JAVASCRIPT

Comprehensive Manual for Developing Dynamic and Responsive Website and Applications.

Suitable for Novice and Experts.

igital Services

JAVASCRIPT

A Comprehensive Manual for Creating Dynamic, Responsive Websites and Applications That is Suitable for Both Novices and Experts

.

ISBN:

978-1-304-32537-2

First Published, 2024

Noogul Digital Publishing

Audience

This tutorial is designed for the aspiring Web Designers and Developers with a need to understand JavaScripting in enough detail along with its simple overview, and practical examples. This tutorial will give you enough ingredients to start with JavaScript from where you can take yourself at higher level of expertise.

Prerequisites

Before proceeding with this tutorial you should have a basic working knowledge with Windows or Linux operating system, additionally you must be familiar with:

- Experience with any text editor like notepad, notepad++, or Edit plus etc.

- Have Basic and Advanced Knowledge of HTML.

- How to create directories and files on your computer.

- How to navigate through different directories.

- How to type content in a file and save them on a computer.

- Understanding about images in different formats like JPEG, PNG format.

- Knowledge of How to use Google Sheets (Excel Sheet).

Contents

Audience

Chapter One

Features of JavaScript

1.
Web Applications

2.
Web Development

3.
Mobile Applications

4.
Game

5.
Presentations

6.
Server Applications

7.
Web Servers

Application of JavaScript

External References

Chapter Two

JavaScript Basics

JavaScript Can Change HTML Content

JavaScript Can Change HTML Attribute Values

JavaScript Can Change HTML Styles (CSS)

JavaScript Can Hide HTML Elements

JavaScript Can Show HTML Elements

The <script> Tag

JavaScript Functions and Events

JavaScript in <head> or <body>

JavaScript in <head>

JavaScript in <body>

JavaScript Expressions

JavaScript Display Possibilities

Using innerHTML

Using document.write()

Using window.alert()

Using console.log()

JavaScript Print

JavaScript Statements

JavaScript Programs

JavaScript Statements

JavaScript White Space

JavaScript Line Length and Line Breaks

JavaScript Code Blocks

JavaScript Keywords

JavaScript Values

JavaScript Literals

Chapter Three

JavaScript Comment

Types of JavaScript Comments

JavaScript Single line Comment

JavaScript Multi line Comment

Using JavaScript Comments to Prevent Code Execution

Commenting Out Function Calls

Commenting Out Function Bodies — Without Return Values

Commenting Out Function Bodies — With Return Values

Writing Effective JavaScript Comments

Chapter Four

JavaScript Variable

JavaScript Keywords

JavaScript Variable Naming Convention

JavaScript Var Keyword

JavaScript Let Keyword

JavaScript Const Keyword

When to Use JavaScript const?

JavaScript Local Variable

Function Scope

JavaScript Global Variable

Internals of global variable in JavaScript

Automatically Global

Global Variables in HTML

How to use variables

Where to use which variable

Chapter Five

JavaScript Operators

JavaScript Assignment

Assignment Examples

JavaScript Arithmetic Operators

JavaScript Assignment Operators

JavaScript Comparison Operators

JavaScript String Addition

Adding Strings and Numbers

JavaScript Logical Operators

JavaScript Bitwise Operators

Bitwise logical operators

JavaScript Bitwise AND

Example

JavaScript Bitwise OR

Example:

JavaScript Bitwise XOR

JavaScript Bitwise NOT (~)

JavaScript (Zero Fill) Bitwise Left Shift (<<)

JavaScript (Sign Preserving) Bitwise Right Shift (>>)

JavaScript (Zero Fill) Right Shift (>>>)

Converting Decimal to Binary

Converting Binary to Decimal

Chapter Six

JavaScript Data Types

JavaScript primitive data types

JavaScript non-primitive data types

Examples

The Concept of Data Types

JavaScript Types are Dynamic

JavaScript Strings

JavaScript String Methods

JavaScript String Length

Extracting String Parts

JavaScript String slice()

Examples

JavaScript String substring()

Replacing String Content

JavaScript String ReplaceAll()

Converting to Upper and Lower Case

JavaScript String concat()

JavaScript String trim()

JavaScript Numbers

JavaScript Random

Exponential Notation

JavaScript BigInt

JavaScript Integer Accuracy

How to Create a BigInt

JavaScript Booleans

The Boolean() Function

NaN data type

Comparisons and Conditions

JavaScript Comparison and Logical Operators

Comparison Operators

How Can it be Used

Conditional (Ternary) Operator

Comparing Different Types

JavaScript if, else, and else if

Conditional Statements

The if Statement

Example

The else Statement

Example

The else if Statement

Example

JavaScript Switch Statement

Example

The break Keyword

The default Keyword

JavaScript Arrays

JavaScript Array

JavaScript Array Methods

JavaScript Objects

The type of Operator

Chapter Seven

JavaScript Functions

Function Syntax

Function declarations

Function Invocation

Invoking a JavaScript Function

The Term “This” in Javascript

Note

The Global Object

Invoking a Function as a Method

Invoking a Function with a Function Constructor

Function Return

The ( ) Operator

Functions Used as Variable Values

Local Variables

Chapter Eight

JavaScript Objects

Real Life Objects, Properties, and Methods

Object Definition

Object Properties

Accessing Object Properties

Object Methods

The this Keyword

Accessing Object Methods

Do Not Declare Strings, Numbers, and Booleans as Objects!

Chapter Nine

JavaScript Events

HTML Events

Mouse events:

onclick Event Type

onsubmit Event Type

onmouseover and onmouseout

Keyboard events:

Form events:

Window/Document events

HTML DOM Events

JavaScript Event Handlers

Chapter Ten

JavaScript Loop

The For Loop

do...while statement

Example:

Differences between do… while and While Loop

While Statement

Example:

Comparison between the while and for loop:

Example: JavaScript For In Loop

for-in Loop Examples

The For Of Loop

Properties of document object

Methods of document object

Accessing field value by document object

JavaScript - document.getElementById() method

JavaScript - document.getElementsByName() method

JavaScript - document.getElementsByTagName() method

Another example of document.getElementsByTagName() method

JavaScript - innerHTML

Example of innerHTML property

Show/Hide Comment Form Example using innerHTML

JavaScript - innerText

JavaScript innerText Example

Understanding the Browser Environment

The user interface

Loader

HTML parsing

CSS parsing

JavaScript parsing

Layout and rendering

Igniting the BOM

The Navigator Object

Window Object

Methods of window object

Example of alert() in javascript

Example of confirm() in javascript

Example of prompt() in javascript

Example of open() in javascript

Example of setTimeout() in javascript

JavaScript History Object

Property of JavaScript history object

Methods of JavaScript history object

Example of history object

JavaScript Navigator Object

Property of JavaScript navigator object

Methods of JavaScript navigator object

Example of navigator object

JavaScript Screen Object

Property of JavaScript Screen Object

Example of JavaScript Screen Object

Approach for Form Validation in JavaScript

JavaScript Form Validation Example

JavaScript Retype Password Validation

JavaScript Number Validation

JavaScript validation with image

JavaScript email validation

JavaScript Classes

Class Declarations

Class Declarations Example

Class Declarations Example: Hoisting

Class Declarations Example: Re-declaring Class

Class expressions

Unnamed Class Expression

Class Expression Example: Re-declaring Class

Named Class Expression Example

JavaScript Objects

Creating Objects in JavaScript

1) JavaScript Object by object literal

2) By creating instance of Object

3) By using an Object constructor

Defining method in JavaScript Object

JavaScript Object Methods

JavaScript Prototype Object

Syntax:

Prototype Chaining

JavaScript Prototype Object

JavaScript Constructor Method

Points to remember

Constructor Method Example

JavaScript static Method

JavaScript static Method Example

Example 4

JavaScript Encapsulation

JavaScript Encapsulation Example

JavaScript Encapsulation Example: Validate

JavaScript Encapsulation Example: Prototype-based approach

JavaScript Inheritance

JavaScript extends Example: inbuilt object

JavaScript extends Example: Custom class

JavaScript extends Example: a Prototype-based approach

JavaScript Polymorphism

JavaScript Abstraction

Chapter Fifteen

JavaScript Cookies

How Cookies Works?

How to create a Cookie in JavaScript?

JavaScript Cookie Example

Cookie Attributes

Cookie expires attribute

Cookie max-age attribute

Cookie path attribute

Cookie path attribute Example

Cookie domain attribute

Cookie with multiple Name-Value pairs

Examples to Store Name-Value pair in a Cookie

Deleting a Cookie in JavaScript

Examples to delete a Cookie

Example 3

Chapter Sixteen

Integrating JavaScript with Google Apps Script

What can Apps Script do?

Custom Menus in Google Workspace

Clickable images and drawings in Google Sheets

Dialogs and Sidebars in Google Workspace Documents.

Alert dialogs

Prompt dialogs

Custom dialogs

Custom sidebars

File-open dialogs

Custom Functions in Google Sheets

Developing a custom function

obtaining a personalized feature via the Google Workspace Marketplace

Using a custom function

Guidelines for custom functions

Naming

Arguments

Return values

Data types

Autocomplete

Using Google Apps Script services

Sharing

Optimization

Google Sheets Macros

Creating macros in Apps Script

Editing macros

Importing functions as macros

Manifest structure for macros

Best practices

Things you can't do

Chapter Seventeen

Developing Web Apps in Apps Script

Requirements for web apps

Request parameters

Deploy a script as a web app

Test a web app deployment

Permissions

Embed your web app in Google Sites or any Site of your Choice.

Web Apps and Browser History

How to create Login and Register Form using Google spreadsheet data

How to Display Google Sheet Data on Webpage

How to Submit HTML Form Data to Google Spreadsheet

How to Submit HTML Form to Gmail

How to Search Google Sheet Contents from HTML Website.

Conclusion

References

Chapter One

JavaScript Introduction

Many websites utilize JavaScript (js), a lightweight object-oriented programming language, to script their webpages. When applied to an HTML document, this fully functional programming language that is interpreted allows for dynamic website interaction.

In 1995, JavaScript was released, allowing users of the Netscape Navigator browser to add applications to web pages. Since then, all other major graphical web browsers have implemented the language. It has enabled the development of contemporary online applications, which allow for immediate user interaction without requiring a page reload. Conventional websites also employ JavaScript to provide a variety of innovative features and interactive elements (Netscape, 2007).

Brendan Eich wrote JavaScript in ten days in May 1995. Eich was employed by Netscape, where he developed JavaScript for Netscape Navigator, the company's web browser. The plan was to use Java to create the main interactive components of the client-side web. JavaScript was intended to serve as the connecting language between those elements and to provide a little bit of interactivity to HTML. JavaScript needed to look like Java because of its supporting function in Java. That eliminated working options like TCL, Python, Perl, and others (W3C, 2021).

JavaScript is a language that improves the Web. The language can assist in transforming a static page of text into an intelligent, dynamic, and engaging experience when used on the client computer. JavaScript applications can be as subtle as greeting a visitor to a website with "Good morning!" when it is morning in the client computer's time zone. Other apps may be considerably more straightforward, like one that downloads a slide show's content in one page and uses JavaScript to manage the presentation's hiding, showing, and "flying slide" transitions.

JavaScript was first known as Mocha. When Netscape Navigator was first released in beta, it was called LiveScript. When Netscape 2 was released in 1995, it was renamed JavaScript. To avoid trademark difficulties, Microsoft swiftly reverse-engineered JavaScript and launched an exact clone of it in Internet Explorer, which they called Jscript.

JavaScript was accepted and standardized as EMCAScript in 1997 after Netscape submitted it to Ecma International, a standards body.

Nevertheless, JavaScript and Java are unrelated programming languages. During the period when Java was becoming more and more prominent in the market, the name was offered and suggested. Databases like CouchDB and MongoDB use JavaScript as their scripting and query language in addition to web browsers. JavaScript founder Brandon Eich is well-known for his remarks against the standardized language's name, referring to ECMAScript (Ecma International) as a "unwanted trade name that sounds like a skin disease."

Not only is ECMAScript a terrible moniker for a programming language, but it's also the name that Netscape gave the language and that most people use to refer to it. If you are familiar with programming in Java or would like to learn at some time, it is a good idea to remember that although there are some similarities between the two languages, they are very different things. When JavaScript first came out, it was widely used to add dynamic elements to websites.

An early outcome of JavaScript being integrated into web browsers was the creation of so-called Dynamic HTML (DHTML), which allowed for a variety of entertaining effects, such as the falling snowflake effect, pop-up windows, and curling web page corners, as well as more practical features like drop-down menus and form validation.

Features of JavaScript

The following are features of JavaScript:

i. All popular web browsers support JavaScript as they provide built-in execution environments.

ii. JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.

iii. JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).

iv. JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.

v. It is a light-weighted and interpreted language.

vi. It is a case-sensitive language.

vii. JavaScript is supportable in several operating systems including, Windows, macOS, etc.

viii. It provides good control to the users over the web browsers.

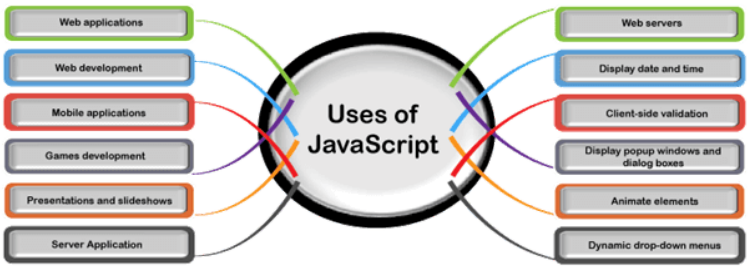
Uses of JavaScript

JavaScript is a programming language that enables the implementation of intricate features on websites. You can be sure that JavaScript is used whenever a website performs any function other than merely displaying static content for you to view. Examples of such functions include interactive maps, scrolling video jukeboxes, animated 2D/3D graphics, and timely content updates. It is the third layer in the stack of common web technologies, the first two of which (CSS and HTML) we have already thoroughly discussed in other Learning Area sections.

Many websites utilize JavaScript, a lightweight object-oriented programming language, to script their webpages. It is a complete programming language that is interpreted. When JavaScript is added to an HTML page, websites may have dynamic interactivity.

JavaScript enables users to create interactive contemporary web apps without constantly refreshing the page. JavaScript is frequently used with the DOM API to refresh a user interface by dynamically modifying HTML and CSS.

It is mostly employed in online programs. Let's talk about JavaScript's applications. An example of how JavaScript is used is seen in the image below.



1. Web Applications

Because browsers are getting better every day, JavaScript has become more and more popular as a tool for creating reliable online applications. We may comprehend it by using Google Maps as an example. With Maps, all a user needs to do is click and drag the mouse to view the information. These ideas are supported by the usage of JavaScript.

2. Web Development

Creating web pages is a typical usage for JavaScript. It enables us to add special effects and dynamic behavior to the webpage. It is mostly used for validation on websites. JavaScript facilitates the execution of intricate

tasks and allows websites to communicate with their users. Another way to load material into a document without refreshing the webpage is by using JavaScript.

3. Mobile Applications

Nowadays, a lot of people utilize their mobile devices to access the internet. We can also create an application for non-web environments using JavaScript. JavaScript is a fantastic technology for developing mobile applications because of its capabilities and applications. The popular JavaScript framework for making mobile apps is called React Native. We can create mobile applications for several operating systems with React Native. Writing separate code for the iOS and Android operating systems is not necessary. It simply has to be written once, and it can operate on several systems.

4. Game

Games may also be made with JavaScript. It offers a number of frameworks and libraries for making games. Either a 2D or 3D game may be played. A few JavaScript game engines like Pixi.js and PhysicsJS assist us in making an online game. Additionally, we may render 2D and 3D pictures on browsers by using the JavaScript API known as the WebGL (web graphics library).

5. Presentations

Moreover, JavaScript facilitates the creation of websites and presentations. You may utilize the libraries, such BespokeJs and RevealJs,

to make an online slide show. Because they are simpler to utilize, we can quickly and simply create something incredible.

With the use of HTML, Reveal.js is used to build stunning and dynamic slide decks. Tablets and mobile devices are ideal for these presentations. All CSS color formats are also supported. Animated bullet lists, responsive scaling, and an extensive feature set are all included in the BespokeJS.

6. Server Applications

Many web applications feature a server-side component. HTTP requests are handled and content is generated using JavaScript. Node.js allows JavaScript to operate on servers as well. The environment that Node.js offers has all the tools needed for JavaScript to execute on servers.

7. Web Servers

Node.js may be used to develop a web server. Because Node.js is event-driven, it doesn't wait for the preceding call's answer. Node.js servers carry large amounts of data quickly and without the need for buffering. The createServer() function in the HTTP module may be used to create the server. Whenever someone attempts to access port 8080, this function gets called. The HTTP server should show HTML in response, and it should also be provided in the HTTP header.

Application of JavaScript

Websites that are interactive are made with JavaScript. There are several other applications for JavaScript that aid in enhancing webpage speed. The following is a list of other uses for JavaScript:

- Client-side validation.

- Dynamic drop-down menus,

- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),

- Displaying clocks

- Displaying date and time.

- To validate the user input before submission of the form.

- Open and close new windows.

- To display dialog boxes and pop-up windows.

- To change the appearance of HTML documents.

- To create the forms that respond to user input without accessing the server.

Prerequisite to Writing Your First JavaScript program

Before starting to build your first JavaScript application, make sure you have all of your tools assembled and ready. By coincidence, the tools we use in this book are also the ones we recommend you download and install. We lead you through the procedure. Please feel free to utilize any comparable or preferred tools you may have.

To understand why we selected these tools and to help you decide whether or not to use them, we still advise you to read this portion of the book. We provide you with some pointers on how to maximize the functionality of each tool when you install it.

Downloading and installing Chrome

Google Chrome is the preferred web browser for dealing with JavaScript. It's acceptable, of course, if you would rather use a different web browser on a daily basis. Every browser will execute JavaScript accurately and quickly. But since Google Chrome will be covered in some detail in this book, we advise you to at least install it on your computer using the steps outlined in this chapter. Because Google Chrome is now the most popular web browser on the Internet and provides great features for making JavaScript writers' tasks simpler, we decided to utilize it in this book. (Yes, its popularity surpasses that of Internet Explorer.)

If you don’t have Chrome installed, follow these steps to install it:

i. Go to www.google.com/chrome.

ii. Hover over the Download tab and choose the appropriate version for your computer.

iii. Open the downloaded file and follow the instructions to install Chrome

Google Chrome parses, compiles, and executes JavaScript code using Google's V8 JavaScript engine. Chrome is either the fastest or among the fastest browsers at running JavaScript, depending on whose benchmarking test you trust. The main browser manufacturers are always trying to surpass one another. The competition has accelerated the pace of every browser's JavaScript engine in recent years, so it doesn't really matter who is the quickest at any one moment. You may visit http://arewefastyet.com to view real-time comparisons of the performance of several browsers in JavaScript tests.

The most popular browsers' JavaScript performance is automatically checked and graphed on this site, which is updated many times a day by Mozilla, the company that created the Firefox browser.

Downloading and installing a code editor

A source code editor, sometimes called a code editor, is essentially a text editor with extra features that make it easier to create and modify computer code. Sublime Text is the one we use. There are several code editors available, so feel free to pick your preferred one if you already know how to use it and find it comfortable. Code editors are very individualized tools, and many programmers will discover that they work better with a particular one solely because it seems more natural to them. If you discover that Sublime Text simply isn't your thing.

Name	Location	Compatible with
Coda	http://panic.com/coda	Mac only
Aptana	www.aptana.com	Mac or Windows
Komodo Edit	www.activestate.com/komodo-edit/downloads	Mac or Windows
Dreamweaver	http://adobe.com/products/dreamweaver.html	Mac or Windows
Eclipse	www.eclipse.org	Mac or Windows
Notepad++	http://notepad-plus-plus.org	Windows only
TextMate	http://macromates.com	Mac only
BBEdit	www.barebones.com/products/bbedit	Mac only
EMacs	www.gnu.org/software/emacs	Mac or Windows
TextPad	www.textpad.com	Windows only
vim	www.vim.org	Mac or Windows
Netbeans	https://netbeans.org	Mac or Windows
Bracket	https://brackets.io/	Mac or Windows
Visual Studio Code	https://code.visualstudio.com/	Mac or Windows

Reading JavaScript Code

Before you get started with writing JavaScript programs, you need to be aware of a few rules of JavaScript:

- JavaScript is case-sensitive. We repeat this several times throughout the book, because it's an error that those who are new to JavaScript make quite frequently. To JavaScript, the words pants and Pants are completely different.

- JavaScript doesn't care much about white space. White space includes spaces, tabs, and line breaks — any character that doesn't have a visual representation. When you're writing JavaScript code, it doesn't matter if you use one space, two spaces, a tab, or even a line break (in most cases) within the code. JavaScript will ignore white space. The one exception is when you're writing out text that you want JavaScript to print to the screen. In this case, the white space you use will show up in the end result. The best practice, with regards to white space in your code, is to use enough space that your code is easy to read and to also be consistent with how you use this space.

- Watch out for reserved words. JavaScript has a list of words that have special meaning to the language. We list these words in Chapter 3. For now, just be aware that some words, such as function, while, break, and with have special meanings.

- JavaScript likes semicolons: JavaScript code is made up of statements. You can think of statements as similar to sentences. They are fundamental building blocks for JavaScript programs in the same way that sentences are the building blocks of paragraphs. In JavaScript, statements end with a semicolon. If you don’t use a semicolon at the end of a statement, JavaScript will put it there for you. This can lead to unexpected results, however, so it’s considered a best practice to always end statements with a semicolon.

Running JavaScript in the Browser Window

While JavaScript is used in many other contexts, web browsers are the most popular location to observe it in the wild. JavaScript was created to handle typical browser actions like clicks and scrolling, control inputs and outputs, manipulate online pages, and control the many functionalities of web browsers!

To run JavaScript in a web browser, you have three options, all of which will be shown in the following pages:

- Put it directly in an HTML event attribute

- Put it between an opening and closing script tag

- Put it in a separate document and include it in your HTML document

Many times, you'll use a combination of all three techniques within any one web page. However, knowing when to use each is important and is a skill that you'll learn with more practice.

Using JavaScript in an HTML event attribute

HTML has a number of unique properties that are intended to cause JavaScript to run in response to events that occur in the web browser or user actions. An HTML button with an event property that reacts to mouse click events is seen here:

<button id="bigButton" onclick="alert('Hello World!');">Click Here</button>

In this case, when a user clicks on the button created by this HTML element, a popup will appear with the words "Hello World!". HTML has over 70 different event attributes. Table 2-3 shows the most commonly used ones.

Commonly Used HTML Event Attributes	
Attribute	**Description**
onload	Runs the script after the pages finishes loading
onfocus	Runs the script when the element gets focus (such as when a text box is active)
onblur	Runs the script when the element loses focus (such as when the user clicks a new text box in a form)
onchange	Runs the script when the value of an element is changed
onselect	Runs the script when text has been submitted
onsubmit	Runs the script when a form has been submitted
onkeydown	Runs the script when a user is pressing a key
onkeypress	Runs the script when a user presses a key
onkeyup	Runs the script when a user releases a key
onclick	Runs the script when a user mouse clicks an element
ondrag	Runs the script when an element is dragged
ondrop	Runs the script when a dragged element is being dropped
onmouseover	Runs the script when a user moves a mouse pointer over an element

Using JavaScript in a script element

The HTML script element allows you to embed JavaScript into an HTML document. Often script elements are placed within the head element, and, in fact, this placement was often stated as a requirement. Today, however, script elements are used within the head element as well as in the body of web pages. The format of the script element is very simple:

<script>

(insert your JavaScript here)

</script>

In this case, however, we place the script element at the bottom of the body element

Embedding JavaScript within a Script Element

<!DOCTYPE html>

<html>

<head>

<title>Hello, HTML!</title>

</head>

<body>

```
<h1>Let's Count to 20 with JavaScript!</h1>

<p id="theCount"></p>

<script>

var count = 0;

while (count < 20) {

count++;

document.getElementById("theCount").innerHTML +=

count + "<br>";

}

</script>

</body>

</html>
```

Script placement and JavaScript execution

Scripts are generally loaded by web browsers and run when they load. The browser reads a web page from top to bottom, exactly like it would read a text page. Occasionally, you should wait for the script to execute

until the browser has finished loading the page's contents. We used the body element's onload event property to do this. Placing the code that has to be run at the conclusion of the code is another popular technique to postpone execution.

Limitations of JavaScript in <script> elements

Although embedding JavaScript into a script element is far more popular and widely accepted than inline scripting (placing JavaScript into event attributes), it still has some significant drawbacks. The main drawback is that these kinds of scripts are only usable on the web page on which they are placed. To put it another way, if you include JavaScript in a script element, you must duplicate and paste the script element precisely onto each page that has it. You can understand how this may turn into a maintenance nightmare given that some websites have hundreds or even thousands of web pages.

When to use JavaScript in <script> elements

There are applications for this JavaScript embedding technique. It is okay and can even speed up the loading and display of your web pages by reducing the number of requests the web server needs to make to the server for those JavaScript elements that do nothing more than call other JavaScript elements.

As the name suggests, single-page applications only have one HTML page, making them excellent candidates for this kind of embedding as there is only one location where the script has to be updated. Generally

speaking, though, you want to try to avoid adding too much JavaScript straight into an HTML page. Your code will become more organized and require less maintenance as a consequence.

Including external JavaScript files

The src property of the script element is the third and most often used method of including JavaScript in HTML texts. The only difference between a script element with JavaScript between the tags and one with a src attribute is that the JavaScript is loaded into the HTML document from a different file.

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension .js.

To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

Here’s an example of a script element with a src attribute:

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

In this case, you would have a separate file, named myScript.js, that would reside in the same folder as your HTML document.

```
<!DOCTYPE html>

<html>

<body>

<h2>Demo External JavaScript</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<p>This example links to "myScript.js".</p>

<p>(myFunction is stored in "myScript.js")</p>

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

</body>

</html>
```

NB:

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

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

iii. *External scripts cannot contain <script> tags.*

The benefits of using external JavaScript files are that using them

- Keeps your HTML files neater and less cluttered

- Makes your life easier because you need to modify JavaScript in only one place when something changes or when you make a bug fix

- It separates HTML and code

- It makes HTML and JavaScript easier to read and maintain

- Cached JavaScript files can speed up page loads

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

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

Creating a .js file

Creating an external JavaScript file is similar to creating an HTML file or another other type of file. To create an external JavaScript file, follow these steps:

i. In Sublime Text or any of your text editor, choose File ➪ New File.

ii. Copy everything between <script> and </script> from MyFirstProgram.html and paste it into your new .js file.

NB: Notice that external JavaScript files don't contain <script> elements, just the JavaScript.

iii. Save your new file as countToTwenty.js in the same folder as MyFirstProgram.html.

iv. In MyFirstProgram.html, modify your script element to add a src attribute, like this:

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

Your copy of MyFirstProgram.html should now look like this:

```
<!DOCTYPE html>

<html>

<head>
```

```
<title>Hello, HTML!</title>

</head>

<body>

<h1>Let's Count to 20 with JavaScript!</h1>

<p id="theCount"></p>

<script>

var count = 0;

while (count < 20) {

count++;

document.getElementById("theCount").innerHTML +=

count + "<br>";

}

</script>

</body>

</html>
```

Your new file, countToTwenty.js, should look like this:

```
function countToTwenty(){

var count = 0;

while (count < 20) {

count++;

document.getElementById("theCount").innerHTML +=

count + "<br>";

}

}
```

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

Example 1: *This example uses a* ***full URL*** *to link to myScript.js:*

```
<!DOCTYPE html>

<html>

<body>

<h2>External JavaScript</h2>

<p id="demo">A Paragraph.</p>



<button type="button" onclick="myFunction()">Click Me</button>

<p>This example uses a full web URL to link to "myScript.js".</p>
```

* (myFunction is stored in "myScript.js") *

<script src="https://www.w3schools.com/js/myScript.js"></script>

</body>

</html>

***Example 2:** This example uses a **file path** to link to myScript.js:*

<!DOCTYPE html>

<html>

<body>

<h2>External JavaScript</h2>

* A Paragraph. *

<button type="button" onclick="myFunction()">Try it</button>

* This example uses a file path to link to "myScript.js". *

* (myFunction is stored in "myScript.js") *

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

</body>

</html>

***Example 3:** This example uses **no path** to link to myScript.js:*

<!DOCTYPE html>

<html>

<body>

<h2>Demo External JavaScript</h2>

* A Paragraph. *

<button type="button" onclick="myFunction()">Try it</button>

* This example links to "myScript.js". *

* (myFunction is stored in "myScript.js") *

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

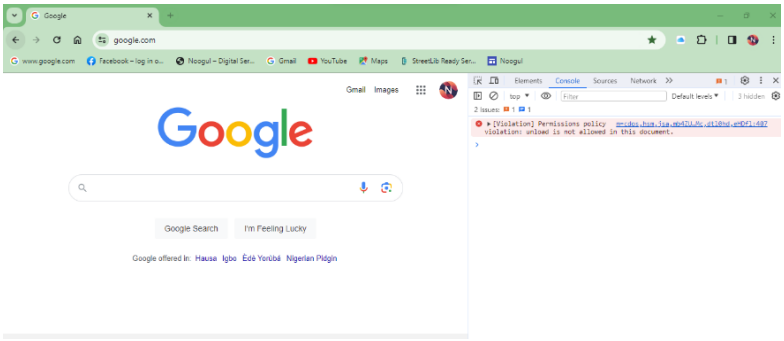
</body>

</html>

Using the JavaScript Developer Console

Sometimes, it's helpful to be able to run JavaScript commands without creating an HTML page and including separate scripts or creating <script> blocks. For these times, you can use the Chrome browser's JavaScript Console. To access the JavaScript Console, find the Chrome menu in the upper-right corner of your browser. It looks like three horizontal lines. Click the Chrome menu and then find More Tools in the drop-down menu. Under More Tools, choose JavaScript Console from the drop-down menu.

And, yes, there is a faster way to open the JavaScript Console. Simply press Alt+Command+J (on Mac) or Control+Shift+J (on Windows).

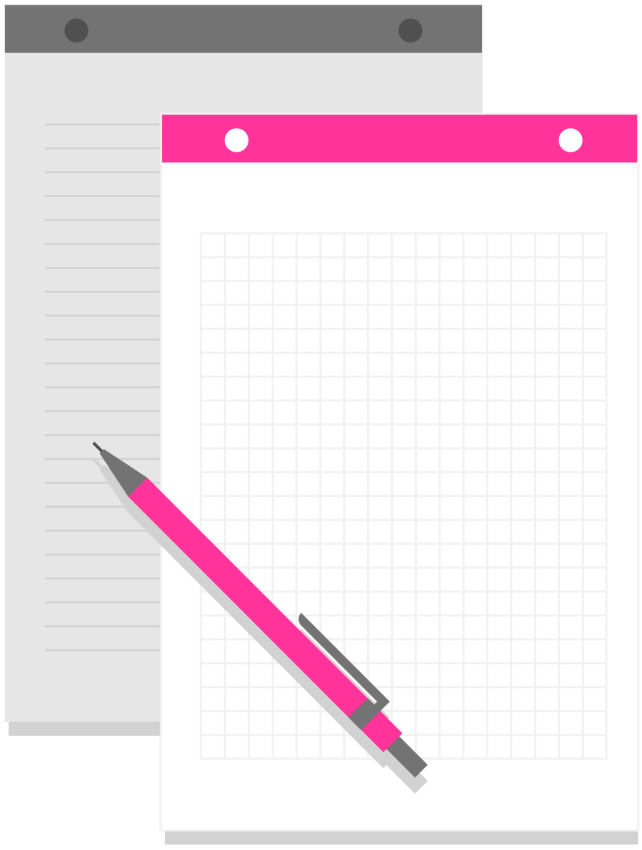


The JavaScript Console is perhaps the best friend of the JavaScript developer. Besides allowing you to test and run JavaScript code quickly and easily, it also is where errors in your code are reported, and it has features that will help you track down and solve problems with your code.

Once you've opened the JavaScript console, you can start inputting commands into it, which will run as soon as you press Enter. To try it out, open the JavaScript console and then type the following commands, pressing Enter after each one:

Chapter Two

JavaScript Basics



This section aims to rapidly expose you to the fundamentals of JavaScript so that you may begin writing programs. This part will teach you the

essential building blocks of JavaScript, avoiding a comprehensive coverage of all theories and notions. Topics such as operators, if statements, loops, variables, data types, swith, functions, objects, arrays, and classes will be covered. Also, you'll discover how to combine them together to create a compact yet reliable software.

As you may have be aware, JavaScript is widely used in the field of software development nowadays. It serves as the cornerstone of front-end web development and is essential to frameworks such as Angular, Vue, and ReactJS. Additionally, it may be used to build robust backends using Node.js platforms, run desktop programs like Slack, Atom, and Spotify, and function as Progressive Web Apps (PWAs) on mobile devices.

In short, it’s everywhere—and for good reason. For starters, compared to other languages like C and Java, JavaScript is generally easier to learn. When we say ‘easier’, we mean in terms of how quickly you can go from being a JavaScript novice to someone who can actually make a living writing professional, high quality JavaScript code. So, in that sense, it’s more accessible than some other languages like C and Java.

JavaScript is also a fun and rewarding language, which is especially important when you’re just getting started in software development. The community support is very good, so if you get stuck, there’s a good chance that the problem and its solution already exist on the web.

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is getElementById().

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

Example:

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



<!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button" onclick='document.getElementById("demo").innerHTML = "Hello JavaScript!"'>Click Me!</button>

</body>

</html>
```

JavaScript accepts both double and single quotes:

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


<!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button" onclick="document.getElementById('demo').innerHTML = 'Hello JavaScript!'">Click Me!</button>

</body>

</html>
```

JavaScript Can Change HTML Attribute Values

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

```
<!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p>JavaScript can change HTML attribute values.</p>

<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>

<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>

</body>

</html>
```

JavaScript Can Change HTML Styles (CSS)

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

```
<!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do? </h2>

<p id="demo">JavaScript can change the style of an HTML element. </p>

<button type="button" onclick="document.getElementById('demo').style.fontSize='35px'">Click Me!</button>

</body>

</html>
```

JavaScript Can Hide HTML Elements

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

```
<!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can hide HTML elements.</p>

<button type="button" onclick="document.getElementById('demo').style.display='none'">Click Me!</button>

</body>

</html>
```

JavaScript Can Show HTML Elements

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

```
<!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p>JavaScript can show hidden HTML elements.</p>

<p id="demo" style="display:none">Hello JavaScript!</p>

<button type="button" onclick="document.getElementById('demo').style.display='block'">Click Me!</button>

</body>

</html>
```

The <script> Tag

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

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

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript in Body</h2>

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

<script>

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

</script>

</body>

</html>
```

Nb: *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.

JavaScript in <head> or <body>

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

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

JavaScript in <head>

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

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

```
<!DOCTYPE html>

<html>
```

```
<head>

<script>

function myFunction() {

document.getElementById("demo").innerHTML = "Paragraph changed.";

}

</script>

</head>

<body>

<h2>Demo JavaScript in Head</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

</body>

</html>
```

JavaScript in <body>

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

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

```
<!DOCTYPE html>

<html>

<body>

<h2>

Demo JavaScript in Body

</h2>

<p id="demo">

A Paragraph

</p>

<button type="button" onclick="myFunction()">

Try it
```

```
</button>

<script>

 function

 myFunction() {

  document.getElementById(

"demo"

).innerHTML =

"Paragraph changed."

;

}

</script>

</body>

</html>
```

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

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:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Expressions</h2>

<p>Expressions compute to values.</p>

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

<script>

document.getElementById("demo").innerHTML = 5 * 10;

</script>
```

```
</body>

</html>
```

Expressions can also contain variable values:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Expressions</h2>

<p>Expressions compute to values.</p>

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

<script>

var x;

x = 5;

document.getElementById("demo").innerHTML = x * 10;

</script>

</body>
```

```
</html>
```

The values can be of various types, such as numbers and strings. For example, "John" + " " + "Doe", evaluates to "John Doe":

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Expressions</h2>

<p>Expressions compute to values.</p>

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

<script>

document.getElementById("demo").innerHTML = "John" + " " +
"Doe";

</script>

</body>

</html>
```

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

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

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:

```
<!DOCTYPE html>

<html>

<body>

<h2>My First Web Page</h2>

<p>My First Paragraph.</p>

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

<script>

document.getElementById("demo").innerHTML = 5 + 6;

</script>

</body>

</html>
```

NB: *Changing the innerHTML property of an HTML element is a common way to display data in HTML.*

Using document.write()

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

```
<!DOCTYPE html>

<html>

<body>

<h2>My First Web Page</h2>

<p>My first paragraph.</p>



<p>Never call document.write after the document has finished loading.

It will overwrite the whole document.</p>

<script>

document.write(5 + 6);

</script>

</body>

</html>
```

NB: Using document.write() after an HTML document is loaded,

will delete all existing HTML:

```
<!DOCTYPE html>

<html>

<body>

<h2>My First Web Page</h2>

<p>My first paragraph.</p>

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

</body>

</html>
```

Using window.alert()

You can use an alert box to display data:

```
<!DOCTYPE html>

<html>

<body>

<h2>My First Web Page</h2>

<p>My first paragraph.</p>

<script>

window.alert(5 + 6);

</script>

</body>

</html>
```

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

```
<!DOCTYPE html>

<html>

<body>
```

```
<h2>My First Web Page</h2>

<p>My first paragraph.</p>

<script>

alert(5 + 6);

</script>

</body>

</html>
```

Using console.log()

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

```
<!DOCTYPE html>

<html>

<body>

<h2>Activate Debugging</h2>
```

```
<p>F12 on your keyboard will activate debugging.</p>

<p>Then select "Console" in the debugger menu.</p>

<p>Then click Run again.</p>

<script>

console.log(5 + 6);

</script>

</body>

</html>
```

JavaScript Print

JavaScript does not have any print object or print methods. You cannot access output devices from JavaScript. The only exception is that you can call the window.print() method in the browser to print the content of the current window.

```
<!DOCTYPE html>

<html>
```

```
<body>

<h2>The window.print() Method</h2>

<p>Click the button to print the current page.</p>

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



</body>

</html>
```

JavaScript Statements

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Statements</h2>

<p>A <b>JavaScript program</b> is a list of <b>statements</b> to be executed by a computer.</p>

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

<script>

let x, y, z; // Statement 1

x = 5; // Statement 2

y = 6; // Statement 3
```

```
z = x + y; // Statement 4

document.getElementById("demo").innerHTML =

"The value of z is " + z + ".";

</script>

</body>

</html>
```

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.

NB: 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":

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Statements</h2>

<p>In HTML, JavaScript statements are executed by the browser.</p>

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

<script>

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

```
</script>

</body>

</html>
```

Most JavaScript programs contain many JavaScript statements.

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

JavaScript code is frequently used to refer to JavaScript applications and statements.

Semicolons;

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Statements</h2>

<p>JavaScript statements are separated by semicolons.</p>

<p id="demo1"></p>

<script>

let a, b, c;

a = 5;

b = 6;

c = a + b;
```

```
document.getElementById("demo1").innerHTML    =
c;

</script>

</body>

</html>
```

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

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Statements</h2>

<p>Multiple statements on one line are allowed.
</p>
```

```
<p id="demo1"></p>

<script>

let a, b, c;

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

document.getElementById("demo1").innerHTML = c;

</script>

</body>

</html>
```

NB: *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:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Statements</h2>

<p>

The best place to break a code line is after an operator or a comma.
```

```
</p>

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

<script>

document.getElementById("demo").innerHTML =

"Hello Dolly!";

</script>

</body>

</html>
```

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:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Statements</h2>

<p>JavaScript code blocks are written between { and }</p>

<button type="button" onclick="myFunction()">Click Me!</button>

<p id="demo1"></p>

<p id="demo2"></p>

<script>

function myFunction() {
```

```
document.getElementById("demo1").innerHTML = "Hello Dolly!";

document.getElementById("demo2").innerHTML = "How are you?";

}

</script>

</body>

</html>
```

NB: *In this tutorial we use 2 spaces of indentation for code blocks.*
You will learn more about functions later in this tutorial.

JavaScript Keywords

JavaScript statements often start with a keyword to identify the JavaScript action to be performed.

Here is a list of some of the keywords you will learn about in this tutorial:

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

JavaScript keywords are reserved words.

Reserved words cannot be used as names for variables.

JavaScript syntax is the set of

rules, how JavaScript programs are constructed:

// How to create variables:

var

x;

let

y;

// How to use variables:

x = 5;

y = 6;

let

z = x + y;

JavaScript Values

The JavaScript syntax defines two types of values:

- Fixed values

- Variable values

Fixed values are called **Literals** .

Variable values are called **Variables** .

JavaScript Literals

The two most important syntax rules for fixed values are:

1. Numbers are written with or without decimals:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Numbers</h2>

<p>Number can be written with or without decimals.</p>

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

```
<script>

document.getElementById("demo").innerHTML      =
10.50;

</script>

</body>

</html>
```

---

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

```
<!DOCTYPE html>


<html>

<body>

<h2>JavaScript Strings</h2>

<p>Strings can be written with double or single
quotes.</p>
```

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

<script>

document.getElementById("demo").innerHTML      =
'John Doe';

</script>

</body>

</html>
```

Chapter Three

JavaScript Comment

Even though JavaScript is designed to be simple and easy to use, complicated code that is difficult to understand at a glance can nevertheless be written. The JavaScript standard offers two methods for creating code comments in certain scenarios so that developers may describe what's occurring in layman's terms.

