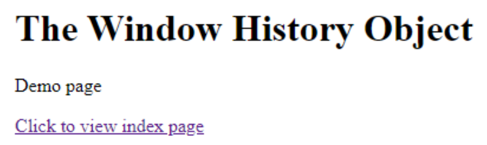
```
<html>
<body>

<h1>The Window History Object</h1>
<h2>The history.back() Method</h2>

<button onclick="history.back()">Go Back</button>

<p>Clicking "Go Back" will not result in any action, because there is no previous page in the
history list.</p>

</body>
</html>
```


The Window History Object

The history.back() Method

Clicking "Go Back" will not result in any action, because there is no previous page in the history list.

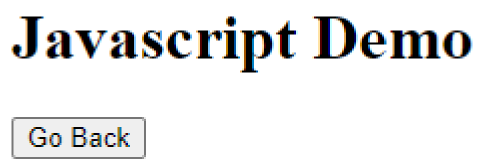
```
<html>
<body>

<h1>The Window History
Object</h1>

<p>Demo page</p>

<a href="index.html">Click to
view index page </a>
</body>
</html>
```


The Window History Object

Demo page

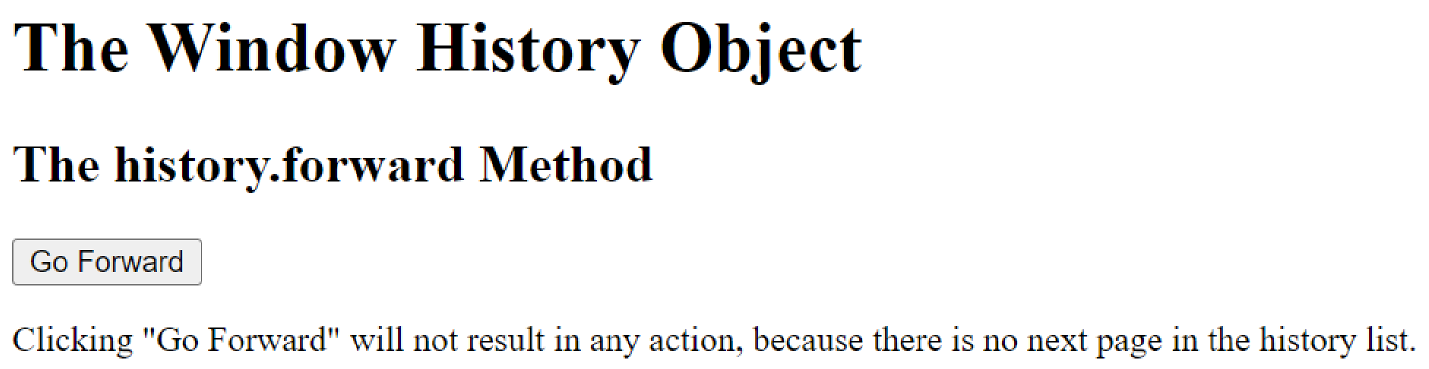
Click to view index page

```
<html>
<body>

<h1>Javascript Demo</h1>
<input type="button" value="Go Back" onclick="goback()"/>
<script>
 function goback() {
                history.back();
}
</script>
</body>
</html>
```


Javascript Demo

forward() method

```
<html>
<body>

<h1>The Window History Object</h1>
<h2>The history.forward Method</h2>

<button onclick="history.forward()">Go Forward</button>

<p>Clicking "Go Forward" will not result in any action, because there is no next page in the
history list.</p>

</body>
</html>
```


The Window History Object

The history.forward Method

Clicking "Go Forward" will not result in any action, because there is no next page in the history list.

Screen Object

- ➢ The JavaScript screen object holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc.

- ➢ The navigator object is the window property, so it can be accessed by:window.screen .

Properties:

- ✓ width:returns the width of the screen
- ✓ height:returns the height of the screen
- ✓ availWidth:returns the available width
- ✓ availHeight:returns the available height
- ✓ colorDepth:returns the color depth
- ✓ pixelDepth:returns the pixel depth.

Example

```
<html>
<body>
<script>
document.writeln("<br/>screen.width: "+screen.width);
document.writeln("<br/>screen.height: "+screen.height);
document.writeln("<br/>screen.availWidth: "+screen.availWidth);
document.writeln("<br/>screen.availHeight: "+screen.availHeight);
document.writeln("<br/>screen.colorDepth: "+screen.colorDepth);
document.writeln("<br/>screen.pixelDepth: "+screen.pixelDepth);
</script>
</body>
</html>
```

```
screen.width: 1280
screen.height: 720
screen.availWidth: 1280
screen.availHeight: 680
screen.colorDepth: 24
screen.pixelDepth: 24
```

Location Object

- The location object contains information about the current URL.
- The location object is a property of the window object.
- The location object is accessed with:

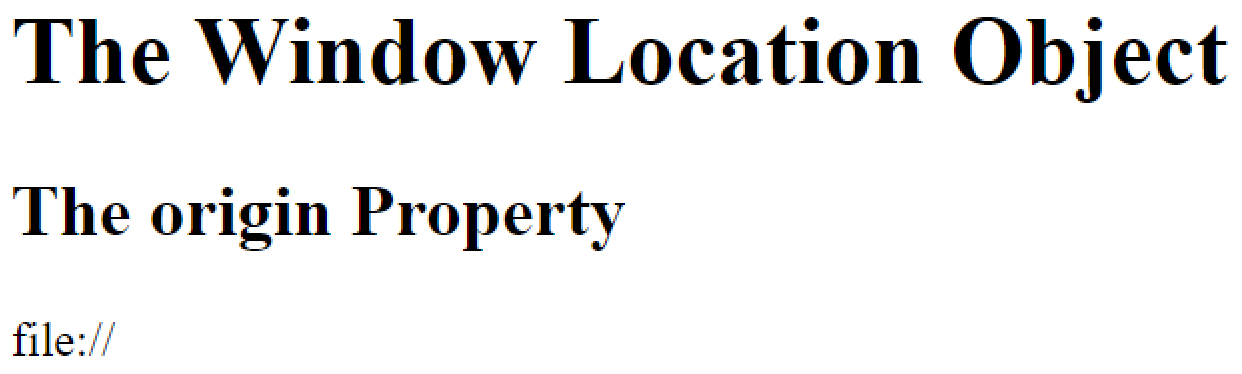
window.location or just location

The location object provides information about the current URL of the page.

```
console.log(location.href); // Output: Current URL   ///C:/Users/DELL/OneDrive/Desktop/FSD-New/Javascript.html
```

Example

```
<html>
<body>

<h1>The Window Location Object</h1>
<h2>The origin Property</h2>

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

<script>
let origin = window.location.origin;
document.getElementById("demo").innerHTML = origin;
</script>

</body>
</html>
```



The Window Location Object

The origin Property

file://

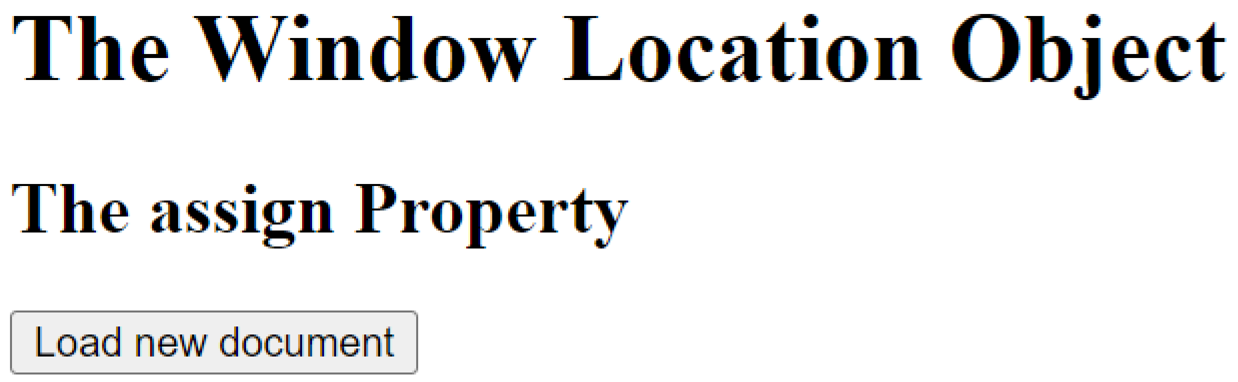
Location Object Methods

The following are the methods:

✓ assign()	:Loads a new document
✓ reload()	:Reloads the current document
✓ replace()	:Replaces the current document with a new one

assign()

```
<html>
<body>

<h1>The Window Location Object</h1>
<h2>The assign Property</h2>

<button onclick="myFunction()">Load new document</button>

<script>
function myFunction() {
  location.assign("https://www.itvedant.com");
}
</script>

</body>
</html>
```

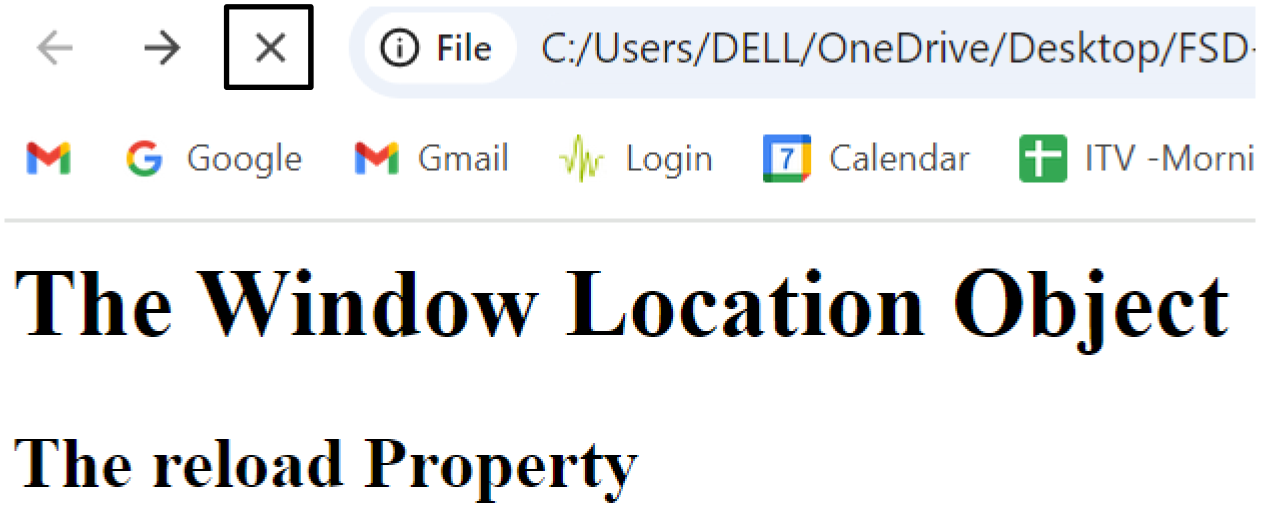


The Window Location Object

The assign Property

reload()

```
<html>
<body>

<h1>The Window Location Object</h1>
<h2>The reload Property</h2>

<script>
location.reload();
</script>

</body>
</html>
```

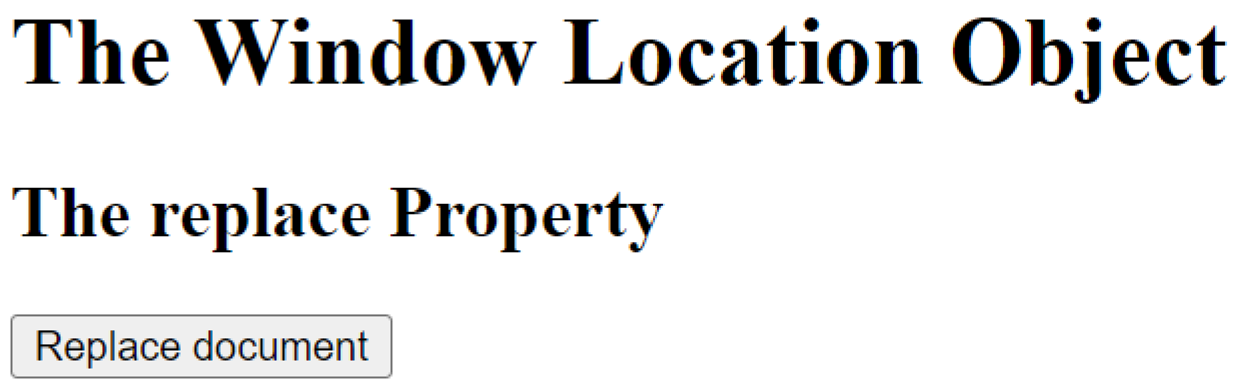


The Window Location Object

The reload Property

replace() method

```
<html>
<body>

<h1>The Window Location Object</h1>
<h2>The replace Property</h2>

<button onclick="myFunction()">Replace document</button>

<script>
function myFunction() {
  location.replace("https://www.itvedant.com")
}
</script>

</body>
</html>
```



The Window Location Object

The replace Property

Dynamic Property Syntax in Object Literals

JavaScript allows you to define object properties dynamically using variables or expressions.

