**Console Logs, Errors, Warnings & More | JavaScript Tutorial In Hindi #2**

In this tutorial, we will learn how to run **JavaScript** in the Chrome Console. We can open our console in the web browser by using: ***Ctrl + Shift + K***or by Right-click on any webpage, click *Inspect*, and then we can see the innards of that site; its source code, the CSS that form its design, the JavaScript code that powers animations, and more. It has a console option as well, where we can run our JavaScript code.

In JavaScript, a **console** is an object which provides access to the browser debugging console. This object provides us with several different methods like **log(), error() ,table()** etc. Each method provides different functionalities. Following is the description of these methods along with examples.

**Console.log():-**

This method is used to log(print) the output to the console. We can put anything inside the log(). It can be an array, object, string, boolean, etc.

**Example:-**

console.log('CodeWithHarry');

console.log(1);

console.log(true);

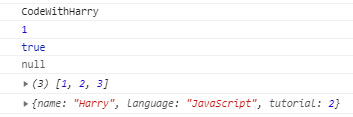
console.log(null); ;

console.log([1, 2, 3]); // array inside log

console.log({name:"Harry", language:"JavaScript", tutorial:2}); // object inside log

Copy

**Output:-**



**Console.table ():-**

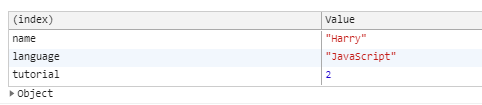
To generate a table inside a console, we use console.table() method. The input must be an array or an object which will be displayed as a table. In the example, we provide the object as an input.

**Example:-**

console.table({name:"Harry", language:"JavaScript", tutorial:2});

Copy

**Output:-**



**Console.assert():-**

This method writes a message to the console that the assertion failed and the message we provide as a parameter, but only if an expression evaluates to *false*. If the expression is true, then nothing will happen.

**Example:-**

console.assert(0>1, "Expression is false")

Copy

**Output:-**

https://api.codewithharry.com/media/videoSeriesFiles/courseFiles/javascript-tutorials-in-hindi-2/base64_94UhtyY.png

**Console.warn():-**

This method is used to log a warning message to the console. By default, the warning message will be highlighted with yellow color.

**Example:-**

console.warn("This is a warning");

Copy

**Output:-**

https://api.codewithharry.com/media/videoSeriesFiles/courseFiles/javascript-tutorials-in-hindi-2/base64_2lVHg0r.png

**Console.clear():-**

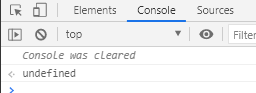
It is used to clear the console. The console will be cleared. In the case of Chrome, a simple overlayed text will be printed on the console.

**Example:-**

console.clear();

Copy

**Output:-**



**Console.time() and Console.timeEnd():-**

With the help of console.time() and console.timeEnd() we can find the amount of time spend by a code on execution.

**Example:-**

console.time();

for (i = 0; i < 100; i++) {

// code

}

console.timeEnd();

Copy

**Output:-**

https://api.codewithharry.com/media/videoSeriesFiles/courseFiles/javascript-tutorials-in-hindi-2/base64_SOCcrce.png

**Console.error():-**

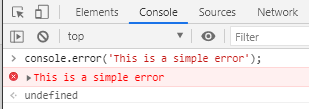
Used to log error message to the console. Useful in the testing of code. By default, the error message will be highlighted with red color.

**Example:-**

console.error("This is a simple error");

Copy

**Output:-**



**Console.count():-**

The console.count() method is used to count the number that the function hit by this counting method.

**Example:-**

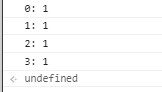
for (i = 0; i<4; i++) {

console.count(i);

}

Copy

**Output:-**



**Console.group() and Console.groupEnd():-**

group() and groupEnd() methods of the console object allow us to group contents in a separate block, indented. Just like the time() and the timeEnd(), they also accept the label, again of the same value.