Even though a browser cannot execute JavaScript comments, it is nevertheless a recommended practice for software developers to include comments. The provision of helpful comments to illustrate how a piece of code works is an indication of high-quality code, and almost all codes could use them.

This Section discusses recommended practices for writing the most impactful JavaScript comments as well as how to create them and use them within programs.

The significant method of message delivery is through the JavaScript comments. To make the code easier for the end user to understand, it is used to include comments, warnings, or other information about the code. The JavaScript engine, which is integrated into the browser, ignores the JavaScript comment.

JavaScript code may be made more understandable and explained with the use of comments. When testing alternative code, JavaScript comments may also be used to stop the code from running.

Developers always wind up generating sophisticated code as websites and systems change. When writing code in this manner, the original author or developer may understand it completely, but a developer who is new to the team or even the author's future self may not understand it at first. In these situations, comments serve as an extremely useful tool for outlining the decisions and methods of thinking that went into producing a block of code.

A Web Developer can write comments in JavaScript — both single-line comments for quick explanation and multi-line comments for a detailed explanation or formal documentation.

Advantages of JavaScript comments

There are mainly two advantages of JavaScript comments.

i. **To make code easy to understand** It can be used to elaborate the code so that end user can easily understand the code.

ii. **To avoid the unnecessary code** It can also be used to avoid the code being executed. Sometimes, we add the code to perform some action. But after sometime, there may be need to disable the code. In such case, it is better to use comments.

Types of JavaScript Comments

There are two types of comments in JavaScript.

i. Single-line Comment

ii. Multi-line Comment

JavaScript Single line Comment

Single line comments are very useful to add a quick comment or explanation over a line of code. Single line comments start with //. Any text between // and the end of the line will be ignored by JavaScript (will not be executed). It is represented by double forward slashes (//). It can be used before and after the statement. Let’s see the example of single-line comment i.e. added before the statement.

```
<!DOCTYPE html>

<html>

<body>

<h1 id="myH"></h1>

<p id="myP"></p>

<script>
```

```
// Change heading:

document.getElementById("myH").innerHTML       =
"JavaScript Comments";

// Change paragraph:

document.getElementById("myP").innerHTML = "My
first paragraph.";

</script>

</body>

</html>
```

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

```
<!DOCTYPE html>

<html>

<body>
```

```
<script>

var a=10;

var b=20;

var c=a+b;//It adds values of a and b variable

document.write(c);//prints sum of 10 and 20

</script>

</body>

</html>
```

JavaScript Multi line Comment

Sometimes, a single line is not enough to document or explain why certain code is written in a particular way or what it does. This is when a JavaScript Developer will opt for writing comments across multiple lines (a JavaScript multi-line comment can also be referred to as block comments). Multi-line comments start with /* and end with */. Any text between /* and */ will be ignored by JavaScript. It can be used to add single as well as multi line comments. So, it is

more convenient. It is represented by forward slash with asterisk then asterisk with forward slash. For example:

```
<!DOCTYPE html>

<html>

<body>

<h1 id="myH"></h1>

<p id="myP"></p>

<script>

/*

The code below will change

the heading with id = "myH"

and the paragraph with id = "myP"

*/

document.getElementById("myH").innerHTML =
"JavaScript Comments";
```

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

</script>

</body>

</html>
```

It can be used before, after and middle of the statement.

```
<html>

<body>

<script>

/* It is multi line comment.

It will not be displayed */

document.write("example of JavaScript multiline
comment");
```

```
</script>

</body>

</html>
```

Using JavaScript Comments to Prevent Code Execution

Since JavaScript comments are not executed, they’re a good way to prevent code execution while testing new features. This strategy allows you to locate bugs, progressively removing comments until you find the problematic code.

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:

```
<!DOCTYPE html>
```

```
<html>

<body>

<h2>JavaScript Comments</h2>

<h1 id="myH"></h1>

<p id="myP"></p>

<script>

//document.getElementById("myH").innerHTML =
"My First Page";

document.getElementById("myP").innerHTML = "My
first paragraph.";

</script>

<p>The line starting with // is not executed.</p>

</body>

</html>
```

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

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Comments</h2>

<h1 id="myH"></h1>

<p id="myP"></p>

<script>

/*

document.getElementById("myH").innerHTML =
"Welcome to my Homepage";

document.getElementById("myP").innerHTML =
"This is my first paragraph.";
```

```
*/

document.getElementById("myP").innerHTML =
"The comment-block is not executed.";

</script>

</body>

</html>
```

Commenting Out Function Calls

When creating a new function, it’s often helpful to make sure that the function doesn’t impact any other code negatively or unexpectedly. A common testing strategy is to comment out the new function, then make sure the rest of a program still runs as expected without it.

This example comments out a call to a newly implemented method to make sure it hasn’t affected the rest of the program before testing the method itself. Despite the function for changing the value of “X” existing in the code,

the commented out method call prevents the function’s execution.

```
function doSomething(x) {

let val = x / 2;

return val;

}

let x = 1;

let y = 4;

// x = doSomething(y); // call is commented out, "x" doesn't change
```

Commenting Out Function Bodies — Without Return Values

It isn’t always necessary to comment out an entire function when testing. If a function doesn’t return a value, you can

comment out the body of the function using a JavaScript multiline comment. In this case, the function itself would be called, but the value of “X” still wouldn’t change because nothing inside the function would execute.

```
let x = 1;

function doSomething(z) {

/*

let a = z * z;

x = a / 2;

*/}

doSomething(12);
```

Commenting Out Function Bodies — With Return Values

The previous technique won’t work the same way if the function immediately assigns its result to a variable; commenting out the function body makes the function return an undefined value. The undefined value then changes the value of “X”, which could confuse testing.

```
function doSomething(z) {

/*

let val = z / 2;

return val;

*/}

let x = 1;

let y = 4;

x = doSomething(y); // "X" would be undefined; may not be desirable
```

Careful commenting allows developers to pinpoint bugs and track how code works, but it’s easy to introduce unexpected behavior without meaning to while commenting, as the previous example shows. Always make sure you account for any ambiguity your comments may create in testing.

Writing Effective JavaScript Comments

Finding the right balance between too many comments and too few, or deciding on which style of comments is best for a certain piece of code, is an ongoing debate among developers, one that is unlikely to be resolved anytime soon. Some codebases follow a formal commenting scheme, while others don’t.

Developers have different preferences for when to use single line or multiline comments, but maintaining a consistent strategy when commenting code ensures that comments are clear regardless of the form they take. Some programmers use single line comments for everything, while others use multiline comments everywhere; some programmers only use block comments for formal documentation, while others use them for any long comment that takes up more than a certain number of lines.

The key to effective JavaScript comments, no matter what style you use, is to be consistent. As long as your comments are clear and give developers a walkthrough of how your code works, it doesn’t matter exactly what those comments look like.

Chapter Four

JavaScript Variable

Millions upon millions of websites and applications are made possible by JavaScript. It is crucial to save data and manage your information when putting such websites into use. JavaScript variables are a means of naming and storing data or information in memory so that it is easier to recall where it is kept.

In a program, variables are named as representatives. Variables are used in programming to represent other thingtws, much as x may indicate the location of the treasure on a pirate map or stand for an as-yet-unknown value in mathematics.

Variables can be thought of as data-containing containers. These containers can be named, so that at a later time, you can use the name of the container to remember and modify the data inside of it.

Consider creating an online store where you must keep track of product details. The product name, manufacturer, launch date, quantity available, and price must all need to be kept on file for each product. Since memory addresses on a computer are lengthy strings of characters and numbers that are impossible for a human to memorize, as a developer, you will find it increasingly challenging to recall where each piece of information is kept in memory.

These memory regions are named using a JavaScript variable, which allows one to simply use the variable name to obtain the information whenever it is needed again.

Variables are containers for values, and as variables are required to modify values, they are essential to the creation of anything interactive or dynamic in the JavaScript language. After using the let or var keywords to declare a variable, you assign a value to it. To put it simply, a JavaScript variable is a storage location name. In JavaScript, variables can be classified as either local or global.

Variables are pleasant terms used to store information. Consider how you address a person by name as opposed to using terms like "human," "one head, two eyes, one nose," and so on. All variables are just descriptive, human-friendly titles for data points. Variables can store values in the following several sorts of data:

- String

- Number
- Boolean
- Array
- Objects and symbols

JavaScript Keywords

JavaScript keywords are used to identify actions to be performed. The let keyword tells the browser to create variables:

```
<!DOCTYPE html>

<html>

<body>

<h2>The <b>let</b> Keyword Creates Variables</h2>

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

<script>

let x, y;

x = 5 + 6;

y = x * 10;
```

```
document.getElementById("demo").innerHTML = y;

</script>

</body>

</html>
```

The var keyword also tells the browser to create variables:

```
<!DOCTYPE html>

<html>

<body>

<h2>The var Keyword Creates Variables</h2>

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

<script>
```

```
var x, y;

x = 5 + 6;

y = x * 10;

document.getElementById("demo").innerHTML = y;

</script>

</body>

</html>
```

JavaScript variables can be created in three different ways using keywords *var* **,** *let* and *const* **.** Initially, variables were created using only the *var* keyword. But due to some historic drawbacks of using *var* , *let* and *const* were implemented for creating variables. It is recommended that Developers don ’ t use the *var* keyword anymore to declare variables. Irrespective of which keyword you use to get your variable declared, the syntax still remains the same.

<keyword> <variable_name> = <value_to_be_stored>

As seen above, a variable can be created by specifying the keyword, followed by a variable name that will be used to store the value and as well retrieve it later, a simple assignment operator (an equal sign) for assigning variable values, and the value to be stored itself.

You can choose to create a local variable or a global variable. Global variables are accessible from anywhere in the program. Local variables are variables declared inside a function, which avoids a conflict with another variable with the same variable name.

You can also declare multiple variables in one statement in JavaScript. Variable declaration can also span multiple lines.

JavaScript Variable Naming Convention

JavaScript variables are powerful programming constructs. However, there is a convention that should be followed by creating a variable in JavaScript.

Following are the rules that a Developer should follow to properly get variables declared:

- Variable name should only contain alphabets, numbers, $ and _

- Variable name should not start with a number

- Variable names are case-sensitive i.e. result and Result are two different variables

- Variables can’t be named as one of the reserved keywords like let, return, const, etc.

- Variable should use camelCase i.e. numOne is preferred over NumOne or numone

- Variable can’t have hyphen - in its name

- Use easy-to-understand names that symbolize the value stored in variables. For e.g. instead of calling a variable phNum, a Developer can call it phoneNumber.

- Don’t use single-letter variable names like x, a, z, etc.

JavaScript Var Keyword

Variables can be created using the var keyword. The only thing to remember is that JavaScript won’t complain or throw an error if a variable is being used before it is declared using the var keyword. In modern programming using JavaScript code, using the var keyword for variables is discouraged and should be replaced with let or const

Variables created using var keyword are also function-scoped or global-scoped i.e. it is very hard to limit wherein a large block of code the variable should be accessible. Hence, code written using the var keyword is hard to maintain.

```
var numOne = 20;

var numTwo = 30;

var result = numOne + numTwo;

console.log('Result is: ', result);
```

Example using var

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Variables</h1>

<p>In this example, x, y, and z are variables.</p>
```

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

<script>

var x = 5;

var y = 6;

var z = x + y;

document.getElementById("demo").innerHTML =

"The value of z is: " + z;

</script>

</body>

</html>
```

Note:

The var keyword was used in all JavaScript code from 1995 to 2015.

The let and const keywords were added to JavaScript in 2015.

The var keyword should only be used in code written for older browsers.

JavaScript Let Keyword

let keyword was introduced to solve hoisting issues that the var keyword had. let variables are block-scoped and are only accessible to where they are declared. This limits the issues of variables being overwritten somewhere else in the code. Apart from this, the variables created using the let keyword follow the same syntax as the ones created using the var keyword. Variables created using let and var keywords can be reassigned a value of a different kind. Hence they are mutable.

```
let numOne = 20;

let numTwo = 30;

var result = numOne + numTwo;

console.log('Result is: ', result); // should print 50
```

numThree = 60;

result = numOne + numThree; // reassign result new value

console.log('Result is: ', result); // should print 80

Example using let

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Variables</h1>

* In this example, x, y, and z are variables. *

* *

<script>

let x = 5;

```
let y = 6;

let z = x + y;

document.getElementById("demo").innerHTML =

"The value of z is: " + z;

</script>

</body>

</html>
```

JavaScript Const Keyword

Sometimes, variables created should not change the value assigned to it. This can’t be achieved if you declare a variable using let and var keywords. In such cases, a variable should be created using the const keyword. A variable created using const∏ can’t change the value assigned to it. It symbolizes constants.

let numOne = 20;

let numTwo = 30;

const result = numOne + numTwo;

console.log('Result is: ', result); // should print 50

numThree = 60;

result = numOne + numThree; // this is not allowed as result is a constant variable

console.log('Result is: ', result); // this will not be executed as above line has error

---

Example using Const

<!DOCTYPE html>

<html>

<body>

```
<h1>JavaScript Variables</h1>

<p>In this example, x, y, and z are variables.</p>

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

<script>

const x = 5;

const y = 6;

const z = x + y;

document.getElementById("demo").innerHTML =

"The value of z is: " + z;

</script>

</body>

</html>
```

const variables are also named differently sometimes when they store a value that would be otherwise hard to remember or store. Like private keys, colors, fonts, etc.

usually have complex values and hence const are appropriate for it.

---

const LIGHT_GRAY = ‘#ccc’;

const DARK_GRAY = ‘#eee’;

When to Use JavaScript const?

If you want a general rule: always declare variables with const. If you think the value of the variable can change, use let. In this example, price1, price2, and total, are variables:

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Variables</h1>

```
<p>In this example, price1, price2, and total are variables.</p>

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

<script>

const price1 = 5;

const price2 = 6;

let total = price1 + price2;

document.getElementById("demo").innerHTML =

"The total is: " + total;

</script>

</body>

</html>
```

The two variables price1 and price2 are declared with the const keyword. These are constant values and cannot

be changed. The variable total is declared with the let keyword. This is a value that can be changed.

JavaScript Local Variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. Variables declared within a JavaScript function, are **LOCAL** to the function: For example:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Scope</h2

<p><b>carName</b> is undefined outside myFunction():</p>

<p id="demo1"></p>

<p id="demo2"></p>

<script>
```

```
myFunction();

function myFunction() {

let carName = "Volvo";

document.getElementById("demo1").innerHTML    =
typeof carName + " " + carName;

}

document.getElementById("demo2").innerHTML    =
typeof carName;

</script>

</body>

</html>
```

When you use JavaScript, local variables are variables that are defined within functions. They have local scope, which means that they can only be used within the functions that define them. Accessing them outside the function will throw an error.

<script> *function abc(){* *var x=10;//local variable* *}* *</script>*	Or *<script>* *If(10<13){* *var y=20;//JavaScript local variable* *}* *</script>*

Local variables have **Function Scope** . They can only be accessed from within the function. Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

Function Scope

JavaScript has function scope: Each function creates a new scope. Variables defined inside a function are not accessible (visible) from outside the function. Variables declared with var, let and const are quite similar when declared inside a function. They all have **Function Scope** :

```
function myFunction() {
  var carName = "Ho
nda
";   // Function Scope
}
```

```
Function myFunction() {
  let carName = "Honda";   // Function Scope
}
```

```
function myFunction() {
  const carName = "Honda";   // Function
```

Scope

}

JavaScript Global Variable

These are variables that are defined in global scope i.e. outside of functions. These variables have a global scope, so they can be accessed by any function directly. In the case of global scope variables, the keyword they are declared with does not matter they all act the same. A variable declared without a keyword is also considered global even though it is declared in the function.

A JavaScript global variable is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

```
<!DOCTYPE html>

<html>

<body>
```

```
<h2>JavaScript Scope</h2>

<p>A GLOBAL variable can be accessed from any script or function.</p>

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

<script>

let carName = "Volvo";

myFunction();

function myFunction() {

document.getElementById("demo").innerHTML = "I can display " + carName;

}

</script>

</body>

</html>
```

```
<html>

<body>

<script>

var value=50;//global variable

function a(){

alert(value);

}

function b(){

alert(value);

}



a();

</script>

</body>
```

</html>

Another example

```
let petName = 'Rocky' // Global variable

myFunction()

function myFunction() {

fruit = 'apple'; // Considered global

console.log(typeof petName +

'- ' +

'My pet name is ' +

petName)

}

console.log(
```

```
typeof petName +

'- ' +

'My pet name is ' +

petName +

'Fruit name is ' +

fruit)
```

In the example above, We can see that the variable *petName* is declared in the global scope and is easily accessed inside functions. Also, the fruit was declared inside the function without any keyword so it was considered global and was accessible inside another function.

Declaring JavaScript global variable within function

To declare JavaScript global variables inside function, you need to use **window object** .

Now it can be declared inside any function and can be accessed from any function. For example:

```
<html>

<body>

<script>

function m(){

window.value=100;//declaring global variable by window object

}

function n(){

alert(window.value);//accessing global variable from other function

}
```

```
m();

n();

</script>

</body>

</html>
```

Internals of global variable in JavaScript

When you declare a variable outside the function, it is added in the window object internally. You can access it through window object also. For example:

```
var value=50;

function a(){

alert(window.value);//accessing  global  variabl
e
```

```
}
```

Automatically Global

If you assign a value to a variable that has not been declared, it will automatically become a **GLOBAL** variable.

This code example will declare a global variable carName, even if the value is assigned inside a function.

<!DOCTYPE html>

<html>

<body>

```
<h2>JavaScript Global Variables</h2>

<p>If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable:</p>

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

<script>

myFunction();
```

```
// code here can use carName as a
global variable

document.getElementById("demo").innerHTM
L = "I can display " + carName;

function myFunction() {

  carName = "Volvo";

}
```

</script>

</body>

</html>

Global Variables in HTML

With JavaScript, the global scope is the JavaScript environment.

In HTML, the global scope is the window object.

Global variables defined with the var keyword belong to the window object:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Scope</h2>

<p>In HTML, global variables defined with <b>var</b>, will become window variables.
</p>
```

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

<script>

var carName = "To

yota

";

// code here can use window.carName

document.getElementById("demo").innerHTML =
```

"I can display " + window.carName;

</script>

</body>

</html>

Global variables defined with the let keyword do not belong to the window object:

<!DOCTYPE html>

```
<html>

<body>

<h2>JavaScript Global Variables</h2>

<p>In HTML, global variables defined with <b>let</b>, will not become window variables.</p>

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

<script>

let carName = "To

yota

";

// code here can not use window.carName

document.getElementById("demo").innerHTML = "I can not display " + window.carName;

</script>

</body>

</html>

How to use variables

- The scope of a variable or function determines

what code has access to it.

- Variables that are created inside a function are local variables, and local variables can only be referred to by the code within the function.

- Variables created outside of functions are global variables, and the code in all functions has access to all global variables.

- If you forget to code the var keyword in a

variable declaration, the JavaScript engine assumes that the variable is global.

This can cause debugging problems.

-

In general, it’s better to pass local variables from one function to another as parameters than it is to use global variables.

That will make your code easier to understand with less chance of errors.

Where to use which variable

- Although it may seem easier to use global variables than to pass data to a function and return data from it, global variables often create problems. That’s because any function can modify a global variable, and it’s all too easy to misspell a variable name or modify the wrong variable, especially in large applications. That, in turn, can create debugging problems.

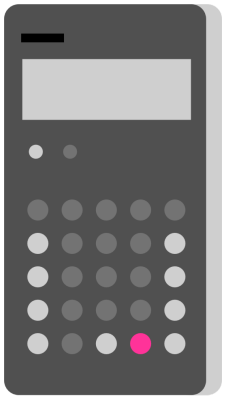
- In contrast, the use of local variables reduces the likelihood of naming conflicts. For instance, two different functions can use the same names for local variables without causing conflicts. That of course, means fewer errors and debugging problems. With just a few exceptions, then, all of the code in your applications should be in functions so all of the variables are local.

- If you misspell the name of a variable that you've already declared, it will be treated as a new global variable. With this in mind, be sure to include the keyword when you declare new variables and always declare a variable before you refer to it in your code.

Note: Use local variables whenever possible. Always use the var keyword to declare a new variable before the variable is referred to by other statements.

Chapter Five

JavaScript Operators



In JavaScript, an operator is a special symbol used to perform operations on operands (values and variables). JavaScript operators, expressions, and statements are the basic building blocks of programs. They help you manipulate and change values, perform math, compare two or more values, and much, much more.

Example: var sum=10+20;

Here, + is the arithmetic operator and = is the assignment operator.

At a high level, an *expression* is a valid unit of code that resolves to a value. There are two types of expressions: those that have side effects (such as assigning values) and those that purely *evaluate* .

The expression `x = 7` is an example of the first type. This expression uses the `=` *operator* to assign the value seven to the variable `x`. The expression itself evaluates to 7.

The expression `3 + 4` is an example of the second type. This expression uses the `+` operator to add `3` and `4` together and produces a value, `7`. However, if it's not eventually part of a bigger construct (for example, a variable declaration like `const z = 3 + 4`), its result will be immediately discarded — this is usually a programmer mistake because the evaluation doesn't produce any effects.

As the examples above also illustrate, all complex expressions are joined by *operators* , such as = and +. In this section, we will introduce the following operators:

- Assignment operators
- Comparison operators
- Arithmetic operators
- Bitwise operators
- Logical operators

- BigInt operators
- String operators
- Conditional (ternary) operator
- Comma operator
- Unary operators
- Relational operators

These operators join operands either formed by higher-precedence operators or one of the basic expressions . A complete and detailed list of operators and expressions is also available in the reference .

The *precedence* of operators determines the order they are applied when evaluating an expression. For example:

```
const x = 1 + 2 * 3;

const y = 2 * 3 + 1;
```

Despite * and + coming in different orders, both expressions would result in 7 because * has precedence over + , so the * -joined expression will always be evaluated first. You can override operator precedence by using parentheses (which creates a grouped expression — the basic expression).

JavaScript Assignment

The

assignment operator

assigns the value of the operand on the right to the operand on the left.

The **Assignment Operator** (=) assigns a value to a variable:

JavaScript provides the assignment operators to assign values to variables with less key strokes. An assignment operator assigns a value to its left operand based on the value of its right operand. The simple assignment operator is equal (=), which assigns the value of its right operand to its left operand. That is, x = f() is an assignment expression that assigns the value of f() to x.

There are also compound assignment operators that are shorthand for the operations listed in the following table:

Name	***Shorthand operator***	***Meaning***
Assignment	x = f()	x = f()
Addition assignment	x += f()	x = x + f()

Subtraction assignment	x -= f()	x = x - f()
Multiplication assignment	x *= f()	x = x * f()
Division assignment	x /= f()	x = x / f()
Remainder assignment	x %= f()	x = x % f()
Exponentiation assignment	x **= f()	x = x ** f()
Left shift assignment	x <<= f()	x = x << f()
Right shift assignment	x >>= f()	x = x >> f()
Unsigned right shift assignment	x >>>= f()	x = x >>> f()
Bitwise AND assignment	x &= f()	x = x & f()
Bitwise XOR assignment	x ^= f()	x = x ^ f()
Bitwise OR assignment	x |= f()	x = x | f()
Logical AND assignment	x &&= f()	x && (x = f())
Logical OR assignment	x ||= f()	x || (x = f())
Nullish coalescing assignment	x ??= f()	x ?? (x = f())

Assignment Examples

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Operators</h1>

<h2>The = Operator</h2>

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

<script>

let x = 10;

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>
```

Example 2:

```
<!DOCTYPE html>

<html>

<body>



<h1>JavaScript Operators</h1>

<h2>The Assignment (=) Operator</h2>



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



<script>

// Assign the value 5 to x

let x = 5;
```

```
// Assign the value 2 to y

let y = 2;

// Assign the value x + y to z

let z = x + y;

// Display z

document.getElementById("demo").innerHTML =
"The sum of x + y is: " + z;

</script>



</body>

</html>
```

Assignment Example 2:

```
<!DOCTYPE html>
```

```
<html>

<body>

<h1>Example: JavaScript Assignment Operators</h1>



<p id="p1"></p>

<p id="p2"></p>

<p id="p3"></p>

<p id="p4"></p>

<p id="p5"></p>

<p id="p6"></p>

<script>

let x = 5, y = 10;

x = y;

document.getElementById("p1").innerHTML = x;
```

```
x += 1;

document.getElementById("p2").innerHTML = x;

x -= 1;

document.getElementById("p3").innerHTML = x;

x *= 5;

document.getElementById("p4").innerHTML = x;

x /= 5;

document.getElementById("p5").innerHTML = x;

x %= 2;

document.getElementById("p6").innerHTML = x;

</script>

</body>

</html>
```

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

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

The commonly used assignment operator is =. You will understand other assignment operators such as +=, -=, *= etc as shown in the table below .

Name	**Example**	**Operat**

or

=	Assignment operator	a = 7; // 7
+=	Addition assignment	a += 5; // a = a + 5
-=	Subtraction Assignment	a -= 2; // a = a - 2
***=**	Multiplication Assignment	a *= 3; // a = a * 3
/=	Division Assignment	a /= 2; // a = a / 2
%=	Remainder Assignment	a %= 2; // a = a % 2
****=**	Exponentiation Assignment	a **= 2; // a = a**2

Example : Arithmetic operators in JavaScript

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Arithmetic</h1>

<h2>Arithmetic Operations</h2>

<p>A typical arithmetic operation takes two numbers (or expressions) and produces a new number.</p>

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

```
<script>

let x = 5;

let y = 3;

// addition

console.log('x + y = ', x + y); // 8

// subtraction

console.log('x - y = ', x - y); // 2

// multiplication

console.log('x * y = ', x * y); // 15

// division

console.log('x / y = ', x / y); // 1.6666666666666667

// remainder

console.log('x % y = ', x % y); // 2

// increment
```

```
console.log('++x = ', ++x); // x is now 6

console.log('x++ = ', x++); // prints 6 and then increased to 7

console.log('x = ', x); // 7

// decrement

console.log('--x = ', --x); // x is now 6

console.log('x-- = ', x--); // prints 6 and then decreased to 5

console.log('x = ', x); // 5

//exponentiation

console.log('x ** y =', x ** y);

document.getElementById("demo").innerHTML = x / y;

</script>

</body>
```

</html>

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

The **Addition Assignment Operator** (+=) adds a value to a variable.

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

Example:

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Arithmetic</h1>

<h2>The += Operator</h2>

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

<script>

var x = 10;

x += 5;

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>
```

JavaScript Comparison Operators

JavaScript provides comparison operators that compare two operands and return a boolean value **true** or **false.**

Operators	***Description***
==	Compares the equality of two operands without considering type.
===	Compares equality of two operands with type.
!=	Compares inequality of two operands.
>	Returns a boolean value true if the left-side value is greater than the right-side value; otherwise, returns false.
<	Returns a boolean value true if the left-side value is less than the right-side value; otherwise, returns false.
>=	Returns a boolean value true if the left-side value is greater than or equal to the right-side value; otherwise, returns false.
<=	Returns a boolean value true if the left-side value is less than or equal to the right-side value; otherwise, returns false.

Comparison operators compare two values and return a boolean value, either true or false.

For example,

Const a=3, b=2;

Console.log(a>b); //true

Here, the comparison operator

>

is used to compare whether

a

is greater than

b.

Operator	***Description***	***Example***

==	Equal to: returns true if the operands are equal	x == y	x
!=	Not equal to: returns true if the operands are not equal	x != y	x
===	Strict equal to: true if the operands are equal and of the same type	x === y	x
!==	Strict not equal to: true if the operands are equal but of different type or not equal at all	x !== y	x
>	Greater than: true if left operand is greater than the right operand	x > y	x
>=	Greater than or equal to: true if left operand is greater than or equal to the right operand	x >= y	x
	Less than: true if the left		x

<	operand is less than the right operand	< y
<=	Less than or equal to: true if the left operand is less than or equal to the right operand	x <= y

Example

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript String Operators</h1>

<p>All conditional operators can be used on both numbers and strings.</p>

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

<script>

let text1 = "A";

let text2 = "B";

let result = text1 < text2;

document.getElementById("demo").innerHTML = "Is A less than B? " + result;
```

</script>

</body>

</html>

Note that strings are compared alphabetically:

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript String Operators</h1>

* Note that strings are compared alphabetically:*
* *

* *

<script>

let text1 = "20";

```
let text2 = "5";

let result = text1 < text2;

document.getElementById("demo").innerHTML = "Is
20 less than 5? " + result;

</script>

</body>

</html>
```

JavaScript String Addition

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

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript String Operators</h1>
```

```
<h2>The + Operator</h2>

<p>The + operator concatenates (adds) strings.
</p>

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

<script>

let text1 = "Jeremiah";

let text2 = "Timothy";

let text3 = "Mathew";

let text4 = text1 + " " + text2 + " " + text3;

document.getElementById("demo").innerHTML =
text4;

</script>

</body>

</html>
```

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

Example

```
let
text1 =
"How are You Doing Today?"
;
text1 +=
"I am Fine Thank You"
;
```

The result of text1 will be:

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript String Operators</h1>

<h2>The += Operator</h2>

<p>The assignment operator += can concatenate strings.</p>

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

<script>

let text1 = "How are you Doing Today? ";

text1 += "I am Fine Thank you";

document.getElementById("demo").innerHTML = text1;

</script>

</body>
```

</html>

Note

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

Adding Strings and Numbers

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

Example

```
let x = 5 + 5;

let y =

"5"

 + 5;

let z =

"Hello"

 + 5;
```

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript String Operators</h1>

<h2>The + Operator</h2>

<p>Adding a number and a string, returns a string.
</p>

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

<script>

let x = 5 + 5;

let y = "5" + 5;

let z = "Hello" + 5;

document.getElementById("demo").innerHTML =
```

```
x + "<br>" + y + "<br>" + z;

</script>

</body>

</html>
```

JavaScript Logical Operators

Logical operators are typically used with Boolean (logical) values; when they are, they return a Boolean value. However, the && and || operators actually return the value of one of the specified operands, so if these operators are used with non-Boolean values, they may return a non-Boolean value. The logical operators are described in the following table. Logical operators perform logical operations and return a boolean value, either true or false.

	Description

Operator	
&&	&& is known as AND operator. It checks whether two operands are non-zero or not (0, false, undefined, null or "" are considered as zero). It returns 1 if they are non-zero; otherwise, returns 0.
||	|| is known as OR operator. It checks whether any one of the two operands is non-zero or not (0, false, undefined, null or "" is considered as zero). It returns 1 if any one of of them is non-zero; otherwise, returns 0.
!	! is known as NOT operator. It reverses the boolean result of the operand (or condition). !false returns true, and !true returns false.

```
<!DOCTYPE html>

<html>

<body>

    <h1>Demo: JavaScript Logical
Operators</h1>
```

<script>

let a = 5, b = 10;

```
document.getElementById("p1").innerHTML = (a !=
b) && (a < b);
```

```
document.getElementById("p2").innerHTML = (a >
b) || (a == b);
```

```
document.getElementById("p3").innerHTML = (a <
b) || (a == b);
```

```
document.getElementById("p4").innerHTML = !(a <
b);
```

```
document.getElementById("p5").innerHTML = !(a >
b);
```

```
</script>


</body>


</html>
```

JavaScript Bitwise Operators

A bitwise operator treats their operands as a set of 32 bits (zeros and ones), rather than as decimal, hexadecimal, or octal numbers. For example, the decimal number nine has a binary representation of 1001. Bitwise operators perform their operations on

such binary representations, but they return standard JavaScript numerical values.

Bitwise operators perform operations on binary representations of numbers.

The following table summarizes JavaScript's bitwise operators.

Operator	***Usage***	***Description***
Bitwise AND	a & b	Returns a one in each bit position for which the corresponding bits of both operands are ones.
Bitwise OR	a | b	Returns a zero in each bit position for which the corresponding bits of both operands are zeros.
Bitwise XOR	a ^ b	Returns a zero in each bit position for which the corresponding bits are the same. [Returns a one in each bit position for which the corresponding bits are different.]
Bitwise NOT	~ a	Inverts the bits of its operand.

Left shift	a << b	Shifts a in binary representation b bits to the left, shifting in zeros from the right.
Sign-propagating right shift	a >> b	Shifts a in binary representation b bits to the right, discarding bits shifted off.
Zero-fill right shift	a >>> b	Shifts a in binary representation b bits to the right, discarding bits shifted off, and shifting in zeros from the left.

Bitwise logical operators

Conceptually, the bitwise logical operators work as follows:

The operands are converted to thirty-two-bit integers and expressed by a series of bits (zeros and ones). Numbers with more than 32 bits get their most significant bits discarded.

Operator	***Name***	***Description***

&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shifts left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off
>>>	Zero fill right shift	Shifts right by pushing zeros in from the left, and let the rightmost bits fall off

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

Operation	***Result***	***Same as***	***Result***
5 & 1	1	0101 & 0001	0001
5 | 1	5	0101 | 0001	0101
~ 5	10	~0101	1010
5 << 1	10	0101 << 1	1010
5 ^ 1	4	0101 ^ 0001	0100
5 >> 1	2	0101 >> 1	0010
5 >>> 1	2	0101 >>> 1	0010

Each bit in the first operand is paired with the corresponding bit in the second operand: first bit to first bit, second bit to second bit, and so on.

The operator is applied to each pair of bits, and the result is constructed bitwise.

For example, the binary representation of nine is 1001, and the binary representation of fifteen is 1111. So, when the

bitwise operators are applied to these values, the results are as follows:

Expression	***Result***	***Binary Description***
15 & 9	9	1111 & 1001 = 1001
15 | 9	15	1111 | 1001 = 1111
15 ^ 9	6	1111 ^ 1001 = 0110
~15	-16	~ 0000 0000 ... 0000 1111 = 1111 1111 ... 1111 0000
~9	-10	~ 0000 0000 ... 0000 1001 = 1111 1111 ... 1111 0110

Note that all 32 bits are inverted using the Bitwise NOT operator, and that values with the most significant (left-most) bit set to 1 represent negative numbers (two's-complement representation). `~x` evaluates to the same value that `-x - 1` evaluates to.

JavaScript Bitwise AND

When a bitwise AND is performed on a pair of bits, it returns 1 if both bits are 1.

One bit example:	***4 bits example:***

Operation	**Result**	**Operation**		**Result**
0 & 0	0	1111 0000	&	0000
0 & 1	0	1111 0001	&	0001
1 & 0	0	1111 0010	&	0010
1 & 1	1	1111 0100	&	0100

Example

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Bitwise AND</h1>

<h2>The & Operator</h2>
```

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

<script>

document.getElementById("demo").innerHTML  =  5
& 1;

</script>

</body>

</html>
```

JavaScript Bitwise OR

When a bitwise OR is performed on a pair of bits, it returns 1 if one of the bits is 1:

One bit example:	***4 bits example:***

Operation	**Operation Result**
0 | 0	1111 | 0000 1111
0 | 1	1111 | 0001 1111
1 | 0	1111 | 0010 1111
1 | 1	1111 | 0100 1111

Example:

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Bitwise OR</h1>

<h2>The | Operator</h2>

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

<script>

document.getElementById("demo").innerHTML = 5 |
1;

</script>

</body>
```

</html>

JavaScript Bitwise XOR

When a bitwise XOR is performed on a pair of bits, it returns 1 if the bits are different:

One bit example:	4 bits example:
Operation Result	**Operation Result**
0 ^ 0 0	1111 ^ 0000 1111
0 ^ 1 1	1111 ^ 0001 1110
1 ^ 0 1	1111 ^ 0010 1101
1 ^ 1 0	1111 ^ 0100 1011

Example:

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Bitwise XOR</h1>

<h2>The ^ Operator</h2>

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

<script>

document.getElementById("demo").innerHTML  =  5
^ 1;

</script>

</body>
```

```
</html>
```

JavaScript Bitwise NOT (~)

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Bitwise NOT</h1>

<h2>The ~ Operator</h2>

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

<script>

document.getElementById("demo").innerHTML = ~
5;

</script>
```

```
</body>

</html>
```

JavaScript (Zero Fill) Bitwise Left Shift (<<)

The left shift (**<<**) operator returns a number or BigInt whose binary representation is the first operand shifted by the specified number of bits to the left. Excess bits shifted off to the left are discarded, and zero bits are shifted in from the right.

The << operator is overloaded for two types of operands: number and BigInt . For numbers, the operator returns a 32-bit integer. For BigInts, the operator returns a BigInt. It first coerces both operands to numeric values and tests the types of them. It performs BigInt left shift if both operands becomes BigInts; otherwise, it converts both operands to 32-bit integers and performs number left shift. A TypeError is thrown if one operand becomes a BigInt but the other becomes a number.

This is a zero fill left shift. One or more zero bits are pushed in from the right, and the leftmost bits fall off:

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Bitwise Left</h1>

<h2>The << Operator</h2>

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

<script>

document.getElementById("demo").innerHTML  =  5
<< 1;

</script>

</body>

</html>
```

JavaScript (Sign Preserving) Bitwise Right Shift (>>)

The right shift (**>>**) operator returns a number or BigInt whose binary representation is the first operand shifted by the specified number of bits to the right. Excess bits shifted off to the right are discarded, and copies of the leftmost bit are shifted in from the left. This operation is also called "sign-propagating right shift" or "arithmetic right shift", because the sign of the resulting number is the same as the sign of the first operand.

The >> operator is overloaded for two types of operands: number and BigInt . For numbers, the operator returns a 32-bit integer. For BigInts, the operator returns a BigInt. It first coerces both operands to numeric values and tests the types of them. It performs BigInt right shift if both operands becomes BigInts; otherwise, it converts both operands to 32-bit integers and performs number right shift. A TypeError is thrown if one operand becomes a BigInt but the other becomes a number.

This is a sign preserving right shift. Copies of the leftmost bit are pushed in from the left, and the rightmost bits fall off:

```
<!DOCTYPE html>

<html>

<body>
```

```
<h1>JavaScript Signed Bitwise Right</h1>

<h2>The >> Operator</h2>

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

<script>

document.getElementById("demo").innerHTML = -5
>> 1;

</script>

</body>

</html>
```

JavaScript (Zero Fill) Right Shift (>>>)

Zero-fill right shift (>>>) operator: It is a binary operator, where the first operand specifies the number and the second operand specifies the number of bits to shift . The operator shifts the bits of the first operand by a number of bits specified by the second operand.

The unsigned right shift (**>>>**) operator returns a number whose binary representation is the first operand shifted by the specified number of bits to the right. Excess bits shifted off to the right are discarded, and zero bits are shifted in from the left. This operation is also called "zero-filling right shift", because the sign bit becomes 0 , so the resulting number is always positive. Unsigned right shift does not accept BigInt values.

Unlike other arithmetic and bitwise operators, the unsigned right shift operator does not accept BigInt values. This is because it fills the leftmost bits with zeroes, but conceptually, BigInts have an infinite number of leading sign bits, so there's no "leftmost bit" to fill with zeroes.