Example:

```
const key = "age";
const person = {
    name: "John",
    [key]: 30  // Dynamically assign the 'age' property
};

document.write(person.age);  // Output: 30
```

In this example, the value of key is used as the property name.

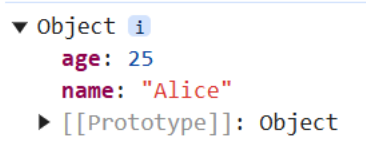
Shorter Property Syntax in Object Literals

In ES6, JavaScript introduced a shorthand syntax for defining properties when the property name and the variable name are the same.

Example:

```
const name = "Alice";
const age = 25;
const person = { name, age };  // Shorter syntax for name: name, age: age

console.log(person);  // Output: { name: "Alice", age: 25 }
```

This shorthand reduces repetition and makes the code cleaner.



```
▼ Object i
    age: 25
    name: "Alice"
  ▶ [[Prototype]]: Object
```

MCQ

MULTIPLE CHOICE QUESTION

JavaScript

The word “document” mainly refers to _____________

A. Dynamic Information

B. Static Information

C. Both Dynamic and Static Information

D. Temporary information

JavaScript

Which object is the main entry point to all client-side JavaScript features and APIs?

A. Standard
B. Location
C. Window
D. Position

JavaScript

Which property in the Window object is used to refer to a Location object?

A. position
B. area
C. window
D. location

JavaScript

Which Window object method is used to display a message in a dialog box?

A. alert()
B. prompt()
C. message()
D. console.log

JavaScript

How to pick a document element based on the value of its id attribute?

A. getElementsbyId()
B. getElementbyId()
C. both getElementsbyId() and getElementbyId()
D. getElement

DOM – Document Object Model

DOM – Document Object Model

What is the DOM?

The DOM is an interface provided by the browser for developers to interact with the page's structure. When a web page is loaded, the browser parses the HTML, CSS, and JavaScript into a DOM, which is an in-memory representation of the page. The DOM can be modified using JavaScript, allowing you to dynamically change the content, structure, and style of a web page after it is loaded.

- In simple terms, the DOM is a tree of objects (nodes) that represents the structure of an HTML document.

DOM and Browser
- When a browser loads a web page, it creates a DOM of the page, which JavaScript can interact with.
- The DOM tree represents the page's structure as a hierarchy of objects.
- Through JavaScript, we can read, modify, delete, and add elements to this tree in real time.

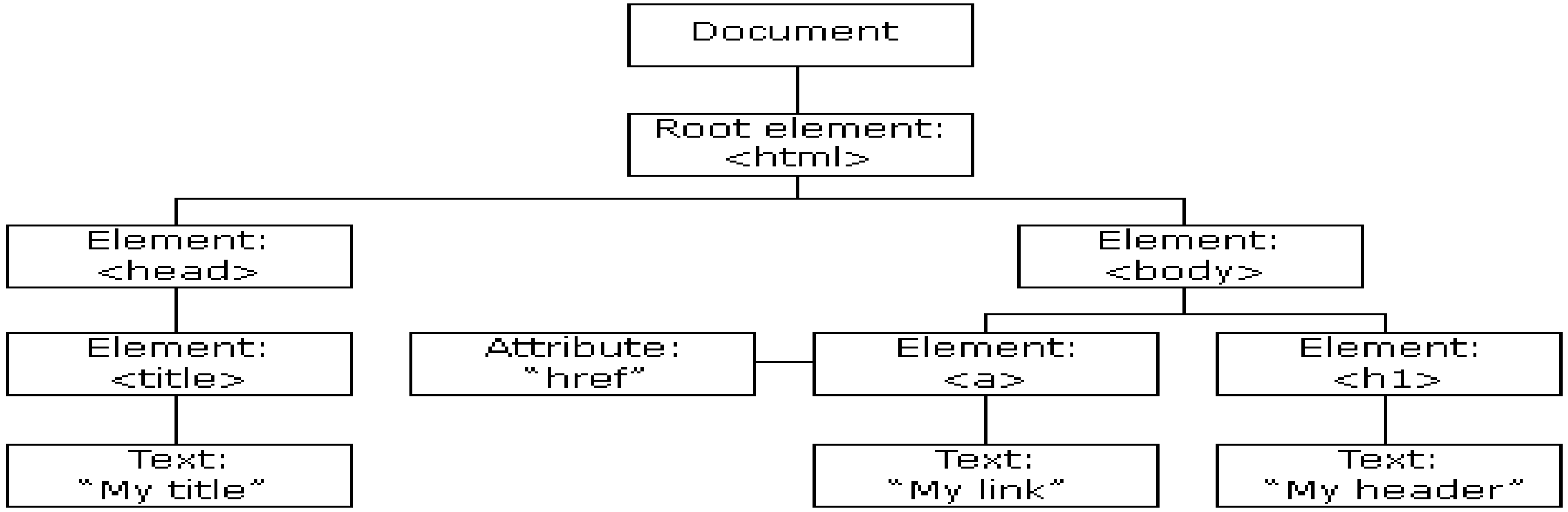
DOM Tree

DOM Structure and Hierarchy

The HTML DOM (Document Object Model)
When a web page is loaded, the browser creates a **Document Object Model** of the page. The **HTML DOM** model is constructed as a tree of **Objects**:

The DOM follows a tree-like structure, which can be broken down into several layers of nodes, each of which can contain other nodes. Here's a breakdown of the different types of nodes and their hierarchy:

- Document Node: This is the root of the DOM tree, representing the entire HTML document.
- Element Nodes: These represent HTML elements such as <html>, <body>, <div>, etc.
- Text Nodes: These represent the actual text content inside HTML elements.
- Attribute Nodes: These represent the attributes of HTML elements (e.g., id, class, etc.).



DOM

Hierarchy Example:
Let's take a look at a simple HTML document:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <div id="content">
      <p class="message">Hello, DOM!</p>
    </div>
  </body>
</html>
```

The corresponding DOM hierarchy would look like this:

```
Document
 └── html
      ├── head
      │    └── title
      └── body
           └── div (id="content")
                └── p (class="message")
                     └── Text ("Hello, DOM!")
```

In this DOM tree:

- Document is the root node.
- The html element is the child of the document node.
- Inside the html element, we have two children: head and body.
- The body contains a div element, which in turn contains a p element, and inside the p element is the text node ("Hello, DOM!").

Accessing and Manipulating DOM Elements

JavaScript allows you to access and manipulate any part of the DOM, such as adding, removing, or updating elements. There are several methods to access DOM elements, and these methods allow you to interact with the document's structure.

Accessing DOM Elements

getElementById(id): This method returns the element with the specified id attribute.
let content = document.getElementById('content');

getElementsByClassName(className): This method returns a live collection of elements with the specified class name.
let messages = document.getElementsByClassName('message');

getElementsByTagName(tagName): This method returns a live collection of elements with the specified tag name.
let paragraphs = document.getElementsByTagName('p');

querySelector(selector): This method returns the first element that matches the specified CSS selector.
let firstParagraph = document.querySelector('p');

querySelectorAll(selector): This method returns a NodeList of all elements that match the specified CSS selector.
let allParagraphs = document.querySelectorAll('p');

Manipulating DOM Elements

Manipulating DOM Elements
Once you've accessed an element, you can modify its properties, styles, content, and more.

Changing Content:

- **innerText:** Sets or retrieves the text content of an element, excluding HTML tags.
- **innerHTML:** Sets or retrieves the HTML content inside an element.

```
let paragraph = document.getElementById('content');
paragraph.innerText = 'New content here!';
paragraph.innerHTML = '<strong>Bold content!</strong>';
----------------------------------------------------------------------
<p id="content">Original Content</p>
  <br>
  <button onclick="changeText()">Set Text</button>
  <button onclick="changeHTML()">Set HTML</button>
  <script>
    function changeText() {
      let paragraph = document.getElementById('content');
      paragraph.innerText = 'New content here!';
    }
    function changeHTML() {
      let paragraph = 
document.getElementById('content');
      paragraph.innerHTML = '<strong>Bold 
content!</strong>';
    }
  </script>
```

Changing Attributes:
setAttribute(attributeName, value): Sets an attribute value.
getAttribute(attributeName): Retrieves the value of a specified attribute.

```
let div = document.querySelector('div');
div.setAttribute('id', 'newContent');
let id = div.getAttribute('id');
console.log(id); // Output: newContent
```

Changing Styles:
style.property: Modifies the inline styles of an element.

```
let div = document.querySelector('div');
div.style.backgroundColor = 'blue';
div.style.color = 'white';
```

DOM Manipulation Methods

JavaScript provides a wide range of methods to manipulate the DOM structure itself, not just the content or style of elements. These methods allow you to add, remove, or replace elements in the DOM.

1. **createElement(tagName):** Creates an element node with the specified tag name.
let newDiv = document.createElement('div');

2. **appendChild(childNode):** Appends a child node to the parent node.
let parentDiv = document.getElementById('parent');
parentDiv.appendChild(newDiv);

3. **removeChild(childNode):** Removes a specified child node from the parent node.
let parentDiv = document.getElementById('parent');
let childDiv = document.getElementById('child');
parentDiv.removeChild(childDiv);

4. **replaceChild(newNode, oldNode):** Replaces the old node with the new node.
let newDiv = document.createElement('div');
let oldDiv = document.getElementById('oldDiv');
document.body.replaceChild(newDiv, oldDiv);

Example for replaceChild

```
<!DOCTYPE html>
<html lang="en">
<head>
       <title>replaceChild Example</title>
</head>
<body>
    <div id="oldDiv" style="padding: 10px; background-
color: lightcoral;">
        This is the old div.
    </div>

    <button onclick="replaceDiv()">Replace Div</button>

    <script>
        function replaceDiv() {
            let newDiv = document.createElement('div');
            newDiv.textContent = "This is the new div.";
            newDiv.style.padding = "10px";
            newDiv.style.backgroundColor = "lightgreen";

            let oldDiv =
document.getElementById('oldDiv');
            document.body.replaceChild(newDiv, oldDiv);
        }
    </script></body></html>
```

DOM Manipulation Methods

5. cloneNode(deep): Clones the specified node. If the deep parameter is true, it clones the node and all its descendants.
let clonedDiv = newDiv.cloneNode(true);

6. **setAttribute(attributeName, value):** Sets an attribute for an element.
newDiv.setAttribute('id', 'newContent');

7. **classList.add(className):** Adds a class to an element.
newDiv.classList.add('highlight');

8. **classList.remove(className):** Removes a class from an element.
newDiv.classList.remove('highlight');

```
<!DOCTYPE html>
<html lang="en">
<head> <title>DOM Manipulation Example</title>
    <style>
        .highlight {
            background-color: yellow;
            padding: 10px;
            border: 2px solid black;          }
    </style> </head> <body>
    <div id="originalDiv" class="highlight">This is the
original div</div>
    <button onclick="manipulateDOM()">Click to Clone and
Modify</button> <script>
        function manipulateDOM() {
            // Select the existing div
            let originalDiv =
document.getElementById('originalDiv');
            // Clone the original div (deep cloning with child
nodes)
            let clonedDiv = originalDiv.cloneNode(true);
            // Set a new attribute to the cloned div
            clonedDiv.setAttribute('id', 'newContent');
            // Add a new class to the cloned div
            clonedDiv.classList.add('highlight');
            // Append the cloned div to the body
            document.body.appendChild(clonedDiv);
            // Wait for 2 seconds, then remove the 'highlight'
class
            setTimeout(() => {
                clonedDiv.classList.remove('highlight');
            }, 2000); } </script> </body> </html>
```

Event Handling in JavaScript

Events are actions that occur in the browser, such as clicks, key presses, and form submissions. JavaScript allows you to attach event listeners to elements to respond to these events.

Event Listeners

Event listeners are functions that wait for a specific event to happen on a DOM element, and when that event occurs, the function is executed.