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Unsigned Bitwise Right</h1>

<h2>The >>> Operator</h2>

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

```
<script>

document.getElementById("demo").innerHTML  =  5
>>> 1;

</script>

</body>

</html>
```

Converting Decimal to Binary

Example 1:

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Convert Decimal to Binary</h1>

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

```
<script>

document.getElementById("demo").innerHTML      =
dec2bin(-5);

function dec2bin(dec){

return (dec >>> 0).toString(2);

}

</script>

</body>

</html>
```

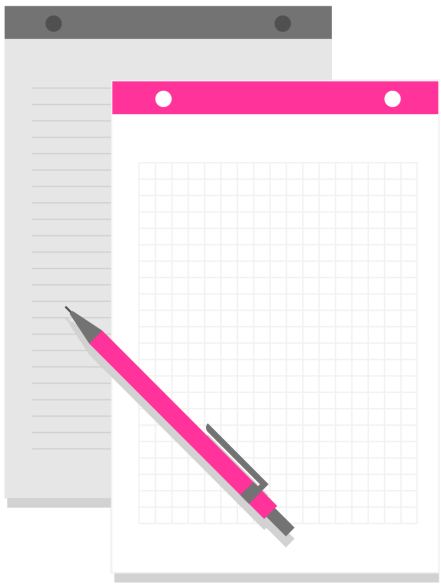
Converting Binary to Decimal

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Convert Binary to Decimal</h2>

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

<script>

document.getElementById("demo").innerHTML      =
bin2dec(101);

function bin2dec(bin){

return parseInt(bin, 2).toString(10);

}

</script>
```

```
</body>

</html>
```

Chapter Six

JavaScript Data Types



A variable ’ s data type is the kind of data the variable can hold and what operations can be done with the value of the variable. The number 10, used in a sentence, is different than the number 10 used in an equation.

Data types are the way JavaScript distinguishes between values that are meant to be words and values that are meant to be treated as mathematical expressions.

Programming languages all have built-in data structures, but these often differ from one language to another. If you think

about all the types of data that you work with on a daily basis, pie charts, recipes, short stories, newspaper articles, and so on, you ’ ll see just how much potential there is for things to get very complicated when it comes to data. The generous creators of JavaScript decided to make things very simple for you. It has just five basic data types.

This section attempts to list the built-in data structures available in JavaScript and what properties they have.

These can be used to build other data structures.

A value in JavaScript is always of a certain type. For example, a string or a number.

Furthermore, JavaScript is what ’ s called a loosely typed language. What *loosely typed* means is that you don ’ t even need to tell JavaScript, or even know, whether a variable you ’ re creating will hold a word, a paragraph, a number, or a different type of data.

Loosely typed doesn

’

t mean that JavaScript doesn

,

t distinguish between words and numbers.

JavaScript just is friendly about it and handles the work of figuring out what type of data you store in your variables largely behind the scenes.

There are eight basic data types in JavaScript.

We can put any type in a variable.

For example, a variable can at one moment be a string and then store a number:

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

1. Primitive data type

2. 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.

There are eight basic data types in JavaScript. They are:

Data Types	***Description***	***Example***
String	represents textual data	'hello', "hello world!" etc
	an integer	3, 3.234, 3e-2

Number	or a floating-point number	etc.
BigInt	an integer with arbitrary precision	900719925124740999n , 1n etc.
Boolean	Any of two values: true or false	true and false
undefined	a data type whose variable is not initialized	let a;
null	denotes a null value	let a = null;
Symbol	data type whose instances are unique and immutable	let value = Symbol('hello');
Object	key-value pairs of collection of data	let student = { };

Here, all data types except

Object

are primitive data types, whereas

Object

is non-primitive.

JavaScript primitive data types

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

Data Type	***Description***
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

JavaScript non-primitive data types

The non-primitive data types are as follows:

Data Type	***Description***
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

Examples

// Numbers:

let

length = 16;

let

weight = 7.5;

// Strings:

let

color =

"Yellow"

;

let

lastName =

"Johnson"

;

// Booleans

let

x =

true

;

let

y =

false

;

// Object:

```
const person = {firstName: "John", lastName: "Doe"};
```

```
// Array object:

const cars = ["Saab"
```

,

"Volvo"

,

"BMW"

];

// Date object:

const

date =

new

Date(

"2022-03-25"

);

The Concept of Data Types

In programming, data types are an important concept. To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

---

let x = 16 + "Volvo" ;

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

JavaScript will treat the example above as:

```
let x = "16" + "Volvo" ;
```

Note

When adding a number and a string, JavaScript will treat the number as a string.

```
<!DOCTYPE html>

<html>

<body>
```

```
<h2>JavaScript</h2>

<p>When adding a number and a string, JavaScript
will treat the number as a string.</p>

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

<script>

let x = 16 + "Volvo";

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>
```

JavaScript Types are Dynamic

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Data Types</h2>

<p>JavaScript has dynamic types. This means that the same variable can be used to hold different data types:</p>

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

<script>

let x; // Now x is undefined

x = 5; // Now x is a Number

x = "John"; // Now x is a String

document.getElementById("demo").innerHTML = x;

</script>

</body>
```

</html>

JavaScript Strings

The String type represents textual data and is encoded as a sequence of 16-bit unsigned integer values representing UTF-16 code units . Each element in the string occupies a position in the string. The first element is at index 0, the next at index 1, and so on. The length of a string is the number of UTF-16 code units in it, which may not correspond to the actual number of Unicode characters; see the String reference page for more details.

JavaScript strings are immutable. This means that once a string is created, it is not possible to modify it. String methods create new strings based on the content of the current string — for example:

- A substring of the original using substring().

- A concatenation of two strings using the concatenation operator (+) or .

String

is used to store text.
In JavaScript, strings are surrounded by quotes:

Single quotes:

'Hello'

Double quotes:

"Hello"

Backticks:

`Hello`

Example:

```
//strings example

Const name =’ram’;

Const name1 =”cow”;
```

```
Const result =’the names are ${name} and ${name1}’;
```

Single quotes and double quotes are practically the same and you can use either of them.

Backticks are generally used when you need to include variables or expressions into a string. This is done by wrapping variables or expressions with ${variable or expression} as shown above.

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Strings</h2>

<p>You can use quotes inside a string, as long as they don't match the quotes surrounding the string:
</p>

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

```
<script>

let answer1 = "It's alright";

let answer2 = "He is called 'Johnny'";

let answer3 = 'He is called "Johnny"';

document.getElementById("demo").innerHTML =

answer1 + "<br>" +

answer2 + "<br>" +

answer3;

</script>

</body>

</html>
```

JavaScript String Methods

In JavaScript, strings are used to represent and work with a sequence of characters. A string can represent an object as well as the primitive data type. JavaScript automatically converts primitive strings to String objects so that it's possible to use String methods and access properties even for primitive strings.

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

JavaScript String Length

The length property returns the length of a string:

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Strings</h1>
```

<h2>The length Property</h2>

The length of the string is:

<script>

let text =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";

```
document.getElementById("demo").innerHTML =
text.length;
```

```
</script>
```

```
</body>
```

```
</html>
```

Extracting String Parts

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

- **slice(** ***start*** **,** ***end*** **)**

- **substring(*start* , *end*)**

- **substr(*start* , *length*)**

JavaScript String slice()

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

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

```
<!DOCTYPE html>

<html>

<body>
```

```
<h1>JavaScript Strings</h1>

<h2>The slice() Method</h2>

<p>The sliced (extracted) part of the string is:</p>

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

<script>

let text = "Apple, JUMP, Kiwi";

let part = text.slice(7,13);

document.getElementById("demo").innerHTML = part;

</script>

</body>

</html>
```

Note

JavaScript counts positions from zero.First position is 0, while Second position is 1.

Examples

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

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Strings</h1>

<h2>The slice() Method</h2>

<p>Extract a part of a string from position 7:</p>

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

<script>

let text = "Apple, Banana, Kiwi";

let part = text.slice(7)
```

```
document.getElementById("demo").innerHTML = part;

</script>

</body>

</html>
```

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

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Strings</h1>

<h2>The slice() Method</h2>

<p>Extract a part of a string counting from the end:</p>

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

```
<script>

let text = "Apple, Banana, Kiwi";

let part = text.slice(-12);

document.getElementById("demo").innerHTML      =
part;

</script>

</body>

</html>
```

JavaScript String substring()

substring() is similar to slice().

The difference is that start and end values less than 0 are treated as 0 in substring().

```
<!DOCTYPE html>
```

```
<html>

<body>

<h1>JavaScript String Methods</h1>

<p>The substring() method extract a part of a string and returns the extracted parts in a new string:</p>

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

<script>

let str = "Apple, Banana, Kiwi";

document.getElementById("demo").innerHTML = str.substring(7,13);

</script>

</body>

</html>
```

Replacing String Content

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

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript String Methods</h1>

<p>Replace "How Can i Help You" with "Are You Happy?" in the paragraph below:</p>

<button onclick="myFunction()">Try it</button>

<button onclick="myFunction()">How can i help you</button>

<p id="demo">Testing JavaScript String Method!</p>

<script>

function myFunction() {
```

```
let text = document.getElementById("demo").innerHTML;

document.getElementById("demo").innerHTML =

text.replace("Testing","Trying it Out ");

}

</script>

</body>

</html>
```

Note

The replace() method does not change the string it is called on.

The replace() method returns a new string.

The replace() method replaces **only the first** match

JavaScript String ReplaceAll()

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

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Strings</h1>

<h2>The replaceAll() Method</h2>

<p>ES2021 intoduced the string method replaceAll().</p>

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

<script>

let text = "I love cats. Cats are very easy to love. Cats are very popular."

text = text.replaceAll("Cats","Dogs");

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

document.getElementById("demo").innerHTML = text;

</script>

</body>

</html>

The replaceAll() method allows you to specify a regular expression instead of a string to be replaced.

If the parameter is a regular expression, the global flag (g) must be set, otherwise a TypeError is thrown.

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Strings</h1>

<h2>The replaceAll() Method</h2>

```
<p>ES2021 intoduced the string method replaceAll().</p>

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

<script>

let text = "I love cats. Cats are very easy to love. Cats are very popular";

text = text.replaceAll(/Cats/g,"Dogs");

text = text.replaceAll(/cats/g,"dogs");

document.getElementById("demo").innerHTML = text;

</script>

</body>

</html>
```

Converting to Upper and Lower Case

A string is converted to upper case with toUpperCase():

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

JavaScript String toUpperCase()

<!DOCTYPE html>

<html>

<body>

```
<h1>JavaScript String Methods</h1>

<p>Convert string to upper case:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo">Hello World!</p>

<script>

function myFunction() {
```

```
let text =
document.getElementById("demo").innerHTML;

document.getElementById("demo").innerHTML =

text.toUpperCase();

}

</script>
```

```
</body>

</html>
```

JavaScript String to LowerCase()

```
<!DOCTYPE html>

<html>
```

<body>

<h1>JavaScript String Methods</h1>

Convert string to lower case:

<button onclick="myFunction()">Try it</button>

Hello World!

<script>

```
function myFunction() {

  let text =
document.getElementById("demo").innerHTML;

document.getElementById("demo").innerHTML =

  text.toLowerCase();

}
```

```
</script>

</body>

</html>
```

JavaScript String concat()

concat() joins two or more strings:

```
<!DOCTYPE html>
```

```
<html>

<body>

<h1>JavaScript String</h1>

<h2>The concat() Method</h2>

<p>The concat() method joins two or
more strings:</p>

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

<script>

let text1 = "Hello";

let text2 = "World!";

let text3 = text1.concat(" ",text2);

document.getElementById("demo").innerHTML =
text3;

```
</script>

</body>

</html>
```

JavaScript String trim()

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

```
<!DOCTYPE html>
```

```
<html>

<body>

<h1>JavaScript Strings</h1>

<h2>The trim() Method</h2>

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

<script>
```

```
let text1 = "     Hello World!
";

let text2 = text1.trim();

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

</script>
```

```
</body>


</html>
```

JavaScript Numbers

Numbers in JavaScript are stored as 64-bit, floating point values. What this means, in English, is that numbers can range from 5e-324 (that ’ s -5 followed by 324 zeros) to 1.7976931348623157e+308 (move the decimal 308 spots to the right to see this giant number). Any number may have decimal points or not. Unlike most programming languages, JavaScript doesn ’ t have separate data types for integers (positive or negative numbers without a fractional part) and floating points (decimals).

The Number type is a double-precision 64-bit binary format IEEE 754 value . It is capable of storing positive floating-point numbers between 2^{-1074} (Number.MIN_VALUE) and 2^{1024} (Number.MAX_VALUE) as well as negative floating-point numbers between -2^{-1074} and -2^{1024}, but it can only safely store integers in the range $-(2^{53} - 1)$ (Number.MIN_SAFE_INTEGER) to $2^{53} - 1$ (Number.MAX_SAFE_INTEGER). Outside this range, JavaScript can no longer safely represent integers; they will instead be represented by a double-precision floating point approximation. You can check if a number is within the range of safe integers using Number.isSafeInteger().

Values outside the range $\pm$ (2^{-1074} to 2^{1024}) are automatically converted:

- Positive values greater than Number.MAX_VALUE are converted to +Infinity.

- Positive values smaller than Number.MIN_VALUE are converted to +0.

- Negative values smaller than -Number.MAX_VALUE are converted to -Infinity.

- Negative values greater than -Number.MIN_VALUE are converted to -0.

+Infinity and -Infinity behave similarly to mathematical infinity, but with some slight differences.

The Number type has only one value with multiple representations: 0 is represented as both -0 and +0 (where 0 is an alias for +0). In practice, there is almost no difference between the different representations; for

example, +0 === -0 is true. However, you are able to notice this when you divide by zero:

NaN ("Not a Number") is a special kind of number value that's typically encountered when the result of an arithmetic operation cannot be expressed as a number. It is also the only value in JavaScript that is not equal to itself.

Although a number is conceptually a "mathematical value" and is always implicitly floating-point-encoded, JavaScript provides bitwise operators . When applying bitwise operators, the number is first converted to a 32-bit integer.

All JavaScript numbers are stored as decimal numbers (floating point).

Numbers can be written with, or without decimals:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Numbers</h2>
```

```
<p>Numbers can be written with, or without decimals:</p>

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

<script>

let x1 = 34.00;

let x2 = 34;

let x3 = 3.14;

document.getElementById("demo").innerHTML =

x1 + "<br>" + x2 + "<br>" + x3;

</script>

</body>

</html>
```

JavaScript Random

The Math.random() static method returns a floating-point, pseudo-random number that's greater than or equal to 0 and less than 1, with approximately uniform distribution over that range which you can then scale to your desired range.

The implementation selects the initial seed to the random number generation algorithm; it cannot be chosen or reset by the user.

Syntax:

Math.random()

Return Value: *The Math.random() method returns a value less than 1.*

```
<!DOCTYPE html>

<html>

<body>
```

```
<h2>JavaScript Math.random()</h2>

<p>Every time you click the button, getRndInteger(min, max) returns a random number between 0

and 9 (both included):</p>

<button onclick="document.getElementById('demo').innerHTML = getRndInteger(0,10)">Click Me</button>

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

<script>

function getRndInteger(min, max) {

return Math.floor(Math.random() * (max - min)) + min;

}

</script>

</body>

</html>
```

As you can see from the examples above, it might be a good idea to create a proper random function to use for all random integer purposes.

This JavaScript function always returns a random number between min (included) and max (excluded):

In the example below: This JavaScript function always returns a random number between min and max (both included):

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Math.random()</h2>

<p>Every time you click the button, getRndInteger(min, max) returns a random number between 1 and 10 (both included):</p>



<button
onclick="document.getElementById('demo').innerHT
```

```
ML = getRndInteger(1,10)">Click Me</button>

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

<script>

function getRndInteger(min, max) {

return Math.floor(Math.random() * (max - min + 1) )
+ min;

}

</script>

</body>

</html>
```

Exponential Notation

Extra large or extra small numbers can be written with scientific (exponential) notation:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Numbers</h2>

<p>Extra large or extra small numbers can be written with scientific (exponential) notation:</p>

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

<script>

let y = 123e5;

let z = 123e-5;
```

document.getElementById("demo").innerHTML =

*y + "
" + z;*

</script>

</body>

</html>

Note

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 BigInt

The BigInt type is a numeric primitive in JavaScript that can represent integers with arbitrary magnitude. With BigInts, you can safely store and operate on large integers even beyond the safe integer limit (Number.MAX_SAFE_INTEGER) for Numbers.

A BigInt is created by appending n to the end of an integer or by calling the BigInt() function.

In JavaScript, Number type can only represent numbers less than (253 - 1) and more than -(253 - 1). However, if you need to use a larger number than that, you can use the BigInt data type.

A BigInt number is created by appending n to the end of an integer.

All JavaScript numbers are stored in a a 64-bit floating-point format.

JavaScript BigInt is a new datatype (ES2020) that can be used to store integer values that are too big to be

represented by a normal JavaScript Number.

```
<!DOCTYPE html>

<html>

<body>

<h1>JavScript Bigint</h1>

<p>A BigInt can not have decimals.</p>

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

<p>You cannot perform math between a BigInt type and a Number type.</p>



<script>

let x = BigInt("12345678901234567890123456789O");

document.getElementById("demo").innerHTML = x;
```

```
</script>

</body>

</html>
```

JavaScript Integer Accuracy

JavaScript integers are only accurate up to 15 digits: In JavaScript, all numbers are stored in a 64-bit floating-point format (IEEE 754 standard).

With this standard, large integer cannot be exactly represented and will be rounded.

Because of this, JavaScript can only safely represent integers:

Up to **9007199254740991** $+(2^{53}-1)$

and

Down to **-9007199254740991** $-(2^{53}-1)$.

Integer values outside this range lose precision.

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Numbers</h1>

<h2>Integer Precision</h2>

<p>Integers (numbers without a period or exponent notation) are accurate up to 15 digits:
</p>

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

<script>

let x = 999999999999999;

let y = 9999999999999999;
```

```
document.getElementById("demo").innerHTML = x
+ "<br>" + y;

</script>

</body>

</html>
```

How to Create a BigInt

A BigInt value, also sometimes just called a BigInt, is a bigint primitive, created by appending n to the end of an integer literal, or by calling the BigInt() function (without the new operator) and giving it an integer value or string value . To create a BigInt, append n to the end of an integer or call BigInt():

```
<!DOCTYPE html>

<html>
```

```
<body>

<h1>JavaScript Numbers</h1>

<h2>Integer and BigInt</h2>

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

<script>

let x = 9999999999999999;

let y = BigInt("9999999999999999");

document.getElementById("demo").innerHTML = x
+ "<br>" + y;

</script>

</body>

</html>
```

JavaScript Booleans

Boolean variables store one of two possible values: either true or false. The term *Boolean* is named after George Boole

(1815 – 1864), who created an algebraic system of logic. Because it ’ s named after a person, you generally write it with an initial capital letter.

The Boolean type represents a logical entity and is inhabited by two values: true and false.

Boolean values are usually used for conditional operations, including ternary operators , if...else, while, etc .

Very often, in programming, you will need a data type that can only have one of two values, like

- YES / NO
- ON / OFF
- TRUE / FALSE

For this, JavaScript has a Boolean data type. It can only take the values true **or** false **.**

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Booleans</h2>

<p>Booleans can have two values: true or false:
</p>

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

<script>

let x = 5;

let y = 5;
```

```
let z = 6;

document.getElementById("demo").innerHTML =

(x == y) + "<br>" + (x == z);

</script>

</body>

</html>
```

The Boolean() Function

Boolean variables are often used for storing the results of comparisons. You can find out the Boolean value of a comparison or convert any value in JavaScript into a Boolean value by using the Boolean() function.

```
var isItGreater = Boolean (3 > 20);
```

```
alert (isItGreater); // returns false
```

```
var areTheySame = Boolean ("tiger" === "Tiger");

alert (areTheySame); // returns false
```

The result of converting a value in JavaScript into a Boolean value using the Boolean() function depends on the value:

> *In JavaScript, the following values always evaluate to a Boolean false value:*

- *NaN*
- *undefined*
- *0 (numeric value zero)*
- *-0*
- "" *(empty string)*

- *false*

Anything that is not one of the preceding values evaluates to a Boolean true.

For example:

- 74
- "Eva"
- "10"

- "NaN"

The number character "0" is not the same as the numeric value 0 (zero).

While 0 will always result in a Boolean value of false, the string "0" will always result in a Boolean true.

Boolean values are primarily used with conditional expressions.

The Following program creates a Boolean variable and then test its value using an if/then statement

---

var b = true;

```
if (b == true) {

alert ("It is true!");

} else {

alert ("It is false.");

}
```

Boolean values are written without quotes around them, like this:

var myVar = true

On the other hand, var myVar =

“

true

”

creates a string variable.

You can use the Boolean() function to find out if an expression (or a variable) is true:

```
<!DOCTYPE html>
```

```
<html>

<body>

<h1>JavaScript Booleans</h1>

<p>Display the value of Boolean(10 > 9):
</p>

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

<script>
```

```
document.getElementById("demo").innerHTML =
Boolean(10 > 9);

    </script>

    </body>

</html>
```

Or even easier:

```
<!DOCTYPE html>

<html>
```

```
<body>

<h1>JavaScript Booleans</h1>

<p>Display the value of 10 > 9:</p>

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

<script>

document.getElementById("demo").innerHTML = 10
> 9;

</script>

</body>

</html>
```

NaN data type

NaN stands for **Not a Number** . It ’ s the result that you get when you try to do math with a string, or when a calculation

fails or can ’ t be done. For example, it ’ s impossible to calculate the square root of a negative number. Trying to do so will result in NaN.

A more common occurrence that will produce NaN is an attempt to perform mathematical operations using strings that can

,

t be converted to numbers.

undefined data type

Even if you create a variable in JavaScript and don

,

t specifically give it a value, it still has a default value. This value is "undefined".

Comparisons and Conditions

The JavaScript Comparisons gives a full overview of comparison operators.

The JavaScript Conditions gives a full overview of conditional statements.

Here are some examples:

Operator	***Description***	***Example***
==	equal to	if (day == "Monday")
>	greater than	if (salary > 9000)
<	less than	if (age < 18)

The Boolean value of an expression is the basis for all

JavaScript comparisons and conditions.

JavaScript Comparison and Logical Operators

Comparison and Logical operators are used to test for true or false.

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that x = 5, the table below explains the comparison operators:

Operator	Description	Comparing	Returns

==	equal to	x == 8	false
		x == 5	true
		x == "5"b	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	true

How Can it be Used

Comparison operators can be used in conditional statements to compare values and take action depending on the result:

```
if (age < 18) text = "Too young to buy alcohol";
```

Conditional (Ternary) Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

variablename = (condition) ? value1:value2

Example:

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Comparison</h1>

<h2>The () ? : Ternary Operator</h2>

<p>Input your age and click the button:</p>

<input id="age" value="18" />
```

```
<button onclick="myFunction()">Try it</button>

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

<script>

function myFunction() {

let age = document.getElementById("age").value;

let voteable = (age < 18) ? "Too young":"Old enough";

document.getElementById("demo").innerHTML = voteable + " to vote.";

}

</script>

</body>

</html>
```

If the variable age is a value below 18, the value of the variable voteable will be "Too young", otherwise the value of voteable will be "Old enough".

Comparing Different Types

Comparing data of different types may give unexpected results. When comparing a string with a number, JavaScript will convert the string to a number when doing the comparison. An empty string converts to 0. A non-numeric string converts to NaN which is always false

---

```
<!DOCTYPE html>

<html>

<body>
```

```
<h2>JavaScript Comparisons</h2>

<p>Input your age and click the button:
</p>

<input id="age" value="18" />

<button onclick="myFunction()">Try
it</button>

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

```
<script>

function myFunction() {

  let voteable;

  let age =
Number(document.getElementById("age").value);

  if (isNaN(age)) {

    voteable = "Input is not a number";
```

```
  } else {


    voteable = (age < 18) ?
"Too young" : "Old enough";


  }


document.getElementById("demo").innerHTML =
voteable + " to vote";


}
```

```
</script>


</body>


</html>
```

JavaScript if, else, and else if

Conditional statements are used to perform different actions based on different conditions.

Conditional Statements

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

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

In JavaScript we have the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true

- Use else to specify a block of code to be executed, if the same condition is false

- Use else if to specify a new condition to test, if the first condition is false

- Use switch to specify many alternative blocks of code to be executed

The if Statement

The **if** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally. Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax

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

true
}

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

Example

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

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript if</h2>

<p>Display "Good day!" if the hour is less than 18:00:</p>

<p id="demo">Good Evening!</p>

<script>
```

```
if (new Date().getHours() < 18) {

document.getElementById("demo").innerHTML      =
"Good day!";

}

</script>

</body>

</html>
```

The else Statement

Use the else statement to specify a block of code to be executed if the condition is false .

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

condition is false
}

Example

If the hour is less than 18, create a "Good day" greeting, otherwise "Good evening":

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript if .. else</h2>

<p>A time-based greeting:</p>

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

<script>

const hour = new Date().getHours();
```

```
let greeting;

if (hour < 18) {

greeting = "Good day";

} else {

greeting = "Good evening";

}

document.getElementById("demo").innerHTML      =
greeting;

</script>

</body>

</html>
```

The else if Statement

Use the else if statement to specify a new condition if the first condition is false.

Syntax

```
if ( condition1 ) {
  //  block of code to be executed if condition1 is true
} else if ( condition2 ) {
   //   block of code to be executed if the condition1 is false and condition2 is true
} else {
   //   block of code to be executed if the condition1 is false and condition2 is false
}
```

Example

If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":

```
<!DOCTYPE html>
```

```
<html>

<body>

<h2>JavaScript if .. else</h2>

<p>A time-based greeting:</p>

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

<script>

const time = new Date().getHours();

let greeting;

if (time < 10) {

greeting = "Good morning";

} else if (time < 20) {

greeting = "Good day";

} else {

greeting = "Good evening";
```

```
}

document.getElementById("demo").innerHTML =
greeting;

</script>

</body>

</html>
```

Note that there is no elseif syntax in JavaScript. However, you can write it with a space between else and if :

```
if (x > 50) {

/* do something */
```

```
} else if (x > 5) {

/* do something */

} else {

/* do something */

}
```

JavaScript Switch Statement

The switch statement evaluates an expression , matching the expression's value against a series of case clauses, and executes statements after the first case clause with a matching value, until a break statement is encountered. The default clause of a switch statement will be jumped to if no case matches the expression's value.

The JavaScript switch statement is used in decision making. The switch statement evaluates an expression and executes the corresponding body that matches the expression's result.

- The switch statement is used to perform different actions based on different conditions. The switch statement evaluates a variable/expression inside parentheses ().

- If the result of the expression is equal to value1, its body is executed.

- If the result of the expression is equal to value2, its body is executed.

- This process goes on. If there is no matching case, the default body executes.

Syntax

switch(

expression

) {

```
  case
x
:

   // code block

   break;
  case
y
:

   // code block
```

break;

default:

//

code block

}

This is how it works:

- The switch expression is evaluated once.

- The value of the expression is compared with the values of each case.

- If there is a match, the associated block of code is executed.

- If there is no match, the default code block is executed.

Example

The getDay() method returns the weekday as a number between 0 and 6. (Sunday=0, Monday=1, Tuesday=2 ..)

This example uses the weekday number to calculate the weekday name:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript switch</h2>

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

<script>

let day;

switch (new Date().getDay()) {

case 0:

day = "Sunday";

break;

case 1:
```

```
day = "Monday";

break;

case 2:

day = "Tuesday";

break;

case 3:

day = "Wednesday";

break;

case 4:

day = "Thursday";

break;

case 5:

day = "Friday";

break;
```

```
case 6:

day = "Saturday";

}

document.getElementById("demo").innerHTML =
"Today is " + day;

</script>



</body>

</html>
```

The break Keyword

When JavaScript reaches a break keyword, it breaks out of the switch block.

This will stop the execution inside the switch block.

It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

Notes:

- The break statement is optional. If the break statement is encountered, the switch statement ends.

- If the break statement is not used, the cases after the matching case are also executed.

- The default clause is also optional.

- If you omit the break statement, the next case will be executed even if the evaluation does not match the case.

The default Keyword

The default keyword specifies the code to run if there is no case match:

Example

The

getDay()

method returns the weekday as a number between 0 and 6.

If today is neither Saturday (6) nor Sunday (0), write a default message:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript switch</h2>

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

<script>

let text;

switch (new Date().getDay()) {

case 6:

text = "Today is Saturday";
```

```
break;

case 0:

text = "Today is Sunday";

break;

default:

text = "Looking forward to the Weekend";

}

document.getElementById("demo").innerHTML =
text;

</script>

</body>

</html>
```

JavaScript Arrays

An array consists of array elements.

Array elements are made up of the

array name and then an index number that is contained in square brackets. The individual value within an array is called an *array element.* Arrays use numbers (called the *index numbers*) to access those elements. The following example illustrates how arrays use index numbers to access elements: JavaScript arrays are written with square brackets.

myArray[0] = "yellow balloon";

myArray[1] = "red balloon";

	myArray[2] = "blue balloon"; *myArray[3] = "pink balloon";*
	In this example, the element with the index number of 0 has a value of "yellow balloon". The element with an index number 3 has a value of "pink balloon". Just as with any variable, you can give an array any name that complies with the rules of naming JavaScript variables. By assigning index numbers in arrays, JavaScript gives you the ability to make a single variable name hold a nearly unlimited list of values.

Array items are separated by commas.

The following code declares (creates) an array called cars , containing three items (car names):

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Arrays</h2>

<p>Array indexes are zero-based, which means the first item is [0].</p>

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

<script>

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

document.getElementById("demo").innerHTML = cars[0];

</script>

</body>

</html>
```

Just so you don ’ t get too carried away, there actually is a limit to the number of elements that you can have in an array, although you ’ re very unlikely to ever reach it. The limit is 4,294,967,295 elements.

In addition to naming requirements (which are the same for any type of variable, as described in chapter 3), arrays have a couple of other rules and special properties that you need to be familiar with:

- Arrays are zero-indexed

- Arrays can store any type of data

JavaScript Array

JavaScript array

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

There are 3 ways to construct array in JavaScript

1. By array literal

2. By creating instance of Array directly (using new keyword)

3. By using an Array constructor (using new keyword)

1) JavaScript 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).

```
<script>

var emp=["Sonoo","Vimal","Ratan"];

for (i=0;i <emp.length ;i++){

document.write(emp[i] + " <br/> ");

}
```

</script>

The .length property returns the length of an array.

2) JavaScript Array directly (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.*

Let's see the example of creating array directly.

```
<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>
```

3) JavaScript array constructor (new keyword)

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

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

```
<script>

var emp=new Array("Jai","Vijay","Smith");

for (i=0;i <emp.length ;i++){
```

```
document.write(emp[i] + " <br> ");

}

</script>
```

Output of the above example

Jai
Vijay
Smith

JavaScript Array Methods

Let's see the list of JavaScript array methods with their description.

Methods	***Description***
concat()	It returns a new array object that contains two or more merged

	arrays.
copywithin()	It copies the part of the given array with its own elements and returns the modified array.
entries()	It creates an iterator object and a loop that iterates over each key/value pair.
every()	It determines whether all the elements of an array are satisfying the provided function conditions.
flat()	It creates a new array carrying sub-array elements concatenated recursively till the specified depth.
flatMap()	It maps all array elements via mapping function, then flattens the result into a new array.
fill()	It fills elements into an array with static values.
from()	It creates a new array carrying the exact copy of another array element.
filter()	It returns the new array containing the elements that pass the provided function conditions.
find()	It returns the value of the first element in the given array that satisfies the specified condition.

findIndex()	It returns the index value of the first element in the given array that satisfies the specified condition.
forEach()	It invokes the provided function once for each element of an array.
includes()	It checks whether the given array contains the specified element.
indexOf()	It searches the specified element in the given array and returns the index of the first match.
isArray()	It tests if the passed value ia an array.
join()	It joins the elements of an array as a string.
keys()	It creates an iterator object that contains only the keys of the array, then loops through these keys.
lastIndexOf()	It searches the specified element in the given array and returns the index of the last match.
map()	It calls the specified function for every array element and returns the new array
of()	It creates a new array from a variable number of arguments, holding any type of argument.

pop()	It removes and returns the last element of an array.
push()	It adds one or more elements to the end of an array.
reverse()	It reverses the elements of given array.
reduce(function, initial)	It executes a provided function for each value from left to right and reduces the array to a single value.
reduceRight()	It executes a provided function for each value from right to left and reduces the array to a single value.
some()	It determines if any element of the array passes the test of the implemented function.
shift()	It removes and returns the first element of an array.
slice()	It returns a new array containing the copy of the part of the given array.
sort()	It returns the element of the given array in a sorted order.
splice()	It add/remove elements to/from the given array.
toLocaleString()	It returns a string containing all the elements of a specified array.
toString()	It converts the elements of a

	specified array into string form, without affecting the original array.
unshift()	It adds one or more elements in the beginning of the given array.
values()	It creates a new iterator object carrying values for each index in the array.

JavaScript Objects

In computer science, an object is a value in memory which is possibly referenced by an identifier . In JavaScript, objects are the only mutable values. Functions are, in fact, also objects with the additional capability of being *callable* . JavaScript objects are written with curly braces {}. Object properties are written as name:value pairs, separated by commas.

An object is a complex data type that allows us to store collections of data. For example,

```
Const student ={

firstName:’ram’,

lastName:null,

class:10

};
```

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Objects</h2>

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

```
<script>

const person = {

firstName : "John",

lastName : "Doe",

age : 50,

eyeColor : "blue"

};

document.getElementById("demo").innerHTML =

person.firstName + " is " + person.age + " years old.";

</script>

</body>

</html>
```

The type of Operator

You can use the JavaScript typeof operator to find the type of a JavaScript variable.

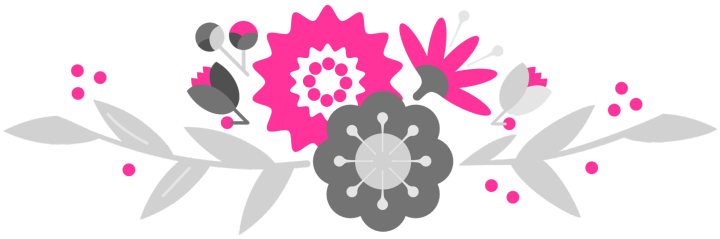
The typeof operator returns the type of a variable or an expression:

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Operators</h1>

<h2>The typeof Operator</h2>

<p>The typeof operator returns the type of a variable or an expression.</p>

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

<script>

document.getElementById("demo").innerHTML =
```

```
typeof "" + "<br>" +

typeof 50 + "<br>" +

typeof "John Doe";

</script>

</body>

</html>
```

Chapter Seven

JavaScript Functions



Functions are one of the fundamental building blocks in JavaScript. A function in JavaScript is similar to a procedure a set of statements that performs a task or calculates a value, but for a procedure to qualify as a function, it should take some input and return an output where there is some obvious relationship between the input and the output. To use a function, you must define it somewhere in the scope from which you wish to call it.

Quite often we need to perform a similar action in many places of the script.

For example, we need to show a nice-looking message when a visitor logs in, logs out and maybe somewhere else.

Functions are the main “ building blocks ” of the program. They allow the code to be called many times without repetition.

Function Syntax

A function is a block of code that performs a specific task. JavaScript functions are basically used to encapsulate logic, making that code more reusable and easier to understand.

The syntax for creating a function in JavaScript is quite simple. Functions can take input in the form of parameters and can return a value or output.

Functions help you organize and structure your code. They also allow for code reuse and make it easier to understand and maintain large codebases.

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(***parameter1, parameter2, ...***)

The code to be executed, by the function, is placed inside curly brackets: {}

---

function

name

(

parameter1, parameter2, parameter3

) {

//

code to be executed

}

Function parameters are listed inside the parentheses () in the function definition.

Function arguments are the values received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

Function declarations

A function definition (also called a function declaration, or function statement) consists of the function keyword, followed by:

- The name of the function.

- A list of parameters to the function, enclosed in parentheses and separated by commas.

- The JavaScript statements that define the function, enclosed in curly brackets, { /* ... */ }.

```
function showMessage() {




  alert( 'Hello everyone!'
);




}
```

The function keyword goes first, then goes the *name of the function* , then a list of *parameters* between the parentheses (comma-separated, empty in the example above, we ’ ll see examples later) and finally the code of the function, also named “ the function body ” , between curly braces.

```
<!DOCTYPE html>

<html>
```

```
<body>

<h1>JavaScript Functions</h1>

<p>Call a function which performs a calculation and returns the result:</p>

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

<script>

function myFunction(p1, p2) {

return p1 * p2;

}

let result = myFunction(4, 3);

document.getElementById("demo").innerHTML = result;

</script>

</body>

</html>
```

Function Invocation

The JavaScript Function Invocation is used to execute the function code and it is common to use the term “ call a function ” instead of “ invoke a function ” . The code inside a function is executed when the function is invoked.

The code inside the function will execute when "something" invokes (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self-invoked)

The function invocation is used to execute the code inside the curly braces in the function definition by adding () after function name after it has been defined to invoke that particular function.

Invoking a JavaScript Function

The code inside a function is not executed when the function is defined. The code inside a function is executed when the function is invoked. It is common to use the term "call a function" instead of "invoke a function". It is also common to say "call upon a function", "start a function", or "execute a function".

The example below will use invoke, because a JavaScript function can be invoked without being called.

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Functions</h2>
```

```
<p>The global function (myFunction) returns the product of the arguments (a ,b):</p>

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

<script>

function myFunction(a, b) {

return a * b;

}

document.getElementById("demo").innerHTML = myFunction(10, 2);

</script>

</body>

</html>
```

Note

This is a common way to invoke a JavaScript function, but not a very good practice. Global variables, methods, or functions can easily create name conflicts and bugs in the global object.

myFunction() and window.myFunction() is the same function:

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Functions</h2>

* Global functions automatically become window methods. Invoking myFunction() is the same as invoking window.myFunction(). *

* *

<script>

function myFunction(a, b) {

*return a * b;*

```
}

document.getElementById("demo").innerHTML =
window.myFunction(10, 2);

</script>

</body>

</html>
```

The function above does not belong to any object. But in JavaScript there is always a default global object.

In HTML the default global object is the HTML page itself, so the function above "belongs" to the HTML page.

In a browser the page object is the browser window. The function above automatically becomes a window function.

The Term

“

This

”

in Javascript

In JavaScript, the this keyword refers to an object. Which object depends on how this is being invoked (used or called).

The this keyword refers to different objects depending on how it is used:

In an object method, this refers to the object.
Alone, this refers to the global object.
In a function, this refers to the global object.
In a function, in strict mode, this is undefined.
In an event, this refers to the element that received the event.
Methods like call(), apply(), and bind() can refer this to any object.

Note

This

is not a variable.
It is a keyword.
You cannot change the value of

this.

The Global Object

When a function is called without an owner object, the value of this becomes the global object.

In a web browser the global object is the browser window.

This example returns the window object as the value of this:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Functions</h2>

<p>In HTML the value of <b>this</b>, in a global function, is the window object.</p>

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

<script>

let x = myFunction();

function myFunction() {

return this;

}
```

```
document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>
```

Invoking a Function as a Method

In JavaScript you can define functions as object methods.

The following example creates an object (myObject), with two properties (firstName and lastName), and a method (fullName):

```
<!DOCTYPE html>

<html>

<body>
```

```
<h2>JavaScript Functions</h2>

<p>myObject.fullName() will return John Doe:</p>

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

<script>

const myObject = {

firstName:"John",

lastName: "Doe",

fullName: function() {

return this.firstName + " " + this.lastName;

}

}

document.getElementById("demo").innerHTML =
myObject.fullName();

</script>

</body>
```

</html>

The fullName method is a function. The function belongs to the object. myObject is the owner of the function. The thing called this, is the object that "owns" the JavaScript code. In this case the value of this is myObject.

Test it! Change the fullName method to return the value of this:

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Functions</h2>

* The value of this, in an object method, is the owner object. *

* *

<script>

```
const myObject = {

firstName:"John",

lastName: "Doe",

fullName: function() {

return this;

}

}

document.getElementById("demo").innerHTML =
myObject.fullName();

</script>

</body>

</html>
```

Invoking a Function with a Function Constructor

If a function invocation is preceded with the new keyword, it is a constructor invocation.

It looks like you create a new function, but since JavaScript functions are objects you actually create a new object:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Functions</h2>

<p>In this example, myFunction is a function constructor:</p>

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

<script>

function myFunction(arg1, arg2) {

this.firstName = arg1;
```

```
this.lastName = arg2;

}

const myObj = new myFunction("John","Doe")

document.getElementById("demo").innerHTML =
myObj.firstName;

</script>

</body>

</html>
```

Function Return

The **return** statement ends function execution and specifies a value to be returned to the function caller.

When JavaScript reaches a return statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a return value. The return value is "returned" back to the "caller":

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Functions</h1>

<p>Call a function which performs a calculation and returns the result:</p>

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

<script>

let x = myFunction(4, 3);

document.getElementById("demo").innerHTML = x;

function myFunction(a, b) {

return a * b;
```

```
}

</script>


</body>

</html>
```

The ( ) Operator

The () operator invokes (calls) the function:

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Functions</h1>
```

* Invoke (call) a function that converts from Fahrenheit to Celsius: *

* *

<script>

function toCelsius(f) {

*return (5/9) * (f-32);*

}

let value = toCelsius(77);

document.getElementById("demo").innerHTML = value;

</script>

</body>

</html>

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

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Functions</h1>

<p>Invoke (call) a function to convert from Fahrenheit to Celsius:</p>

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

<script>

function toCelsius(f) {

return (5/9) * (f-32);

}

let value = toCelsius();

document.getElementById("demo").innerHTML = value;

</script>
```

</body>

</html>

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

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Functions</h1>

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

* *

<script>

function toCelsius(f) {

*return (5/9) * (f-32);*

}

```
let value = toCelsius;

document.getElementById("demo").innerHTML =
value;

</script>

</body>

</html>
```

Functions Used as Variable Values

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

Example

Instead of using a variable to store the return value of

a function:

```
let

x = toCelsius(77);



let

text =

"The temperature is "

+ x +

" Celsius"

;
```

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Functions</h1>

<p>Using a function as a variable:</p>

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

<script>

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

document.getElementById("demo").innerHTML      =
text;

function toCelsius(fahrenheit) {

return (5/9) * (fahrenheit-32);

}

</script>
```

</body>

</html>

Local Variables

Variables declared within a JavaScript function, become LOCAL to the function. Local variables can only be accessed from within the function.

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Functions</h1>

<p>Outside myFunction() carName is undefined.
</p>

<p id="demo1"></p>

<p id="demo2"></p>

<script>

let text = "Outside: " + typeof carName;
```

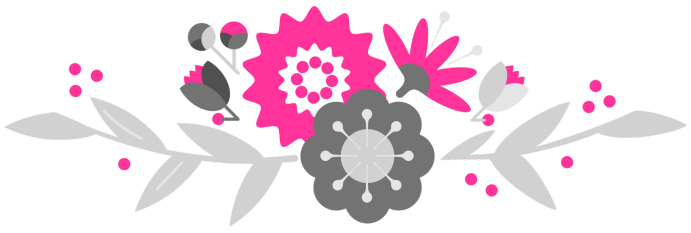
```
document.getElementById("demo1").innerHTML = text;

function myFunction() {

let carName = "Volvo";

let text = "Inside: " + typeof carName + " " + carName;

document.getElementById("demo2").innerHTML = text;

}

myFunction();

</script>

</body>

</html>
```

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

Chapter Eight

JavaScript Objects



JavaScript is designed on a simple object-based paradigm. An object is a collection of properties , and a property is an association between a name (or *key*) and a value. A property's value can be a function, in which case the property is known as a method .

Objects in JavaScript, just as in many other programming languages, can be compared to objects in real life. In JavaScript, an object is a standalone entity, with properties and type. Compare it with a cup, for example. A cup is an object, with properties. A cup has a color, a design, weight, a material it is made of, etc. The same way, JavaScript objects can have properties, which define their characteristics.

In addition to objects that are predefined in the browser, you can define your own objects. This chapter describes how to use objects, properties, and methods, and how to create your own objects.

JavaScript object is a non-primitive data-type

that allows you to store multiple collections of data.

There are eight different forms of data in JavaScript, as we learned in the chapter on data types. Because seven of them only have one value — a string, a number, or anything else — they are referred to as "primitive."

On the other hand, more sophisticated entities and keyed collections of diverse data are stored as objects. Objects are present in nearly every area of JavaScript. We thus need to comprehend them before delving further into anything else.

An object can be created with figure brackets { … } with an optional list of *properties* . A property is a “ key: value ” pair, where key is a string (also called a “ property name ”), and value can be anything.

We can imagine an object as a cabinet with signed files. Every piece of data is stored in its file by the key. It ’ s easy to find a file by its name or add/remove a file

Real Life Objects, Properties, and Methods

An automobile is a thing in reality. A automobile includes features like color and weight, as well as functions like start and stop: Although the attributes of all automobiles are the same, each car has a different value for each property. The procedures are the same in every automobile, although they are carried out at various periods.

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Objects</h2>

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

<script>

// Create an object:

const car = {type:"Toyota",
model:"Camry", color:"white"};

// Display some data from the
object:
```

document.getElementById("demo").innerHTML = "The car type is " + car.type;

</script>

</body>

</html>

This code assigns many values (Toyota, Camry, white) to a variable named car.

The values are written as name:value pairs (name and value separated by a colon).

Object Definition

You define (and create) a JavaScript object with an object literal:

<!DOCTYPE html>

<html>

```
<body>

<h2>JavaScript Objects</h2>

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

<script>

// Create an object:

const person = {
```

```
  firstName: "John",

  lastName: "Doe",

  age: 50,

  eyeColor: "blue"

};

// Display some data from the
object:
```

```
document.getElementById("demo").innerHTML =

        person.firstName + " is " + person.age + " years old.";

    </script>

    </body>

</html>
```

Object Properties

The name:values pairs in JavaScript objects are called properties:

Property	***Property Value***
firstName	John
lastName	Doe
age	50
eyeColor	blue

Accessing Object Properties

You can access object properties in two ways:

objectName.propertyName

or

objectName["propertyName"]

Example:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Objects</h2>

<p>There are two different ways to access an object property.</p>

<p>You can use person.property or person["property"].</p>

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

<script>

// Create an object:
```

```
const person = {

firstName: "John",

lastName : "Doe",

id : 5566

};

// Display some data from the object:

document.getElementById("demo").innerHTML =

person.firstName + " " + person.lastName;

</script>

</body>

</html>
```

Example 2:

```
<!DOCTYPE html>
```

```
<html>

<body>

<h2>JavaScript Objects</h2>

<p>There are two different ways to access an object property.</p>

<p>You can use person.property or person["property"].</p>

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

<script>

// Create an object:

const person = {

firstName: "John",

lastName : "Doe",

id : 5566

};
```

```
// Display some data from the object:

document.getElementById("demo").innerHTML =

person["firstName"] + " " + person["lastName"];

</script>

</body>

</html>
```

Object Methods

Collections of key/value pairs make up objects in JavaScript. In addition to all standard JavaScript data types, including strings, integers, and Booleans, the values can also have methods and properties. The property turns into a method when the value is a function. To explain the activities of the objects, you often use methods.

In JavaScript, every object is descended from the parent constructor called Object. Working with individual objects is made simple by the several helpful built-in methods that Object provides. While Object methods are used directly on the Object constructor and take the object instance as an argument, Array prototype methods like as sort() and reverse() are used on the array instance. We call this a static method. Moreover, objects can have methods. Actions that may be carried out on objects are called methods. Properties hold methods as function definitions.

Property	***Property Value***
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

Example

```
const
person = {
firstName:
"John"
,
lastName :
"Doe"
,
id       : 5566,
fullName :
function
```

() {

return

this

.firstName +

" "

+

this

.lastName;

}

```
};
```

In the example above, this refers to the person object.

I.E. this.firstName means the firstName property of this.

I.E. this.firstName means the firstName property of person.

Understanding the Term : “ THIS ”

In JavaScript, this keyword refers to an **object** . Which object depends on how this is being invoked (used or called).

The this keyword refers to different objects depending on how it is used:

In an object method, this refers to the object.
Alone, this refers to the global object.
In a function, this refers to the global object.
In a function, in strict mode, this is undefined.

In an event, this refers to the element that received the event.
Methods like call(), apply(), and bind() can refer this to any object.

The this Keyword

In a function definition, this refers to the "owner" of the function.

In the example above, this is the person object that "owns" the fullName function.

In other words, this.firstName means the firstName property of this object.

Accessing Object Methods

You access an object method with the following syntax:

Using:

objectName.methodName()

Example:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Objects</h2>

<p>An object method is a function definition, stored as a property value.</p>

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

<script>

// Create an object:

const person = {

firstName: "John",

lastName: "Doe",
```

```
id: 5566,

fullName: function() {

return this.firstName + " " + this.lastName;

}

};

// Display data from the object:

document.getElementById("demo").innerHTML =
person.fullName();

</script>

</body>

</html>
```

If you access a method without the () parentheses, it will return the function definition:

```
<!DOCTYPE html>

<html>
```

```
<body>

<h2>JavaScript Objects</h2>

<p>If you access an object method without (), it will return the function definition:</p>

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

<script>

// Create an object:

const person = {

firstName: "John",

lastName : "Doe",

id : 5566,

fullName : function() {

return this.firstName + " " + this.lastName;

}
```

```
};

// Display data from the object:

document.getElementById("demo").innerHTML =
person.fullName;

</script>

</body>

</html>
```

Do Not Declare Strings, Numbers, and Booleans as Objects!

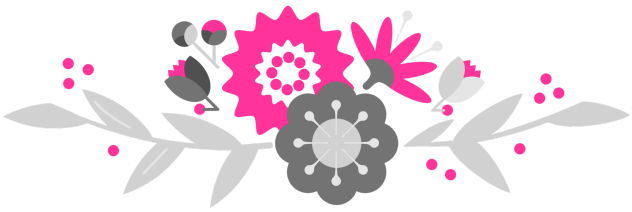
When a JavaScript variable is declared with the keyword " new ", the variable is created as an object:

```
x = new String();        // Declares x as a String object
y = new Number();        // Declares y as a Number object
z = new Boolean();       // Declares z as a Boolean object
```

Avoid String , Number , and Boolean objects. They complicate your code and slow down execution speed.

Chapter Nine

JavaScript Events



Events are things that happen in the system you are programming — the system produces (or "fires") a signal of some kind when an event occurs, and provides a mechanism by which an action can be automatically taken (that is, some code running) when the event occurs. Events are fired inside the browser window, and tend to be attached to a specific item that resides in it. This might be a single element, a set of elements, the HTML document loaded in the current tab, or the entire browser window. There are many different types of events that can occur.

For example, if the user clicks a button on a webpage, you might want to react to that action by displaying an information box. In this article, we discuss some important concepts surrounding events, and look at how they work in browsers. This won't be an exhaustive study; just what you need to know at this stage.

For example:

- The user selects, clicks, or hovers the cursor over a certain element.

- The user chooses a key on the keyboard.

- The user resizes or closes the browser window.

- A web page finishes loading.

- A form is submitted.

- A video is played, paused, or ends.

- An error occurs.

You can gather from this (and from glancing at the MDN event reference) that there are a lot of events that can be fired.

To react to an event, you attach an event handler to it. This is a block of code (usually a JavaScript function that you as a programmer create) that runs when the event fires. When such a block of code is defined to run in response to an event, we say we are registering an event handler. Note: Event handlers are sometimes called event listeners — they are pretty much interchangeable for our purposes, although strictly speaking, they work together. The listener listens out for the event happening, and the handler is the code that is run in response to it happening.

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

HTML Events

The change in the state of an object is known as an Event. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML , js react over these events and allow the execution. This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers.

For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

Mouse events:

Event Performed	***Event Handler***	***Description***
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Example:

```
<html>
```

```
<head> JavaScript Events </head>

<body>

<script language="Javascript" type="text/Javascript">

<!--

function clickevent()

{

document.write("Tutorial on Javascripting Events");

}

//-->

</script>

<form>

<input  type="button"  onclick="clickevent()"  value="Hello
How are You?"/>

</form>

</body>

</html>
```

---

onsubmit Event Type

onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

Example

The following example shows how to use onsubmit. Here we are calling a validate() function before submitting a form data to the webserver. If validate() function returns true, the form will be submitted, otherwise it will not submit the data.

```
<html>

<head>

<script type = "text/javascript">

<!--

function validation() {

all validation goes here
```

```
.........

return either true or false

}

//-->

</script>

</head>

<body>

<form method = "POST" action = "t.cgi" onsubmit =
"return validate()">

.......

<input type = "submit" value = "Submit" />

</form>

</body>

</html>
```

onmouseover and onmouseout

These two event types will help you create nice effects with images or even with text as well. The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element. Try the following example.

```
<html>

<head>

<script type = "text/javascript">

<!--

function over() {

document.write ("Mouse Over");

}

function out() {

document.write ("Mouse Out");

}

//-->

</script>
```

```
</head>

<body>

<p>Bring your mouse inside the division to see the result:
</p>

<div onmouseover = "over()" onmouseout = "out()">

<h2> This is inside the division </h2>

</div>

</body>

</html>
```

Keyboard events:

Event Performed	***Event Handler***	***Description***
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form events:

Event Performed	***Event Handler***	***Description***
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Window/Document events

Event Performed	***Event Handler***	***Description***
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading

- An HTML input field was changed

- An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

With single quotes:

<element event='some JavaScript'>

With double quotes:

<element event="some JavaScript">

In the following example, an onclick attribute (with code), is added to a <button> element:

```
<!DOCTYPE html>

<html>

<body>

<button
onclick="document.getElementById('demo').innerHTML=Date()">T
he time is?</button>

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

</body>

</html>
```

In the example above, the JavaScript code changes the content of the element with id="demo".

In the next example, the code changes the content of its own element (using this.innerHTML):

Example1:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript HTML Events</h2>

<button onclick="this.innerHTML=Date()">The time is?
</button>

</body>

</html>
```

Example 2:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript HTML Events</h2>
```

```
<p>Click the button to display the date.</p>

<button onclick="displayDate()">The time is?</button>

<script>

function displayDate() {

document.getElementById("demo").innerHTML = Date();

}

</script>

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

</body>

</html>
```

HTML DOM Events

HTML or DOM events are widely used in JavaScript code. JavaScript code is executed with HTML/DOM events. So before learning JavaScript, let ’ s have some idea about events.

DOM Events allow JavaScript to add event listener or event handlers to HTML elements.

Examples

In HTML onclick is the event listener, myFunction is the event handler:

<button onclick="myFunction()">Click me</button>

In JavaScript click is the event, myFunction is the event handler:

button.addEventListener("click" , myFunction);

You might be wondering when to use touch events versus mouse events, since they're so similar.

Touch events are only triggered on touch-enabled devices like smartphones and touch-screen laptops. Mouse events like click and mousemove are triggered on the majority of browsers and devices. However, in most smartphones, the mouseover event isn't triggered at all, because they can't detect a finger hovering over the phone. Some smartphones are adding sensors for that though, so more smartphones will detect mouseover in the future.

In most cases, you'll want to listen to mouse events instead of touch events, because those are the most universal.

Event	***Occurs When***	***Belongs To***
abort	The loading of a media is aborted	UiEvent, Event
afterprint	A page has started printing	Event
animationend	A CSS animation has completed	AnimationEvent
animationiteration	A CSS animation is repeated	AnimationEvent
animationstart	A CSS animation has started	AnimationEvent

beforeprint	A page is about to be printed	Event
beforeunload	Before a document is about to be unloaded	UiEvent, Event
blur	An element loses focus	FocusEvent
canplay	The browser can start playing a media (has buffered enough to begin)	Event
canplaythrough	The browser can play through a media without stopping for buffering	Event
change	The content of a form element has changed	Event
click	An element is clicked on	MouseEvent
contextmenu	An element is right-clicked to open a context menu	MouseEvent

copy	The content of an element is copied	ClipboardEvent
cut	The content of an element is cutted	ClipboardEvent
dblclick	An element is double-clicked	MouseEvent
drag	An element is being dragged	DragEvent
dragend	Dragging of an element has ended	DragEvent
dragenter	A dragged element enters the drop target	DragEvent
dragleave	A dragged element leaves the drop target	DragEvent
dragover	A dragged element is over the drop target	DragEvent
dragstart	Dragging of an element has started	DragEvent
drop	A dragged element is	DragEvent

	dropped on the target	
durationchange	The duration of a media is changed	Event
ended	A media has reach the end ("thanks for listening")	Event
error	An error has occurred while loading a file	ProgressEvent, UiEvent, Event
focus	An element gets focus	FocusEvent
focusin	An element is about to get focus	FocusEvent
focusout	An element is about to lose focus	FocusEvent
fullscreenchange	An element is displayed in fullscreen mode	Event
fullscreenerror	An element can not be displayed in fullscreen mode	Event
hashchange	There has	HashChangeEvent

	been changes to the anchor part of a URL	
input	An element gets user input	InputEvent, Event
invalid	An element is invalid	Event
keydown	A key is down	KeyboardEvent
keypress	A key is pressed	KeyboardEvent
keyup	A key is released	KeyboardEvent
load	An object has loaded	UiEvent, Event
loadeddata	Media data is loaded	Event
loadedmetadata	Meta data (like dimensions and duration) are loaded	Event
loadstart	The browser starts looking for the specified media	ProgressEvent
message	A message is received through the event source	Event

mousedown	The mouse button is pressed over an element	MouseEvent
mouseenter	The pointer is moved onto an element	MouseEvent
mouseleave	The pointer is moved out of an element	MouseEvent
mousemove	The pointer is moved over an element	MouseEvent
mouseover	The pointer is moved onto an element	MouseEvent
mouseout	The pointer is moved out of an element	MouseEvent
mouseup	A user releases a mouse button over an element	MouseEvent
mousewheel	Deprecated. Use the wheel event instead	WheelEvent
offline	The browser starts working offline	Event

online	The browser starts working online	Event
open	A connection with the event source is opened	Event
pagehide	User navigates away from a webpage	PageTransitionEvent
pageshow	User navigates to a webpage	PageTransitionEvent
paste	Some content is pasted in an element	ClipboardEvent
pause	A media is paused	Event
play	The media has started or is no longer paused	Event
playing	The media is playing after beeing paused or buffered	Event
popstate	The window's history changes	PopStateEvent
progress	The browser is downloading	Event

	media data	
ratechange	The playing speed of a media is changed	Event
resize	The document view is resized	UiEvent, Event
reset	A form is reset	Event
scroll	An scrollbar is being scrolled	UiEvent, Event
search	Something is written in a search field	Event
seeked	Skipping to a media position is finished	Event
seeking	Skipping to a media position is started	Event
select	User selects some text	UiEvent, Event
show	A <menu> element is shown as a context menu	Event
stalled	The browser is trying to get unavailable media data	Event

storage	A Web Storage area is updated	StorageEvent
submit	A form is submitted	Event
suspend	The browser is intentionally not getting media data	Event
timeupdate	The playing position has changed (the user moves to a different point in the media)	Event
toggle	The user opens or closes the <details> element	Event
touchcancel	The touch is interrupted	TouchEvent
touchend	A finger is removed from a touch screen	TouchEvent
touchmove	A finger is dragged across the screen	TouchEvent
touchstart	A finger is	TouchEvent

	placed on a touch screen	
transitionend	A CSS transition has completed	TransitionEvent
unload	A page has unloaded	UiEvent, Event
volumechange	The volume of a media is changed (includes muting)	Event
waiting	A media is paused but is expected to resume (e.g. buffering)	Event
wheel	The mouse wheel rolls up or down over an element	WheelEvent

JavaScript Event Handlers

Event handlers can be used to handle and verify user input, user actions, and browser actions:

- Things that should be done every time a page loads

- Things that should be done when the page is closed

- Action that should be performed when a user clicks a button

- Content that should be verified when a user inputs data

- And more ...

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can execute JavaScript code directly

- HTML event attributes can call JavaScript functions

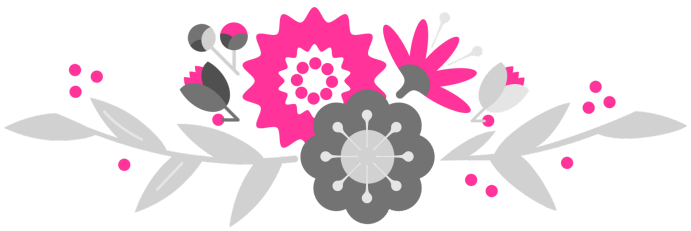
- You can assign your own event handler functions to HTML elements

- You can prevent events from being sent or being handled

- And more ...

Chapter Ten

JavaScript Loop



JavaScript Loops are powerful tools for performing repetitive tasks efficiently. Loops in JavaScript execute a block of code again and again while the condition is true.

For example, suppose we want to print “Hello World” 5 times. This can be done using JS Loop easily. In Loop, the statement needs to be written only once and the loop will be executed 5 times as shown below:

```
FOR (LET I = 0; I < 5; I++) {

CONSOLE.LOG("HELLO WORLD!");
```

```
}
```

Output

```
Hello world!

Hello world!

Hello world!
```

```
Hello world!

Hello world!
```

Looping in programming languages is a feature that facilitates the execution of code blocks repeatedly until some condition becomes false.

For example, if you want to show a message 100 times, then you can use a loop.

It's just a simple example; you can achieve much more with loops.

The JavaScript loops are used to iterate the piece of code using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. for loop

2. while loop

3. do-while loop

4. for-in loop

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

Often this is the case when working with arrays:

Instead of writing:

```
text += cars[0] + "<br>";

text += cars[1] + "<br>";

text += cars[2] + "<br>";

text += cars[3] + "<br>";

text += cars[4] + "<br>";

text += cars[5] + "<br>";
```

You can write:

```
<!DOCTYPE html>
```

```
<html>

<body>

<h2>JavaScript For Loop</h2>

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

<script>

const cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat",
"Audi"];

let text = "";

for (let i = 0; i < cars.length; i++) {

text += cars[i] + "<br>";

}

document.getElementById("demo").innerHTML      =
text;

</script>

</body>
```

</html>

The For Loop

The for statement creates a loop with 3 optional expressions:

When a for loop executes, the following occurs:

1. The initializing expression initialization, if any, is executed. This expression usually initializes one or more loop counters, but the syntax allows an expression of any degree of complexity. This expression can also declare variables.

2.

The

condition

expression is evaluated.
If the value of

condition

is true, the loop statements execute.
Otherwise, the

for

loop terminates.
(If the

condition

expression is omitted entirely, the condition is assumed to be true.)

3.

The

statement

executes.
To execute multiple statements, use a

block statement

({ }) to group those statements.

4.

If present, the update expression

afterthought

is executed.

5.

Control returns to Step 2

---

for (

expression 1

;

expression 2

;

expression 3

) {

//

code block to be executed

}

Expression 1 is executed (one time) before the execution of the code block.

Expression 2 defines the condition for executing the code block.

Expression 3 is executed (every time) after the code block has been executed.

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript For Loop</h2>

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

<script>

let text = "";

for (let i = 0; i < 5; i++) {

text += "The number is " + i + "<br>";

}
```

```
document.getElementById("demo").innerHTML =
text;

</script>

</body>

</html>
```

From the example above, you can read:

Expression 1 sets a variable before the loop starts (let i = 0).

Expression 2 defines the condition for the loop to run (i must be less than 5).

Expression 3 increases a value (i++) each time the code block in the loop has been executed.

- The initialExpression initializes and/or declares variables and executes only once.

- The condition is evaluated.

- If the condition is false, the for loop is terminated.

- If the condition is true, the block of code inside of

the for loop is executed.

- The updateExpression updates the value of initialExpression when the condition is true.

- The condition is evaluated again.
  This process continues until the condition is false.

The

for loop

runs until the given condition becomes false

.
It is similar to the for loops in C++ and Java.

JavaScript for loop is used to iterate the elements/code block a fixed number of times. It is used if the number of the iteration is known.

for statement creates the loop that accepts three optional expressions and a code block that will be executed in a loop. The syntax of for statement is given below.

```
for (statement 1 ; statement 2 ;
statement 3){

    code here...
```

}

- **Statement 1:**

  It

  is the initialization of the counter.
  It is executed once before the execution of the code block.

- **Statement 2:**

  It defines the testing condition for executing the code block

- **Statement 3:**

  It is the increment or decrement of the counter & executed (every time) after the code block has been executed.

  Example:

```
// JavaScript program to illustrate for loop
```

```
let x;

// for loop begins when x=2

// and runs till x <=4

for (x = 2; x <= 4; x++) {
```

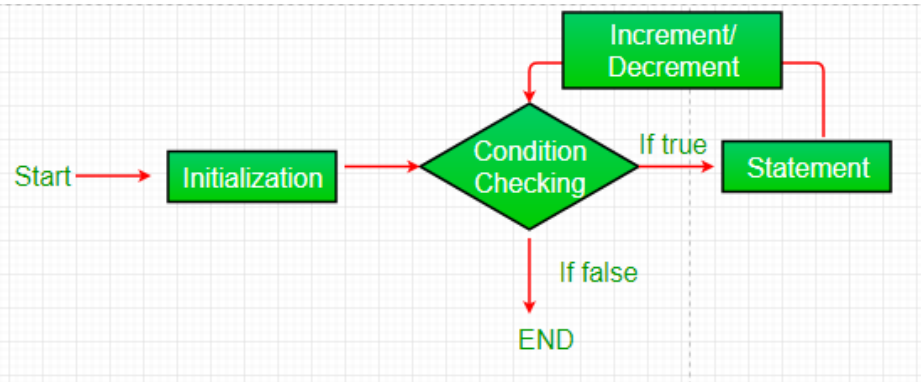
```
    console.log("Value of x:" + x);


}
```

Output:

```
Value of x:2

Value of x:3

Value of x:4
```

Flow chart

This flowchart below shows the working of the for loop in JavaScript.
You can see the control flow in the For loop.


Increment/
Decrement
Start
Initialization
Condition
Checking
If true
Statement
If false
END

Example:

HTML

In the example below, the function contains a for statement that counts the number of selected options in a scrolling list (a <select> element that allows multiple selections).

<form name="selectForm">

<label for="musicTypes"

>Choose some music types, then click the button below:</label

>

<select id="musicTypes" name="musicTypes" multiple>

<option selected>R&B</option>

<option>Jazz</option>

<option>Blues</option>

<option>New Age</option>

<option>Classical</option>

<option>Opera</option>

</select>

<button id="btn" type="button">How many are selected?</button>

</form>

JavaScript

Here, the for statement declares the variable i and initializes it to 0. It checks that i is less than the number of options in the <select> element, performs the succeeding if statement, and increments i by 1 after each pass through the loop.

```
function countSelected(selectObject) {

let numberSelected = 0;

for (let i = 0; i < selectObject.options.length; i++) {

if (selectObject.options[i].selected) {

numberSelected++;

}

}

return numberSelected;

}



const btn = document.getElementById("btn");



btn.addEventListener("click", () => {
```

```
const musicTypes = document.selectForm.musicTypes;

console.log(`You have selected ${countSelected(musicTypes)} option(s).`);

});
```

do...while statement

The do ...
while loop works in much the same way as the while loop, except that it puts the statements before the expression to test against.

The effect is that the statements within a do ...
while loop will always execute as least once.

Example:

Listing 6-6: Using a do . . while Loop

```
<html>

<head>

<title>Let's Count</title>
```

```
</head>

<body>

<script>

var i = 0;

do {
```

```
    i++;

    document.write(I + "<br>");

    } while (i<10);

    </script>

    </body>

</html>
```

In JavaScript, a do while loop is a control statement that let the code to run repeatedly in response to a specified boolean condition. It resembles an iterative if statement. You can use the do...while loop to run a certain block of code at least once.

Two primary categories of loops exist.

- **Access regulated loops** The test condition is examined in this kind of loop prior to the loop body's entry. Entry-controlled loops are those seen in while and for loops.

- **Exit Controlled Loops:** At the conclusion of the loop body, the test condition is examined or tested in this kind of loop. Consequently, regardless of whether the

test condition is true or false, the loop body will run at least once. An exit-controlled loop is a do-while loop.

The do...while statement repeats until a specified condition evaluates to false.

A do...while statement looks as follows:

```
do {

   // Statements

}

while(conditions)
```

---

statement is always executed once before the condition is checked. (To execute multiple statements, use a block statement ({ }) to group those statements.)

If condition is true, the statement executes again. At the end of every execution, the condition is checked. When the condition is false, execution stops, and control passes to the statement following do...while.

Example:

In the following example, the

do

loop iterates at least once and reiterates until

i

is no longer less than

5

.

```
let i = 0;

do {
```

```
    i += 1;


    console.log(i);


} while (i < 5);
```

The example below will illustrate the use of a do...while loop

```
let test = 1;
```

```
do {

  console.log(test);

  test++;

} while(test<=5)
```

The main difference between do

...

while and while loop is that it is guaranteed that do

...

while loop will run at least once.
Whereas, the while loop will not run even once if the given condition is not satisfied.

The example below

will try to understand the difference between two loops

```
let test = 1;



do {



  console.log(test);



} while(test<1)
```

```
while(test<1){

  console.log(test);

}
```

Output:

1

Explanation: We can see that even if the condition is not satisfied in the do...while loop the code still runs once, but in the case of while loop, first the condition is checked before entering into the loop. Since the condition does not match therefore the while loop is not executed.

Differences between do

...

while and While Loop

	Do….While Loop	***While Loop***
	It is an exit-controlled loop	It is an entry-controlled loop.
	The number of iterations will be at least one irrespective of the condition	The number of iterations depends upon the condition specified
	The block code is controlled at the end	The block of code is controlled at

	starting

Note:

When we are writing conditions for the loop we should always add a code that terminates the code execution otherwise the loop will always be true and the browser will crash.

Supported Browser:

- Chrome

- Edge
- Safari
- Firefox
- Internet Explorer

While Statement

The while statement creates a loop that runs as long as a condition evaluates to true.

Listing 6-5 shows a webpage containing an example of the while loop.

```
<html>
```

<head>

<title>Guess the Word</title>

</head>

<body>

<script>

```
var guessedWord = prompt("What word
am I thinking

of?");

while (guessedWord != "sandwich") { //
as long as the

guessed word is not sandwich

prompt("No.
That's not it.
Try again.");
```

```
}



alert("Congratulations!
That's exactly right!"); //



do this after exiting the loop



</script>



</body>
```

</html>

A

while

statement executes its statements as long as a specified condition evaluates to

true.
A

while

statement looks as follows:

```
while (condition) {
    Code block to be executed
}
```

If the condition becomes false, statement within the loop stops executing and control passes to the statement

following the loop.

Here’s an example of a while loop that counts from 1 to 5.

```
let count = 1;

while (count <= 5) {

console.log(count);
```

```
        count++;



    }
```

Output:

```
1

2
```

3

4

5

Explanation: Here,

- Count is initialized to 1.

- The loop runs as long as count is less than or equal to 5.

- Inside the loop, console.log(count) outputs the current value of count.

- After each iteration, count is incremented by 1 (count++).

The condition test occurs *before* statement in the loop is executed. If the condition returns true, statement is executed and the condition is tested again. If the condition

returns false, execution stops, and control is passed to the statement following while.

To execute multiple statements, use a block statement ({ }) to group those statements

Example:

The following

while

loop iterates as long as

n

is less than

3

:

```
let n = 0;
```

```
let x = 0;
```

```
while (n < 3) {


  n++;


  x += n;


}
```

With each iteration, the loop increments

n

and adds that value to

x.
Therefore,

x

and

n

take on the following values:

- After the first pass:

  n

  =

  1

and

x

=

1

- After the second pass:

n

=

2

and

x

=

3

- After the third pass:

  $n = 3$ and $x = $

6

After completing the third pass, the condition

$n < 3$

is no longer

true, so the loop terminates.

Avoid infinite loops.

Make sure the condition in a loop eventually becomes false — otherwise, the loop will never terminate! The statements in the following while loop execute forever because the condition never becomes false :

```
// Infinite loops are bad!

while (true) {

console.log("Hello, world!");

}
```

Comparison between

the

while and

for

loop:

Both while and for loops are used for repetitive tasks in JavaScript, but they have different syntax and are typically used in different scenarios.

1.

Initialization

:

- **While Loop**

  : The initialization of variables happens before the loop.

- **For Loop**

  : The initialization of variables is done within the loop syntax.

2.

Condition

:

- Both loops require a condition that determines whether the loop should continue or terminate.

3.

Increment/Decrement

:

- **While Loop**

  : The increment/decrement of loop control variable(s) must be done manually within the loop block.

- **For Loop**

  : The increment/decrement of loop control variable(s) is part of the loop syntax.

4.

Use Cases

:

- **While Loop**

  : Typically used when you don’t know the number of iterations in advance or when you want to create an infinite loop.

- **For Loop**

  : Generally used when you know the number of iterations in advance or when iterating over arrays or other collections.

for…….in Loop

The for ... in statements loop through the properties in an object. You can also use a for ... in statement to loop through the values of an array.

The for ... in loop has an interesting quirk. It doesn ’ t care about the order of properties or elements that it ’ s looping through. For this reason, and because using for ... in loop is

slower, you ’ re much better off using a standard for loop to loop through array elements.

Objects are data containers that have properties (what they are) and methods (what they do). Web browsers have a set of built-in objects that programmers can use to control the function of the browser. The most basic of these is the Document object. The write method of the Document object, for example, tells your browser to insert a specified value into the HTML document.

The Document object also has properties that it uses to track and give programmers information about the current document. The Document.images collection, for example, contains all of the img tags in the current HTML document.

<html>

<head>

<title>document properties</title>

<style>

.columns {

-webkit-column-count: 6; // Chrome, Safari, Opera

-moz-column-count: 6; // Firefox

```
column-count: 6;



}



</style>



</head>



<body>
```

```
<div class="columns">



<script>



for (var prop in document){



document.write (prop + "<br>");



}
```

```
</script>

</div>

</body>

</html>
```

The for...in loop iterates over the properties of an object. For each property, the code in the code block is executed.

Syntax

```
for (let i in obj1) {


    // Prints all the keys in


    // obj1 on the console


    console.log(i);


}
```

Example: JavaScript For In Loop

The for in statement loops through the properties of an object:

```
<!DOCTYPE html>
```

```
<html>

<body>

<h2>JavaScript For In Loop</h2>

<p>The for in statement loops
through the properties of an object:</p>

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

<script>
```

```
const person = {fname:"John",
lname:"Doe", age:25};

let txt = "";

for (let x in person) {

  txt += person[x] + " ";

}
```

```
document.getElementById("demo").innerHTML = txt;
</script>
</body>
</html>
```

Example Explained

- The **for in** loop iterates over a **person** object

- Each iteration returns a **key** (x)

- The key is used to access the

  value

  of the key

- The value of the key is

  person[x]

for in Loop Important Points

- Use the for-in loop to iterate over non-array objects.
  Even though we can use a for-in loop for an array, it is generally not recommended.
  Instead, use a for loop for looping over an array.

- The properties iterated with the for-in loop also include the properties of the objects higher in the Prototype chain.

- The order in which properties are iterated may

not match the properties that are defined in the object.

for-in Loop Examples

Example:

A simple example to illustrate the for-in loop over an array.

```
const array = [1, 2, 3, 4, 5];

for (const element of array) {

console.log(element);

}
```

Output:

1

2

3

4

Example: For-in loop iterates over the properties of an object and its prototype chain's properties. If we want to display both properties of the "student1" object which belongs to that object only and the prototype chain, then we can perform it by for in loop.

```
const courses = {

// Declaring a courses object

firstCourse: "C++ STL",

secondCourse: "DSA Self Paced",

thirdCourse: "CS Core Subjects"

};

// Creating a new empty object with
```

```
// prototype set to courses object

const student1 = Object.create(courses);

// Defining student1 properties and methods

student1.id = 123;

student1.firstName = "Prakhar";

student1.showEnrolledCourses = function () {

console.log(courses);

}

// Iterating over all properties of

// student1 object

for (let prop in student1) {

console.log(prop + " -> "

+ student1[prop]);

}
```

Output:

```
        id -> 123

        firstName -> Prakhar

        showEnrolledCourses -> function ()
{
            console.log(courses);

        }
```

firstCourse -> C++ STL

secondCourse -> DSA Self Paced

thirdCourse -> CS Core Subjects

Supported Browsers:

The browser supported are listed below:

- Google Chrome
- Edge
- Firefox
- Opera

- Safari

The For Of Loop

The JavaScript for of statement loops through the values of an iterable object, such as arrays, strings, maps, sets, NodeLists, and more: . It provides a simpler syntax compared to traditional for loops.

Syntax

```
for ( variable of iterableObjectName) {
```

```
  //

  code block to be executed


}
```

Parameters

:

- **Variable** : Represents the current value of each iteration from the iterable. For every iteration the value of the next property is assigned to the variable. *Variable* can be declared with const , let , or var .

- **Iterable** : Any object that can be iterated over (e.g., arrays, strings, maps).

Examples :

Iterate over an array:

```
const arr = [ "Fred", "Tom", "Bob" ];

for (let i of arr) {
```

```
  console.log(i);



}



// Output:



// Fred



// Tom
```

```
// Bob



const array = [1, 2, 3, 4, 5];



for (const item of array) {



console.log(item);



}
```

Output

1

2

3

4

5

Explanation:

The code initializes an array with values 1 through 5.

It then iterates over each element of the array using a for...of loop, logging each element to the console.

Iterate over a Map **:**

```
const m = new Map();
```

```
m.set(1, "black");

m.set(2, "red");

for (let n of m) {

  console.log(n);
```

```
}



// Output:



// [1, black]



// [2, red]



const map = new Map([
```

```
    ['key1', 'value1'],

    ['key2', 'value2'],

    ['key3', 'value3']

]);

for (const [key, value] of map) {

    console.log(`Key: ${key}, Value:
${value}`);
```

```
}
```

Output

```
Key: key1, Value: value1

Key: key2, Value: value2

Key: key3, Value: value3
```

Explanation:

Here,

- map is the Map object you want to iterate over.

- for (const [key, value] of map) initiates the for...of loop, where [key, value] represents each key-value pair in the Map during each iteration.

- Inside the loop, console.log(Key: ${key}, Value: ${value}); prints each key-value pair to the console during each iteration of the loop.

Iterate over a Set **:**

```
const s = new Set();

s.add(1);
```

```
s.add("red");




for (let n of s) {


  console.log(n);


}


// Output:
```

```
// 1



// red



const str = "Hello";



for (const char of str) {
```

```
console.log(char);



}
```

Output

```
H

e
```

l

l

o

Explanation:

Here,

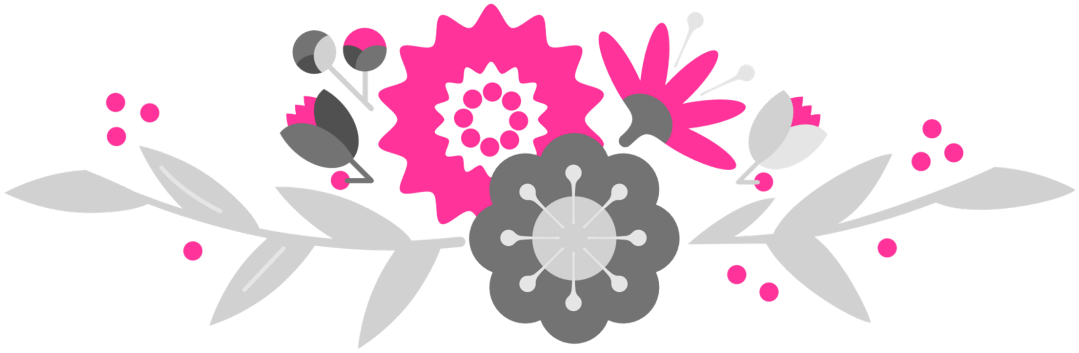
- str is the string you want to loop over.

- for (const char of str) initiates the for...of loop, where char represents each character in the string during each iteration.

- console.log(char) prints each character to the console during each iteration of the loop.

Chapter Eleven

Utilizing the JavaScript DOM



Web documents include a programming interface called the Document Object Model (DOM). Programs can alter the document's structure, design, and content by using it as a representation of the page. In order to enable computer languages to communicate with the page, the document is represented by the

DOM as nodes and objects. An HTML document becomes a document object when it loads in the browser. The HTML document is represented by this root element. It contains attributes and functions. We are able to incorporate dynamic material into our webpage with the aid of document objects.

---

window.document

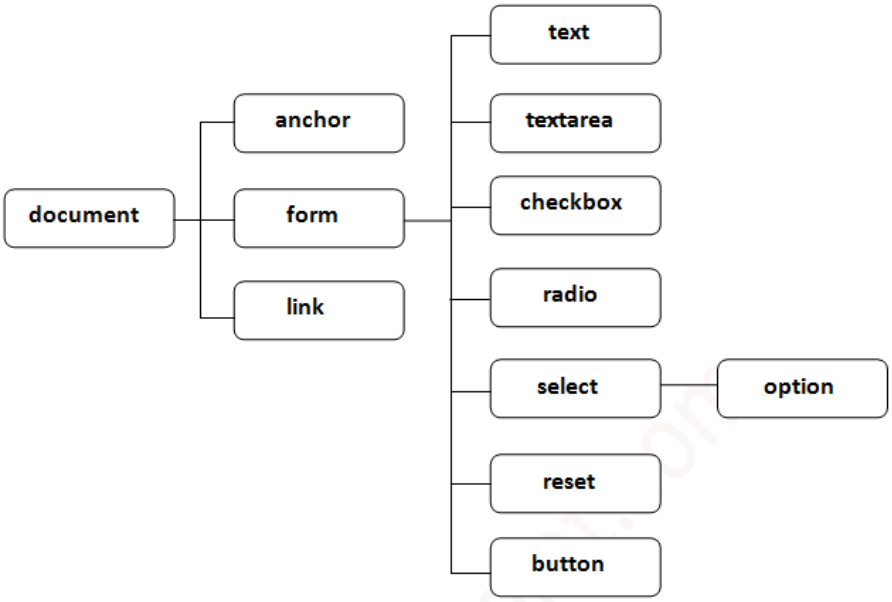
is same as

document

According to W3C - "The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

Properties of document object

Below are the properties of document object that can be accessed and modified by the document object.



Methods of document object

We can access and change the contents of document by its methods.

The important methods of document object are as follows:

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

Accessing field value by document object

In the example below, it gets the value of input text by user. Here, we are using **document.form1.name.value** to get

the value of name field.

Here, **document** is the root element that represents the html document.

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

function printvalue(){

var name=document.form1.name.value;

        alert("Welcome: "+name);

        }

        </script>

        <form name="form1" >

        Enter Name: <input type="text" name="name" />

        <input
         type="button" onclick="printvalue()" value="print name" />
```

</form>

---

form1 *is the name of the form.*

name *is the attribute name of the input text.*

value *is the property, that returns the value of the input text.*

JavaScript - document.getElementById() method

The **document.getElementById()** method returns the element of specified id.