```
let button = document.querySelector('button');
button.addEventListener('click', function() {
  alert('Button clicked!');
});
```

- The addEventListener method takes two parameters:
  - The type of event (e.g., click, submit, etc.).
  - The callback function that will be executed when the event occurs.

Removing Event Listeners

To remove an event listener, use the removeEventListener() method.

```
button.removeEventListener('click', function() {
  alert('Button clicked!');
});
```

```
<!DOCTYPE html>
<html lang="en">
<head>
       <title>Button Click Example</title>
</head>
<body>
    <button>Click Me</button>
    <script>
        let button =
document.querySelector('button');
        button.addEventListener('click',
function() {
            alert('Button clicked!');
        });
    </script>
</body>
</html>
```

Event Handling in JavaScript

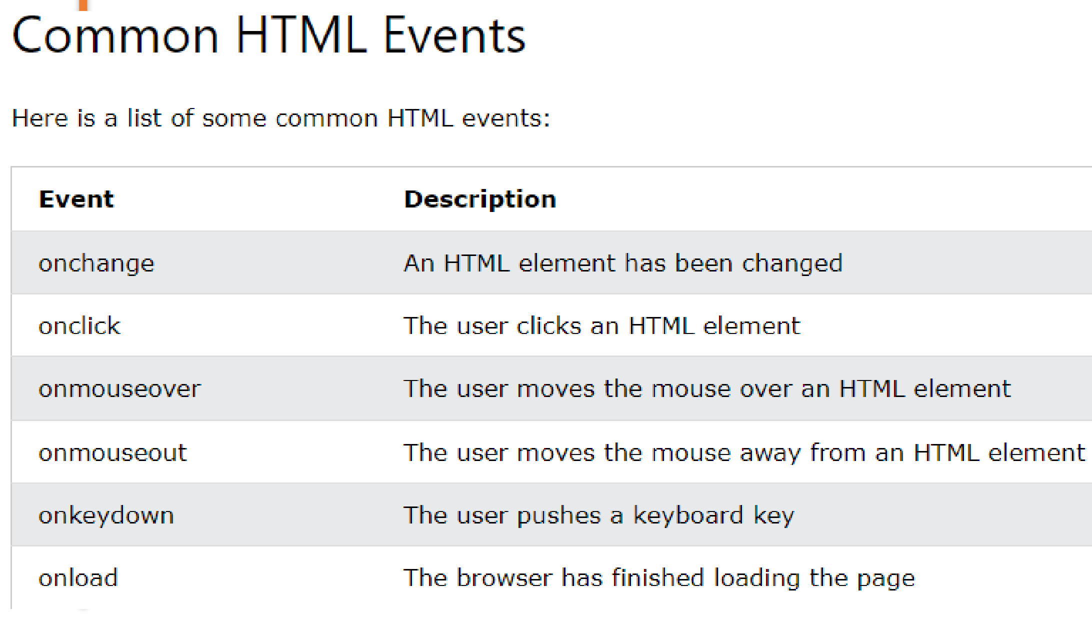
Event Types
JavaScript supports a variety of event types. Some common ones include:

Mouse Events:
click: Triggered when the mouse button is pressed and released on an element.
dblclick: Triggered when the mouse button is clicked twice in quick succession.
mouseover: Triggered when the mouse pointer enters an element.
mouseout: Triggered when the mouse pointer leaves an element.

Keyboard Events:
keypress: Triggered when a key is pressed down (but not released).
keydown: Triggered when a key is pressed down.
keyup: Triggered when a key is released.

Form Events:
submit: Triggered when a form is submitted.
focus: Triggered when an element (like an input field) gains focus.
blur: Triggered when an element loses focus.

Window Events:
load: Triggered when the page finishes loading.
resize: Triggered when the window is resized.



Common HTML Events

Here is a list of some common HTML events:

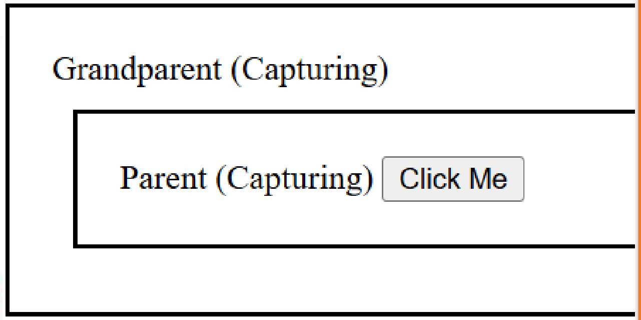
Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

Event Propagation and Delegation

Event Propagation
When an event occurs, it goes through a process called propagation. There are two phases:

Capturing Phase: The event starts from the root of the DOM tree and travels down to the target element.
Bubbling Phase: The event starts from the target element and bubbles up to the root.
By default, events use the bubbling phase, but you can enable the capturing phase by setting the third parameter of addEventListener() to true.

```
button.addEventListener('click', function() {
  alert('Button clicked!');
}, true); // This enables capturing phase
```



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Event Capturing Example</title>
    <style> div {
            padding: 20px;
            margin: 10px;
            border: 2px solid black;      }      </style>
</head> <body>
    <div id="grandparent">
        Grandparent (Capturing)
        <div id="parent">
            Parent (Capturing)
            <button id="child">Click Me</button>
        </div>
    </div>
    <script>
        document.getElementById("grandparent").addEventListe
ner("click", function() {
            alert("Grandparent Clicked (Capturing Phase)");
        }, true); // Capturing phase enabled

        document.getElementById("parent").addEventListener("
click", function() {
            alert("Parent Clicked (Capturing Phase)");
        }, true); // Capturing phase enabled
        document.getElementById("child").addEventListener("c
lick", function() {
            alert("Button Clicked!");
        }, true); // Capturing phase enabled
    </script> </body> </html>
```

Event Propagation and Delegation

Event Delegation

Event delegation is a technique that involves adding a single event listener to a parent element instead of individual child elements. This is particularly useful when you have a large number of elements, such as a list of items, and you want to listen for events on those items.

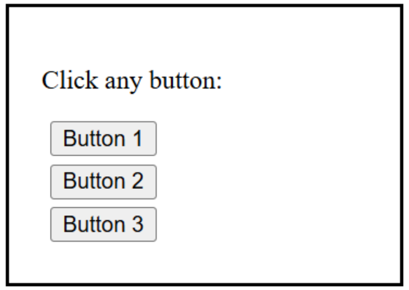
Event delegation works by taking advantage of event bubbling. When an event occurs on a child element, it bubbles up to the parent element, which can handle the event.

```
document.querySelector('#parent').addEventListener('click'
, function(event) {
  if (event.target && event.target.matches('button')) {
    alert('Button clicked!');
  }
});
```

In this example:

The event listener is attached to the **#parent** element.
If a **button** inside the parent is clicked, it triggers the event handler.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Event Delegation Example</title>
    <style>
        #parent {
            padding: 20px;
            border: 2px solid black;
            width: 200px;
        }
        button {
            display: block;
            margin: 5px;
        }
    </style></head>
<body>
    <div id="parent">
        <p>Click any button:</p>
        <button>Button 1</button>
        <button>Button 2</button>
        <button>Button 3</button>
    </div>
    <script>
        document.querySelector('#parent').addEventListener('
click', function(event) {
            if (event.target &&
event.target.matches('button')) {
                alert(event.target.innerText + ' clicked!');
            }
        });
    </script></body></html>
```


Click any button:

MCQ

MULTIPLE CHOICE QUESTION

JavaScript

How is everything treated in HTML DOM?

A. Node

B. Attributes

C. Elements

D. Arrays

JavaScript

How are the objects organized in the HTML DOM?

A. Class-wise

B. Queue

C. Hierarchy

D. Stack

JavaScript

Which object is the top of the hierarchy?

A. Window Object
B. Document Object
C. Form Object
D. Form Control Elements

JavaScript

Identify the correct code in order to fetch the value entered in username text field?

```
< body>
< form name="login">
Enter Your Name< input value="Akash" id="p_name" name="uname">
< /form>
< /body>
```

A. document.login.uname.value

B. document.getElementById ("p_name").value

C. document.getElementByName ("name").value

D. Both A and B

JavaScript

The HTML DOM is a standard object model and programming interface for HTML.

A. TRUE

B. FALSE

C. Can be true or false

D. Can not say

Modern JavaScript

Modern JavaScript

1. Template Strings (Template Literals)

Definition: Template strings (also called template literals) are a feature introduced in ECMAScript 6 (ES6) that allows you to embed expressions inside strings. They provide an easier way to work with strings and can span multiple lines without the need for escape characters.

Basic Syntax

Template strings are enclosed in backticks (\``), and you can embed expressions using ${}`.

Basic Example:

```
let name = "John";
let age = 25;
let greeting = `Hello, my name is ${name} and I am ${age} years old.`;
console.log(greeting);
// Output: Hello, my name is John and I am 25 years old.
```

Real-World Scenario: Template literals are useful in generating dynamic HTML content or rendering a string with variable data, like building messages, invoices, or content in a web application.

Multi-line Strings Example:

```
let message = `This is a message
spanning multiple lines.
It makes code more readable.`;
console.log(message);
```

Expression Inside Template Literals:

You can use variables, arithmetic operations, or even function calls inside template strings.