The example above has used

document.form1.name.value

---

to get the value of the input value. Instead of this, we can use document.getElementById() method to get value of the input text. But we need to define id for the input field. Let's see the simple example of document.getElementById() method that prints cube of the given number.

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

function getcube(){
```

```
var  number=document.getElementById("number").
value;

alert(number*number*number);

}

</script>

<form>

Enter                                                    No:
<input  type="text"  id="number"  name="number"/
><br/>

<input type="button" value="cube" onclick="getcu
be()"/>

</form>
```

JavaScript - document.getElementsByName() method

The **document.getElementsByName()** method returns all the element of specified name.

The syntax of the getElementsByName() method is given below:

---

document.getElementsByName("name")

Example of document.getElementsByName() method

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

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

function totalelements()

{

var allgenders=document.getElementsByName("gender");
```

```
alert("Total Genders:"+allgenders.length);

}

</script>

<form>

Male:<input type="radio" name="gender" value="male">

Female:
<input type="radio" name="gender" value="female">

<input type="button" onclick="totalelements()" value="Total G
enders">

</form>
```

JavaScript - document.getElementsByTagName() method

The **document.getElementsByTagName()** method returns all the element of specified tag name.

The syntax of the getElementsByTagName() method is given below:

---

document.getElementsByTagName("name")

Example of document.getElementsByTagName() method

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

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

function countpara(){

var totalpara=document.getElementsByTagName("p");

alert("total p tags are: "+totalpara.length);

}

</script>

<p>This is a pragraph</p>
```

```
<p>Here we are going to count total number of paragraphs by getElementByTag
me() method.</p>

<p>Let's see the simple example</p>

<button onclick="countpara()">count paragraph</button>
```

Another example of document.getElementsByTagName() method

In this example, we going to count total number of h2 and h3 tags used in the document.

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

function counth2(){

var totalh2=document.getElementsByTagName("h2");

alert("total h2 tags are: "+totalh2.length);

}

function counth3(){

var totalh3=document.getElementsByTagName("h3");

alert("total h3 tags are: "+totalh3.length);

}

</script>

<h2>This is h2 tag</h2>

<h2>This is h2 tag</h2>

<h3>This is h3 tag</h3>

<h3>This is h3 tag</h3>
```

```
<h3>This is h3 tag</h3>

<button onclick="counth2()">count h2</button>

<button onclick="counth3()">count h3</button>
```

JavaScript - innerHTML

The **innerHTML** property can be used to write the dynamic html on the html document.

It is used mostly in the web pages to generate the dynamic html such as registration form, comment form, links etc.

Example of innerHTML property

In this example, we are going to create the html form when user clicks on the button. In this example, we are dynamically writing the html form inside the div name having the id mylocation. We are identifing this position by calling the document.getElementById() method.

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

function showcommentform() {

var data="Name:<input type='text' name='name'>
<br>Comment:<br><textarea rows='5' cols='80'>
</textarea>

<br><input type='submit' value='Post Comment'>";

document.getElementById('mylocation').innerHTML=data;
```

```
}

</script>

<form name="myForm">

<input type="button" value="comment" onclick="showcommen
tform()">

<div id="mylocation"></div>

</form>
```

Show/Hide Comment Form Example using innerHTML

```
<!DOCTYPE html >

<html>

<head>

<title> First JS </title>

<script>

var flag=true;

function commentform(){

var cform=" <form  action='Comment' > Enter Name:
<br><input  type='text' name='name' /><br/>
```

*Enter Email: **
<input** type='email' name='email' **/>
** Enter Comment: **
***

***<textarea** rows='5' cols='70' **></textarea>
 <input** type='submit' value='Post Comment' **/> </form>** ";*

if(flag){

document.getElementById("mylocation").innerHTML=cfo rm;

flag=false;

}else{

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

flag=true;

}

}

</script>

</head>

<body>

<button** onclick="commentform()" **>** Comment **</button>

```
<div  id="mylocation" ></div>

</body>

</html>
```

JavaScript - innerText

The **innerText** property can be used to write the dynamic text on the html document. Here, text will not be interpreted as html text but a normal text.

It is used mostly in the web pages to generate the dynamic content such as writing the validation message, password strength etc.

JavaScript innerText Example

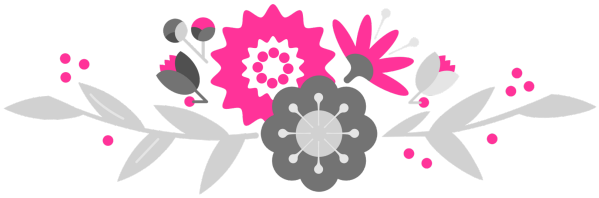
In this example, we are going to display the password strength when releases the key after press.

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

function validate() {

var msg;

if(document.myForm.userPass.value.length > 5){

msg="good";

}

else{

msg="poor";
```

```
}

document.getElementById('mylocation').innerText=msg;

 }



</script>

<form  name="myForm" >

<input
 type="password" value="" name="userPass" onkeyup="validat
e()" >

Strength: <span  id="mylocation" > no strength </span>

</form>
```

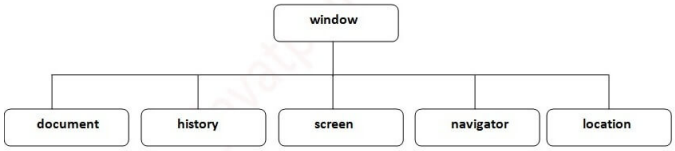
Chapter Twelve

Javascript Browser Object Model.



A JavaScript programming interface tool for interacting with web browsers is called Browser Object Model (BOM). This facilitates the JavaScript code, allowing access to and manipulation of the browser window, frames, and other browser-related objects.

JavaScript may interface with browser capabilities using the Browser Object Model (BOM). You can build and resize windows, show alarm messages, and alter the page that is currently displayed in the browser by using the BOM.



The default object of browser is window means you can call all the functions of window by specifying window or directly.

For example:

```
window.alert(" Hey Friend");

// Same as

alert(" Hey Friend ");
```

You can use a lot of properties (other objects) defined underneath the window object like document, history, screen, navigator, location, innerHeight, innerWidth,

Understanding the Browser Environment

Web browsers are complicated pieces of software. When they work well, they operate seamlessly and integrate all their functions into a smooth and seemingly simple web browsing experience. We all know that web browsers have an occasional hiccup and sometimes even crash. To understand why this happens, and to be able to make better use of browsers, it ’ s important to know the many different parts of the web browser and how these parts interact with each other.

The user interface

The part of the web browser that you interact with when you type in a URL, click the home button, create or use a bookmark, or change your browser settings is called the user interface, or browser chrome (not to be confused with Google ’ s Chrome browser).

The browser chrome consists of the web browser ’ s menus, window frames, toolbars, and buttons that are outside of the main content window where web pages load.

Loader

The *loader* is the part of a web browser that communicates with web servers and downloads web pages, scripts, CSS, graphics, and all the other components of a web page. Most often, loading is the part of displaying a web page that creates the longest wait time for the user.

The *HTML page* is the first part of a web page that must be downloaded, as it contains links and embedded scripts and styles that need to be processed in order to display the page.

It displays a graphical view of everything that happens during the loading of a web page, along with a timeline showing how long the loading of each part takes.

Once the HTML document is downloaded, browsers will open several connections to the server in order to download the other parts of the web page as quickly as possible. Generally, the parts of a web page that are linked from an HTML document (also known as the *resources*) are loaded in the order in which they appear in the HTML document.

For example, a script that is linked in the head element of the page will be loaded before one that ’ s linked at the bottom of the page. The load order of resources is critical to the efficiency and speed at which the page can be displayed to the user. In order for a web page to be displayed correctly, the CSS styles that apply to that page need to be loaded and parsed. Because of this, CSS should always be loaded in the head element at the top of the web page.

JavaScript sometimes affects the display of a web page as well, but more often, it affects only the functionality. When a script will affect the display of a web page, it should be loaded in the head of the document (after the CSS).

Scripts that aren ’ t critical to how the web page appears should be linked from the very end of the body element (right before the </body>), so as to not create a blocking scenario in which the browser waits for scripts to load before displaying anything to the user.

HTML parsing

After a web page is downloaded, the HTML parsing component of the browser goes to work parsing the HTML to create a model (called the Document Object Model or DOM) of the web page. The DOM, which is covered in detail in Chapter Eleven, is like a map of your web page. JavaScript programmers use this map to manipulate and access all the different parts of a web page.

Upon completion of the HTML parsing, the browser begins downloading the other components of the web page.

CSS parsing

Once the CSS for a web page is completely downloaded, the web browser will parse the styles and figure out which ones apply to the HTML document. CSS parsing is a complex process involving multiple passes over a document in order to apply each style correctly and to take into account how the styles impact each other.

JavaScript parsing

The next step in displaying a web page is the JavaScript parsing. The JavaScript parser compiles and runs every script in your web page in the order in which it appears in the document. If your JavaScript code adds or removes elements, text, or styles within the HTML DOM, the browser will update the HTML and CSS renderings accordingly.

Layout and rendering

Finally, once all the web page ’ s resources have been loaded and parsed, the browser determines how to display the page and then displays it. Unless you ’ ve specified that a script included earlier in the document should wait until the end to be executed, the layout and rendering of your scripts will occur in the order they ’ re included in the document.

In general, it ’ s better to display a web page to the user as quickly as possible, even if the page may not be fully functional when it first appears. Modern websites frequently employ this strategy specifically (called *deferred loading*) to improve the perceived performance of their pages.

If you ’ ve ever opened a web page and had to wait for a moment before you can use a form or interactive element, you ’ ve seen deferred loading in action.

Igniting the BOM

JavaScript programmers can find out information about a user ’ s web browser and control aspects of the user ’ s experience through an API called the Browser Object Model.

There is no official standard for the Browser Object Model. Different browsers implement it in different ways. However, there are some generally accepted standards for how JavaScript interacts with web browsers.

The Navigator Object

The Navigator object provides JavaScript with access to information about the user ' s web browser. The Navigator object takes its name from the first web browser to implement it, Netscape Navigator. The Navigator object isn ' t built into JavaScript. Rather, it ' s a feature of web browsers that is accessible using JavaScript. Nearly every web browser (and every modern web browser) has adopted the same terminology to refer to this highest - level browser object.

The Navigator object accesses helpful information such as

- The name of the web browser
- The version of the web browser

- The physical location of the computer the browser is running on (if the user allows the browser to access geolocation data).

- The language of the browser

- The type of computer the browser is running on.

Window Object

In JavaScript, the Window object represents the window that contains a Document Object Model document. The window object represents a window in browser. An object of window is created automatically by the browser. Window is the object of browser, it is not the object of JavaScript. The JavaScript objects are string, array, date etc.

The Window object provides access to various properties and methods that enable interaction with the browser environment, including manipulating the document, handling events, managing timers, displaying dialog boxes, and more.

Methods of window object

The important methods of window object are as follows:

Method	***Description***
alert()	displays the alert box containing message with ok button.
confirm()	displays the confirm dialog box containing message with ok and cancel button.
prompt()	displays a dialog box to get input from the user.
open()	opens the new window.
close()	closes the current window.
setTimeout()	performs action after specified time like calling function, evaluating expressions etc.

Example of alert() in javascript

It displays alert dialog box. It has message and ok button.

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

function msg(){
```

```
 alert("Hello Alert Box");

}

</script>

<input type="button" value="click" onclick="msg()"/>
```

Example of confirm() in javascript

It displays the confirm dialog box. It has message with ok and cancel buttons.

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

function msg(){

var v= confirm("Are u sure?");

if(v==true){

alert("ok");

}

else{
```

```
alert("cancel");

}

}

</script>

<input type="button" value="delete record" onclick="msg()"/>
```

Example of prompt() in javascript

It displays prompt dialog box for input. It has message and textfield.

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

function msg(){

var v= prompt("Who are you?");

alert("I am "+v);

}

</script>

<input type="button" value="click" onclick="msg()"/>
```

Example of open() in javascript

It displays the content in a new window.

```
<script type="text/javascript">
function msg(){
open("http://www.google.com");
}
</script>
<input type="button" value="javatpoint" onclick="msg()"/>
```

Example of setTimeout() in javascript

It performs its task after the given milliseconds.

```
<script type="text/javascript">
function msg(){
setTimeout(
function(){
```

```
alert("Welcome to Javatpoint after 2 seconds")

},2000);

}

</script>

<input type="button" value="click" onclick="msg()"/>
```

JavaScript 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

There are only 1 property of history object.

No.	*Property*	*Description*
1	length	returns the length of the history URLs.

Methods of JavaScript history object

There are only 3 methods of history object.

No.	Method	Description
1	forward()	loads the next page.
2	back()	loads the previous page.
3	go()	loads the given page number.

Example of history object

Let’s see the different usage of history object.

---

1. history.back();//for previous page

2. history.forward();//for next page

3. history.go(2);//for next 2nd page

4. history.go(-2);//for previous 2nd page

JavaScript Navigator Object

The **JavaScript navigator object** is used for browser detection. It can be used to get browser information such as appName, appCodeName, userAgent etc.

The navigator object is the window property, so it can be accessed by:

```
window.navigator
```

or

```
navigator
```

Property of JavaScript navigator object

There are many properties of navigator object that returns information of the browser.

No.	***Property***	***Description***
1	appName	returns the name

2	appVersion	returns the version
3	appCodeName	returns the code name
4	cookieEnabled	returns true if cookie is enabled otherwise false
5	userAgent	returns the user agent
6	language	returns the language. It is supported in Netscape and Firefox only.
7	userLanguage	returns the user language. It is supported in IE only.
8	plugins	returns the plugins. It is supported in Netscape and Firefox only.
9	systemLanguage	returns the system language. It is supported in IE only.
10	mimeTypes[]	returns the array of mime type. It is supported in Netscape and Firefox only.
11	platform	returns the platform e.g. Win32.
12	online	returns true if browser is online otherwise false.

Methods of JavaScript navigator object

The methods of navigator object are given below.

No.	***Method***	***Description***
1	javaEnabled()	checks if java is enabled.
2	taintEnabled()	checks if taint is enabled. It is deprecated since JavaScript 1.2.

Example of navigator object

Let’s see the different usage of history object.

```
<script>

document.writeln("                                    <br/>
navigator.appCodeName: "+navigator.appCodeName);

document.writeln("                                    <br/>
navigator.appName: "+navigator.appName);

document.writeln("                                    <br/>
navigator.appVersion: "+navigator.appVersion);

document.writeln("                                    <br/>
navigator.cookieEnabled: "+navigator.cookieEnabled);

document.writeln("                                    <br/>
navigator.language: "+navigator.language);

document.writeln("                                    <br/>
navigator.userAgent: "+navigator.userAgent);
```

```
document.writeln("<br/>
navigator.platform: "+navigator.platform);

document.writeln("<br/>
navigator.onLine: "+navigator.onLine);

</script>
```

JavaScript 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

or

screen

Property of JavaScript Screen Object

There are many properties of screen object that returns information of the browser.

No.	***Property***	***Description***
1	width	returns the width of the screen
2	height	returns the height of the screen
3	availWidth	returns the available width
4	availHeight	returns the available height
5	colorDepth	returns the color depth
6	pixelDepth	returns the pixel depth.

Example of JavaScript Screen Object

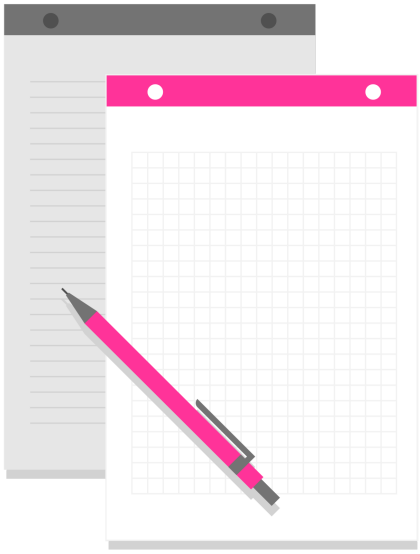
Let’s see the different usage of screen object.

```
<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>
```

Chapter Thirteen

JavaScript Form Validation



JavaScript Form Validation ensures data integrity by verifying user input before submission.

It validates fields like passwords, emails, and selections, providing alerts for invalid data, and enhancing user experience and data accuracy.

It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.

JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation. Most of the web developers prefer JavaScript form validation.

Through JavaScript, we can validate name, password, email, date, mobile numbers and more fields.

Approach for Form Validation in JavaScript

In this approach, we are following these steps

- **Data Retrieval:**

  It retrieves the values of various form fields like name, email, course selection, password, and address using document.forms.RegForm.

- **Data Validation:**

  -

  Name Validation:

Checks if the name field is empty or contains any digits.

-

Address Validation:

Ensures the address field is not empty.

-

Email Validation:

Verifies if the email field is not empty and contains the ‘@’ symbol.

-

Password Validation:

Validates that the password field is not empty and has a

minimum length of 6 characters.

-

Course Selection Validation:

Ensures that a course is selected from the dropdown.

- **Error Handling:**

-

If any of the validation criteria fail, it displays an alert message using window.alert.

-

Sets focus back to the respective field that failed validation, ensuring the user’s attention is drawn to the problematic field.

- **Submission Control:**

  - Returns true if all validation checks pass, indicating that the form can be submitted.
  Otherwise, it returns false, preventing form submission.

- **Focus Adjustment:**

  -

Sets focus to the first field that failed validation, ensuring the user's attention is drawn to the problematic field.

JavaScript Form Validation Example

In this example, we are going to validate the name and password. The name can't be empty and password can't be less than 6 characters long.

Here, we are validating the form on form submit.
The user will not be forwarded to the next page until given values are correct.

Here, we are validating the form on form submit.
The user will not be forwarded to the next page until given values are correct.

```
<script>

function validateform(){

var name=document.myform.name.value;
```

```
var password=document.myform.password.value;

if (name==null || name==""){

  alert("Name can't be blank");

  return false;

}else if(password.length <6 ){

  alert("Password must be at least 6 characters long.");

  return false;

  }

}

</script>

<body>

<form
 name="myform" method="post" action="abc.jsp" onsubmit="return vali
dateform()"  >

Name:  <input  type="text" name="name" ><br/>

Password:  <input  type="password" name="password" ><br/>

<input  type="submit" value="register" >

</form>
```

JavaScript Retype Password Validation

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

function matchpass(){

var firstpassword=document.f1.password.value;

var secondpassword=document.f1.password2.valu

if(firstpassword==secondpassword){

return true;

}

else{

alert("password must be same!");

return false;

}

}

</script>

<form
 name="f1" action="register.jsp" onsubmit="return matchpass()"
>

Password:  <input  type="password"  name="password"  />
<br/>
```

Re-enter Password: ***<input*** *type="password" name="password2"* ***/>
***

<input *type="submit"* ***>***

</form>

JavaScript Number Validation

Let's validate the textfield for numeric value only. Here, we are using isNaN() function.

```
<script>

function validate(){

var num=document.myform.num.value;

if (isNaN(num)){

  document.getElementById("numloc").innerHTML="Enter Numeric v
alue only";

 return false;

}else{

 return true;

 }

}

</script>

<form  name="myform" onsubmit="return validate()"  >

Number:  <input  type="text" name="num" ><span  id="numloc"
></span><br/>

<input  type="submit" value="submit" >
```

```
</form>
```

JavaScript validation with image

Let’s see an interactive JavaScript form validation example that displays correct and incorrect image if input is correct or incorrect.

```
<script>

function validate(){

var name=document.f1.name.value;

var password=document.f1.password.value;

var status=false;



if(name.length <1 ){

document.getElementById("nameloc").innerHTML=

" <img src='unchecked.gif'/> Please enter your name";

status=false;

}else{

document.getElementById("nameloc").innerHTML=" <img src='checked.
gif'/>";
```

```
status=true;

}

if(password.length <6 ){

document.getElementById("passwordloc").innerHTML=

" <img src='unchecked.gif'/> Password must be at least 6 char long";

status=false;

}else{

document.getElementById("passwordloc").innerHTML="  <img  src='chec
ked.gif'/>";

}

return status;

}

</script>



<form  name="f1" action="#" onsubmit="return validate()" >

<table>

<tr><td> Enter Name: </td><td><input  type="text" name="name"
/>

<span  id="nameloc" ></span></td></tr>

<tr><td>  Enter  Password:  </td><td><input
 type="password" name="password" />
```

```
<span id="passwordloc" ></span></td></tr>

<tr><td colspan="2" ><input type="submit" value="register" />
</td></tr>

</table>

</form>
```

JavaScript email validation

We can validate the email by the help of JavaScript. There are many criteria that need to be follow to validate the email id such as:

- email id must contain the @ and character
- There must be at least one character before and after the @.
- There must be at least two characters after. (dot).

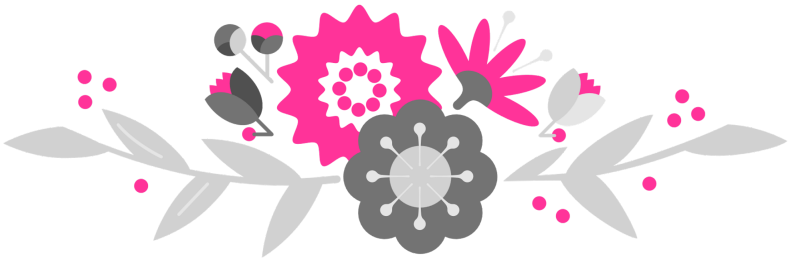
Let's see the simple example to validate the email field.

```
<script>

function validateemail()
```

```
{

var x=document.myform.email.value;

var atposition=x.indexOf("@");

var dotposition=x.lastIndexOf(".");

if (atposition <1  || dotposition <atposition +2 || dotposition+2 >
=x.length){

                alert("Please enter a valid e-
mail address \n atpostion:"+atposition+"\n dotposition:"+dotposition);


  return false;

  }

}

</script>

<body>

<form
 name="myform"  method="post" action="#" onsubmit="return valid
ateemail();" >

Email:  <input  type="text" name="email" ><br/>

<input  type="submit" value="register" >

</form>
```

Chapter Fourteen

JavaScript Object Oriented Programming



Object-Oriented Programming is a programming style based on classes and objects. These group data (properties) and methods (actions) inside a box. OOP was developed to make code more flexible and easier to maintain. JavaScript is prototype-based procedural language, which means it supports both functional and object-oriented programming.

JavaScript Classes

In JavaScript, classes are the special type of functions. We can define the class just like function declarations and function expressions.

The JavaScript class contains various class members within a body including methods or constructor. The class is executed in strict mode. So, the code containing the silent error or mistake throws an error.

The class syntax contains two components:

- Class declarations
- Class expressions

Class Declarations

A class can be defined by using a class declaration. A class keyword is used to declare a class with any particular name. According to JavaScript naming conventions, the name of the class always starts with an uppercase letter.

Class Declarations Example

Let's see a simple example of declaring the class.

<script>

```
//Declaring class
class Employee
 {
//Initializing an object
   constructor(id,name)     {
     this.id=id;
     this.name=name;
 }
//Declaring method
   detail()    {
 document.writeln(this.id+" "+this.name+" <br> ")
   }
 }
//passing object to a variable
var e1=new Employee(101,"Martin Roy");
var e2=new Employee(102,"Duke William");
e1.detail(); //calling method
e2.detail();
</script>
```

Class Declarations Example: Hoisting

Unlike function declaration, the class declaration is not a part of JavaScript hoisting. So, it is required to declare the class before invoking it.

Let's see an example.

```
<script>
//Here, we are invoking the class before declaring it.
var e1=new Employee(101,"Martin Roy");
var e2=new Employee(102,"Duke William");
e1.detail(); //calling method
e2.detail();

//Declaring class
class Employee
{
//Initializing an object
    constructor(id,name)
    {
      this.id=id;
      this.name=name;
    }
    detail()
```

```
  {
 document.writeln(this.id+" "+this.name+" <br> ")
  }
 }
</script>
```

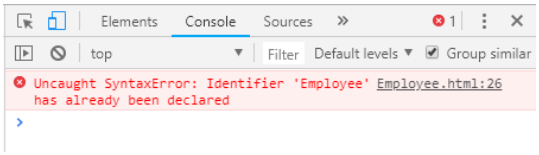
Class Declarations Example: Re-declaring Class

A class can be declared once only. If we try to declare class more than one time, it throws an error.

```
<script>
//Declaring class
class Employee
 {
//Initializing an object
   constructor(id,name)
   {
     this.id=id;
     this.name=name;
   }
   detail()
   {
```

```
  document.writeln(this.id+" "+this.name+" <br> ")
    }
  }
//passing object to a variable
var e1=new Employee(101,"Martin Roy");
var e2=new Employee(102,"Duke William");
e1.detail(); //calling method
e2.detail();
//Re-declaring class
class Employee
  {
  }
</script>
```

OUTPUT



Class expressions

Another way to define a class is by using a class expression. Here, it is not mandatory to assign the name of the class. So, the class expression can be named or unnamed. The class expression allows us to fetch the class name. However, this will not be possible with class declaration.

Unnamed Class Expression

The class can be expressed without assigning any name to it.

Let's see an example.

```
<script>
var emp = class {
  constructor(id, name) {
    this.id = id;
    this.name = name;
  }
};
document.writeln(emp.name);
</script>
```

Class Expression Example: Re-declaring Class

Unlike class declaration, the class expression allows us to re-declare the same class. So, if we try to declare the class more than one time, it throws an error.

```
<script>
//Declaring class
var emp=class
  {
//Initializing an object
  constructor(id,name)
    {
      this.id=id;
      this.name=name;
    }
//Declaring method
detail()
    {
  document.writeln(this.id+" "+this.name+" <br> ")
    }
  }
//passing object to a variable
var e1=new emp(101,"Martin Roy");
var e2=new emp(102,"Duke William");
e1.detail(); //calling method
e2.detail();
```

```
//Re-declaring class
var emp=class
  {
//Initializing an object
    constructor(id,name)
    {
      this.id=id;
      this.name=name;
    }
    detail()
    {
  document.writeln(this.id+" "+this.name+" <br> ")
    }
  }
//passing object to a variable
var e1=new emp(103,"James Bella");
var e2=new emp(104,"Nick Johnson");
e1.detail(); //calling method
e2.detail();
</script>
```

Named Class Expression Example

We can express the class with the particular name. Here, the scope of the class name is up to the class body. The class is retrieved using class.name property.

```
<script>
var emp = class Employee {
  constructor(id, name) {
    this.id = id;
    this.name = name;
  }
};
document.writeln(emp.name);
/*document.writeln(Employee.name);
Error occurs on console:
"ReferenceError: Employee is not defined
*/
</script>
```

JavaScript Objects

A JavaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

Creating Objects in JavaScript

There are 3 ways to create objects.

1. By object literal

2. By creating instance of Object directly (using new keyword)

3. By using an object constructor (using new keyword)

1) JavaScript Object 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).

Let's see the simple example of creating object in JavaScript.

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

2) By creating instance of Object

The syntax of creating object directly is given below:

```
var objectname=new Object();
```

Here, ***new keyword*** *is used to create object.*

```
<script>
```

```
var emp=new Object();

emp.id=101;

emp.name="Ravi Malik";

emp.salary=50000;

document.write(emp.id+" "+emp.name+" "+emp.salary);

</script>
```

3) By using an Object constructor

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword. 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>
```

Defining method in JavaScript Object

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.

The example of defining method in object is given below.

```
<script>

function emp(id,name,salary){

this.id=id;

this.name=name;

this.salary=salary;



this.changeSalary=changeSalary;

function changeSalary(otherSalary){
```

```
this.salary=otherSalary;

}

}

e=new emp(103,"Sonoo Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);

e.changeSalary(45000);

document.write(" <br> "+e.id+" "+e.name+" "+e.salary);

</script>
```

JavaScript Object Methods

The various methods of Object are as follows:

S.No	Methods	Description
1	Object.assign()	This method is used to copy enumerable and own properties from a source object to a target object
2	Object.create()	This method is used to create a new object with the specified prototype object and properties.
3	Object.defineProperty()	This method is used to describe some behavioral attributes of the property.
4	Object.defineProperties()	This method is used to create or configure multiple object properties.
5	Object.entries()	This method returns an array with arrays of the key, value pairs.

6	Object.freeze()	This method prevents existing properties from being removed.
7	Object.getOwnPropertyDescriptor()	This method returns a property descriptor for the specified property of the specified object.
8	Object.getOwnPropertyDescriptors()	This method returns all own property descriptors of a given object.
9	Object.getOwnPropertyNames()	This method returns an array of all properties (enumerable or not) found.
10	Object.getOwnPropertySymbols()	This method returns an array of all own symbol key properties.
11	Object.getPrototypeOf()	This method returns the prototype of the specified object.
12	Object.is()	This method determines whether two values are the same value.
13	Object.isExtensible()	This method determines if an object is extensible
14	Object.isFrozen()	This method determines if an object was frozen.
15	Object.isSealed()	This method determines if an object is sealed.
16	Object.keys()	This method returns an array of a given object's own property names.
17	Object.preventExtensions()	This method is used to prevent any extensions of an object.
18	Object.seal()	This method prevents new properties from being added and marks all existing properties as non-configurable.
19	Object.setPrototypeOf()	This method sets the prototype of a specified object to another object.
20	Object.values()	This method returns an array of values.

JavaScript Prototype Object

JavaScript is a prototype-based language that facilitates the objects to acquire properties and features from one another. Here, each object contains a prototype object.

In JavaScript, whenever a function is created the prototype property is added to that function automatically. This property is a prototype object that holds a constructor property.

Syntax:

```
ClassName.prototype.methodName
```

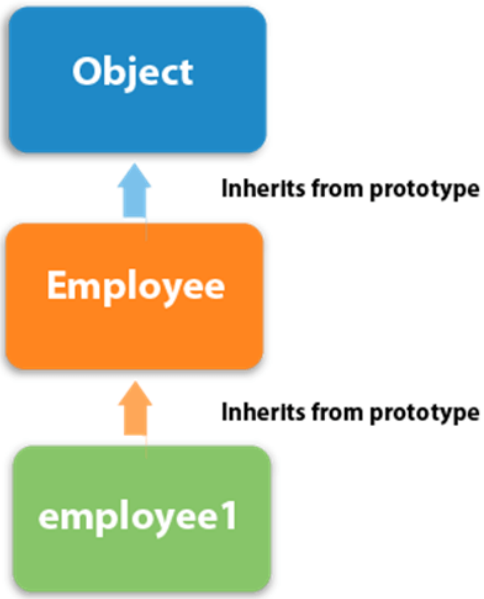
What is the requirement of a prototype object?

Whenever an object is created in JavaScript, its corresponding functions are loaded into memory. So, a new copy of the function is created on each object creation.

In a prototype-based approach, all the objects share the same function. This ignores the requirement of creating a new copy of function for each object. Thus, the functions are loaded once into the memory.

Prototype Chaining

In JavaScript, each object contains a prototype object that acquires properties and methods from it. Again, an object's prototype object may contain a prototype object that also acquires properties and methods, and so on. It can be seen as prototype chaining.



JavaScript Prototype Object

Example 1

Let's see an example to add a new method to the constructor function.

```
<script>
function Employee(firstName,lastName)
{
  this.firstName=firstName;
  this.lastName=lastName;
}
Employee.prototype.fullName=function()
  {
    return this.firstName+" "+this.lastName;
  }

var employee1=new Employee("Martin","Roy");
var employee2=new Employee("Duke", "William");
document.writeln(employee1.fullName()+" <br> ");
document.writeln(employee2.fullName());
</script>
```

Output:

Martin Roy

Duke William

Example 2

Let's see an example to add a new property to the constructor function.

```
<script>

function Employee(firstName,lastName)

{

  this.firstName=firstName;

  this.lastName=lastName;

}



Employee.prototype.company="Javatpoint"



var employee1=new Employee("Martin","Roy");

var employee2=new Employee("Duke", "William");

document.writeln(employee1.firstName+" "+employee1.lastName+" "+employ
1.company+" <br> ");

document.writeln(employee2.firstName+" "+employee2.lastName+" "+employ
2.company);

</script>
```

Output

Martin Roy Javatpoint

Duke William Javatpoint

JavaScript Constructor Method

A JavaScript constructor method is a special type of method which is used to initialize and create an object. It is called when memory is allocated for an object.

Points to remember

- The constructor keyword is used to declare a constructor method.
- The class can contain one constructor method only.
- JavaScript allows us to use parent class constructor through super keyword.

Constructor Method Example

Let's see a simple example of a constructor method.

```
<script>
class Employee {
  constructor() {
    this.id=101;
    this.name = "Martin Roy";
  }
}
var emp = new Employee();
document.writeln(emp.id+" "+emp.name);
</script>
```

Output

101 Martin Roy

Constructor Method Example: super keyword

The super keyword is used to call the parent class constructor. Let's see an example.

```
<script>
class CompanyName
{
  constructor()
  {
    this.company="Javatpoint";
  }
}
class Employee extends CompanyName {
  constructor(id,name) {
   super();
    this.id=id;
    this.name=name;
  }
}
var emp = new Employee(1,"John");
```

document.writeln(emp.id+" "+emp.name+" "+emp.company);

</script>

Output

1 John Javatpoint

Note -

If we didn't specify any constructor method, JavaScript use default constructor method.

JavaScript static Method

The JavaScript provides static methods that belong to the class instead of an instance of that class. So, an instance is not required to call the static method. These methods are called directly on the class itself.

Points to remember

- The static keyword is used to declare a static method.
- The static method can be of any name.
- A class can contain more than one static method.
- If we declare more than one static method with a similar name, the JavaScript always invokes the last one.
- The static method can be used to create utility functions.

- We can use this keyword to call a static method within another static method.

- We cannot use this keyword directly to call a static method within the non-static method. In such case, we can call the static method either using the class name or as the property of the constructor.

JavaScript static Method Example

Let's see a simple example of a static method.

```
<script>
class Test
{
  static display()
  {
    return "static method is invoked"
  }
}
document.writeln(Test.display());
```

```
</script>
```

Output

static method is invoked

Example 2

Le's see an example to invoke more than one static method.

```
<script>
class Test
{
  static display1()
  {
```

```
    return "static method is invoked"
  }
  static display2()
  {
    return "static method is invoked again"
  }
}
document.writeln(Test.display1()+" <br> ");
document.writeln(Test.display2());
</script>
```

Output

static method is invoked

static method is invoked again

Example 3

Let's see an example to invoke more than one static method with similar names.

```
<script>

class Test

{

  static display()

  {

    return "static method is invoked"

  }

  static display()

  {

    return "static method is invoked again"

  }

}

document.writeln(Test.display());

</script>
```

Output

static method is invoked again

Example 4

Let's see an example to invoke a static method within the constructor.

```
<script>
class Test {
  constructor() {
  document.writeln(Test.display()+" <br> ");
  document.writeln(this.constructor.display());
  }
  static display() {
     return "static method is invoked"
  }
}
var t=new Test();
</script>
```

Output

static method is invoked

static method is invoked

Example 5

Let's see an example to invoke a static method within the non-static method.

```
<script>
class Test {
  static display() {
      return "static method is invoked"
  }

 show() {
  document.writeln(Test.display()+" <br> ");
  }
}
var t=new Test();
t.show();
</script>
```

Output

static method is invoked

JavaScript Encapsulation

The JavaScript Encapsulation is a process of binding the data (i.e. variables) with the functions acting on that data. It allows us to control the data and validate it. To achieve an encapsulation in JavaScript: -

- Use var keyword to make data members private.

- Use setter methods to set the data and getter methods to get that data.

The encapsulation allows us to handle an object using the following properties:

Read/Write - Here, we use setter methods to write the data and getter methods read that data.

Read Only - In this case, we use getter methods only.

Write Only - In this case, we use setter methods only.

JavaScript Encapsulation Example

Let's see a simple example of encapsulation that contains two data members with its setter and getter methods.

```
<script>
class Student
 {
  constructor()
  {
    var name;
    var marks;
  }
     getName()
     {
       return this.name;
     }
```

```
  setName(name)
  {
    this.name=name;
  }

  getMarks()
  {
    return this.marks;
  }
 setMarks(marks)
{
  this.marks=marks;
 }

 }
 var stud=new Student();
  stud.setName("John");
  stud.setMarks(80);
```

```
document.writeln(stud.getName()+" "+stud.getMarks());

</script>
```

Output

John 80

JavaScript Encapsulation Example: Validate

In this example, we validate the marks of the student.

```
<!DOCTYPE html>

<html>

<body>

<script>

class Student

{

constructor()

{

var name;
```

```
    var marks;
    }
    getName()
    {
    return this.name;
    }
    setName(name)
    {
    this.name=name;
    }

getMarks()
    {
    return this.marks;
    }
    setMarks(marks)
    {
    if(marks<0||marks>100)
    {
```

```
alert("Invalid Marks");

}

else

{

this.marks=marks;

}

}

}

var stud=new Student();

stud.setName("John");

stud.setMarks(110);//alert() invokes

document.writeln(stud.getName()+"
"+stud.getMarks());

</script>



</body>

</html>
```

Output

John undefined

JavaScript Encapsulation Example: Prototype-based approach

Here, we perform prototype-based encapsulation.

```
<!DOCTYPE html>

<html>

<body>

<script>

function Student(name,marks)

{
```

```
var s_name=name;

var s_marks=marks;

Object.defineProperty(this,"name",{

 get:function()

 {

  return s_name;

 },
```

```
    set:function(s_name)

    {

      this.s_name=s_name;

    }

});

    Object.defineProperty(this,"marks",{

    get:function()

    {
```

```
    return s_marks;

  },

 set:function(s_marks)

 {

  this.s_marks=s_marks;

 }

});
```

```
}

  var stud=new Student("John",80);

  document.writeln(stud.name+" "+stud.marks);

</script>

</body>

</html>
```

JavaScript Inheritance

The JavaScript inheritance is a mechanism that allows us to create new classes on the basis of already existing classes. It provides flexibility to the child class to reuse the methods and variables of a parent class.

The JavaScript **extends** keyword is used to create a child class on the basis of a parent class. It facilitates child class to acquire all the properties and behavior of its parent class.

Points to remember

- It maintains an IS-A relationship.
- The extends keyword is used in class expressions or class declarations.
- Using extends keyword, we can acquire all the properties and behavior of the inbuilt object as well as custom classes.
- We can also use a prototype-based approach to achieve inheritance.

JavaScript extends Example: inbuilt object

In this example, we extends **Date** object to display today's date.