```
let num1 = 5;
let num2 = 10;
let result = `The sum of ${num1} and ${num2} is ${num1 +
num2}.`;
console.log(result);
// Output: The sum of 5 and 10 is 15.
```

Modern JavaScript

2. Destructuring Arrays and Objects

Definition: Destructuring allows you to unpack values from arrays or properties from objects into distinct variables. It simplifies your code by allowing you to extract multiple values from complex structures in a single statement.

Array Destructuring:

Basic Syntax:

```
let arr = [1, 2, 3];
let [a, b, c] = arr;
console.log(a, b, c);  // Output: 1 2 3
```

Skipping Items:

```
let arr = [1, 2, 3, 4];
let [, , c] = arr;  // Skips the first two elements
console.log(c);  // Output: 3
```

Default Values:

```
let arr = [1, 2];
let [a, b = 5] = arr;  // b has a default value of 5
console.log(b);  // Output: 2
```

Rest Pattern:

```
let arr = [1, 2, 3, 4];
let [first, second, ...rest] = arr;
console.log(rest);  // Output: [3, 4]
```

Modern JavaScript

2. Destructuring Arrays and Objects

Basic Syntax:

```
let person = { name: "Alice", age: 30 };
let { name, age } = person;
document.write(name, age);  // Output: Alice 30
```

Renaming Variables:

```
let person = { name: "Alice", age: 30 };
let { name: personName, age: personAge } = person;
document.write(personName, personAge);  // Output: Alice 30
```

Default Values:

```
let person = { name: "Alice" };
let { name, age = 25 } = person;
document.write(age);  // Output: 25
```

Real-World Scenario: Destructuring is commonly used in JavaScript frameworks like React for handling props and state, allowing easy extraction of values from complex objects.

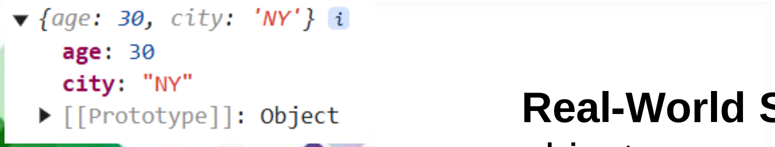
Modern JavaScript

3. Rest and Spread Operators

Rest Operator (...)
The rest operator is used to collect multiple elements into an array. It's commonly used in function parameters, array destructuring, and object destructuring.

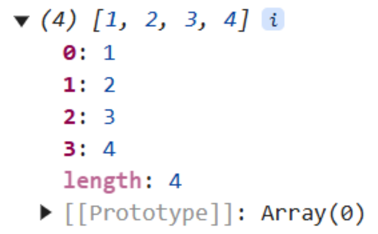
In Function Parameters:
```
function sum(...numbers) {
  return numbers.reduce((total, num) => total + num, 0);
}
document.write(sum(1, 2, 3, 4));  // Output: 10
```

In Object Destructuring:
```
let person = { name: "Alice", age: 30, city: "NY" };
let { name, ...rest } = person;
console.log(rest);  // Output: { age: 30, city: 'NY' }
```

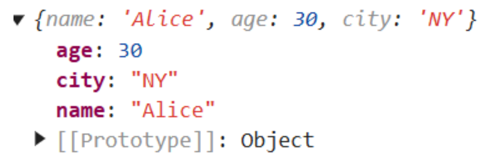

```
▼ {age: 30, city: 'NY'} i
    age: 30
    city: "NY"
  ▶ [[Prototype]]: Object
```

Spread Operator (...)
The spread operator is used to unpack elements from an array or object. It can be used in function calls, array literals, and object literals.

In Arrays:
```
let arr1 = [1, 2];
let arr2 = [...arr1, 3, 4];
console.log(arr2);  // Output: [1, 2, 3, 4]
```


```
▼ (4) [1, 2, 3, 4] i
    0: 1
    1: 2
    2: 3
    3: 4
    length: 4
  ▶ [[Prototype]]: Array(0)
```

In Objects:
```
let person = { name: "Alice", age: 30 };
let updatedPerson = { ...person, city: "NY" };
console.log(updatedPerson);  // Output: { name: "Alice", age: 30, city: "NY" }
```


```
▼ {name: 'Alice', age: 30, city: 'NY'}
    age: 30
    city: "NY"
    name: "Alice"
  ▶ [[Prototype]]: Object
```

Real-World Scenario: These operators are used for tasks like combining arrays, merging object properties, and handling variable-length arguments in functions.

Modern JavaScript

4. JavaScript Callbacks

Definition: A callback is a function passed into another function as an argument, which is then executed later, usually after some operation is complete.

Basic Example:

```
function greet(name, callback) {
document.write(`Hello, ${name}`);
  callback();
}

function sayGoodbye() {
 document.write("Goodbye!");
}

greet("Alice", sayGoodbye);
// Output:
// Hello, Alice
// Goodbye!
```

Real-World Scenario: Callbacks are widely used in asynchronous operations like reading files, making HTTP requests, or handling user interactions.

5. JavaScript Asynchronous Programming

Asynchronous programming allows JavaScript to perform tasks like reading files or making HTTP requests without blocking the execution of other code.

Callbacks in Asynchronous Programming:

```
function fetchData(callback) {
  setTimeout(() => {
    callback("Data fetched");
  }, 2000);
}

fetchData(data => {
  document.write(data);  // Output: Data fetched (after 2 seconds)
});
```

Real-World Scenario: Asynchronous programming is crucial in tasks like API calls, animations, or waiting for user input.

Modern JavaScript

6. JavaScript Promises

Definition: A Promise is an object representing the eventual completion (or failure) of an asynchronous operation. It is a better way to handle asynchronous code compared to callbacks.

Basic Syntax:

```
let promise = new Promise((resolve, reject) => {
  let success = true;
  if (success) {
    resolve("Operation successful!");
  } else {
    reject("Operation failed.");
  }
});

promise
  .then(result => document.write(result))
  .catch(error => document.write(error));
                                        // Output: Operation successful!
```

Chaining Promises:

```
let promise = new Promise((resolve, reject) =>
resolve("Step 1"))
  .then(result => {
    document.write(result);  // Output: Step 1
    return "Step 2";
  })
  .then(result => document.write(result))  // Output:
Step 2
  .catch(error => document.write(error));
```

Real-World Scenario: Promises are used for handling API responses, loading resources, and chaining multiple asynchronous tasks.

Modern JavaScript

7. JavaScript Async/Await

Definition: async and await make working with promises easier. async makes a function return a promise, and await pauses the execution of the function until the promise is resolved.

Basic Syntax:

```
async function fetchData() {
  let response = await fetch("https://api.example.com");
  let data = await response.json();
  document.write(data);
}
fetchData();
```

Real-World Scenario: Async/await simplifies code that involves chaining multiple asynchronous actions, such as fetching data from APIs or handling file operations.

Program in next page

Modern JavaScript

7. Working with Fetch API

Definition: The fetch() function is used to make network requests to retrieve or send data to a server asynchronously.

Basic Syntax:

```
fetch("https://api.example.com")
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error("Error:", error));
```

Using async/await with Fetch:

```
async function getData() {
  try {
    let response = await fetch("https://api.example.com");
    let data = await response.json();
    console.log(data);
  } catch (error) {
    console.error("Error:", error);
  }
}
getData();
```

Output

```
{
"userId": 1,
"id": 1,
"title": "delectus aut autem",
"completed": false
}
```

Real-World Scenario: The fetch() API is essential for making AJAX calls to server-side applications or consuming RESTful APIs.

```
<!DOCTYPE html>
<html lang="en">
<head>
     <title>Fetch API Example</title>
</head>
<body>
    <h2>Fetching Data Using Async/Await</h2>
    <div id="dataContainer">Loading...</div>

    <script>
        async function fetchData() {
            try {
                let response = await
fetch("https://jsonplaceholder.typicode.com/todos/1");
                let data = await response.json();
                document.getElementById("dataContainer")
.innerText = JSON.stringify(data, null, 2);
            } catch (error) {
                document.getElementById("dataContainer")
.innerText = "Error fetching data!";
                console.error("Error:", error);
            }
        }

        fetchData();
    </script>
</body>
</html>
```

(pretty-printed) JSON string - JSON.stringify()

JSON.stringify(value, replacer, space)

value → The object to be converted into a JSON string.
replacer → (Optional) A function or array that filters properties.
space → (Optional) The number of spaces used for indentation.

RESTful APIs

What are RESTful APIs?

A **RESTful API (Representational State Transfer API)** is a web service that follows REST (REpresentational State Transfer) principles. It allows different systems to communicate over the internet using **HTTP methods** like GET, POST, PUT, and DELETE.

Key Features of RESTful APIs:

Stateless – Each request from a client contains all the necessary information, and the server does not store client state.

Client-Server Architecture – The client and server are separate, allowing for scalability.

Resource-Based – Data is represented as resources (e.g., users, orders), accessed via URLs (Uniform Resource Locators).

Uses HTTP Methods:

GET – Read / Retrieve data.

POST – Create a new resource (Create/Read/Retrieve/Edit data)

PUT – Update an existing resource(Update Data)

DELETE – Remove a resource. (Delete Data)

Uses JSON or XML – Most REST APIs use JSON (JavaScript Object Notation) for data exchange.

Example of a REST API Endpoint:

Request:

GET https://api.example.com/users/1

Response (JSON):

```
{
  "id": 1,
  "name": "John Doe",
  "email": "john@example.com"
}
```

Why Use RESTful APIs?

Scalable – Supports multiple clients (web, mobile, etc.).
Flexible – Works with various programming languages.
Lightweight – Uses simple HTTP requests.

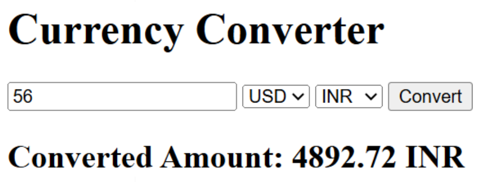
Modern JavaScript

8. Creating a Currency Converter Using Fetch API

Now, let’s use the fetch() function to create a simple currency converter that fetches data from an API and converts amounts between two currencies.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Currency Converter</title>
</head>
<body>
  <h1>Currency Converter</h1>
  <input type="number" id="amount" placeholder="Enter amount">
  <select id="fromCurrency">
    <option value="USD">USD</option>
    <option value="EUR">EUR</option>
    <option value="INR">INR</option>
  </select>
```



Currency Converter

56 USD INR Convert

Converted Amount: 4892.72 INR

```
  <select id="toCurrency">
    <option value="USD">USD</option>
    <option value="EUR">EUR</option>
    <option value="INR">INR</option>
  </select>
  <button id="convertButton">Convert</button>
  <h2 id="result"></h2>
<script>
    document.getElementById("convertButton").addEventListener("click",
convertCurrency);

    async function convertCurrency() {
      let amount = document.getElementById("amount").value;
      let fromCurrency = document.getElementById("fromCurrency").value;
      let toCurrency = document.getElementById("toCurrency").value;

      let url = `https://api.exchangerate-api.com/v4/latest/${fromCurrency}`;
      let response = await fetch(url);
      let data = await response.json();

      let conversionRate = data.rates[toCurrency];
      let convertedAmount = amount * conversionRate;

      document.getElementById("result").textContent = `Converted Amount:
${convertedAmount} ${toCurrency}`;
    }
  </script>
</body>
</html>
```

How it Works:

- The user enters an amount and selects the currencies they want to convert from and to.
- The fetch() function retrieves the current exchange rates from the ExchangeRate-API.
- The result is displayed dynamically based on the selected currencies.

Important

Form Validations

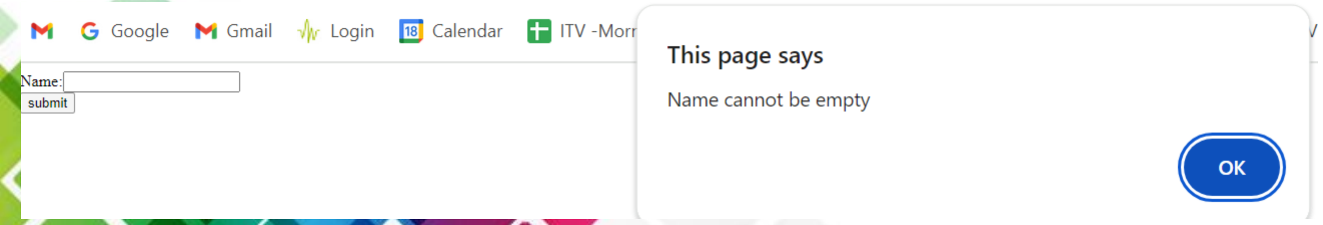
Name / username - Validation

Form Input

```
<form name="myform">

<Label>Name:</label><input type="text" name="fname" id =
"fname"/><br/>

<input type="submit" value="submit" onclick="validate()"/>

</form>
```

Method - 1

```
<script>
    function validate() {
    var n = document.myform.fname.value;
  if (n == "" || n == null) {
  alert("Name cannot be empty");
  }
}
</script>
```


Google Gmail Login Calendar ITV -Morr

Name:
submit

This page says

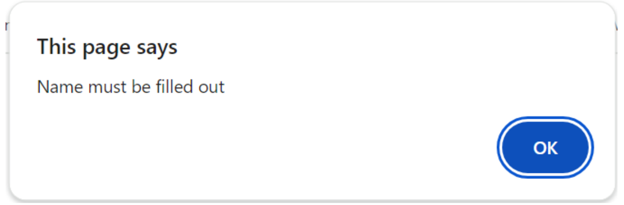
Name cannot be empty

OK

Input Validation

Method – 2

```
<script>
function validate() {
  let x = document.forms["myform"]["fname"].value;
  if (x == "") {
              alert("Name must be filled out");
    return false;
  }
}
</script>
```