```
<!DOCTYPE html>
```

```
<html>

<body>

<script>

class Moment extends Date {

  constructor() {

    super();

  }}
```

```
        var m=new Moment();



        document.writeln("Current date:")



        document.writeln(m.getDate()+"-"+
(m.getMonth()+1)+"-"+m.getFullYear());



        </script>



        </body>



    </html>
```

Let's see one more example to display the year value from the given date.

```
<!DOCTYPE html>

<html>

<body>

<script>

class Moment extends Date {

  constructor(year) {

    super(year);

  }}
```

```
var m=new Moment("August 15, 1947 20:22:10");

document.writeln("Year value:")

document.writeln(m.getFullYear());

</script>

</body>

</html>
```

JavaScript extends Example: Custom class

In this example, we declare sub-class that extends the properties of its parent class.

```
<!DOCTYPE html>

<html>

<body>

<script>

class Bike

{
```

```
  constructor()

  {

    this.company="Honda";

  }

}

class Vehicle extends Bike {

  constructor(name,price) {
```

```
        super();


        this.name=name;


        this.price=price;


      }


    }


    var v = new Vehicle("Shine","70000");


    document.writeln(v.company+" "+v.name+"
"+v.price);
```

```
</script>


</body>


</html>
```

JavaScript extends Example: a Prototype-based approach

Here, we perform prototype-based inheritance. In this approach, there is no need to use class and extends keywords.

```
<script>

//Constructor function

function Bike(company)

{

   this.company=company;

}

Bike.prototype.getCompany=function()

{
```

```
  return this.company;

}

//Another constructor function

function Vehicle(name,price) {

 this.name=name;

  this.price=price;

  }

var bike = new Bike("Honda");

Vehicle.prototype=bike; //Now Bike treats as a parentof Vehicle.

var vehicle=new Vehicle("Shine",70000);

document.writeln(vehicle.getCompany()+" "+vehicle.name+" "+vehi
cle.price);

</script>
```

JavaScript Polymorphism

The polymorphism is a core concept of an object-oriented paradigm that provides a way to perform a single action in different forms. It provides an ability to call the same method on different JavaScript objects. As JavaScript is not a type-safe language, we can pass any type of data members with the methods.

JavaScript Polymorphism Example 1

Let's see an example where a child class object invokes the parent class method.

```
<!DOCTYPE html>

<html>
```

```
<body>

<script>

class A

  {

    display()

    {

     document.writeln("A is invoked");
```

```
    }


  }


class B extends A


  {


  }


var b=new B();
```

```
b.display();


</script>


</body>


</html>
```

Example 2

Let's see an example where a child and parent class contains the same method. Here, the object of child class invokes both classes method.

```
<script>
class A
  {
    display()
   {
     document.writeln("A is invoked <br> ");
   }
  }
class B extends A
  {
   display()
   {
     document.writeln("B is invoked");
   }
  }
```

```
var a=[new A(), new B()]

a.forEach(function(msg)

{

msg.display();

});

</script>
```

***O** utput*

A is invoked

B is invoked

Example 3

Let's see the same example with prototype-based approach.

```
<!DOCTYPE html>

<html>

<body>

<script>

function A()
```

```
{



}

A.prototype.display=function()

{

  return "A is invoked";

}
```

```
function B()

{

}

B.prototype=Object.create(A.prototype);

var a=[new A(), new B()]

a.forEach(function(msg)

{
```

```
    document.writeln(msg.display()+"<br>");


});


<script>


</body>


</html>
```

JavaScript Abstraction

An abstraction is a way of hiding the implementation details and showing only the functionality to the users. In other words, it ignores the irrelevant details and shows only the required one.

Points to remember

- We cannot create an instance of Abstract Class.
- It reduces the duplication of code.

JavaScript Abstraction Example

Example 1

Let's check whether we can create an instance of Abstract class or not.

```
<script>

//Creating a constructor function

function Vehicle()

{

    this.vehicleName= vehicleName;

    throw new Error("You cannot create an instance of Abstrac
t class");

}

Vehicle.prototype.display=function()

{

    return this.vehicleName;

}

var vehicle=new Vehicle();

  </script>
```

Example 2

Let's see an example to achieve abstraction.

```
<!DOCTYPE html>

<html>

<body>

<script>

//Creating a constructor function
```

```
function Vehicle()

{

    this.vehicleName="vehicleName";

    throw new Error("You cannot create an
instance of Abstract Class");

}

Vehicle.prototype.display=function()
```

```
{

    return "Vehicle is: "+this.vehicleName;

}

//Creating a constructor function

function Bike(vehicleName)

{

    this.vehicleName=vehicleName;
```

```
}

//Creating object without using the function constructor

Bike.prototype=Object.create(Vehicle.prototype);

var bike=new Bike("Honda");

document.writeln(bike.display());

</script>
```

```
</body>

</html>
```

Example 3

In this example, we use instanceof operator to test whether the object refers to the corresponding class.

```
<script>

//Creating a constructor function

 function Vehicle()

{

    this.vehicleName=vehicleName;
```

```
    throw new Error("You cannot create an instance of Abstrac
t class");

}

//Creating a constructor function

function Bike(vehicleName)

{

    this.vehicleName=vehicleName;

}

Bike.prototype=Object.create(Vehicle.prototype);

var bike=new Bike("Honda");

document.writeln(bike instanceof Vehicle);

document.writeln(bike instanceof Bike);

  </script>
```

Chapter Fifteen

JavaScript Cookies

A cookie is an amount of information that persists between a server-side and a client-side. A web browser stores this information at the time of browsing.

A cookie contains the information as a string generally in the form of a name-value pair separated by semi-colons. It maintains the state of a user and remembers the user's information among all the web pages.

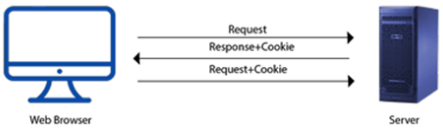
How Cookies Works?

- When a user sends a request to the server, then each of that request is treated as a new request sent by the different user.

- So, to recognize the old user, we need to add the cookie with the response from the server.

- browser at the client-side.

- Now, whenever a user sends a request to the server, the cookie is added with that request automatically. Due to the cookie, the server recognizes the users.



How to create a Cookie in JavaScript?

In JavaScript, we can create, read, update and delete a cookie by using **document.cookie** property.

The following syntax is used to create a cookie:

```
document.cookie="name=value";
```

JavaScript Cookie Example

Example 1

Let's see an example to set and get a cookie.

```
<!DOCTYPE html >
<html>
<head>
</head>
<body>
<input  type="button" value="setCookie" onclick="setCookie()" >
<input  type="button" value="getCookie" onclick="getCookie()" >
   <script>
   function setCookie()
   {
      document.cookie="username=Duke Martin";
   }
```

```
    function getCookie()
    {
       if(document.cookie.length!=0)
       {
       alert(document.cookie);
       }
       else
       {
       alert("Cookie not available");
       }
    }
     </script>
</body>
</html>
```

Example 2

Here, we display the cookie's name-value pair separately.

```
<!DOCTYPE html >
```

```
<html>
<head>
</head>
<body>
<input type="button" value="setCookie" onclick="setCookie()" >
<input type="button" value="getCookie" onclick="getCookie()" >
 <script>
function setCookie()
{
   document.cookie="username=Duke Martin";
}
function getCookie()
{
   if(document.cookie.length!=0)
   {
      var array=document.cookie.split("=");
   alert("Name="+array[0]+" "+"Value="+array[1];
   }
   else
   {
   alert("Cookie not available");
```

```
    }
  }
  </script>

</body>
</html>
```

Example 3

In this example, we provide choices of color and pass the selected color value to the cookie. Now, cookie stores the last choice of a user in a browser. So, on reloading the web page, the user's last choice will be shown on the screen.

```
<!DOCTYPE html >
<html>
<head>
</head>
<body>
    <select  id="color" onchange="display()" >
        <option  value="Select Color" > Select Color </option>
        <option  value="yellow" > Yellow </option>
```

```
    <option  value="green" > Green </option>
    <option  value="red" > Red </option>
</select>
<script  type="text/javascript" >
  function display()
  {
     var value = document.getElementById("color").value;
     if (value != "Select Color")
     {
        document.bgColor = value;
        document.cookie = "color=" + value;
     }
  }
  window.onload = function ()
  {
     if (document.cookie.length != 0)
     {
        var array = document.cookie.split("=");
        document.getElementById("color").value = array[1];
        document.bgColor = array[1];
     }
```

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

Cookie Attributes

JavaScript provides some optional attributes that enhance the functionality of cookies. Here, is the list of some attributes with their description.

Attributes	*Description*
expires	It maintains the state of a cookie up to the specified date and time.
max-age	It maintains the state of a cookie up to the specified time. Here, time is given in seconds.
path	It expands the scope of the cookie to all the pages of a website.
domain	It is used to specify the domain for which the cookie is valid.

Cookie expires attribute

The cookie expires attribute provides one of the ways to create a persistent cookie. Here, a date and time are declared that represents the active period of a cookie. Once the declared time is passed, a cookie is deleted automatically.

Let's see an example of cookie expires attribute.

```
<!DOCTYPE html >
<html>
<head>
</head>
<body>
<input  type="button" value="setCookie" onclick="setCookie()" >
<input  type="button" value="getCookie" onclick="getCookie()" >
    <script>
   function setCookie()
   {
       document.cookie="username=Duke Martin;expires=Sun, 20 Aug 2030 12:
00:00 UTC";
   }
   function getCookie()
   {
      if(document.cookie.length!=0)
      {
         var array=document.cookie.split("=");
      alert("Name="+array[0]+" "+"Value="+array[1]);
      }
      else
```

```
    {
    alert("Cookie not available");
    }
  }
  </script>
</body>
</html>
```

Cookie max-age attribute

The cookie max-age attribute provides another way to create a persistent cookie. Here, time is declared in seconds. A cookie is valid up to the declared time only.

Let's see an example of cookie max-age attribute.

```
<!DOCTYPE html >
<html>
<head>
</head>
<body>
```

```
<input type="button" value="setCookie" onclick="setCookie()" >
<input type="button" value="getCookie" onclick="getCookie()" >
  <script>
  function setCookie()
  {
                    document.cookie="username=Duke  Martin;max-
age=" + (60 * 60 * 24 * 365) + ";"
  }
  function getCookie()
  {
    if(document.cookie.length!=0)
    {
      var array=document.cookie.split("=");
    alert("Name="+array[0]+" "+"Value="+array[1]);
    }
    else
    {
    alert("Cookie not available");
    }
  }
  </script>
```

```
</body>

</html>
```

Cookie path attribute

If a cookie is created for a webpage, by default, it is valid only for the current directory and sub-directory. JavaScript provides a path attribute to expand the scope of cookie up to all the pages of a website.

Cookie path attribute Example

Let's understand the path attribute with the help of an example.

Here, if we create a cookie for webpage2.html, it is valid only for itself and its sub-directory (i.e., webpage3.html). It is not valid for webpage1.html file.

In this example, we use path attribute to enhance the visibility of cookies up to all the pages. Here, you all just need to do is to maintain the above directory structure and put the below program in all three web pages. Now, the cookie is valid for each web page.

```
<!DOCTYPE html >

<html>

<head>

</head>

<body>
```

```
<input type="button" value="setCookie" onclick="seCookie()" >

<input type="button" value="getCookie" onclick="getCookie()" >

 <script>

function setCookie()

{

                document.cookie="username=Duke Martin;max-
age=" + (60 * 60 * 24 * 365) + ";path=/;"

}

function getCookie()

{

   if(document.cookie.length!=0)

   {

      var array=document.cookie.split("=");

   alert("Name="+array[0]+" "+"Value="+array[1]);

   }

   else

   {

   alert("Cookie not available");

   }

}

 </script>

</body>
```

```
</html>
```

Cookie domain attribute

A JavaScript domain attribute specifies the domain for which the cookie is valid. Let's suppose if we provide any domain name to the attribute such like:

```
domain=example.com
```

Here, the cookie is valid for the given domain and all its sub-domains.

However, if we provide any sub-domain to the attribute such like:

```
omain=training.example.com
```

Here, the cookie is valid only for the given sub-domain. So, it's a better approach to provide domain name instead of sub-domain.

Cookie with multiple Name-Value pairs

In JavaScript, a cookie can contain only a single name-value pair. However, to store more than one name-value pair, we can use the following approach: -

- Serialize the custom object in a JSON string, parse it and then store in a cookie.

- For each name-value pair, use a separate cookie.

Examples to Store Name-Value pair in a Cookie

Example 1

Let's see an example to check whether a cookie contains more than one name-value pair.

```
<!DOCTYPE html >
<html>
<head>
</head>
<body>
   Name:  <input  type="text" id="name" ><br>
   Email:  <input  type="email" id="email" ><br>
   Course:  <input  type="text" id="course" ><br>
<input  type="button" value="Set Cookie" onclick="setCookie()" >
<input  type="button" value="Get Cookie" onclick="getCookie()" >
<script>
```

```
    function setCookie()

    {

//Declaring 3 key-value pairs

        var info="Name="+ document.getElementById("name").value+
entById("course").value;

//Providing all 3 key-value pairs to a single cookie

        document.cookie=info;

    }

    function getCookie()

    {

        if(document.cookie.length!=0)

        {

       //Invoking key-value pair stored in a cookie

        alert(document.cookie);

        }

        else

        {

        alert("Cookie not available")

        }

    }
```

</script>

</body>

```
</html>
```

Example 2

Let's see an example to store different name-value pairs in a cookie using JSON.

```
<!DOCTYPE html >
<html>
<head>
</head>
<body>
   Name:  <input  type="text" id="name" ><br>
   Email:  <input  type="email" id="email" ><br>
   Course:  <input  type="text" id="course" ><br>
<input  type="button" value="Set Cookie" onclick="setCookie()" >
<input  type="button" value="Get Cookie" onclick="getCookie()" >
<script>
   function setCookie()
{
   var obj = {};//Creating custom object
```

```
 obj.name = document.getElementById("name").value;

obj.email = document.getElementById("email").value;

 obj.course = document.getElementById("course").value;

            //Converting JavaScript object to JSON string

            var jsonString = JSON.stringify(obj);

               document.cookie = jsonString;

            }

                  function getCookie()

            {

               if( document.cookie.length!=0)

               {

            //Parsing JSON string to JSON object

               var obj = JSON.parse(document.cookie);



                  alert("Name="+obj.name+" "+"Email="+obj.email+" "+"Course="+o
            bj.course);

               }

               else

               {

                  alert("Cookie not available");

               }
```

```
}

    </script>

</body>

</html>
```

Example 3

Let's see an example to store each name-value pair in a different cookie.

```
<!DOCTYPE html >

<html>
<head>
</head>
<body>
   Name:  <input  type="text" id="name" ><br>
   Email:  <input  type="email" id="email" ><br>
   Course:  <input  type="text" id="course" ><br>
<input  type="button" value="Set Cookie" onclick="setCookie()" >
<input  type="button" value="Get Cookie" onclick="getCookie()" >
<script>
 function setCookie()
{
   document.cookie = "name=" + document.getElementById("name").
   document.cookie = "email=" + document.getElementById("email").
```

```
    document.cookie = "course=" + document.getElementById("course"
}
function getCookie()
{
    if (document.cookie.length != 0)
    {
        alert("Name="+document.getElementById("name").value+" Ema
ntById("course").value);
    }
    else
    {
       alert("Cookie not available");
    }
}
  </script>
</body>
</html>
```

Deleting a Cookie in JavaScript

In the previous section, we learned the different ways to set and update a cookie in JavaScript. Apart from that, JavaScript also allows us to delete a cookie. Here, we see all the possible ways to delete a cookie.

Different ways to delete a Cookie

These are the following ways to delete a cookie:

- A cookie can be deleted by using expire attribute.
- A cookie can also be deleted by using max-age attribute.
- We can delete a cookie explicitly, by using a web browser.

Examples to delete a Cookie

Example 1

In this example, we use expire attribute to delete a cookie by providing expiry date (i.e. any past date) to it.

```
<!DOCTYPE html >

<html>

<head>

</head>

<body>

<input  type="button" value="Set Cookie" onclick="setCookie()" >

<input  type="button" value="Get Cookie" onclick="getCookie()" >

<script>

function setCookie()

{

   document.cookie="name=Martin Roy; expires=Sun, 20 Aug 2000 12:0
0:00 UTC";

}
```

```
function getCookie()   {
   if(document.cookie.length!=0)
   {
   alert(document.cookie);
   }
   else
   {
      alert("Cookie not avaliable");
   }
}
</script>
</body>
</html>
```

Example 2

In this example, we use **max-age** attribute to delete a cookie by providing zero or negative number (that represents seconds) to it.

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

```
<head>
</head>
<body>
<input type="button" value="Set Cookie" onclick="setCookie()" >
<input type="button" value="Get Cookie" onclick="getCookie()" >

<script>
function setCookie()
{
   document.cookie="name=Martin Roy;max-age=0";
}
function getCookie()
{
   if(document.cookie.length!=0)
   {
   alert(document.cookie);
   }
   else
   {
      alert("Cookie not avaliable");
   }
```

```
}

</script>

</body>

</html>
```

Example 3

Let's see an example to set, get and delete multiple cookies.

```
<!DOCTYPE html >

<html>

<head>

</head>

<body>

<input  type="button" value="Set Cookie1" onclick="setCookie1()"
>

<input  type="button" value="Get Cookie1" onclick="getCookie1()"
>

<input
 type="button" value="Delete Cookie1" onclick="deleteCookie1()" >


<br>

<input  type="button" value="Set Cookie2" onclick="setCookie2()"
>
```

```
<input  type="button" value="Get Cookie2" onclick="getCookie2()"
>

<input
 type="button" value="Delete Cookie2" onclick="deleteCookie2()" >


<br>

<input
 type="button" value="Display all cookies" onclick="displayCookie()"
>

<script>

function setCookie1()

{

   document.cookie="name=Bernice Johnson";

    cookie1=  document.cookie;

}

function setCookie2()

{

   document.cookie="name=Othniel Williams";

    cookie2=  document.cookie;

}

function getCookie1()

{

   if(cookie1.length!=0)

   {
```

```
    alert(cookie1);
    }
    else
    {
        alert("Cookie not available");
    }
}
function getCookie2()
{
    if(cookie2.length!=0)
    {
    alert(cookie2);
    }
    else
    {
        alert("Cookie not available");
    }
}
function deleteCookie1()
{
    document.cookie=cookie1+";max-age=0";
```

```
    cookie1=document.cookie;
    alert("Cookie1 is deleted");
}

function deleteCookie2()
{
    document.cookie=cookie2+";max-age=0";
    cookie2=document.cookie;
   alert("Cookie2 is deleted");
}
function displayCookie()
{
if(cookie1!=0&&cookie2!=0)
{
    alert(cookie1+" "+cookie2);
}
else if(cookie1!=0)
{
    alert(cookie1);
}
else if(cookie2!=0)
```

```
{
   alert(cookie2);
}
else{
   alert("Cookie not available");
}

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

Example 4

Let's see an example to delete a cookie explicitly.

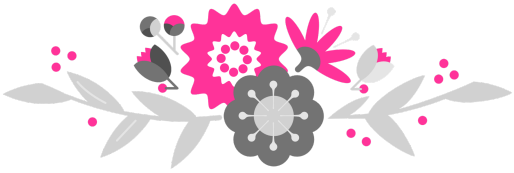
```
<!DOCTYPE html >
<html>
<head>
</head>
```

```
<body>


<input  type="button" value="Set Cookie" onclick="setCookie()" >

<input  type="button" value="Get Cookie" onclick="getCookie()" >

<script>

function setCookie()

{

   document.cookie="name=Martin Roy";

}

function getCookie()

{

   if(document.cookie.length!=0)

   {

   alert(document.cookie);

   }

   else

   {

      alert("Cookie not avaliable");

   }

}

</script>
```

```
</body>

</html>
```

Chapter Sixteen

Integrating JavaScript with Google Apps Script



A rapid application development framework called Google Apps Script makes it simple and quick to construct business apps that work with Google Workspace. Developers may use built-in libraries for their preferred Google Workspace apps, such as Gmail, Calendar, Drive, and more, while writing code in the contemporary JavaScript language. Your scripts run on Google's servers, and they provide you with a code editor directly in your browser, so there's nothing to install.

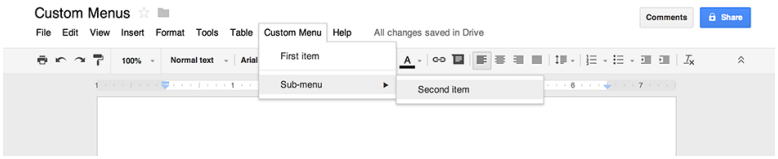
What can Apps Script do?

Apps Script has several uses.
You can, among other things:

- Customize Google Sheets, Forms, and menus, dialog boxes, and sidebars.
- Create bespoke macros and functions for Google Sheets.
- Release web applications, either separate or integrated with Google Sites.
- Engage with AdSense, Analytics, Calendar, Drive, Gmail, and Maps, among other Google services.
- Create add-ons and submit them to the marketplace for Google Workspace.

Custom Menus in Google Workspace

By including user-interface components that, when clicked, carry out an Apps Script action, scripts can expand the functionality of several Google products. In Google Docs, Sheets, Slides, or Forms, the most typical example is executing a script from a custom menu item. However, in Google Sheets, script functionalities may also be activated by clicking on drawings and photos.



Google Docs, Sheets, Slides, and Forms may have additional menus added by Apps Script, each of which is linked to a scripted function. (In Google Forms, users who enter the form to reply are not able to see the custom menus; only editors who open the form to make changes may see them.)

A script that is connected to a document, spreadsheet, or form can only generate a menu in that manner. Write the menu code within a onOpen() method so that it appears when the user opens a file.

The example below demonstrates adding a menu with one item, a visual divider, and a sub-menu with an additional item. (Note that sub-menus are not supported in Google Sheets and that you must use the addMenu() syntax instead, unless you're using the latest version.) An alert dialog is displayed by the relevant function when the user selects either menu option. See the guide to dialogs and sidebars for further details on the many kinds of dialogs that are available for opening.

```
function onOpen() {
  var ui = SpreadsheetApp.getUi();
  // Or DocumentApp, SlidesApp or FormApp.
  ui.createMenu('Custom Menu')
      .addItem('First item', 'menuItem1')
      .addSeparator()
      .addSubMenu(ui.createMenu('Sub-menu')
          .addItem('Second item', 'menuItem2'))
      .addToUi();
}
function menuItem1() {
  SpreadsheetApp.getUi() // Or DocumentApp, SlidesApp or
FormApp.
      .alert('You clicked the first menu item!');
}
```

```
function menuItem2() {

  SpreadsheetApp.getUi() // Or DocumentApp, SlidesApp or
FormApp.

    .alert('You clicked the second menu item!');

}
```

A spreadsheet, presentation, document, or form may only have one menu with a certain name. A menu with the same name that is added by the same script or another script replaces the previous one. Although you may build your onOpen() code to bypass the menu in the future if a certain property is specified, menus cannot be deleted while the file is open.

Clickable images and drawings in Google Sheets

If the script is linked to the spreadsheet, you may also use an Apps Script function to an image or drawing in Google Sheets. How to set this up is shown in the example below.

- To build a script that is connected to the spreadsheet, choose Extensions > Apps Script from the Google Sheets menu.

- In the script editor, remove any existing code and insert the following code.

```
function showMessageBox() {

  Browser.msgBox('You clicked it!');

}
```

- Go back to Sheets and choose Insert > Image or Insert > Drawing to add an image or drawing.

- Click the picture or drawing once it has been inserted. There's a little menu choice that drops down in the upper right corner. Select Assign script by clicking on it.

- Enter the name of the Apps Script function you wish to perform, showMessageBox in this example, without parenthesis, in the dialog box that pops up. Press OK.

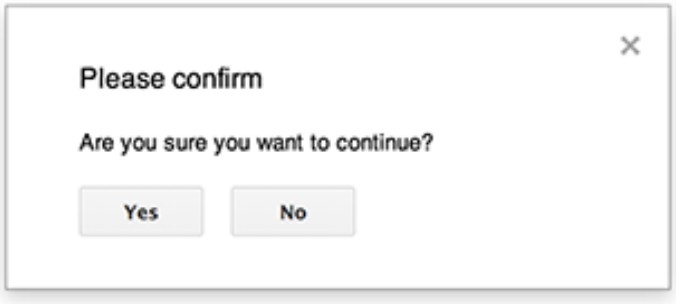
- Re-click the picture or illustration. At this point, the function runs.

Dialogs and Sidebars in Google Workspace Documents.

Custom HTML service pages can be shown in dialogs and sidebars, as well as pre-built alerts and prompts, using scripts that are linked to Google Docs, Sheets, or Forms. These components are usually accessed by menu items. (Note that user-interface components in Google Forms are not visible to users opening the form to answer; rather, they are only available to an editor who opens the form to amend it.)

Alert dialogs

An alert is a pre-made dialog box that appears in the editor of Google Sheets, Docs, Slides, or Forms. It shows a message and a "OK" button; alternate buttons and a title are not required. It is comparable to using client-side JavaScript in a web browser to call window.alert().



When the alert dialog is active, the server-side script is suspended. When the user exits the dialog, the script continues, but JDBC connections are lost during the suspension.

Google Docs, Forms, Slides, and Sheets all employ the Ui.alert() method, which has three variations, as the example below illustrates. You can use a value from the Ui.ButtonSet enum as the buttons parameter to override the default "OK" button. The Ui.Button enum and the return value for alert() should be compared to determine which button the user pressed.

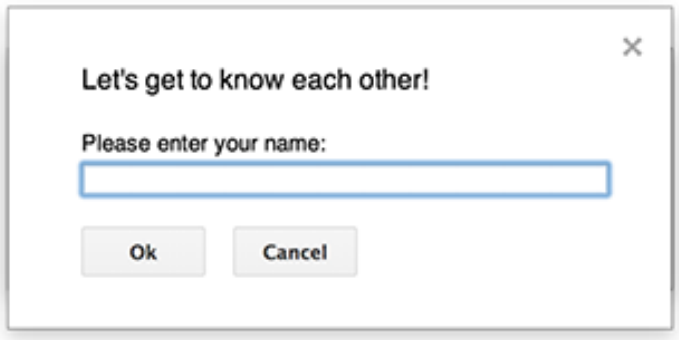
```
function onOpen() {

  SpreadsheetApp.getUi() // Or DocumentApp or SlidesApp or FormApp.

      .createMenu('Custom Menu')

      .addItem('Show alert', 'showAlert')

      .addToUi();

}
```

```
function showAlert() {
  var ui = SpreadsheetApp.getUi(); // Same variations.

  var result = ui.alert(
     'Please confirm',
     'Are you sure you want to continue?',
      ui.ButtonSet.YES_NO);

  // Process the user's response.
  if (result == ui.Button.YES) {
    // User clicked "Yes".
    ui.alert('Confirmation received.');
  } else {
    // User clicked "No" or X in the title bar.
    ui.alert('Permission denied.');
```

```
  }

}
```

Prompt dialogs

A pre-made dialog box known as a prompt appears inside the editor of Google Documents, Sheets, Slides, or Forms. It shows a text-input area, a message, and a "OK" button; other buttons and a title are optional. It is comparable to using client-side JavaScript in a web browser to use window.prompt().



Requests that the server-side script be suspended while the dialog is active. When the user exits the dialog, the script continues, but JDBC connections are lost during the suspension.

Google Docs, Forms, Slides, and Sheets all employ the Ui.prompt() function, which has three variations, as the example below illustrates. You can use a value from the Ui.ButtonSet enum as the buttons parameter to override the default "OK" button. After capturing the prompt() return value, call PromptResponse to

assess the user's answer.Use getResponseText() to get the user's input, then compare PromptResponse's response value.the Ui.Button enum using getSelectedButton().

```
function onOpen() {

  SpreadsheetApp.getUi() // Or DocumentApp or SlidesApp or FormApp.

    .createMenu('Custom Menu')

    .addItem('Show prompt', 'showPrompt')

    .addToUi();

}



function showPrompt() {

  var ui = SpreadsheetApp.getUi(); // Same variations.



  var result = ui.prompt(

    'Let\'s get to know each other!',

    'Please enter your name:',
```

```
    ui.ButtonSet.OK_CANCEL);


// Process the user's response.
var button = result.getSelectedButton();
var text = result.getResponseText();
if (button == ui.Button.OK) {
  // User clicked "OK".
  ui.alert('Your name is ' + text + '.');
} else if (button == ui.Button.CANCEL) {
  // User clicked "Cancel".
  ui.alert('I didn\'t get your name.');
} else if (button == ui.Button.CLOSE) {
  // User clicked X in the title bar.
  ui.alert('You closed the dialog.');
}
```

```
}
```

Custom dialogs

An HTML service user interface may be shown inside a Google Docs, Sheets, Slides, or Forms editor using a custom dialog.

When a custom dialog is active, the server-side script is not suspended. The google.script API for HTML-service interfaces allows the client-side component to call the server-side script asynchronously.

By using google.script.host.close() on the client side of an HTML-service interface, the dialog may shut on its own. Only the user or the dialog itself has the ability to close it; other interfaces cannot.

Google Docs, Forms, Slides, and Sheets all utilize the Ui.showModalDialog() function to open the dialog, as seen in the sample below.

Code.gs

	Page.html
function onOpen() { *SpreadsheetApp.getUi() // Or DocumentApp or SlidesApp or FormApp.* *.createMenu('Custom Menu')* *.addItem('Show dialog', 'showDialog')* *.addToUi();* *}* *function showDialog() {* *var html = HtmlService.createHtmlOutputFromFile('Page')* *.setWidth(400)* *.setHeight(300);* *SpreadsheetApp.getUi() // Or DocumentApp or SlidesApp or FormApp.* *.showModalDialog(html, 'My custom dialog');* *}*	Hello, world! <input type="button" value="Close" onclick="google.script.host.close()" />

Custom sidebars

Within an editor for Google Docs, Forms, Slides, and Sheets, a sidebar can show an HTML service user interface.

The server-side script is not suspended by sidebars when the dialog is active. The google.script API for HTML-service interfaces allows the client-side component to call the server-side script asynchronously.

Calling google.script.host.close() on the client side of an HTML-service interface allows the sidebar to shut itself. Only the user or the sidebar itself has the ability to close it; other interfaces cannot.

The method Ui.showSidebar() is used by Google Docs, Forms, Slides, and Sheets to open the sidebar, as demonstrated in the sample below.

Code.gs	***Page.html***
function onOpen() { *SpreadsheetApp.getUi() // Or DocumentApp or SlidesApp or FormApp.* *.createMenu('Custom Menu')* *.addItem('Show dialog',*	*Hello, world!* *<input type="button" value="Close" onclick="google.script.host.close()" />*

```
'showDialog')
          .addToUi();
    }
    function showDialog() {
      var html =
HtmlService.createHtmlOutputFromFile('Page')
          .setWidth(400)
          .setHeight(300);
      SpreadsheetApp.getUi() // Or
DocumentApp or SlidesApp or FormApp.
          .showModalDialog(html, 'My
custom dialog');
    }
```

File-open dialogs

A "file-open" dialog for data housed on Google servers, Google Picker provides access to Google Drive, Google Image Search, Google Video Search, and many services.

Picker's client-side JavaScript API may be utilized in HTML services to construct a custom dialog that allows users to upload new files or pick existing ones, as seen in the example below. The selection is then passed back to your script for additional usage.

To obtain an API key and enable Picker, go to following guidelines:

- Verify that your script project is using a standard GCP project.
- Enable the "Google Picker API" in your Google Cloud project.
- While your Google Cloud project is still open, select

APIs & Services

, then click

Credentials

.

- Click

  Create credentials > API key

  .
  This action creates the key, but you should edit the key to add both application restrictions and an API restriction to the key.

- In the API key dialog, click

  Close

  .

- Next to the API key you created, click More >

  Edit API key

  .

- Under

  Application restrictions

  , complete the following steps:

  a)

  Select

HTTP referrers (web sites)

.

b)

Under

Website restrictions

, click

Add an item

.

c)

Click

Referrer

and enter *.google.com.

d)

Add another item and enter *.googleusercontent.com as the referrer.

e)

Click

Done

.

- Under

API restrictions

, complete the following steps:

a)

Select

Restrict key

.

b)

In the

Select APIs

section, select

Google Picker API

and click

OK

.

Note:

The Google Picker API does not appear unless you have enabled it because the list only shows APIs that have been enabled for the Cloud project.

- Under **API key**, click Copy to clipboard .

- At the bottom, click **Save**.

NB: The following example invokes ScriptApp.getOAuthToken() in order to provide Picker with the user's OAuth 2.0 access token. This method avoids Picker from having to display its own authorization dialog, but it requires Apps Script to have the OAuth scope that Picker requires. If not, as this "hello world" example illustrates, your Picker code will need to define its own OAuth scopes.

Code.gs

```
/**
 * Creates a custom menu in Google Sheets when the spreadsheet opens.
 */
function onOpen() {
  try {
    SpreadsheetApp.getUi().createMenu('Picker')
        .addItem('Start', 'showPicker')
        .addToUi();
  } catch (e) {
    // TODO (Developer) - Handle exception
    console.log('Failed with error: %s', e.error);
```

```
  }

}


/**

 * Displays an HTML-service dialog in Google Sheets that contains client-side

 * JavaScript code for the Google Picker API.

 */

function showPicker() {

  try {

    const html = HtmlService.createHtmlOutputFromFile('dialog.html')

        .setWidth(600)

        .setHeight(425)

        .setSandboxMode(HtmlService.SandboxMode.IFRAME);

    SpreadsheetApp.getUi().showModalDialog(html, 'Select a file');

  } catch (e) {
```

```
      // TODO (Developer) - Handle exception
      console.log('Failed with error: %s', e.error);
    }
  }

  /**
   * Gets the user's OAuth 2.0 access token so that it can be passed to Picker.
   * This technique keeps Picker from needing to show its own authorization
   * dialog, but is only possible if the OAuth scope that Picker needs is
   * available in Apps Script.
  In this case, the function includes an unused call
   * to a DriveApp method to ensure that Apps Script requests access to all files
   * in the user's Drive.
   *
   * @return {string} The user's OAuth 2.0 access token.
   */
```

```
function getOAuthToken() {
  try {
    DriveApp.getRootFolder();
    return ScriptApp.getOAuthToken();
  } catch (e) {
    // TODO (Developer) - Handle exception
    console.log('Failed with error: %s', e.error);
  }
}
```

Dialog.html

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

```
<head>

  <link rel="stylesheet"
href="https://ssl.gstatic.com/docs/script/css/add-ons.css">

  <script>

    // IMPORTANT: Replace the value for DEVELOPER_KEY with
the API key obtained

    // from the Google Developers Console.

    var DEVELOPER_KEY = 'ABC123 ...
';

    var DIALOG_DIMENSIONS = {width: 600, height: 425};

    var pickerApiLoaded = false;



    /**

     * Loads the Google Picker API.

     */

    function onApiLoad() {

      gapi.load('picker', {'callback': function() {

        pickerApiLoaded = true;
```

```
    }});

    }

    /**
     * Gets the user's OAuth 2.0 access token from the server-side script so that
     * it can be passed to Picker. This technique keeps Picker from needing to
     * show its own authorization dialog, but is only possible if the OAuth scope
     * that Picker needs is available in Apps Script. Otherwise, your Picker code
     * will need to declare its own OAuth scopes.
     */
    function getOAuthToken() {
      google.script.run.withSuccessHandler(createPicker)
          .withFailureHandler(showError).getOAuthToken();
    }
```

```
/**
 * Creates a Picker that can access the user's spreadsheets.
 This function
 * uses advanced options to hide the Picker's left navigation panel and
 * default title bar.
 *
 * @param {string} token An OAuth 2.0 access token that lets Picker access the
 *     file type specified in the addView call.
 */
function createPicker(token) {
  if (pickerApiLoaded && token) {
    var picker = new google.picker.PickerBuilder()
        // Instruct Picker to display only spreadsheets in Drive. For other
        // views, see https://developers.google.com/picker/docs/#otherviews
        .addView(google.picker.ViewId.SPREADSHEETS)
```

```
                // Hide the navigation panel so that Picker fills more of
the dialog.
                .enableFeature(google.picker.Feature.NAV_HIDDEN)
                // Hide the title bar since an Apps Script dialog already
has a title.
                .hideTitleBar()
                .setOAuthToken(token)
                .setDeveloperKey(DEVELOPER_KEY)
                .setCallback(pickerCallback)
                .setOrigin(google.script.host.origin)
                // Instruct Picker to fill the dialog, minus 2 pixels for
the border.
                .setSize(DIALOG_DIMENSIONS.width - 2,
                    DIALOG_DIMENSIONS.height - 2)
                .build();
            picker.setVisible(true);
          } else {
            showError('Unable to load the file picker.');
```

```
    }

  }

  /**
   * A callback function that extracts the chosen document's metadata from the
   * response object.
   * For details on the response object, see
   * https://developers.google.com/picker/docs/result
   *
   * @param {object} data The response object.
   */
  function pickerCallback(data) {
    var action = data[google.picker.Response.ACTION];
    if (action == google.picker.Action.PICKED) {
      var doc = data[google.picker.Response.DOCUMENTS][0];
      var id = doc[google.picker.Document.ID];
      var url = doc[google.picker.Document.URL];
```

```
      var title = doc[google.picker.Document.NAME];
      document.getElementById('result').innerHTML =
          '<b>You chose:</b><br>Name: <a href="' + url +
'">' + title +
          '</a><br>ID: ' + id;
    } else if (action == google.picker.Action.CANCEL) {
      document.getElementById('result').innerHTML = 'Picker
canceled.';
    }
  }

  /**
   * Displays an error message within the #result element.
   *
   * @param {string} message The error message to display.
   */
  function showError(message) {
    document.getElementById('result').innerHTML = 'Error: '
```

```
+ message;

            }

          </script>

        </head>

        <body>

          <div>

            <button onclick="getOAuthToken()">Select a
file</button>

            <p id="result"></p>

          </div>

          <script src="https://apis.google.com/js/api.js?
onload=onApiLoad"></script>

        </body>

        </html>
```

Custom Functions in Google Sheets

Numerous built-in functions, like AVERAGE, SUM, and VLOOKUP, are available in Google Sheets. If they don't meet your needs, you may create custom functions using Google Apps Script, such as converting meters to miles or retrieving real-time material from the Internet, and use them in Google Sheets exactly like a built-in function.

Conventional JavaScript is used to construct custom functions. Previous Chapters in this book has explained in details all you need to know, on JavaScript.

The example bellow illustrates a straightforward custom function called DOUBLE that multiplies a value input by two (2):

```
/**
 * Multiplies an input value by 2.
 * @param {number} input The number to double.
 * @return The input multiplied by 2.
 * @customfunction
*/
function DOUBLE(input) {
  return input * 2;
```

```
}
```

Developing a custom function

To write a custom function:

- In Google Sheets, create or open a spreadsheet.
- Choose Extensions > Apps Script from the menu.
- Open the script editor and remove any code.
  Just copy and paste the code for the aforementioned DOUBLE function into the script editor.

- Click Save at the top.

obtaining a personalized feature via the Google Workspace Marketplace

A number of customized features are available as Google Sheets add-ons through the Google Workspace Marketplace. To utilize or investigate these extras:

- In Google Sheets, create or open a spreadsheet.

- Click Add-ons > Get Add-ons at the top.

- Click the search box located in the upper right corner of the Google Workspace Marketplace after it has opened.

- Enter "custom function" after typing it.

- Click Install to install any custom function add-ons you find appealing.

- You may see a dialog box informing you that the add-on needs permission.
  If so, carefully read the message before clicking "Allow."

- The spreadsheet starts to display the add-on.

- Open a separate spreadsheet and select Add-ons > Manage Add-ons at the top to utilize the add-on.

- Click Options more_vert > Use in this document after selecting the add-on you wish to use.

Using a custom function

Once a custom function has been developed or installed from the Google Workspace Marketplace, using it is just as simple as using one that is pre-installed:

- To utilize the function, click the cell in question.