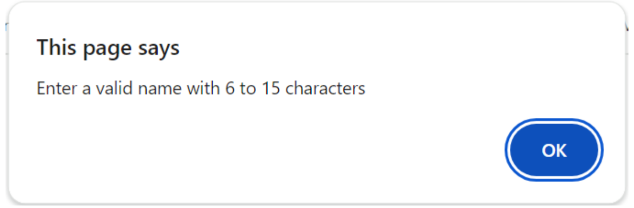

This page says

Name must be filled out

OK

Method – 3

```
<script>
function validate() {
var namelen = /^([a-zA-Z]{6,15})+$/;
var name = document.getElementById('fname').value;
if (!namelen.test(name)) {
alert("Enter a valid name with 6 to 15 characters");
return false;
}
}
</script>
```


This page says

Enter a valid name with 6 to 15 characters

OK

```
var username = /^([a-zA-Z0-9]{6,15})+$/;
```

Email ID - Validation

var email=/^(([a-zA-Z0-9\._]{6,16})+\@([a-z]{3,7})+\.([a-z A-Z]{2,3}))+$/;

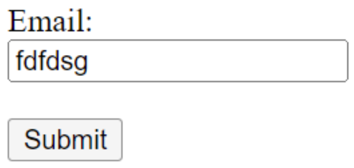
Form Input

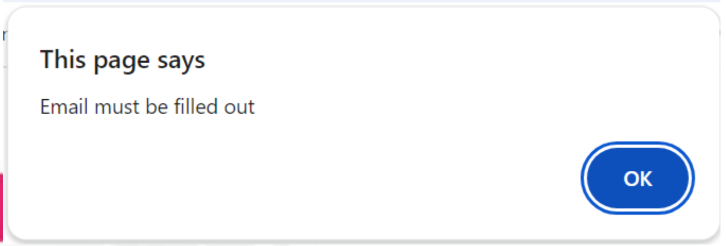
Method - 1

```
<form name="myform">

 <label>Email:</label><br>
 <input type="email" id="email" name="email"><br><br>
<input type="submit" value="submit" onclick="validate()"/>

</form>
```

Method – 2

```
<input type="text" id="email" name="email"><br><br>
<input type="submit" value="submit" onclick="validate()"/>
<script>
      function validate() {
         var z = document.myform.email.value;
            if (z == "" || z == null) {
           alert("Email cannot be empty");
      }
}
</script>
```


Email:
fdfdsg
Submit


This page says
Email must be filled out
OK

Input Validation

Method – 3

```
<form id="myForm">
        <label for="email">Email:</label><br>
        <input type="text"  id="email" name="email"><br><br>
        <input type="submit" value="Submit">
    </form>

    <script>
        document.getElementById("myForm").addEventListener("submit"
, function(event) {
            var emailPattern = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-
zA-Z]{2,}$/;
            var email = document.getElementById("email").value.trim(); //
Trim spaces

            if (!emailPattern.test(email)) {
                alert("Please enter a valid email address.");
                event.preventDefault(); // Prevent form submission if invalid
            } else {
                alert("Email is valid! Form submitted successfully.");
            }
        });
    </script>
```

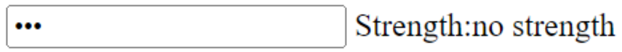
Password - Validation

Form Input

```
form name="myForm">
<input type="password" value="" name="userPass" onkeyup="validate()">
Strength:<span id="status">no strength</span>
</form>
```

Method – 1

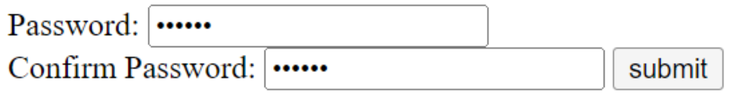
```
<script>
function validate() {
var pwd;
if(document.myForm.userPass.value.length>5){
pwd="Strong password";
}
else{
pwd="Weak password";
}
document.getElementById('status').innerText=pwd;
}
</script>
```

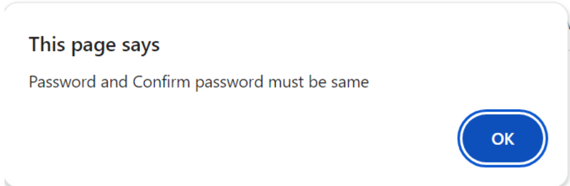

Strength:no strength

Input Validation

Method – 2

```
<form name="myform" id="myForm">
Password: <input type="password" value="" name="pass1"> <br/>
Confirm Password: <input type="password" value="" name="pass2">
<input type="submit" value="submit" onclick="validate()"/>
</form>
<script>
function validate() {
var password=document.myform.pass1.value;
var cpassword=document.myform.pass2.value;
if(password.length<6){
alert("Password must contain 6 characters");
return false;
}
if (password==cpassword) {
alert("Both passwords are same");
}
else {
alert("Password and Confirm password must be same");
}
}
</script>
```


Password: ••••••
Confirm Password: ••••••


This page says
Password and Confirm password must be same

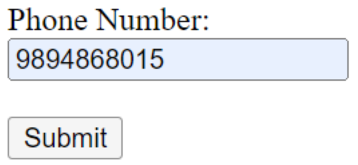
Phone Number - Validation

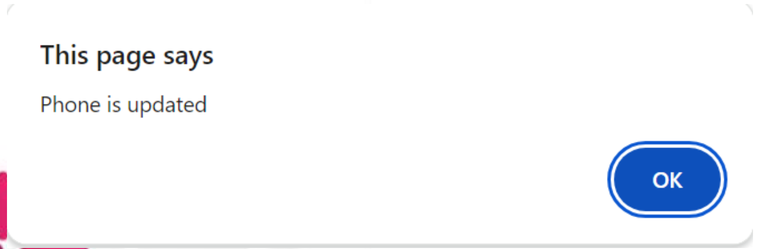
Form Input

```
<form name="myForm">
<label>Phone Number:</label><br>
<input type="text" id="phone" name="phone"><br><br>
<input type="submit" value="Submit" onclick="validate()">
</form>
```

Method – 1

```
<script>
function validate() {
var mobile = document.myForm.phone.value;
if (mobile.length == 10) {
alert("Thank you for providing your phone number");
} else {
alert("Enter a 10-digit mobile number");
return false;
}
}
</script>
```

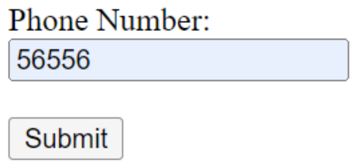

Phone Number:
9894868015

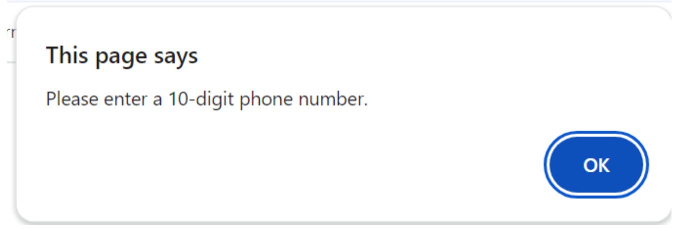

This page says
Phone is updated

Input Validation

Method – 2

```
<script>
function validate() {
var pn = document.forms["myForm"]["phone"].value;
var pptn = /^\d{10}$/; // Change this regex pattern based on your phone number format

if (!pptn.test(pn)) {
alert("Please enter a 10-digit phone number.");
return false;
}
alert("Phone is updated");}
</script>
```


Phone Number:
56556

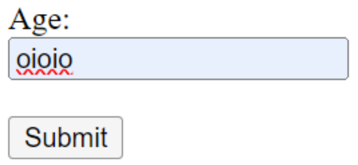

This page says
Please enter a 10-digit phone number.

Age / year of passing - Validation

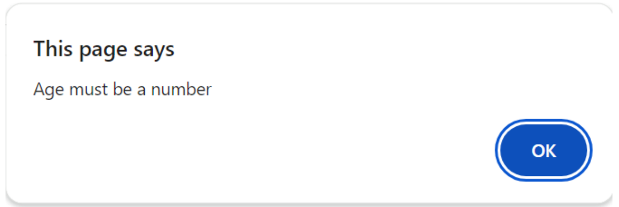
Form Input

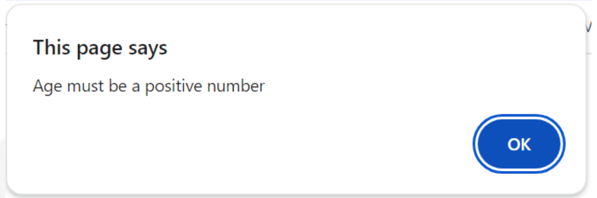
```
<form name="myForm">
<label for="age">Age:</label><br>
<input type="text" id="age" name="age"><br><br>
<input type="submit" value="Submit" onclick="validate()"> </form>
```

Method – 1


Age:
oioio
Submit

```
<script>
function validate() {
var age = document.forms["myForm"]["age"].value;
if (isNaN(age)) {
alert("Age must be a number");
return false;
}
if (age < 0) {
alert("Age must be a positive number");
return false;
}
// Additional validation rules can be added here based on your specific
requirements
return true;
}
</script>
```

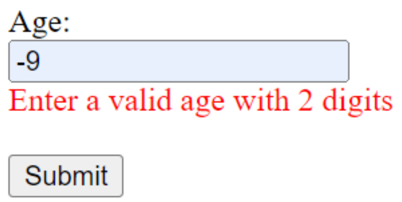

This page says
Age must be a number
OK


This page says
Age must be a positive number
OK

Input Validation

Method – 2

```
<form name="myForm" onsubmit="return validateForm()">
<label for="age">Age:</label><br>
<input type="text" id="age" name="age"><br>
<span id="spage" style="color: red;"></span><br><br>
<input type="submit" value="Submit">
</form>

<script>
function validateForm() {
var age = document.getElementById('age').value;
var agelen = /^([0-9]{2})+$/;

if (!agelen.test(age)) {
document.getElementById('spage').innerHTML = "Enter a valid
age with 2 digits";
return false;
} else {
document.getElementById('spage').innerHTML = "";
}
return true;
}
</script>
```


Age:
-9
Enter a valid age with 2 digits
Submit

var yearofpassing = /^([0-9]{4})+$/;

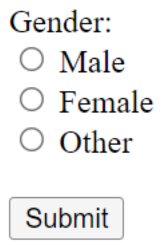
Gender (Radio button) - Validation

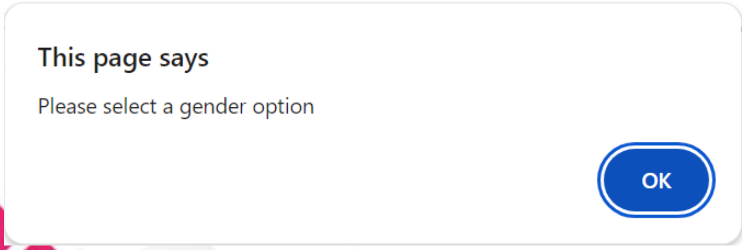
Form Input

```
<form name="myForm">
<label>Gender:</label><br> <input type="radio" id="male" name="gender"
value="male"> <label>Male</label><br>
<input type="radio" id="female" name="gender" value="female">
<label>Female</label><br>
<input type="radio" id="other" name="gender" value="other">
<label>Other</label><br><br>
<input type="submit" value="Submit" onclick="validate()">
</form>
```

Method – 1

```
<script>
function validate() {
var gender = document.forms["myForm"]["gender"].value;
if (!gender) {
alert("Please select a gender option");
return false;
}
return true;
}
</script>
```



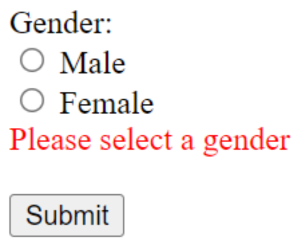

This page says

Please select a gender option

Input Validation

Method – 2

```
<form name="myForm" onsubmit="return validate()">
<label>Gender:</label><br>
<input type="radio" id="male" name="gen" value="male">
<label>Male</label><br>
<input type="radio" id="female" name="gen" value="female">
<label>Female</label><br>
<span id="spgen" style="color: red;"></span><br><br>
<input type="submit" value="Submit" onclick="validate()">
</form>
<script>
function validate() {
var d = document.getElementsByName('gen');
var k = 0;
for (var i = 0; i < d.length; i++) {
if (d[i].checked) {
k++;
}
}
if (k == 0) {
document.getElementById('spgen').innerHTML = "Please select a
gender";
return false;
} else {
document.getElementById('spgen').innerHTML = "";
}
return true;
}
</script>
```



Please select a gender

Qualification (Dropdown) - Validation

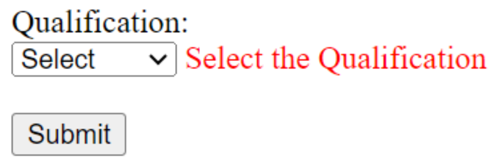
Form Input

```
<form name="myForm" onsubmit="return validate()">
<label for="quali">Qualification:</label><br>
<select id="quali">
<option value="select">Select</option>
<option value="bachelors">Bachelors</option>
<option value="masters">Masters</option>
<option value="phd">PhD</option>
</select>
<span id="spquali" style="color: red;"></span><br><br>
<input type="submit" value="Submit">
</form>
```

Input Validation

Method – 1

```
<script>
function validate() {
var selectedQualification = document.getElementById('quali').value;

if (selectedQualification == "select") {
document.getElementById('spquali').innerHTML = "Select the
Qualification";
return false;
} else {
document.getElementById('spquali').innerHTML = "";
return true;
}
}
</script>
```


Qualification:
Select Select the Qualification
Submit

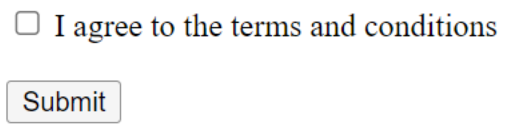
Input Validation

Skills (Checkbox) - Validation

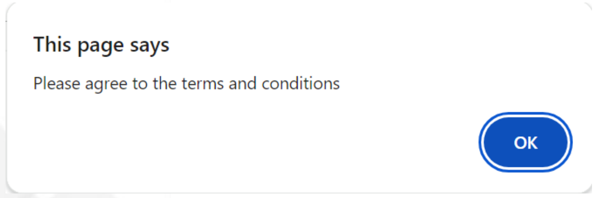
Form Input

Method – 1

```
<form name="myForm" onsubmit="return validateForm()">
<input type="checkbox" id="checkbox" name="checkbox">
<label for="checkbox">I agree to the terms and conditions</label><br>
<br/>
<input type="submit" value="Submit">
</form>

<script>
function validateForm() {
var checkbox = document.getElementById('checkbox');

if (!checkbox.checked) {
alert("Please agree to the terms and conditions");
return false;
} else {
alert("Thank you for agreeing the terms and conditions");
return true;
}
}
</script>
```



☐ I agree to the terms and conditions

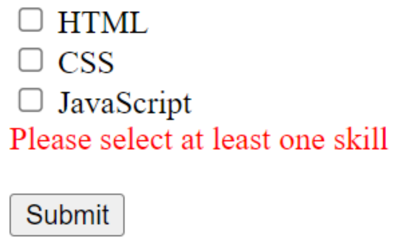


This page says

Please agree to the terms and conditions

Method – 2

```
<form name="myForm" onsubmit="return validate()">
<input type="checkbox" id="skill1" name="skill[]" value="HTML">
<label for="skill1">HTML</label><br>
<input type="checkbox" id="skill2" name="skill[]" value="CSS">
<label for="skill2">CSS</label><br>
<input type="checkbox" id="skill3" name="skill[]" value="JavaScript">
<label for="skill3">JavaScript</label><br>
<span id="spskill" style="color: red;"></span><br><br>
<input type="submit" value="Submit"> </form>
<script>
function validate() {
var skills = document.getElementsByName('skill[]');
var numChecked = 0;
for (var i = 0; i < skills.length; i++) {
if (skills[i].checked) {
numChecked++;
}
}
if (numChecked === 0) {
document.getElementById('spskill').innerHTML = "Please select at
least one skill";
return false;
} else {
document.getElementById('spskill').innerHTML = "";
return true;
}
}
</script>
```

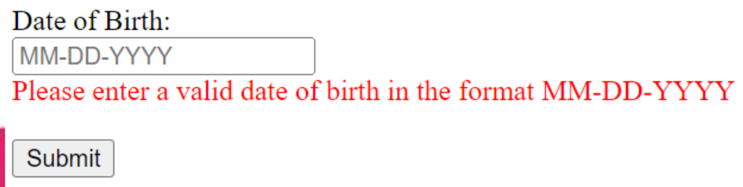


☐ HTML
☐ CSS
☐ JavaScript
Please select at least one skill

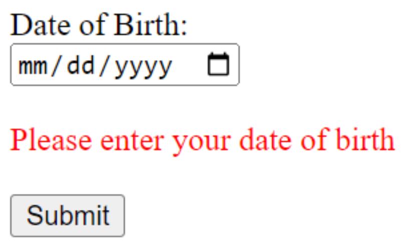
Date - Validation

Form Input
Method – 1

```
<form name="myForm" onsubmit="return validate()">
<label for="dob">Date of Birth:</label><br>
<input type="text" id="dob" name="dob" placeholder="MM-DD-
YYYY"><br>
<span id="spdob" style="color: red;"></span><br><br>
<input type="submit" value="Submit">
</form>

<script>
function validate() {
var dob = document.getElementById('dob').value;
var doblen = /^([0-9]{1,2}\-[0-9]{1,2}\-[0-9]{4})+$/;

if (!doblen.test(dob)) {
document.getElementById('spdob').innerHTML = "Please enter a valid
date of birth in the format MM-DD-YYYY";
return false;
} else {
document.getElementById('spdob').innerHTML = "";
return true;
}
}
</script>
```


Date of Birth:
MM-DD-YYYY
Please enter a valid date of birth in the format MM-DD-YYYY
Submit

Input Validation


Date of Birth:
mm/dd/yyyy
Please enter your date of birth
Submit

Method – 2

```
<form name="myForm" onsubmit="return validate()">
<label for="dob">Date of Birth:</label><br>
<input type="date" id="dob" name="dob"><br><br>
<span id="spdob" style="color: red;"></span><br><br>
<input type="submit" value="Submit">
</form>

<script>
function validate() {
var dob = document.forms["myForm"]["dob"].value;

if (dob === "") {
document.getElementById('spdob').innerHTML = "Please enter your
date of birth";
return false;
}

// You can add additional validation logic for the date format if needed

document.getElementById('spdob').innerHTML = "";
return true;
}
</script>
```

Age Calculation & Validation

Form Input

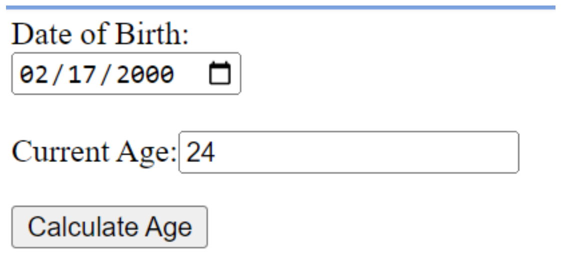
Method – 1

```
<form name="myForm" onsubmit="return validate()">
<label for="dob">Date of Birth:</label><br>
<input type="date" id="dob" name="dob"><br><br>
Current Age:<input type="text" id="age" readonly></p>
<input type="submit" value="Calculate Age">
</form>
```

Input Validation

Javascript

```
<script>
function validate() {
var dob = new Date(document.forms["myForm"]["dob"].value);
var today = new Date();
var age = today.getFullYear() - dob.getFullYear();
var monthDiff = today.getMonth() - dob.getMonth();
if (monthDiff < 0 || (monthDiff === 0 && today.getDate() <
dob.getDate())) {
age--;
}
alert("Age is: " + age);
document.getElementById("age").value = age;
return false;
}
</script>
```


Date of Birth:
02/17/2000

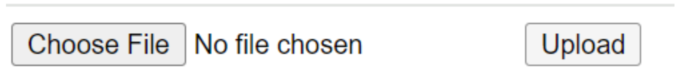
Current Age: 24

File Validation

Form Input

Method – 1

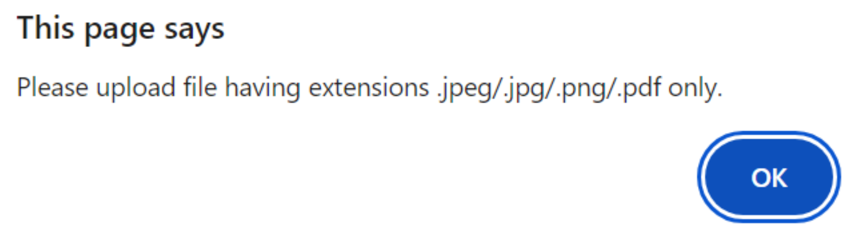
```
<form action="upload.php" method="post" enctype="multipart/form-data"
onsubmit="return validateFileUpload()">
    <input type="file" id="fileInput" name="fileInput">
    <input type="submit" value="Upload">
</form>
```



Input Validation

Javascript

```
<script>
function validateFileUpload() {
    var fileInput = document.getElementById('fileInput');
    var filePath = fileInput.value;
    var allowedExtensions = /(\.jpg|\.jpeg|\.png|\.pdf)$/i;
    if (!allowedExtensions.exec(filePath)) {
        alert('Please upload file having extensions .jpeg/.jpg/.png/.pdf
only.');
        fileInput.value = '';
        return false;
    }
    else {
        // File type is valid, you can submit the form or do further
processing
        return true;
    }
}
</script>
```