- Press Enter after typing the equals symbol (=), the function name, and any input value (for example, =DOUBLE(A1)).

- The cell will show Loading... for a brief while before returning the outcome.

Guidelines for custom functions

There are several rules to follow before creating your own custom function.

Naming

Apart from the customary practices for designating JavaScript functions, take note of the following:

- A custom function's name must be unique from the names of built-in functions, such as SUM().

- A custom function's name cannot conclude in an underscore (_), as in Apps Script, this indicates a private function.

- The syntax function myFunction(), not var myFunction = new Function(), is required when declaring the name of a custom function.

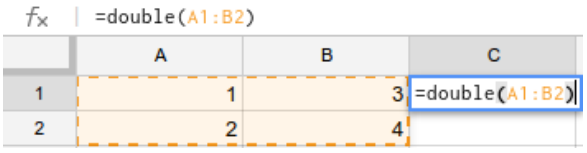
- Although the names of spreadsheet functions are often capitalized, capitalization is not important.

Arguments

A custom function can accept parameters as input values, just like a built-in function can:

- The cell value will be the argument if you call your function with a reference to a single cell as the parameter (e.g., =DOUBLE(A1)).

- The parameter for your function will be a two-dimensional array containing the values of the cells if you call it with a reference to a range of cells as the argument (e.g., =DOUBLE(A1:B10)). For instance, in the image below, Apps Script interprets the parameters in =DOUBLE(A1:B2) as double([[1,3],[2,4]]). Keep in mind that in order to accept an array as input, the DOUBLE example code from above would need to be updated.



- Arguments to custom functions must be deterministic. That is, you cannot pass in built-in spreadsheet functions like NOW() or RAND() as parameters to a custom function since they produce a different answer each time they compute. A custom function will always show Loading... if it attempts to return a value based on one of these volatile built-in functions.

Return values

Each custom function has to give back a value to the display in a way that:

- The cell from where the custom function was called displays the value if the function returns one.

- When a custom function produces a two-dimensional array of values, as long as the neighboring cells are vacant, the values spill over into them. The custom method will instead produce an exception if doing so will cause the array to overwrite the contents of any existing cells.

- Cells that it returns a value to are the only ones that a custom function can impact. Put differently, only the cells it is called from and their neighboring cells may be edited by a custom function. It cannot modify any other cells. Instead, utilize a custom menu to launch a function in order to change any cell.

- A call to a custom function needs to return in 30 seconds or less. The cell will show the following error if it doesn't: The custom function is executing with an internal error.

Data types

Depending on the type of data, Google Sheets saves it in several forms. Apps Script handles these values as the proper JavaScript data type when they are used in custom methods. These are the most frequently misunderstood areas:

- Date objects in Apps Script are created from times and dates in Sheets. The custom function will have to adjust if there is a time zone difference between the spreadsheet and the script, which is an uncommon issue.

- Although handling duration values in Sheets might be challenging, they also become Date objects.

- In Apps Script, percentage values in Sheets are converted to decimal numbers. For instance, in Apps Script, a cell having a value of 10% becomes 0.1.

Autocomplete

Similar to built-in functions, custom functions in Google Sheets may also use autocomplete. You will get a list of pre-built and custom functions that match the function name you write in a cell.

If a custom function's script has a JsDoc @customfunction tag, like in the DOUBLE() example below, it will show up in this list.

```
/**
```

```
 * Multiplies the input value by 2.
 *
 * @param {number} input The value to multiply.
 * @return The input multiplied by 2.
 * @customfunction
 */
function DOUBLE(input) {
  return input * 2;
}
```

Using Google Apps Script services

More complicated operations can be carried out using custom functions by calling certain Google Apps Script services. For instance, to translate a Spanish sentence from English, a custom function can make a call to the Language service.

Custom functions never request permission from users to access personal data, in contrast to the majority of other types of Apps Scripts. As a result, they are

limited to calling the following services, which do not possess access to personal data:

Supported services	***Notes***
Cache	Works, but not particularly useful in custom functions
HTML	Can generate HTML, but cannot display it (rarely useful)
JDBC	
Language	
Lock	Works, but not particularly useful in custom functions
Maps	Can calculate directions, but not display maps
	getUserProperties() only gets the properties of the spreadsheet owner.

Properties	Spreadsheet editors can't set user properties in a custom function.
	Read only (can use most get*() methods, but not set*()).
Spreadsheet	Cannot open other spreadsheets (SpreadsheetApp.openById() or SpreadsheetApp.openByUrl()).
URL Fetch	
Utilities	
XML	

Should your customized function provide an error message The service requires user authorization and hence cannot be utilized in a custom function; you do not have permission to use it.

Rather than building a new function, construct a custom menu that calls an Apps Script function to use a service not on the above list. When a function is called from a menu, it can utilize all Apps Script services and will prompt the user for permission if needed.

Sharing

Initially, custom functions are restricted to the spreadsheet in which they were developed. This implies that unless you employ one of the following techniques, a custom function created in one spreadsheet cannot be utilized in another spreadsheet:

- To access the script editor, click Extensions > Apps Script. Once the script editor is open, copy and paste the script content from the first spreadsheet into the script editor of another spreadsheet.

- Click File > Make a copy to create a copy of the spreadsheet with the custom function in it. Scripts that are connected to a spreadsheet are copied along with it. The script may be copied by anybody with access to the spreadsheet. (Partners with view-only access are unable to access the script editor within the spreadsheet. But when they copy something, they get ownership of the copy and are able to view the script.)

- Release the script as an Add-on for Google Sheets Editor.

Optimization

Every time a spreadsheet with a custom function is utilized, Google Sheets calls the Apps Script server once again. This procedure can be very slow if your spreadsheet has thousands (or hundreds!) of calls to custom functions.

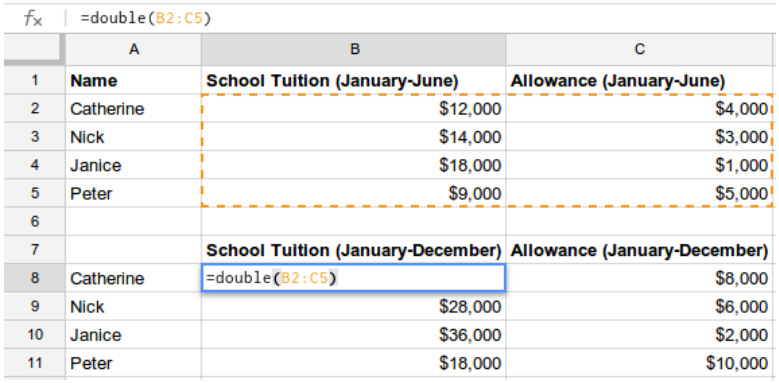
Therefore, think about changing a custom function such that it takes a range as input in the form of a two-dimensional array and returns a two-dimensional array that can overflow into the proper cells if you want to use it again on a huge range of data.

For instance, the above-described DOUBLE() function may be changed to take a single cell or a range of cells like this:

```
/**
 * Multiplies the input value by 2.
 *
 * @param {number|Array<Array<number>>} input The
value or range of cells
 *     to multiply.
 * @return The input multiplied by 2.
 * @customfunction
 */
function DOUBLE(input) {
```

```
  return Array.isArray(input) ?
    input.map(row => row.map(cell => cell * 2)) :
    input * 2;
}
```

The aforementioned technique calls DOUBLE recursively on each value in the two-dimensional array of cells by using the map function of JavaScript's Array object. The findings are returned as a two-dimensional array. As seen in the picture below, you may do this by using DOUBLE only once and have it compute for a huge number of cells at once. (Alternatively, you could use nested if statements in place of the map call to get the same result.)



fx =double(B2:C5)

	A	B	C
1	**Name**	**School Tuition (January-June)**	**Allowance (January-June)**
2	Catherine	$12,000	$4,000
3	Nick	$14,000	$3,000
4	Janice	$18,000	$1,000
5	Peter	$9,000	$5,000
6			
7		**School Tuition (January-December)**	**Allowance (January-December)**
8	Catherine	=double(B2:C5)	$8,000
9	Nick	$28,000	$6,000
10	Janice	$36,000	$2,000
11	Peter	$18,000	$10,000

Similar to that, the custom code below employs a two-dimensional array to efficiently retrieve live material from the Internet and show two columns of results with only one function call. The process would take a much longer if every cell needed to call a different function, as the Apps Script server would need to download and process the XML feed each time.

```
/**
```

```
 * Show the title and date for the first page of posts on the
 * Developer blog.
 *
 * @return Two columns of data representing posts on the
 *     Developer blog.
 * @customfunction
 */
function getBlogPosts() {
  var array = [];
  var url = 'https://gsuite-
developers.googleblog.com/atom.xml';
  var xml = UrlFetchApp.fetch(url).getContentText();
  var document = XmlService.parse(xml);
  var root = document.getRootElement();
  var atom =
XmlService.getNamespace('http://www.w3.org/2005/Atom');
```

```
    var entries = document.getRootElement().getChildren('entry',
atom);

    for (var i = 0; i < entries.length; i++) {

      var title = entries[i].getChild('title', atom).getText();

      var date = entries[i].getChild('published', atom).getValue();

      array.push([title, date]);

    }

    return array;

  }
```

Almost every custom function that is often used in a spreadsheet can benefit from these strategies, while the specifics of implementation will depend on how the function behaves.

Google Sheets Macros

A Google Sheets macro is a set of recorded operations. Once recorded, you may use a menu item or shortcut key to subsequently repeat those activities by activating a macro.

You may record macros in Google Sheets that replicate a defined sequence of user interface interactions. After recording a macro, you may associate it with a Ctrl+Alt+Shift+Number keyboard shortcut.

That shortcut allows you to easily run the same macro steps again, usually on new data or in a different location. Additionally, the Google Sheets Extensions > Macros menu allows you to initiate the macro.

Google Sheets automatically generates an Apps Script function (the macro function) that duplicates the macro steps when you record a macro. The macro function is introduced in a file called macros.gs to an Apps Script project that is tied to the sheet. The macro function is inserted to the project file if one already exists, if it is bound to the sheet with that name. Moreover, Google Sheets immediately modifies the script project manifest, logging the macro's name and keyboard shortcut.

You can edit recorded macros directly in the Apps Script editor because they are all defined fully within Apps Script. With Apps Script, you can even create macros from start or convert functions that you've already written into macros.

Creating macros in Apps Script

Apps Script functions are capable of being used as macro functions. Importing an existing function from the Google Sheets editor is the simplest method to accomplish this.

As an alternative, you can follow these instructions to create macros in the Apps Script editor:

- To access the script tied to the sheet in the Apps Script editor, choose Extensions > Apps Script in the Google Sheets user interface.

- Compose the macro function. Macro functions ought to be empty—that is, they ought to return nothing.

- To construct the macro and attach it to the macro function, edit your script manifest. Give it a special name and keyboard shortcut.

- The script project should be saved. After that, the macro may be used in the sheet.

- To ensure that it performs as anticipated, test the macro function in the sheet.

Editing macros

The following steps may be used to edit macros that are associated to a sheet:

- To manage macros in the Google Sheets user interface, go to Extensions > Macros.

- Locate the macro that needs editing, then choose more_vert > Edit macro. This launches the project file containing the macro function in the Apps Script editor.

- To alter the behavior of the macro, edit the macro function.

- The script project should be saved. After that, the macro may be used in the sheet.

- To ensure that it performs as anticipated, test the macro function in the sheet.

Importing functions as macros

You may import a function as a new macro into an existing script that is connected to a sheet and then give it a keyboard shortcut. To do this, modify the manifest file and update the sheets.macros[] property with a new element.

As an alternative, use the following procedures to import a function from the Sheets UI as a macro:

- In the Google Sheets UI, select **Extensions** > **Macros** > **Import**.

- Select a function from the list presented and then click **Add function**.

- Select clear to close the dialog.

- Select **Extensions** > **Macros** > **Manage macros** .

- Locate the function you just imported in the list. Assign a unique keyboard shortcut to the macro. You can also change the macro name here; the name defaults to the name of the function.

- Click **Update** to save the macro configuration.

Manifest structure for macros

The part of a manifest that defines Google Sheets macros is displayed in the sample snippet of a manifest file that follows. The name of the macro function and its corresponding keyboard shortcut are defined in the sheets section of the manifest.

```
{
  ...
  "sheets": {
    "macros": [{
```

```
    "menuName": "QuickRowSum",
    "functionName": "calculateRowSum",
    "defaultShortcut": "Ctrl+Alt+Shift+1"
  }, {
    "menuName": "Headerfy",
    "functionName": "updateToHeaderStyle",
    "defaultShortcut": "Ctrl+Alt+Shift+2"
  }]
}
```

Best practices

The following principles should be followed when you create or manage macros in Apps Script.

- When macros are lightweight, they function better. Try to keep a macro's number of activities to a minimum.

- When repetitive processes need to be performed repeatedly with minimal or no setup, macros are the ideal choice. Instead, think about employing a custom menu item for additional actions.

- Never forget that there may only be ten macros with keyboard shortcuts on a particular sheet at any one moment, and that each macro's shortcut must be distinct. The Extensions > Macros menu is the sole way to run any new macros.

-

If you choose the entire range before executing the macro, you may apply macros that alter a single cell to a range of cells.

This implies that writing macros to do the same action over a preset range of cells is frequently superfluous.

(Google Workspace, 2024)

Things you can't do

The following are some limitations on the use of macros:

-

Utilize macros outside of script bounds:

Scripts that are attached to certain Google Sheets define macros. If a macro definition is found in a standalone script or web application, it is disregarded.

- **Set macros in the add-ons for Sheets:**

Spreading macro definitions using a Sheets add-on is not possible. Users of a Sheets add-on project disregard any macro definitions in that add-on.

- **Make macros available in script libraries**

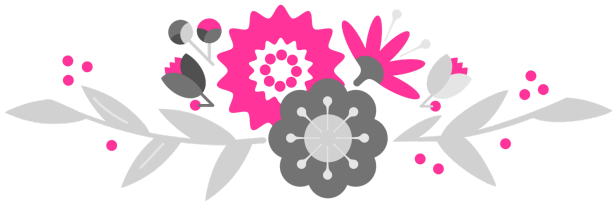
: Distribution of macro definitions using Apps Script libraries is not possible.

- **Outside of Google Sheets, use macros:**

There are no macros in Google Docs, Forms, or Slides; they are exclusive to Google Sheets.

Chapter Seventeen

Developing Web Apps in Apps Script



A program that can be accessed using a web browser is called a web application. Web applications resemble mobile apps when accessed through a mobile device's browser, but they are not the same.

A script may be made into a web application and published online if its user interface is developed. For instance, it would be ideal to offer a script that allows customers to book meetings with support team members as a web application, allowing consumers to access it straight from their browsers.

As long as they fulfill the specifications listed below, scripts that are independent or integrated with Google Workspace applications can be developed into web applications.

Requirements for web apps

If a script satisfies certain criteria, it can be released as a web application:

- A doGet(e) or doPost(e) function is present.
- The function returns either a TextOutput object from the Content service or an HTML service HtmlOutput object.

Request parameters

Apps Script executes the function doGet(e) when a user visits an app or when a program submits an HTTP GET request to the app. Rather of sending an HTTP POST request to the app, Apps Script does doPost(e). The e argument in all scenarios denotes an event parameter that may include details about any request parameters. The table below displays the event object's structure:

Fields	

e.queryString	The value of the query string portion of the URL, or null if no query string is specified name=alice&n=1&n=2
e.parameter	An object of key/value pairs that correspond to the request parameters. Only the first value is returned for parameters that have multiple values. {"name": "alice", "n": "1"}
e.parameters	An object similar to e.parameter, but with an array of values for each key {"name": ["alice"], "n": ["1", "2"]}
e.pathInfo	The URL path after /exec or /dev. For example, if the URL path ends in /exec/hello, the path info is hello.

e.contextPath	Not used, always the empty string.
e.contentLength	The length of the request body for POST requests, or -1 for GET requests 332
e.postData.length	The same as e.contentLength 332
e.postData.type	The MIME type of the POST body text/csv
e.postData.contents	The content text of the POST body

e.postData.name	Alice,21 Always the value "postData" postData

For instance, you could pass parameters such as username and age to a URL as shown below:

---

https://script.google.com/.../exec?username=jsmith&age=21

Then, you can display the parameters like so:

```
function doGet(e) {
  var params = JSON.stringify(e);
  return ContentService.createTextOutput(params).setMimeType(ContentService.MimeType.JSON);
}
```

In the above example,

doGet(e)

returns the following output:


```
{
 "queryString": "username=jsmith&age=21",
 "parameter": {
  "username": "jsmith",
  "age": "21"
 },
 "contextPath": "",
 "parameters": {
  "username": [
   "jsmith"
  ],
  "age": [
   "21"
  ]
```

```
  },
  "contentLength": -1
}
```

Source: (Google Workspace, 2024)

Deploy a script as a web app

To deploy a script as a web app, follow these steps:

1. At the top right of the script project, click **Deploy** > **New deployment**.

2. Next to "Select type," click Enable deployment types settings > **Web app**.

3. Enter the information about your web app in the fields under "Deployment configuration."

4. Click **Deploy**.

You can share the web app URL with those you would like to use your app, provided you have granted them access.

Test a web app deployment

To test your script as a web app, follow the steps below:

1. At the top right of the script project, click **Deploy > Test deployments**.

2. Next to "Select type," click Enable deployment types settings **> Web app**.

3. Under the web app URL, click **Copy**.

4. Paste the URL in your browser and test your web app.

This URL ends in /dev and can only be accessed by users who have edit access to the script.
This instance of the app always runs the most recently saved code and is only intended for testing during development.

Permissions

A web application's permissions vary based on how you choose to run it:

- Run the application in my place. In this instance, regardless of who views the web application, the script always runs as you, the script owner.

- Run the application as the user visiting the website—In this scenario, the script executes using the user's identity who is now utilizing the website. When the user grants access using this permission method, the web application displays the email address of the script owner.

Embed your web app in Google Sites or any Site of your Choice.

A web application has to be launched before it can be embedded into Google Sites. The Deployed URL from the Deploy dialog is also required.

Use these procedures to embed a web application into a Sites page:

1. Open the Sites page where you'd like to add the web app.

2. Select

Insert > Embed URL

.

3.

Paste in the web app URL and then click

ADD

.

The page preview displays the web application in a frame. Before the web app runs normally after you publish the page, visitors to your website might need to grant the web app permission. Users are prompted for consent by unauthorized online applications.

Web Apps and Browser History

A online application using Apps Script may be desired if it simulates a multi-page application or has a dynamic user interface that is controlled by URL parameters. To achieve this effectively, you may construct a state object that will serve as the user interface (UI) or page for your app, and when the user interacts with it, you can push the state into the browser history.

In order to ensure that your web application shows the right user interface (UI) as the user moves the browser's buttons back and forth, you may also listen to past events. You may allow the user to launch your app in a certain state by having your app dynamically generate its user interface (UI) based on URL parameters by querying the parameters during load time.

Two asynchronous client-side JavaScript APIs are offered by Apps Script to help develop online applications that are connected to the user's browsing history:

- Google.script.history offers techniques to enable dynamic reaction to modifications in the browser history. This entails adding states—basic objects that you may define—to the history of the browser, swapping out the current state at the top of the history stack, and configuring a callback function for the listener to use in response to history modifications.

- If the URL parameters and URL fragment are available for the current page, they may be retrieved using google.script.url.

Only web apps have access to these historical APIs. Neither sidebars nor dialog boxes nor add-ons are supported. It is also not advised to utilize this capability in web apps that are integrated with Google Sites.

How to create Login and Register Form using Google spreadsheet data

The example below illustrates how to create a Login and register form using google spreadsheet data. You can access the Video Tutorial on https://youtu.be/2qRti1S9rK8 . The codes are displayed below:

Example 1:

Code.gs

```
function doGet(e) {

  var x = HtmlService.createTemplateFromFile("");

  var y = x.evaluate();

  var z =
y.setXFrameOptionsMode(HtmlService.XFrameOptionsMode.ALLOWALL);

  return z;

}

function checkLogin(username, password) {

  var url = '';
```

```
var ss= SpreadsheetApp.openByUrl(url);

var webAppSheet = ss.getSheetByName("DATA");

var getLastRow =  webAppSheet.getLastRow();

var found_record = '';

for(var i = 1; i <= getLastRow; i++)

{

 if(webAppSheet.getRange(i, 1).getValue().toUpperCase() ==
username.toUpperCase() &&

  webAppSheet.getRange(i, 2).getValue().toUpperCase() ==
password.toUpperCase())
```

```
 {

   found_record = 'TRUE';

 }

}

if(found_record == '')

{

  found_record = 'FALSE';

}
```

```
  return found_record;




}

function AddRecord(usernamee, passwordd, email, phone) {

  var url = '';

  var ss= SpreadsheetApp.openByUrl(url);

  var webAppSheet = ss.getSheetByName("DATA");

  webAppSheet.appendRow([usernamee,passwordd,email,phone]);
```

```
}
```

Index.html

```
<!DOCTYPE html>

<html>

  <head>

    <base target="_top">

    <script>
```

```
function AddRow()

{

var usernamee =
document.getElementById("usernamee").value;

var passwordd = document.getElementById("passwordd").value;

var email = document.getElementById("email").value;

var phone = document.getElementById("phone").value;

if (usernamee==""|| passwordd==""|| email==""|| phone=="")
{

  return false;
```

```
                }

                else {

google.script.run.AddRecord(usernamee,passwordd,email,phone);

                document.getElementById("page2_id1").className =
"page2_id1-off";

                document.getElementById("page3_id1").className =
"page3_id1";

                }

            }
```

```
 function LoginUser()

{

var username = document.getElementById("username").value;

var password = document.getElementById("password").value;

google.script.run.withSuccessHandler(function(output)

{

  if(output == 'TRUE')

  {
```

```
                document.getElementById("displayusername").innerHTML =
username;



                document.getElementById("page1_id1").className =
"page1_class1-off";



                document.getElementById("page4_id1").className =
"page4_id1";



            }



            else if(output == 'FALSE')



            {



              document.getElementById("errorMessage").innerHTML =
"Invalid data";



            }
```

```
        }).checkLogin(username, password);



        }



    function function1(){



        document.getElementById("page1_id1").className =
"page1_class1-off";



        document.getElementById("page2_id1").className =
"page2_id1";



    }



    function function3(){
```

```
document.getElementById("page3_id1").className = "page3_id1-
off";

document.getElementById("page1_id1").className = "page1_id1";

}

</script>

<style>

/*page1*/

.page1_class1-off{
```

```
    display: none;

  }

/*page2*/

.page2_class1{

    display: none;

  }
```

```
.page2_id1-off{

    display:none;

}



/*page3*/

.page3_class1{

    display:none;



}
```

```
.page3_id1-off{

    display:none;

}

/*page4*/

.page4_class1{

    display:none;

}
```

```
.page4_id1-off{

    display:none;

}

input[type=text]:hover{

        border-bottom:2px solid black;

    }

input[type=number]:hover{

        border-bottom:2px solid black;
```

```
    }

input[type=password]:hover{

      border-bottom:2px solid black;

    }

.user{

display: inline-block;

width: 75px;

height: 75px;
```

border: 8px solid black;

border-radius: 50%;

position: relative;

overflow: hidden;

box-sizing: border-box;

}

*/*the head/*/*

```
.user::before{

content: '';

display: inline-block;

width: 24px;

height: 24px;

background: black;

border-radius: 50%;

position: absolute;

left: calc(50% - 11px);
```

```
top: 10px;

}



/*the body*/

.user::after{

content: '';

display: inline-block;

width:50px;

height:40px;
```

background: black;

border-radius: 50%;

position: absolute;

left: calc(50% - 24px);

top: 39px;

}

</style>

```
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">



         </head>



         <body>



           <br><br>



        <!--page1-->



        <center>



           <div class="page1_class1" id="page1_id1"
style="background:none;border:2px solid gray;border-radius: 20px;width:
250px;padding-top: 10px;padding-bottom: 20px;padding-left: 20px;padding-
right: 20px;">



             <h1>Login Form</h1>
```


<input type="text" id="username" placeholder="Username" style="border-top: none;border-right: none;border-left: none;outline: none; text-align: center;font-size:0.9em ;width: 50%;font-weight:bold;"/>

<input type="password" id="password" placeholder="Password" style="border-top: none;border-right: none;border-left: none;outline: none; text-align: center;font-size:0.9em ;width: 50%;font-weight:bold;"/>

<input type="submit" value="Login" onclick="LoginUser()" style="float: right;padding-top: 1px;padding-bottom: 1px;padding-left: 10px;padding-right: 10px;font-size: 0.9em;font-weight:bold;" />

If you don't have an account,<input type="button"

```
onClick="function1()" value="Create New" style="margin-top: 5px;font-
weight:bold;" />



            </div>



            <!--page2-->

            <div class="page2_class1" id="page2_id1"
style="background:none;border:2px solid gray;border-radius: 20px;width:
250px;padding-top: 10px;padding-bottom: 20px;padding-left: 20px;padding-
right: 20px;">

            <h1>Create Account</h1>

              <input type="text" id="usernamee" placeholder="Name"
style="border-top: none;border-right: none;border-left: none;outline: none;
text-align: center;font-size:0.9em ;width: 50%;font-weight:bold;"/><br>

              <br>
```

<input type="password" id="passwordd" placeholder="Create password" style="border-top: none;border-right: none;border-left: none;outline: none; text-align: center;font-size: 0.9;width: 50%;font-weight:bold;" />

<input type="text" id="email" placeholder="Email" style="border-top: none;border-right: none;border-left: none;outline: none; text-align: center;font-size:0.9em ;width: 50%;font-weight:bold;"/>

<input type="number" id="phone" placeholder="Phone no." style="border-top: none;border-right: none;border-left: none;outline: none; text-align: center;font-size:0.9em ;width: 50%;font-weight:bold;" />

Password must contain letters and numbers.
It will not work without letters and numbers.

<input type="submit" value="Create" onclick="AddRow()" style="float: right;padding-top: 1px;padding-bottom: 1px;padding-left: 10px;padding-right: 10px;font-size: 0.9em;font-weight:bold;" />

```
<br>

</div>

<!--page3-->

<div class="page3_class1" id="page3_id1"
style="background:none;border:2px solid gray;border-radius: 20px;width:
250px;padding-top: 10px;padding-bottom: 20px;padding-left: 20px;padding-
right: 20px;"><center>

<h2> Your account has been successfully created.
Login to your account</h2>

<input type="submit"  onClick="function3()" value="Login"
style="font-weight:bold;"><br>

</div>
```

```
<!--page4-->

<div class="page4_class1" id="page4_id1"
style="background:none;border:2px solid gray;border-radius: 20px;width:
250px;padding-top: 10px;padding-bottom: 20px;padding-left: 20px;padding-
right: 20px;" ><center>

<br>

<h2>Hi <b id="displayusername" style="color:red;"></b>!
</h2>

<div class="user"></div>

<h2> You are successfully logged in.</h2>

<h2>**********</h2>
```

```
<br>

</div>

</center>

</body>

</html>
```

Example 2:

These example applies some CSS properties to the Login and Registration form.

Code.gs

```
function doGet(e) {

  var x = HtmlService.createTemplateFromFile("index");

  var y = x.evaluate();

  var z =
y.setXFrameOptionsMode(HtmlService.XFrameOptionsMode.ALLOWALL);

  return z;

}

function checkLogin(username, password) {

  var url = 'Enter Your Google Sheet URL';
```

```
var ss= SpreadsheetApp.openByUrl(url);

var webAppSheet = ss.getSheetByName("Noogul");

var getLastRow =  webAppSheet.getLastRow();

var found_record = '';

for(var i = 1; i <= getLastRow; i++)

{

 if(webAppSheet.getRange(i, 1).getValue().toUpperCase() ==
username.toUpperCase() &&

  webAppSheet.getRange(i, 2).getValue().toUpperCase() ==
password.toUpperCase())

 {
```

```
   found_record = 'TRUE';




 }


}


if(found_record == '')


{


  found_record = 'FALSE';


}
```

```
  return found_record;

}

function AddRecord(usernamee, passwordd, email, phone) {

  var url = 'Enter Your Google Sheet URL';

  var ss= SpreadsheetApp.openByUrl(url);

  var webAppSheet = ss.getSheetByName("Noogul");

  webAppSheet.appendRow([usernamee,passwordd,email,phone]);

}
```

Index.html

```
<!DOCTYPE html>

<html>

  <head>

  <base target="_top">

  <script>

   function AddRow()
```

```
{

var usernamee =
document.getElementById("usernamee").value;

var passwordd = document.getElementById("passwordd").value;

var email = document.getElementById("email").value;

var phone = document.getElementById("phone").value;

if (usernamee==""|| passwordd==""|| email==""|| phone=="")
{

  return false;

}
```

else {

google.script.run.AddRecord(usernamee,passwordd,email,phone);

document.getElementById("page2_id1").className = "page2_id1-off";

document.getElementById("page3_id1").className = "page3_id1";

}

}

function LoginUser()

{

```
var username = document.getElementById("username").value;

var password = document.getElementById("password").value;

google.script.run.withSuccessHandler(function(output)

{

  if(output == 'TRUE')

  {

    var url1 ='Enter Redirect Url address Here';

    var winRef = window.open(url1);
```

```
                winRef ?
            google.script.host.close() : window.onload=function()
{document.getElementById('url').href = url1;}



            }



            else if(output == 'FALSE')

            {

              document.getElementById("errorMessage").innerHTML =
"Invalid Data";

            }

          }).checkLogin(username, password);
```

```
        }

        function function1(){

            document.getElementById("page1_id1").className =
"page1_class1-off";

            document.getElementById("page2_id1").className =
"page2_id1";

        }

        function function3(){

          document.getElementById("page3_id1").className = "page3_id1-
```

```
off";

            document.getElementById("page1_id1").className = "page1_id1";

        }

        </script>

        <style>

        /*page1*/

        .page1_class1-off{

          display: none;
```

```
}



/*page2*/

.page2_class1{

    display: none;

}

.page2_id1-off{

    display:none;
```

```
}



/*page3*/

.page3_class1{

    display:none;



}

.page3_id1-off{

    display:none;
```

```
}



/*page4*/

.page4_class1{

    display:none;



}

.page4_id1-off{

    display:none;
```

```
}




input[type=text]:hover{

    border-bottom:2px solid black;

}

input[type=number]:hover{

    border-bottom:2px solid black;

}

input[type=password]:hover{
```

```
        border-bottom:2px solid black;

    }



.user{

display: inline-block;

width: 75px;

height: 75px;

border: 8px solid black;

border-radius: 50%;
```

```
position: relative;

overflow: hidden;

box-sizing: border-box;

}

/*the head/*/

.user::before{

content: '';
```

display: inline-block;

width: 24px;

height: 24px;

background: black;

border-radius: 50%;

position: absolute;

left: calc(50% - 11px);

top: 10px;

}

```
/*the body*/

.user::after{

content: '';

display: inline-block;

width:50px;

height:40px;

background: black;

border-radius: 50%;
```

```
            position: absolute;

            left: calc(50% - 24px);

            top: 39px;

            }

            </style>

            <meta name="viewport" content="width=device-width, initial-
scale=1.0">

              </head>
```

<body>

<!--page1-->

<div class="page1_class1" id="page1_id1" style="width: 80%;

color: black;

padding: 14px 20px;

margin: 8px 0;

border-radius: 20px;width: 250px

border-radius: 4px;

cursor: pointer;">

* *

* *

*
*

<input type="text" id="username" placeholder="Username" style="width: 80%;

color: black;

padding: 14px 20px;

margin: 8px 0;

border-radius: 20px;width: 250px

border-radius: 4px;

*cursor: pointer;"/>
*

*
*

<input type="password" id="password" placeholder="Password" style="width: 80%;

color: black;

padding: 14px 20px;

margin: 8px 0;

border-radius: 20px;width: 250px

border-radius: 4px;

cursor: pointer;"/>

*

*

<input type="submit" value="Login" onclick="LoginUser()"
style="width: 80%;

background-color: dodgerblue;

color: white;

padding: 15px 20px;

border: none;

cursor: pointer;

width: 100%;

*opacity: 0.9;" />
*

*
*

If you don't have an account,<input type="button" onClick="function1()" value="Create New" style="margin-top: 5px;font-weight:bold;" />

* *

<!--page2-->

<div class="page2_class1" id="page2_id1" style="width: 50%;

color: black;

padding: 14px 20px;

margin: 8px 0;

border-radius: 20px;width: 250px

border-radius: 4px;

cursor: pointer;">

<h1>Create Account</h1>

<input type="text" id="usernamee" placeholder="Username" style="width: 80%;

color: black;

padding: 14px 20px;

margin: 8px 0;

border-radius: 20px;width: 250px

border-radius: 4px;

*cursor: pointer;"/>
*

*
*

<input type="password" id="passwordd" placeholder="Create password" style="width: 80%;

color: black;

padding: 14px 20px;

margin: 8px 0;

border-radius: 20px;width: 250px

border-radius: 4px;

*cursor: pointer;" />
*

*
*

<input type="text" id="email" placeholder="Email" style="width: 80%;

color: black;

padding: 14px 20px;

margin: 8px 0;

border-radius: 20px;width: 250px

border-radius: 4px;

*cursor: pointer;"/>
*

*
*

<input type="number" id="phone" placeholder="Phone no." style="width: 80%;

color: black;

padding: 14px 20px;

margin: 8px 0;

border-radius: 20px;width: 250px

border-radius: 4px;

*cursor: pointer;" />

*

Password must contain letters and numbers.
*It will not work without letters and numbers.
*

*
*

<input type="submit" value="Create" onclick="AddRow()"

style="width: 80%;

background-color: dodgerblue;

color: white;

padding: 15px 20px;

border: none;

cursor: pointer;

width: 100%;

opacity: 0.9;" />

*
*

```
</div>



<!--page3-->



 <div class="page3_class1" id="page3_id1"
style="background:none;border:2px solid gray;border-radius: 20px;width:
250px;padding-top: 10px;padding-bottom: 20px;padding-left: 20px;padding-
right: 20px;"><center>



   <h2> Your account has been successfully created.
  Login to your account</h2>



   <input type="submit"  onClick="function3()" value="Login"
style="font-weight:bold;"><br>



  </div>
```

```
<!--page4-->

<div class="page4_class1" id="page4_id1" style="width: 50%;

 background-color: #FFFAF0;

 color: Green;

 padding: 14px 20px;

 margin: 8px 0;

 border: none;

 border-radius: 4px;

 cursor: pointer;" ><center>
```

```
<br>

<h2>Hi <b id="displayusername" style="color:red;"></b>!
</h2>

<div class="user"></div>

<h2> You are successfully logged in.</h2>

<nav>

<ul>

<li><a href="Optional Url if Not Redirected">DASHBOARD</a>
</li> <BR>

<BR>
```

*
UPLOAD CONTENT
*

*
*

*
BLOG
*

*
*

*
*

</nav>

*
*

* *

* *

</body>

</html>

How to Display Google Sheet Data on Webpage

You must first have a Google Spreadsheet with some data in it. For this, you may utilize an already-existing one or make a new one. By doing this, the webpage will be able to show every Google Sheet on an HTML website. The Youtube Video Link Illustrates the application of these codes https://youtu.be/-dkewMUYU7A

Code.gs

```
function doGet(e) {

  var x = HtmlService.createTemplateFromFile("");

  var y = x.evaluate();

  var z =
y.setXFrameOptionsMode(HtmlService.XFrameOptionsMode.ALLOWALL);

  return z;

}

function getSheetData()  {
```

```
  var a= SpreadsheetApp.getActiveSpreadsheet();

  var b = a.getSheetByName('Sheet1');

  var c = b.getDataRange();

  return c.getValues();

}
```

Index.html

```
<!DOCTYPE html>
```

```
<html>

 <head>

  <base target="_top">

 </head>

 <body>

  <center><br>

  <h2>Display google sheet data on webpage</h2>

  <table border="3" style="border:2px solid black"
cellpadding="5px" >
```

```
    <?var tableData = getSheetData();?>


    <?for(var i = 0; i < tableData.length; i++) { ?>


      <?if(i == 0) { ?>


        <tr>


        <?for(var j = 0; j < tableData[i].length; j++) { ?>


        <th><?= tableData[i][j] ?></th>


        <?
} ?>


        </tr>


      <?
} else { ?>
```

```
        <tr>

        <?for(var j = 0; j < tableData[i].length; j++) { ?>

        <td><?= tableData[i][j] ?></td>

        <?
} ?>.

        </tr>

    <?
} ?>

   <?
} ?>

   </table></center>
```

```
</body>

</html>
```

How to Submit HTML Form Data to Google Spreadsheet

The following example https://youtu.be/7Vn3ycGS04I submits filled details of HTML form data to google Spreadsheet. Efficient data collection and management is essential for both individuals and enterprises in the current digital era. HTML forms are a common tool for data collection, and Google Sheets is a great place to store this data. We'll walk you through setting up a data entry form on your website and moving the data to Google Sheets with ease. You'll have an effective tool to expedite your data collecting procedure by the conclusion of this session.

Recognizing the Value of HTML Form Integration with Google Sheets

Let's quickly go over the benefits of integrating HTML forms with Google Sheets before getting into the technical details.

This strategy has several advantages:

i. **Effective Data Gathering:** Since HTML forms are so widely available and easy to use, gathering data is a snap.

ii. **Real-time Updates:** Information input into the form may be immediately reflected in the Google Sheet, giving you access to the most recent information available.

iii. **Single Data Storage:** Google Sheets eliminates the need for dispersed files and manual data entry by offering a single area for organizing and storing your data.