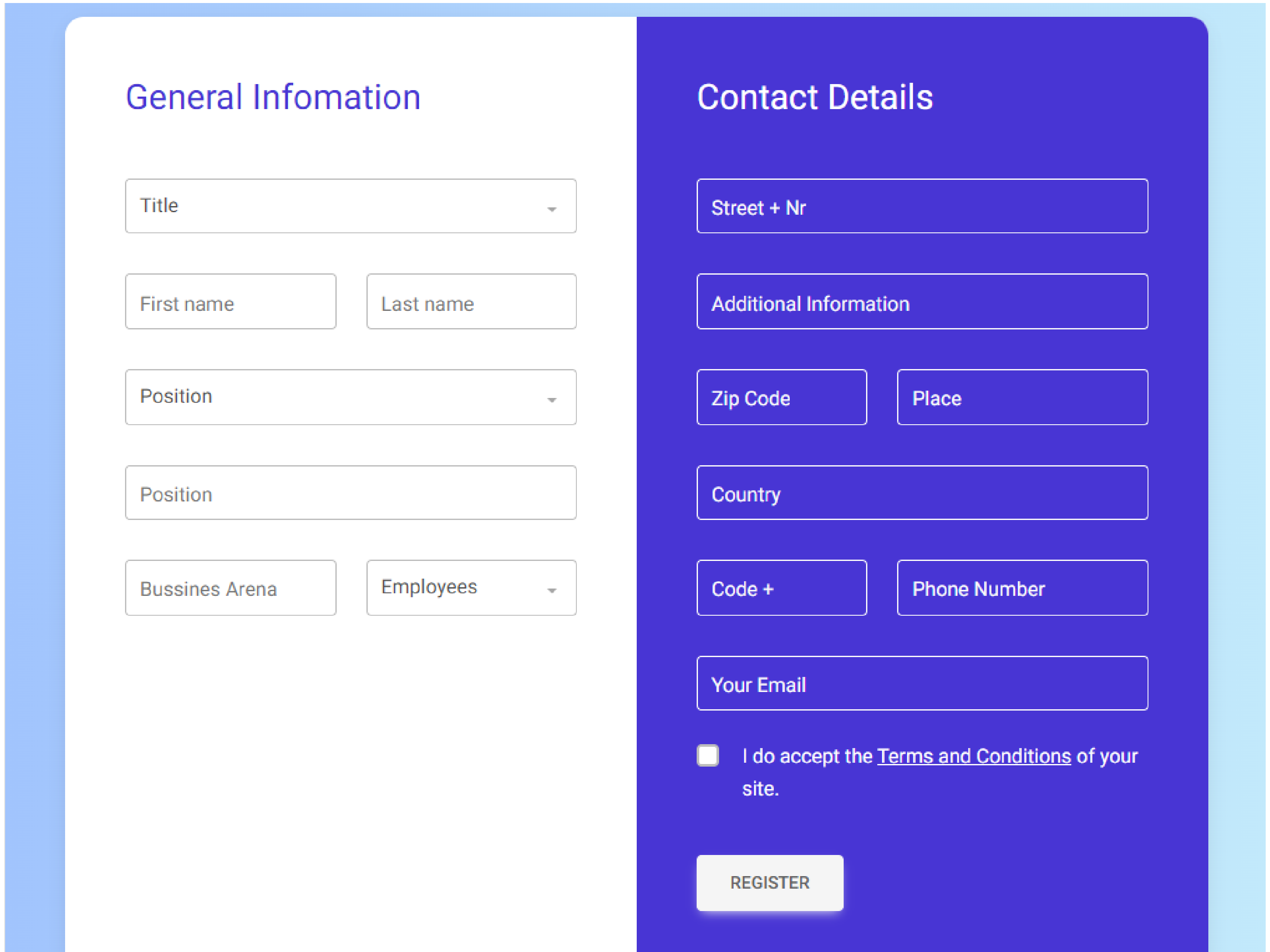

This page says

Please upload file having extensions .jpeg/.jpg/.png/.pdf only.

Projects

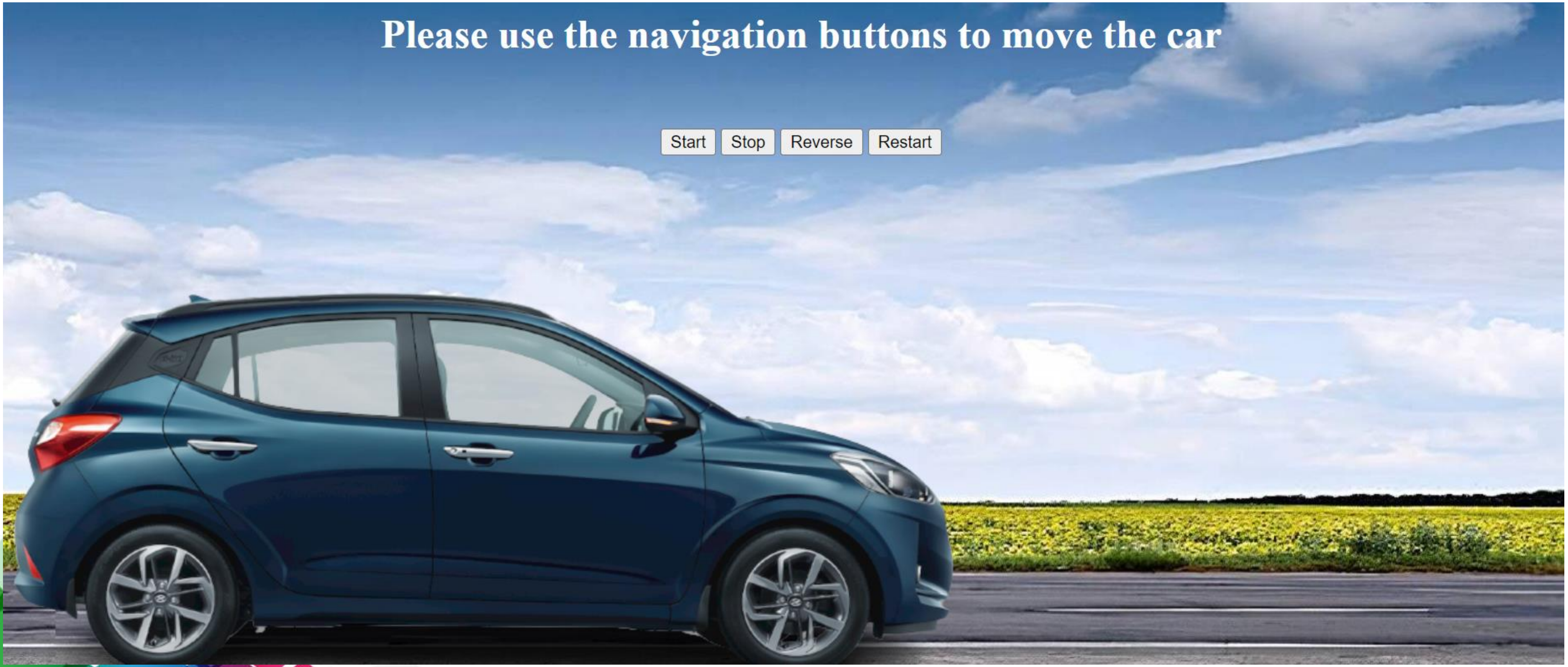
Task

Create a Student Registration Form like the below and add the Validation with JavaScript



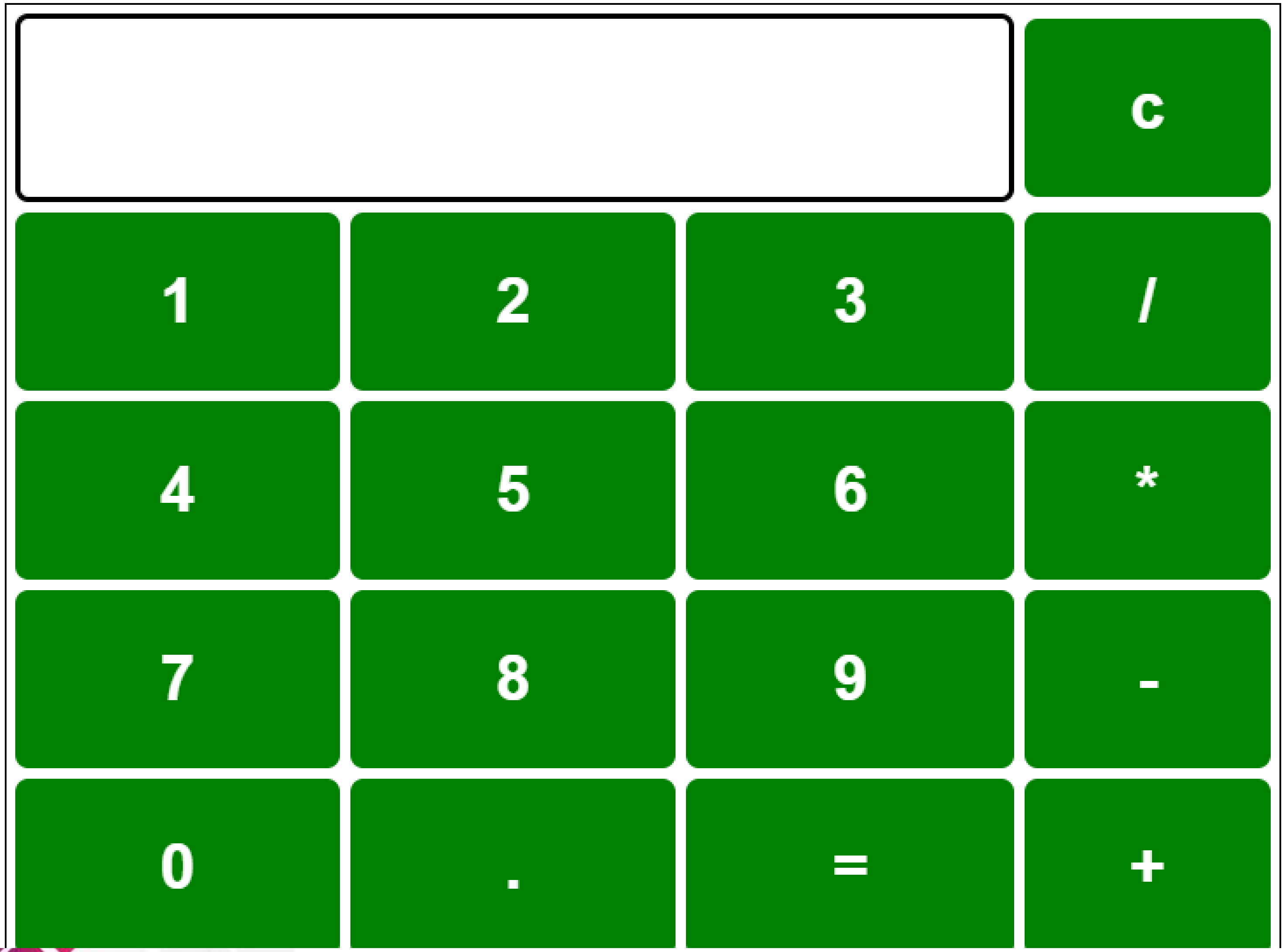
Task

Please create a simulation of a car like the below using HTML, CSS and JavaScript.



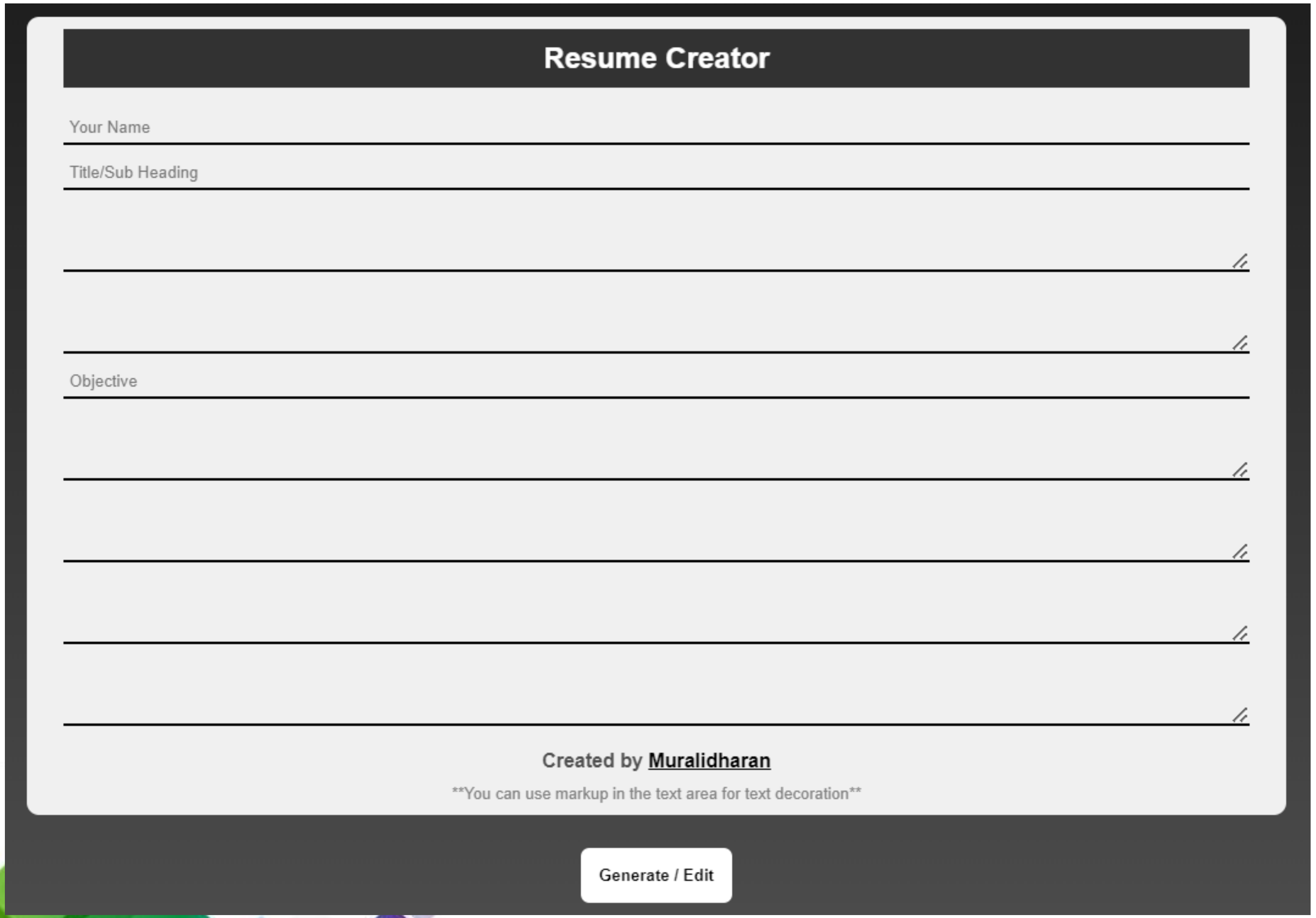
Task

Create a calculator using JavaScript



Task

Create a Resume Generator similar to the one below. Once you click the Generate Button, it should create the resume with print options like the right side image.