Example 1:

Code.gs

```
var sheetName = 'Sheet1'
```

```
var scriptProp = PropertiesService.getScriptProperties()

function intialSetup () {

  var activeSpreadsheet = SpreadsheetApp.getActiveSpreadsheet()

  scriptProp.setProperty('key', activeSpreadsheet.getId())
```

```
}
```

```
function doPost (e) {
```

```
  var lock = LockService.getScriptLock()
```

```
  lock.tryLock(10000)
```

```
  try {

    var doc = SpreadsheetApp.openById(scriptProp.getProperty('key'))

    var sheet = doc.getSheetByName(sheetName)

    var headers = sheet.getRange(1, 1, 1,
sheet.getLastColumn()).getValues()[0]

    var nextRow = sheet.getLastRow() + 1
```

```
var newRow = headers.map(function(header) {

  return header === 'timestamp' ?
new Date() : e.parameter[header]

})

sheet.getRange(nextRow, 1, 1,
```

```
newRow.length).setValues([newRow])

        return ContentService

            .createTextOutput(JSON.stringify({ 'result': 'success', 'row':
nextRow }))

            .setMimeType(ContentService.MimeType.JSON)

        }
```

```
  catch (e) {

return ContentService

  .createTextOutput(JSON.stringify({ 'result': 'error', 'error': e }))

  .setMimeType(ContentService.MimeType.JSON)
```

```
    }

    finally {

  lock.releaseLock()

    }

}
```

Form.html

```
<!DOCTYPE html>

<html>

<head>

  <title>database</title>

    <style>

input[type=text], select {
```

```
  width: 80%;

  padding: 12px 20px;

  margin: 8px 0;

  display: inline-block;

  border: 1px solid #ccc;

  border-radius: 4px;

  box-sizing: border-box;

}
```

```
    input[type=email], select {

width: 80%;

padding: 12px 20px;

margin: 8px 0;

display: inline-block;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;
```

```
}

textarea {

  width: 80%;

  height: 250px;

  padding: 12px 20px;

  box-sizing: border-box;

  border: 2px solid #ccc;

  border-radius: 4px;

  background-color: #f8f8f8;
```

```
  font-size: 16px;

  resize: none;

}

input[type=submit] {

  width: 80%;

  background-color: #4CAF50;

  color: white;
```

```
  padding: 14px 20px;

  margin: 8px 0;

  border: none;

  border-radius: 4px;

  cursor: pointer;

}

input[type=submit]:hover {
```

```
  background-color: #45a049;

}

div {

  border-radius: 5px;

  background-color: #f2f2f2;

  padding: 20px;

}

body {font-family: Arial, Helvetica, sans-serif;}
```

```
* {box-sizing: border-box;}

</style>

</head>

<body>

  <form method="post" autocomplete="off" name="google-sheet">
```

*

*

*<h1>Registration Form</h1>
*

<table style =

width: 80%;

padding: 12px 20px;

margin: 8px 0;

display: inline-block;

border: 1px solid #ccc;

border-radius: 4px;

```
box-sizing: border-box;



}>



                        <tr><td></td>


                    <td> <h3>Full Name:</h3><input type="text"
name="Name"  placeholder="Enter Author Full Name" required="" /></td>


                    </tr>


                    <tr><td></td>
```

<td><h3>Email:</h3><input type="email" name="email" autocomplete="off" placeholder="Enter Your Correct Email Address" required="" /></td>

</tr>

<tr><td></td>

<td><h3>Title:</h3><input type="text" name="Title" placeholder="Enter The Title of your Content" required="" /></td>

</tr>

<tr><td></td>

<td><h3> Sub-Title:</h3><input type="text" name="Sub-Title" placeholder="Enter The Sub-Title of your Content (Optional)" /></td>

</tr>

<tr><td></td>

*<td> <h3>Beneficiary Name:</h3><input type="text" name="Beneficiary Name" placeholder="Designate who gets paid when your Book sells" required="" />
*

*<h3>Banking Country:</h3><input type="text" name="Country" placeholder="Country of Residence" required="" />
*

*<h3>Beneficiary Account Number:</h3><input type="text" name="Account Number" placeholder="Account Number of Beneficiary" required="" />
*

*<h3>Bank Name:</h3><input type="text" name="Bank Name" placeholder="Enter Bank Name" required="" />
*

*<h3>Bank Branch:</h3><input type="text" name="Bank Branch" placeholder="What Branch is your Bank Located" required="" />
*

<h3>Bank Swift Code:</h3><input type="text"

*name="Swift Code" placeholder="Your Bank Swift Code" required="" />
*

<h3>Tax ID (Optional):</h3><input type="text" name="Tax ID" placeholder="Enter your Tax ID (Optional)" />

</td>

</tr>

<tr><td></td>

*<td ><h2 Style ="background-color: #45a049"; >Additional Payment Information (Optional)

</h2><h4>Paypal Payment (Payable in USD only.*
$5 Processing Fee).</h4>

</td>

</tr>

<tr><td></td>

*<td> <h3>Send Payments to:</h3><input type="text" name=" Paypal Email Address " placeholder="Enter your Paypal Email Address" />
*

</td>

</tr>

<tr><td></td>

*<td>
*

<input type="checkbox" id="Concent" name="Consent" value="Concent">

<label for="Concent">I Concent - to receive emails from Noogul, including exclusive offers, newsletters, promotions, and other notifications </label>

</tr>

</table>

*

*

<input type="submit" name="submit" value="Submit"/>

* *

</form>

<script type="text/javascript">

const scriptURL = ‘

Enter the Appscript URL where filled contents will be submitted to'

const form = document.forms['google-sheet']

form.addEventListener('submit', e => {

```
                e.preventDefault()

                fetch(scriptURL, { method: 'POST', body: new
FormData(form)})

                  .then(response => alert("You have successfully
submitted."))

                  .catch(error => console.error('Error!', error.message))

              })

        </script>

    </body>

</html>
```

Example 2

: The example below shows how to submit HTML Forms to google sheets with CSS Properties.

Code.gs

```
var sheetName = 'Publish'

var scriptProp = PropertiesService.getScriptProperties()

function intialSetup () {

var activeSpreadsheet =
SpreadsheetApp.getActiveSpreadsheet()
```

```
  scriptProp.setProperty('key', activeSpreadsheet.getId())

}

function doPost (e) {

  var lock = LockService.getScriptLock()

  lock.tryLock(10000)

  try {

  var doc =
SpreadsheetApp.openById(scriptProp.getProperty('key'))
```

```
        var sheet = doc.getSheetByName(sheetName)

        var headers = sheet.getRange(1, 1, 1,
sheet.getLastColumn()).getValues()[0]

        var nextRow = sheet.getLastRow() + 1

        var newRow = headers.map(function(header) {

          return header === 'timestamp' ?
      new Date() : e.parameter[header]

        })
```

```
        sheet.getRange(nextRow, 1, 1,
newRow.length).setValues([newRow])

        return ContentService
          .createTextOutput(JSON.stringify({ 'result': 'success', 'row':
nextRow }))
          .setMimeType(ContentService.MimeType.JSON)
      }

      catch (e) {
        return ContentService
```

```
      .createTextOutput(JSON.stringify({ 'result': 'error', 'error': e }))

      .setMimeType(ContentService.MimeType.JSON)

  }



  finally {

  lock.releaseLock()

  }

}
```

Publish.html

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>database</title>

```
</head>

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">

    <style>

        body {

  background-color: #FF3399;

}

input[type=text], select {

  width: 80%;
```

```
  padding: 12px 20px;

  margin: 8px 0;

  display: inline-block;

  border: 1px solid #ccc;

  border-radius: 4px;

  box-sizing: border-box;

}
```

```
    input[type=email], select {

width: 80%;

padding: 12px 20px;

margin: 8px 0;

display: inline-block;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;
```

```
}

    input[type=email], select {

  width: 80%;

  padding: 12px 20px;

  margin: 8px 0;

  display: inline-block;

  border: 1px solid #ccc;

  border-radius: 4px;
```

```
  box-sizing: border-box;

}

  textarea {

  width: 80%;

  height: 250px;

  padding: 12px 20px;

  box-sizing: border-box;
```

```
  border: 2px solid #ccc;

  border-radius: 4px;

  background-color: #f8f8f8;

  font-size: 16px;

  resize: none;

}

input[type=submit] {

  width: 80%;
```

```
    background-color: #4CAF50;

    color: white;

    padding: 14px 20px;

    margin: 8px 0;

    border: none;

    border-radius: 4px;

    cursor: pointer;

}
```

```
input[type=submit]:hover {

  background-color: #45a049;

}

div {

  border-radius: 5px;

  background-color: #f2f2f2;

  padding: 20px;

}
```

```
body {font-family: Arial, Helvetica, sans-serif;}

* {box-sizing: border-box;}

</style>

<body>

  <form method="post" autocomplete="off" name="google-sheet">
```

*

*

* *

<table style =

width: 80%;

padding: 12px 20px;

margin: 8px 0;

display: inline-block;

```
border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;

}>

<tr><td></td>

<td> <h3>Full Name:</h3><input type="text"
name="Name"  placeholder="Enter Author Full Name" required="" /></td>

</tr>
```

<tr><td></td>

<td><h3>Email:</h3><input type="email" name="email" autocomplete="off" placeholder="Enter Your Correct Email Address" required="" /></td>

</tr>

<tr><td></td>

<td><h3>Title:</h3><input type="text" name="Title" placeholder="Enter The Title of your Content" required="" /></td>

</tr>

<tr><td></td>

<td><h3> Sub-Title:</h3><input type="text" name="Sub-

```
Title"  placeholder="Enter The Sub-Title of your Content (Optional)"  /></td>

                    </tr>

                       <tr><td ></td>

                    <td  style ="width: 80%"><select name="category"
required="" >

            <option value="Select Category">Select  Category >>>>>
</option>

            <option value="Art and Photography">Art and
Photography</option>

            <option value="Business and Economics">Business and
Economics</option>

            <option value="Biographies and Memoirs">Biographies and
Memoirs</option>
```

<option value="Children's">Children's</option>

<option value="Comics & Graphic Novels">Comics & Graphic Novels</option>

<option value="Computers & Technology">Computers & Technology</option>

<option value="Religion & Spirituality">Religion & Spirituality</option>

<option value="Science & Medicine">Science & Medicine</option>

<option value="Social Science">Social Science</option>

<option value="Psychology">Psychology</option>

<option value="Sports">Sports</option>

<option value="Travel & Adventure">Travel & Adventure</option>

<option value="Young Adult">Young Adult</option>

</select>

</td>

</tr>

<tr><td></td>

<td> <h3> Contributors (Optional):</h3><input

type="text" name="Contributors" placeholder="Specify Either Co-author, Editor or Illustrator with their names separated with Comma (,)" /></td>

</tr>

*<tr><td></td>

*

*<td><h3>Project Details(Synopsis) :</h3><textarea id="subject" name="Synopsis" placeholder="Provide all important metadata to help readers find your book.
" style="height:250px"></textarea></td>*

</tr>

<tr><td></td>

*<td><h3>Keywords:</h3><input type="text" name="Keywords" placeholder="Add Keywords...."
required="" /></td>*

```
                    </tr>

                    <tr><td ></td>

                    <td  style ="width: 100%"><select name="Language"
required="" >

            <option value="Select Language">Select  Language >>>>>
</option>

            <option value="English">English</option>

            <option value="German">German</option>

            <option value="Spanish">Spanish</option>
```

<option value="Chinese">Chinese</option>

<option value="French">French</option>

<option value="Iranian">Iranian</option>

<option value="Hausa">Hausa</option>

<option value="Igbo">Igbo</option>

<option value="Yoruba">Yoruba</option>

<option value="Abkhazian">Abkhazian</option>

<option value="Acoli">Acoli</option>

<option value="Afrikaans">Afrikaans</option>

<option value="Bambara">Bambara</option>

<option value="Zapotec">Zapotec</option>

<option value="Yapeze">Yapeze</option>

<option value="Twi">Twi</option>

</select>

</td>

<tr><td></td>

<td><h3>Other Language: </h3><input type="text" name="others" placeholder="Kindly indicate other Languags if not Specified (Optional)" /></td>

</tr>

<tr><td ></td>

<td style ="width: 80%;" ><select name="Audience" required="" >

<option value="Select Audience">Select Audience >>>>> </option>

<option value="General/Trade - Adult Fiction and Non-Fiction">General/Trade - Adult Fiction and Non-Fiction</option>

<option value="Children/Juvenile - Children's Books age range 2 - 12 years, not for educational purposes">Children/Juvenile - Children's Books age range 2 - 12 years, not for educational purposes</option>

<option value="Young Adult - Teen Fiction and Non-Fiction, age range 12 - 20 years, not for educational Purposes">Young Adult - Teen Fiction and Non-Fiction, age range 12 - 20 years, not for educational Purposes</option>

<option value="Primary and Secondary/ Elementary and High School Educational Material, age range 5 - 18years">Primary and Secondary/ Elementary and High School Educational Material, age range 5 - 18years</option>

</select>

</td>

*<tr><td></td>

*

<td><h3>Table of Contents :</h3><textarea id="subject" name="Contents" placeholder="Provide your Table of Content (Optional).
" style="height:250px"></textarea></td>

</tr>

<tr><td></td>

*<td ><h2 Style ="background-color: #45a049";>Select a Goal

</h2> Start by telling us what you plan to do with your Book.*
From printing your own copies to selling around the world or on your own website, we've got you covered!
* *

<input type="checkbox" id="Global Distribution" name="Global Distribution" value="Global Distribution">

<label for="Global Distribution">Global Distribution (Sell your Book through 40,000+ global retailers using Noogul distribution service.
Please note that a title page, copyright page, and ISBN are required.

</label> </td>

</tr>

<tr><td></td>

*<td>
*

<input type="checkbox" id="Print Your Book" name="Print Your Book" value="Print Your Book">

<label for="Print Your Book"><b>Print Your Book (Upload your Book files to your account and purchase copies).
</label>

</tr>

<tr><td></td>

*<td ><h2 Style ="background-color: #45a049"; >Book Specification

</h2> Select your book's specifications, add content and design your cover and interior.
*

<input type="checkbox" id="5.5 X 8.25" name="5.5 X 8.25" value="5.5 X 8.25 ">

*<label for="5.5 X 8.25">5.5 X 8.25 inch </label>

 <img src="7.jpg" alt="Print Book " style="width:250px;height:250px;"> (Fiction, Poetry, and Non-Fiction Less than 50,000 words).*
Ideal to present or showcase your fiction, non fictional, catalogs or manuals book cover design </td>

</tr>

<tr><td></td>

*<td>
*

<input type="checkbox" id="6 x 9" name="6 x 9" value="6 x 9">

*<label for="6 x 9">6 x 9 inch </label>

 <img src="8.jpg" alt="6 x 9 " style="width:200px;height:300px;"> (Self-help, Biographies, Business and Academic Less than 75,000 words) </td>*

</tr>

<tr><td></td>

*<td>
*

<input type="checkbox" id="6.625 x 10.25 inch" name="6.625 x 10.25 inch" value="6.625 x 10.25 inch">

*<label for="6.625 x 10.25 inch">6.625 x 10.25 inch </label>

 <img src="9.jpg" alt="6.625 x 10.25 inch " style="width:300px;height:200px;"> (168.3 x 260.4 mm) with 0.75 inch spine width for each book to showcase collection or compilation of your graphic novels cover design. </td>*

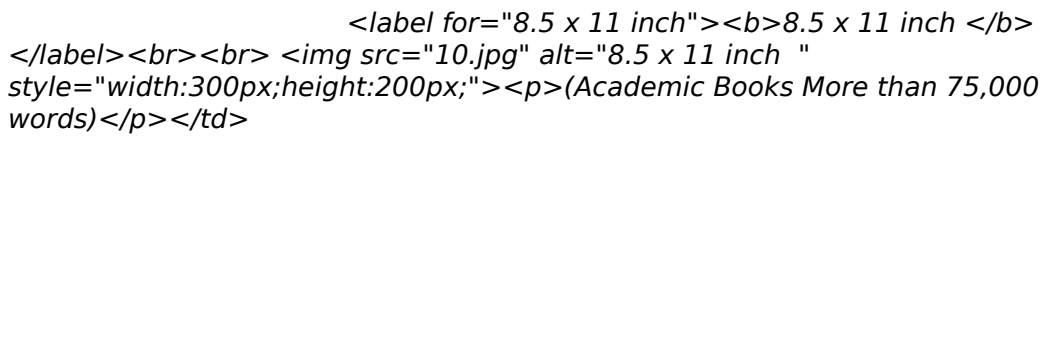
</tr>

<tr><td></td>

<td>

<input type="checkbox" id="8.5 x 11 inch" name="8.5 x 11 inch" value="8.5 x 11 inch">

<label for="8.5 x 11 inch">8.5 x 11 inch </label>

</tr>

<tr><td></td>

<td>

<input type="checkbox" id="11 x 8.5 inch" name="11 x 8.5 inch" value="11 x 8.5 inch">

*<label for="11 x 8.5 inch">11 x 8.5 inch </label>

 <img src="11.jpg" alt="11 x 8.5 inch " style="width:300px;height:200px;"> (280 x 216 mm) Horizontal Book Box with 0.75 inch (19 mm) spine thickness.*

Ideal to showcase your 260 ~ 360 pages photo albums, catalogs, children story books, series of book in a package cover design. </td>

</tr>

<tr><td></td>

*<td ><h2 Style ="background-color: #45a049"; >Binding Type

</h2> Select the color and paper for your interior, and the binding and finish for your cover.*

*Note that if an option is unavailable for your Book size, it will not be available in this step.
*

<input type="checkbox" id="Paper Back" name="Paper Back" value="Paper Back ">

*<label for="Paper Back">Paper Back </label>

 <img src="8.jpg" alt="Paper Back " style="width:200px;height:300px;"></td>*

</tr>

<tr><td></td>

*<td>
*

<input type="checkbox" id="Thick Back" name="Thick Back" value="Thick Back">

*<label for="Thick Back">Thick Back *
*</label>

 <img src="12.jpg" alt="Thick Back "*
style="width:300px;height:200px;"></td>

</tr>

<tr><td></td>

<td ><h2 Style ="background-color: #45a049"; >Book
*Interior

</h2> Standard ink is recommended for Books using*
text and limited graphics, while Premium is ideal for rich colors and more
*complex graphics.
*

<input type="checkbox" id="Black & White
Premium" name="Black & White Premium" value="Black & White Premium ">

<label for="Black & White Premium">Black &
*White Premium </label>

 <img src="13.png" alt="Black &*

White Premium " style="width:300px;height:200px;"></td>

</tr>

<tr><td></td>

*<td>
*

<input type="checkbox" id="Premium Color" name="Premium Color" value="Premium Color">

*<label for="Premium Color">Premium Color </label>

 <img src="5.png" alt="Premium Color " style="width:300px;height:200px;"></td>*

</tr>

<tr><td></td>

<td ><h2 Style ="background-color: #45a049";
*>Paper Type

</h2> For Premium ink, we suggest the heavier 80# paper.*
*For more economical options, use Standard ink and the 60# paper weight.
*

<input type="checkbox" id="80# White-Coated Paper" name="80# White-Coated Paper" value="80# White-Coated Paper">

*<label for="80# White-Coated Paper">80# White-Coated Paper </label>

 <img src="14.jpg" alt="80# White-Coated Paper " style="width:300px;height:200px;"></td>*

</tr>

<tr><td></td>

*<td>
*

<input type="checkbox" id="60# Cream Paper" name="60# Cream Paper" value="60# Cream Paper">

*<label for="60# Cream Paper">60# Cream Paper   </label>

 <img src="15.jpg" alt="60# Cream Paper " style="width:300px;height:200px;"></td>*

</tr>

<tr><td></td>

*<td ><h2 Style ="background-color: #45a049"; >Book Cover Lamination

</h2> Select the cover finish for your Book. *

<input type="checkbox" id="Glossy Finish" name="Glossy Finish" value="Glossy Finish">

<label for="Glossy Finish">Glossy Finish(Recommended for darker colored covers and designs with spatial detailing) </label> </td>

</tr>

<tr><td></td>

<td>

<input type="checkbox" id="Matte Finish" name="Matte Finish" value="Matte Finish">

<label for="Matte Finish">Matte Finish (Recommended for lighter colored covers, especially with graphic designs) </label>

 </td>

</tr>

<tr><td></td>

<td ><h2 Style ="background-color: #45a049";
*>Copyright

</h2> Select the copyright license that best suits your work.*
For more information about copyright, please visit our Terms of Agreement.
* *

<input type="checkbox" id="All Right Reserved" name="All Right Reserved" value="All Right Reserved">

<label for="All Right Reserved">All Rights Reserved - Standard Copyright License (All Rights Reserved licensing.
Your work cannot be distributed, remixed, or otherwise used without your express consent.) </label> </td>

</tr>

<tr><td></td>

*<td>
*

<input type="checkbox" id="Some Right Reserved" name="Some Right Reserved" value="Some Right Reserved">

<label for="Some Right Reserved">Some Rights Reserved - Creative Commons (CC BY) (Some rights are reserved, based on the specific Creative Commons Licensing you select.
Please Specify) </label>

</tr>

<tr><td></td>

*<td>
*

<input type="checkbox" id="No Right Reserved" name="No Right Reserved" value="No Right Reserved">

<label for="No Right Reserved ">No Rights

Reserved - Public Domain (No rights are reserved and the work is freely available for anyone to use, distribute, and alter in any way).</label>

</tr>

<tr><td></td>

*<td ><h2 Style ="background-color: #45a049"; >Add Pricing and Payees

</h2> Set the price for each currency manually or select a revenue goal for each Book sale.*
* *

</td>

</tr>

<tr><td ></td>

```
                    <td  style ="width: 80%;"  ><select  name="Pricing"
required=""  >

            <option value="Select Currency">Select  Currency >>>>>
</option>

            <option value="NAIRA">NAIRA &#8358;</option>

            <option value="EURO">EURO &#8364;</option>

            <option value="INDIAN RUPEE">INDIAN RUPEE &#8377;</option>

            <option value="GBP">GBP &#xA3; </option>

            <option value="USD">USD &dollar;</option>

            <option value="CENT">CENT &#162;</option>
```

<option value="BITCOIN">BITCOIN ₿</option>

<option value="CNY">CNY 元</option>

<option value="WON">WON ￦</option>

<option value="CEDI">CEDI ₵ </option>

<option value="ARS">ARS $</option>

<option value="BRL">BRL R</option>

<option value="CAD">CAD $</option>

<option value="JMD">JMD J </option>

<option value="PHP">PHP ₱</option>

<option value="NOK">NOK k</option>

<option value="VEF">VEF B</option>

*</select>
*

<input type="text" name="Price" placeholder="Add the price of your Product" required="" />

</td>

<tr><td></td>

*<td ><h2 Style ="background-color: #45a049"; >Profit Payment Information

</h2><h4>Designate who gets paid when your Book sells.</h4>*

</td>

</tr>

<tr><td></td>

*<td> <h3>Beneficiary Name:</h3><input type="text" name="Beneficiary Name" placeholder="Designate who gets paid when your Book sells" required="" />
*

*<h3>Banking Country:</h3><input type="text" name="Country" placeholder="Country of Residence" required="" />
*

*<h3>Beneficiary Account Number:</h3><input type="text" name="Account Number" placeholder="Account Number of Beneficiary" required="" />
*

<h3>Bank Name:</h3><input type="text" name="Bank

*Name" placeholder="Enter Bank Name" required="" />
*

*<h3>Bank Branch:</h3><input type="text" name="Bank Branch" placeholder="What Branch is your Bank Located" required="" />
*

*<h3>Bank Swift Code:</h3><input type="text" name="Swift Code" placeholder="Your Bank Swift Code" required="" />
*

<h3>Tax ID (Optional):</h3><input type="text" name="Tax ID" placeholder="Enter your Tax ID (Optional)" />

</td>

</tr>

<tr><td></td>

<td ><h2 Style ="background-color: #45a049";

*>Additional Payment Information (Optional)

</h2><h4>Paypal Payment (Payable in USD only.*
$5 Processing Fee).</h4>

</td>

</tr>

<tr><td></td>

*<td> <h3>Send Payments to:</h3><input type="text" name="Beneficiary Name" placeholder="Enter your Paypal Email Address" />
*

</td>

</tr>

<tr><td></td>

<td>

<input type="checkbox" id="Concent" name="Consent" value="Concent">

<label for="Concent">I Concent - to receive emails from Noogul, including exclusive offers, newsletters, promotions, and other notifications </label>

</tr>

</table>

*<input type="submit" name="submit" value="Submit"/>
*
*
<INPUT type="reset" Style="width: 40%;*

background-color: #FF6347;

color: white;

padding: 14px 20px;

margin: 8px 0;

border: none;

border-radius: 4px;

cursor: pointer;">

* *

</form>

<script type="text/javascript">

const scriptURL = ‘Enter the Appscript URL where filled contents will be submitted to'

const form = document.forms['google-sheet']

```
                form.addEventListener('submit', e => {

                  e.preventDefault()

                  fetch(scriptURL, { method: 'POST', body: new
FormData(form)})

                    .then(response => alert("You have successfully
submitted."))

                  .catch(error => console.error('Error!', error.message))

                })

            </script>
```

```
</body>


</html>
```

How to Submit HTML Form to Gmail

The example below gives a detailed template of how to submit html forms to Gmail or any email provider of your choice.

Code.gs

```
function formatMailBody(obj,order) {


  var result = "";
```

```
    for (var idx in order) {

      var key = order[idx];

      result += "<h4 style='text-transform: capitalize; margin-bottom:
0'>" + key + "</h4><div>" + sanitizeInput(obj[key]) + "</div>";

    }

    return result;

  }

  function sanitizeInput(rawInput) {

    var placeholder = HtmlService.createHtmlOutput(" ");
```

```
  placeholder.appendUntrusted(rawInput);



  return placeholder.getContent();

}



function doPost(e) {



  try {

    Logger.log(e);
```

```
record_data(e);




var mailData = e.parameters;




var orderParameter = e.parameters.formDataNameOrder;

var dataOrder;

if (orderParameter) {

  dataOrder = JSON.parse(orderParameter);

}
```

```
    var sendEmailTo = (typeof TO_ADDRESS !== "undefined") ?
TO_ADDRESS : mailData.formGoogleSendEmail;

    if (sendEmailTo) {

      MailApp.sendEmail({

        to: String(sendEmailTo),

        subject: "Contact form submitted",
```

```
    htmlBody: formatMailBody(mailData, dataOrder)

  });

}

return ContentService    // return json success results

    .createTextOutput(

      JSON.stringify({"result":"success",

                      "data": JSON.stringify(e.parameters) }))

    .setMimeType(ContentService.MimeType.JSON);
```

```
} catch(error) { // if error return this

  Logger.log(error);

  return ContentService

      .createTextOutput(JSON.stringify({"result":"error", "error":
error}))

      .setMimeType(ContentService.MimeType.JSON);

 }

}
```

```
function record_data(e) {

  var lock = LockService.getDocumentLock();

  lock.waitLock(30000); // hold off up to 30 sec to avoid concurrent
writing

  try {

    Logger.log(JSON.stringify(e)); // log the POST data in case we need
to debug it

    var doc = SpreadsheetApp.getActiveSpreadsheet();

    var sheetName = e.parameters.formGoogleSheetName ||
"sheetname";
```

```
        var sheet = doc.getSheetByName(sheetName);

        var oldHeader = sheet.getRange(1, 1, 1,
sheet.getLastColumn()).getValues()[0];

        var newHeader = oldHeader.slice();

        var fieldsFromForm = getDataColumns(e.parameters);

        var row = [new Date()]; // first element in the row should always
be a timestamp

        for (var i = 1; i < oldHeader.length; i++) { // start at 1 to avoid
Timestamp column
```

```
var field = oldHeader[i];

var output = getFieldFromData(field, e.parameters);

row.push(output);



var formIndex = fieldsFromForm.indexOf(field);

if (formIndex > -1) {

  fieldsFromForm.splice(formIndex, 1);

}
```

```
}

for (var i = 0; i < fieldsFromForm.length; i++) {

  var field = fieldsFromForm[i];

  var output = getFieldFromData(field, e.parameters);

  row.push(output);

  newHeader.push(field);

}

var nextRow = sheet.getLastRow() + 1; // get next row
```

```
      sheet.getRange(nextRow, 1, 1, row.length).setValues([row]);

      if (newHeader.length > oldHeader.length) {

        sheet.getRange(1, 1, 1,
newHeader.length).setValues([newHeader]);

      }

    }

    catch(error) {

      Logger.log(error);
```

```
  }

  finally {

    lock.releaseLock();

    return;

  }

}

function getDataColumns(data) {
```

```
    return Object.keys(data).filter(function(column) {

      return !(column === 'formDataNameOrder' || column ===
'formGoogleSheetName' || column === 'formGoogleSendEmail' || column
=== 'honeypot');

    });

  }

  function getFieldFromData(field, data) {

    var values = data[field] || '';

    var output = values.join ?
  values.join(', ') : values;
```

```
    return output;


}
```

Contactus.html

```
<!doctype html>

<html>

<head>

  <meta charset="utf-8">

  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

```
<title>Contact Us</title>

</head>

<body>

<br>

<center>

<form class="gform pure-form pure-form-stacked"
method="POST" data-email=""

action="">

<div class="form-elements"><br>
```

```
<div style="background:none;border:8px solid gray;border-radius: 30px;width: 400px;padding-top: 15px;padding-bottom: 40px;padding-left: 20px;padding-right: 20px;"><br>

<b class="content-head" style="font-size: 2.3em;">Contact Us!</b>

<br><br>

<b style="margin-left: -255px;font-size: 1.3em;">Name</b> <br>

<input type="text" name="name" required="" style="font-size: 1em;width: 300px;"/><br><br>

<b style="margin-left: -255px;font-size: 1.3em;">Email</b> <br>

<input type="email" name="email" required="" style="font-size: 1em;width: 300px;"/><br><br>
```

Phone
*
*

<input type="number" name="phone" required=""
*style="font-size: 1em;width: 300px;"/>

*

How can we
*help you?
*

<textarea name="message" rows="5" cols="22" required=""
*style="font-size: 1.3em;width: 300px;"></textarea>

*

<button style="float: right;width: 100px;height: 27px;font-size:
1.1em;margin-right: 48px;">

```
send</button><br>


</div>


</div>


<!-- Thankyou_message --><br>


<div class="thankyou_message"
style="display:none;background:none;border:8px solid gray;border-radius:
40px;width: 400px;padding-top: 15px;padding-bottom: 40px;padding-left:
20px;padding-right: 20px;"><br><br>


<h1>Thanks for contacting us!</h1>


<h1>*******</h1>
```

```
  </div>


</form>


<script data-cfasync="false" type="text/javascript">




 (function() {


function validEmail(email) {


  var re = /^([\w-]+(?:\.[\w-]+)*)@((?:[\w-]+\.)*\w[\w-]{0,66})\.([a-z]{2,6}(?:\.[a-z]{2})?)$/i;


  return re.test(email);


}
```

```
function validateHuman(honeypot) {

  if (honeypot) {

    console.log("Robot Detected!");

    return true;

  } else {

    console.log("Welcome Human!");

  }

}
```

```
function getFormData(form) {

  var elements = form.elements;

  var fields = Object.keys(elements).filter(function(k) {

      return (elements[k].name !== "honeypot");

  }).map(function(k) {

    if(elements[k].name !== undefined) {

      return elements[k].name;

    }else if(elements[k].length > 0){
```

```
    return elements[k].item(0).name;

  }

}).filter(function(item, pos, self) {

  return self.indexOf(item) == pos && item;

});

var formData = {};

fields.forEach(function(name){
```

```
var element = elements[name];

formData[name] = element.value;

if (element.length) {

  var data = [];

  for (var i = 0; i < element.length; i++) {

    var item = element.item(i);

    if (item.checked || item.selected) {

      data.push(item.value);

    }
```

```
            }

            formData[name] = data.join(', ');

        }

    });

    // add form-specific values into the data

    formData.formDataNameOrder = JSON.stringify(fields);

    formData.formGoogleSheetName = form.dataset.sheet ||
"sheetname"; // default sheet name

    formData.formGoogleSendEmail = form.dataset.email || ""; // no
```

```
email by default

    console.log(formData);

    return formData;

  }


  function handleFormSubmit(event) {

    event.preventDefault();

    var form = event.target;

    var data = getFormData(form);
```

```
if( data.email && !validEmail(data.email) ) {

  var invalidEmail = form.querySelector(".email-invalid");

  if (invalidEmail) {

    invalidEmail.style.display = "block";

    return false;

  }

} else {

  disableAllButtons(form);
```

```
        var url = form.action;

        var xhr = new XMLHttpRequest();

        xhr.open('POST', url);

        xhr.setRequestHeader("Content-Type", "application/x-www-form-
urlencoded");

        xhr.onreadystatechange = function() {

            console.log(xhr.status, xhr.statusText);

            console.log(xhr.responseText);

            var formElements = form.querySelector(".form-elements")
```

```
                    if (formElements) {

                      formElements.style.display = "none"; // hide form

                    }

                    var thankYouMessage =
form.querySelector(".thankyou_message");

                    if (thankYouMessage) {

                      thankYouMessage.style.display = "block";

                    }

                    return;

                };
```

```
        var encoded = Object.keys(data).map(function(k) {

            return encodeURIComponent(k) + "=" +
encodeURIComponent(data[k]);

        }).join('&');

        xhr.send(encoded);

    }

}

function loaded() {
```

```
        console.log("Contact form submission handler loaded
successfully.");

        var forms = document.querySelectorAll("form.gform");

        for (var i = 0; i < forms.length; i++) {

          forms[i].addEventListener("submit", handleFormSubmit, false);

        }

      };

      document.addEventListener("DOMContentLoaded", loaded, false);

      function disableAllButtons(form) {
```

```
    var buttons = form.querySelectorAll("button");

    for (var i = 0; i < buttons.length; i++) {

      buttons[i].disabled = true;

    }

  }

})();

  </script>
```

```
</center>


</body>


</html>
```

Example 2: Sample Contact form with CSS attributes

Contactus.html

```
<!DOCTYPE html>
```

```
<html>

<head>

<style>

* {

  box-sizing: border-box;

}



input[type=text], select, textarea {

  width: 100%;
```

```
  padding: 12px;

  border: 1px solid #ccc;

  border-radius: 4px;

  resize: vertical;

}

label {

  padding: 12px 12px 12px 0;
```

```
  display: inline-block;

}

input[type=submit] {

  background-color: #04AA6D;

  color: white;

  padding: 12px 20px;

  border: none;

  border-radius: 4px;
```

```
  cursor: pointer;

  float: right;

}

input[type=submit]:hover {

  background-color: #45a049;

}

.container {
```

```
  border-radius: 5px;

  background-color: #f2f2f2;

  padding: 20px;

}

.col-25 {

  float: left;

  width: 25%;

  margin-top: 6px;
```

```
}



.col-75 {

  float: left;

  width: 75%;

  margin-top: 6px;

}



/* Clear floats after the columns */
```

```
.row::after {

  content: "";

  display: table;

  clear: both;

}

/* Responsive layout - when the screen is less than 600px wide, make the two columns stack on top of each other instead of next to each other */

@media screen and (max-width: 600px) {
```

```
  .col-25, .col-75, input[type=submit] {

    width: 100%;

    margin-top: 0;

  }

}

</style>

</head>

<body>
```

```
<h2>Responsive Form</h2>

<p>Resize the browser window to see the effect.
When the screen is less than 600px wide, make the two columns stack on top of each other instead of next to each other.</p>

<div class="container">

<form action="/action_page.php">

<div class="row">

<div class="col-25">

<label for="fname">First Name</label>

</div>
```

```
      <div class="col-75">

        <input type="text" id="fname" name="firstname"
placeholder="Your name..">

      </div>

    </div>

    <div class="row">

      <div class="col-25">

        <label for="lname">Last Name</label>

      </div>
```

```
<div class="col-75">

<input type="text" id="lname" name="lastname"
placeholder="Your last name..">

</div>

</div>

<div class="row">

<div class="col-25">

<label for="country">Country</label>

</div>
```

```
  <div class="col-75">

    <select id="country" name="country">

      <option value="australia">Australia</option>

      <option value="canada">Canada</option>

      <option value="usa">USA</option>

    </select>

  </div>

</div>

<div class="row">
```

```
                <div class="col-25">

                  <label for="subject">Subject</label>

                </div>

                <div class="col-75">

                  <textarea id="subject" name="subject" placeholder="Write
something.."
                style="height:200px"></textarea>

                </div>

              </div>

              <br>
```

```
    <div class="row">

      <input type="submit" value="Submit">

    </div>

    </form>

  </div>

  </body>

</html>
```

How to Search Google Sheet Contents from HTML Website.

This example displays contents of Google Sh

Code.gs

```
function doGet() { return
HtmlService.createTemplateFromFile("Index").evaluate().setXFrameOptionsMode(HtmlServ
LL); }
```

```
/* PROCESS FORM */
```

```
function processForm(formObject){
```

```
  var result = "";
```

```
  if(formObject.searchtext){///Execute if form passes search text

     result = search(formObject.searchtext);

  }

  return result;

}

//SEARCH FOR MATCHED CONTENTS

function search(searchtext){

  var spreadsheetId   = 'Enter Your Spreadsheet ID'; //** CHANGE !!!
```

```
var dataRage        = 'Data!A2:Y';                          //** CHANGE !!!

var data = Sheets.Spreadsheets.Values.get(spreadsheetId, dataRage).values;

var ar = [];

data.forEach(function(f) {

  if (~f.indexOf(searchtext)) {

    ar.push(f);

  }
```

```
  });

  return ar;

}
```

Page.html

```
<!DOCTYPE html>

<html>

  <head>

    <base target="_top">
```

```
                <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>




                <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.bundle.min.
js" integrity="sha384-
xrRywqdh3PHs8keKZN+8zzc5TX0GRTLCcmivcbNJWm2rs5C8PRhcEn3czEjhAO9o"
crossorigin="anonymous"></script>









                <!--##JAVASCRIPT FUNCTIONS ----------------------------------------------
--- -->



                <script>



                  //PREVENT FORMS FROM SUBMITTING / PREVENT DEFAULT
BEHAVIOUR
```

```
function preventFormSubmit() {

  var forms = document.querySelectorAll('form');

  for (var i = 0; i < forms.length; i++) {

    forms[i].addEventListener('submit', function(event) {

    event.preventDefault();

    });

  }

}

window.addEventListener("load", preventFormSubmit, true);
```

```
//HANDLE FORM SUBMISSION

function handleFormSubmit(formObject) {

google.script.run.withSuccessHandler(createTable).processForm(formObject);

  document.getElementById("search-form").reset();

}
```

```
//CREATE THE DATA TABLE

function createTable(dataArray) {

if(dataArray && dataArray !== undefined && dataArray.length
!= 0){

var result = "<table class='table table-sm table-striped'
id='dtable' style='font-size:0.8em'>"+

"<thead style='white-space: nowrap'>"+

"<tr>"+ //Change table headings
to match witht he Google Sheet

"<th scope='col'>EMAIL</th>"+

"<th scope='col'>PROJECT TITLE</th>"+
```

```
"<th scope='col'>AUTHOR</th>"+

"<th scope='col'>PRICE</th>"+

"<th scope='col'>UNITS</th>"+

"<th scope='col'>TOTAL</th>"+

 "<th scope='col'>ESTIMATED REVENUE</th>"+

"<th scope='col'>BENEFICIARY</th>"+

"<th scope='col'>BANK</th>"+

"<th scope='col'>ACCOUNT NUM</th>"+

"<th scope='col'>SWIFT CODE</th>"+
```

```
                "<th scope='col'>COUNTRY</th>"+

                "<th scope='col'>PAYMENT STATUS</th>"+

              "</tr>"+

            "</thead>";

for(var i=0; i<dataArray.length; i++) {

    result += "<tr>";

    for(var j=0; j<dataArray[i].length; j++){
```

```
            result += "<td>"+dataArray[i][j]+"</td>";

        }

        result += "</tr>";

    }

    result += "</table>";

    var div = document.getElementById('search-results');

    div.innerHTML = result;

}else{

    var div = document.getElementById('search-results');
```

```
                    //div.empty()

                    div.innerHTML = "Data not found!";

                  }

                 }

                </script>

                <!--##JAVASCRIPT FUNCTIONS ~ END ------------------------------------
------------ -->

            </head>
```

<body>

<!-- ## SEARCH FORM -- -->

<form id="search-form" class="form-inline" onsubmit="handleFormSubmit(this)">

* *

<label for="searchtext"style="font-size:30px;">Sales and Payments</label>

* *

* *

<input type="text" class="form-control" id="searchtext" name="searchtext" placeholder="Enter Email Address"style="width: 50%;

color: black;

padding: 14px 20px;

margin: 8px 0;

```
        border-radius: 20px;width: 250px

        border-radius: 4px;

        cursor: pointer;" >

</div>

<button type="submit"style="width: 50%;

        background-color: #4CAF50;

        color: #FFFFFF;

        padding: 14px 20px;
```

```
                                margin: 8px 0;



                                border: none;



                                border-radius: 4px;



                                cursor: pointer;"  class="btn btn-primary mb-
2">SALES OVERVIEW</button>



                            </form>



                            <!-- ## SEARCH FORM ~ END ----------------------------------------
-- -->









                        </div>



                    </div>
```

```
                    <div class="row">

                      <div class="col">

                        <!-- ## TABLE OF SEARCH RESULTS ---------------------------------
------------ -->

                        <div id="search-results" class="table-responsive">

                          <!-- The Data Table is inserted here by JavaScript -->

                        </div>

                        <!-- ## TABLE OF SEARCH RESULTS ~ END ------------------------
---------------------- -->
```

```
        </div>

      </div>

    </div>

  </body>

</html>
```

Conclusion

JavaScript is a dynamic programming language for computers. It is most frequently utilized as a lightweight component of web pages, whose implementations enable client-side script to create dynamic pages and engage with the user. It is an object-oriented programming language that is interpreted.

The fact that JavaScript doesn't require pricey development tools is one of its main advantages. A basic text editor like Notepad can be used as a starting point. You don't even need to purchase a compiler because it is an interpreted language that runs within a web browser.

Millions of Web pages utilize JavaScript to enhance their appearance, verify forms, identify browsers, set cookies, and do a lot more. The most widely used programming language on the Internet, JavaScript is compatible with all of the main browsers, including Internet Explorer, Mozilla Firefox, and Opera.

References

Ecma International. (2024). "*ECMAScript® 2025 Language Specification*". *27 March 2024.*

Google Workspace. (2024). *Google Apps Script overview.* Retrieved May 23, 2024, from Apps Script.

Netscape, S. (2007). "*Netscape and Sun announce JavaScript, the Open, Cross-platform Object Scripting Language for Enterprise Networks and the Internet*" . https://web.archive.org/web/20070916144913/https://wp.netscape.com/newsref/pr/newsrelease67.html.

W3C. (2021, 5 31). *World Wide Web Consortium* "*How to code: The best ways to learn programming in 2021*" . Retrieved 12 1, 2022

JAVASCRIPT

Comprehensive Manual for Developing Dynamic and Responsive Website and Applications.

Suitable for Novice and Experts.

igital Services