Resume Creator

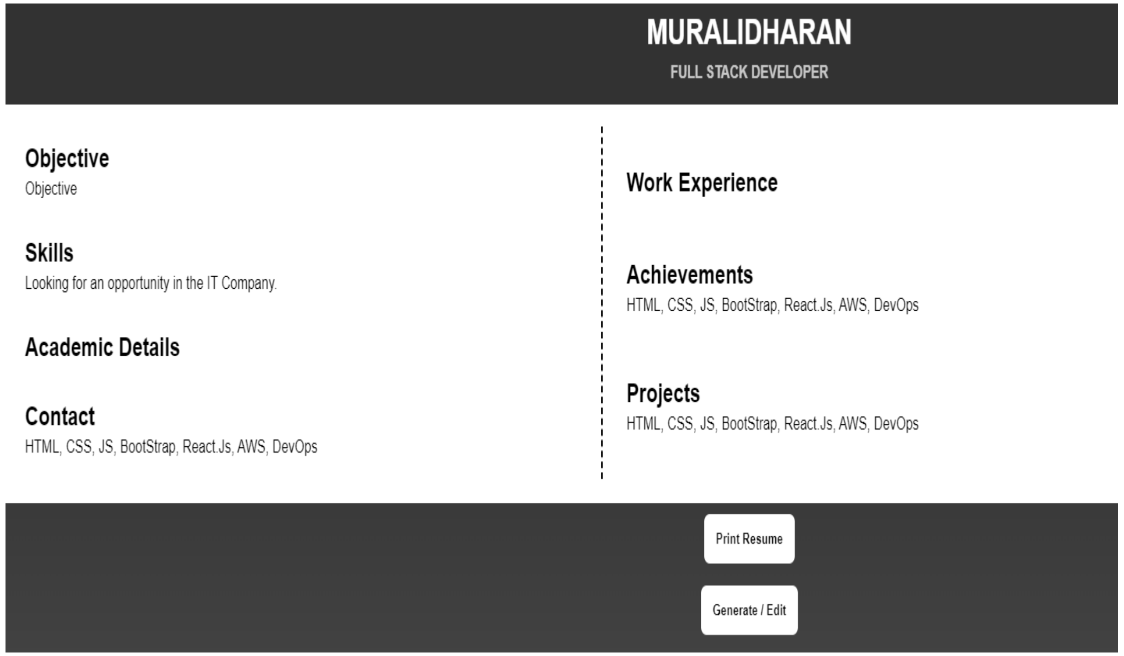
Your Name

Title/Sub Heading

Objective

Created by Muralidharan

You can use markup in the text area for text decoration



MURALIDHARAN

FULL STACK DEVELOPER

Objective

Objective

Skills

Looking for an opportunity in the IT Company.

Academic Details

Contact

HTML, CSS, JS, BootStrap, React.Js, AWS, DevOps

Work Experience

Achievements

HTML, CSS, JS, BootStrap, React.Js, AWS, DevOps

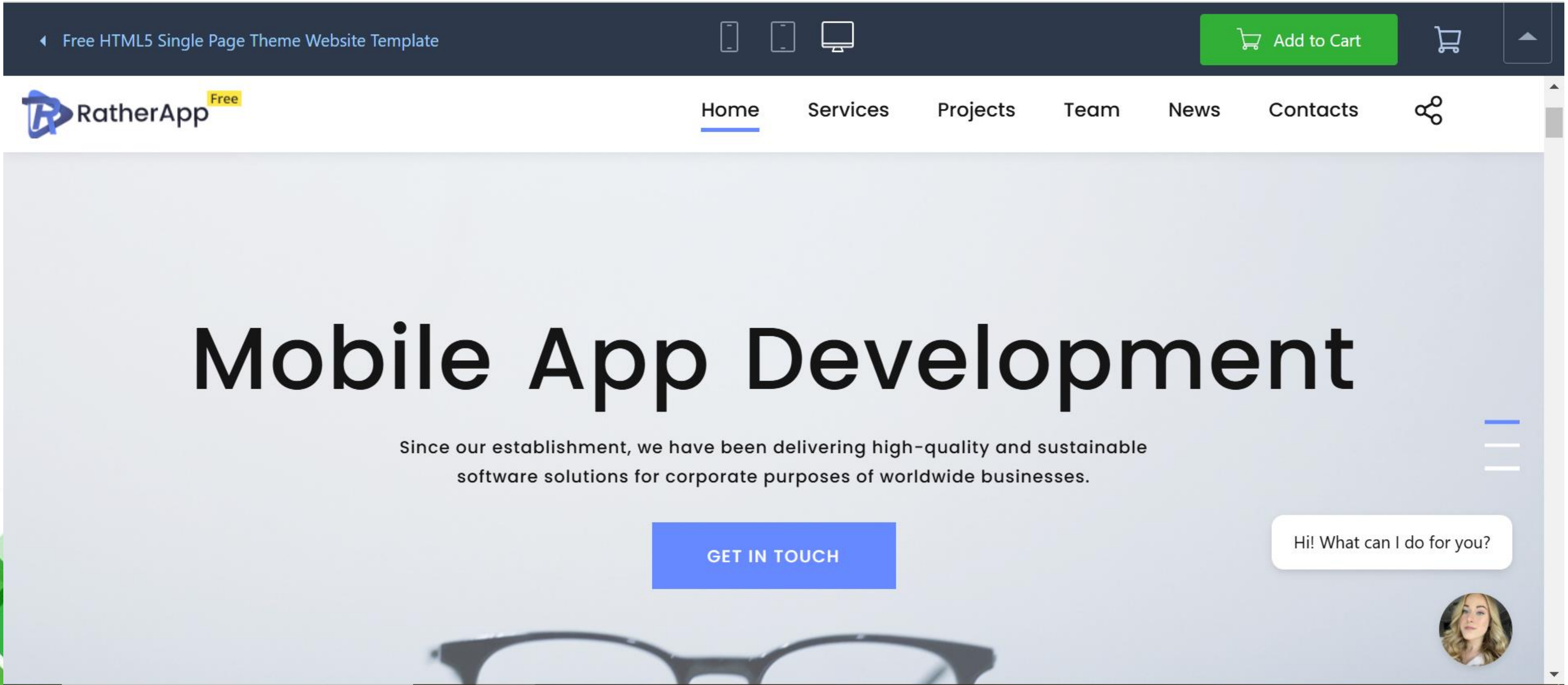
Projects

HTML, CSS, JS, BootStrap, React.Js, AWS, DevOps

Task

Create a website similar to this for your Own Software Company

https://demo.templatemonster.com/demo/51694.html?_gl=1*owqtzh*_gcl_au*MTQ2OTg4NjY1OS4xNzM0MDcyNjA5*_ga*MjM2NDA0NTQ5LjE3MzQwNzI2MDU.*_ga_FTPYEGT5LY*MTczNDA3MjYwNC4xLjEuMTczNDA3Mjc3MS40MC4wLjA.



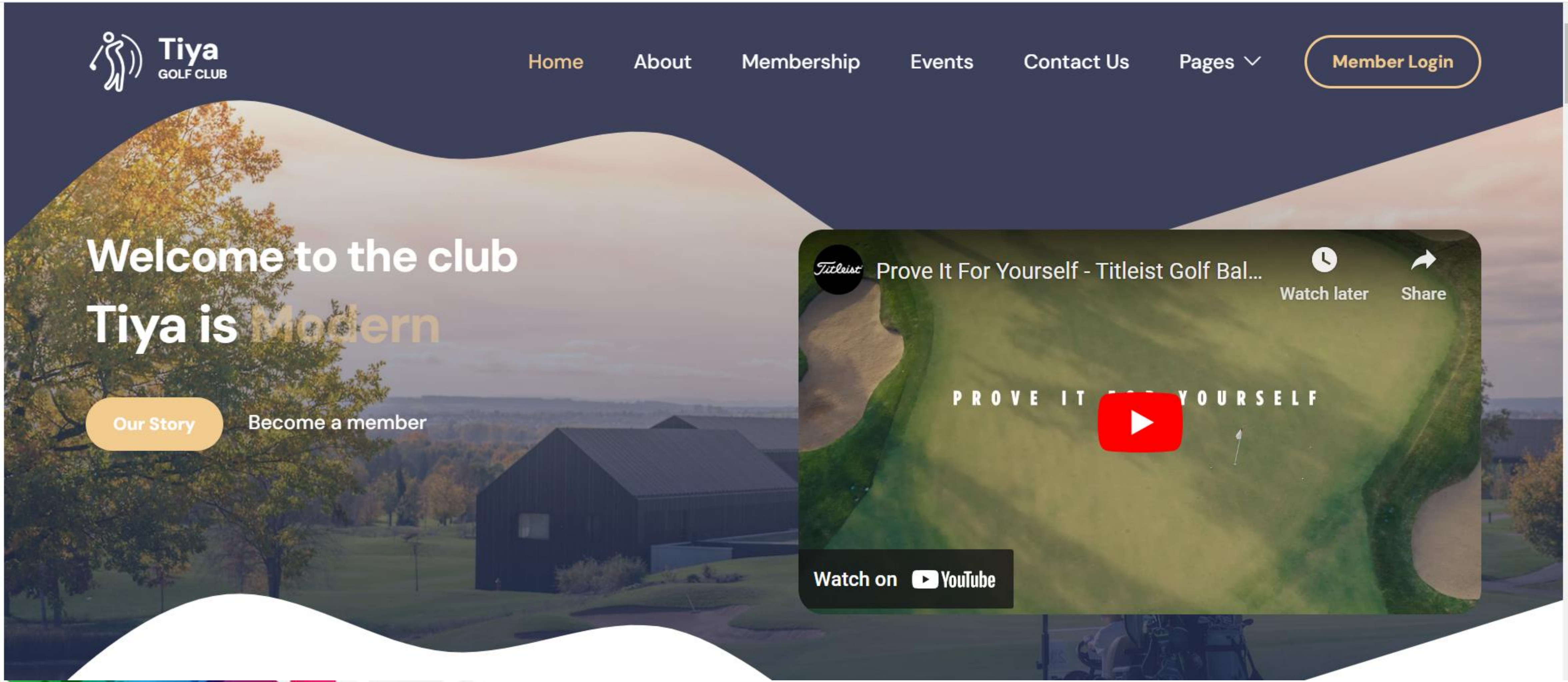
Mobile App Development

Since our establishment, we have been delivering high-quality and sustainable software solutions for corporate purposes of worldwide businesses.

Task

Create a website similar to this for your Football Club

https://templatemo.com/live/templatemo_587_tiya_golf_club



Questions

THANK YOU :)