**Example:-**

console.group('simple');

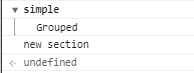
console.log('Grouped');

console.groupEnd('simple');

console.log('new section');

Copy

**Output:-**



**Custom Console logs:-**

If the user has even a little idea about CSS, they can add Styling to the console logs to make logs Custom. The Syntax for it is to add the CSS styling as a parameter to the logs, which will replace %c in the logs as shown in the example below:

**Example:-**

const spacing = '8px';

const mystyle =

`padding: ${spacing}; background-color: white; color: blue ; font-style:

italic; border: 1px solid black dotted; font-size: 2em;`;

console.log('%cCode With Harry', mystyle);

Copy

**Output:-**

https://api.codewithharry.com/media/videoSeriesFiles/courseFiles/javascript-tutorials-in-hindi-2/base64_SuZEB0r.png

**Code index.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Tutorial on Js</title>

</head>

<body>

<h1>This is Js tutorial by Harry</h1>

</body>

<!-- <script src="js/tut2.js"></script> -->

<!-- <script src="js/tut3.js"></script> -->

<!-- <script src="js/tut4.js"></script> -->

<!-- <script src="js/tut5.js"></script> -->

<!-- <script src="js/tut6.js"></script> -->

<!-- <script src="js/tut7.js"></script> -->

<!-- <script src="js/tut8.js"></script> -->

<!-- <script src="js/tut9.js"></script> -->

<!-- <script src="js/tut10.js"></script> -->

<script src="js/tut11.js"></script>

</html>

Copy

**Js code as described/written in the video**

console.time('Your code Took');

console.log('Hello console');

console.log(4+34);

console.log(34);

console.log(true);

console.log([34,2,1,2]);

console.log({harry: 'this', marks:34});

console.table({harry: 'this', marks:34});

console.warn('This is a warning');

// console.clear();

console.timeEnd('Your code Took');

// console.assert(566<189, 'Age >189 is not possible')

// console.error('This is an error')

/\*

this

is a

multiline comment

\*/

**Variables: let, const & var in JavaScript | JavaScript Tutorial In Hindi #3**

In today's tutorial, we are going to study ***variables in JavaScript*** and their different types. As we know, that variable is the name of the storage location. When we want to save some data, we store it in a variable. In any programming language, we typically do lots of calculations. The calculation results are stored in the computer's memory. Just like human memory, the memory of the computer also consists of millions of cells. The calculated values are stored in these memory cells. To make the usage and retrieval of these values easy, these memory locations are given names. The names given to these locations are called variables.

Data types in JavaScript are either Variables or Constants. ES6 has made major changes in JavaScript's syntax and has also brought new features. Initially, we used to declare variables with a keyword **"var".** However, ES6 has brought a new variable declaration keyword, ***"let"*** and ***"const."***

**Following are some rules while declaring a JavaScript variable:**

1. A variable name must start with a letter (a to z or A to Z), underscore (\_), or dollar( $ ) sign.
2. A variable name cannot start with a number. After the first letter, we can use digits (0 to 9), for example, message1.
3. JavaScript variables are case sensitive. For example, a and A are different variables.

So, let us explore these variable declaration keywords in detail.

**Var:-**

Before the advent of ES6, var declarations were used to declare a variable. The properties of var is that it has visibility linked to the function to which it belongs. We can change its value, and it can be redeclared in the same scope. Scope means where these variables are available for use. There are two types of scope, local and global. Var declarations are globally scoped, and when they are defined inside a function, they are locally scoped.

**Example of var;-**

var age = 25; // Number

var name = "John"; // String

var developer = true;// Boolean

var location = null; // Null

var task; // undefined

Copy

When the variable is declared, the JavaScript engine assigns it a memory or space. Because of this, once a variable is declared, it takes a value of undefined even before the assignment. We assigned data to the variable by using the assignment operator **"="**. Datatypes in JavaScript are:

1. Number i.e., 11,23,45,6
2. Strings, i.e., "Hello World."
3. Boolean, i.e., true, false
4. Undefined
5. Null
6. Any of the complex data structures (Array and Objects)

**let:-**

The variable type let is introduced in ES6. It shares a lot of similarities with var, but unlike var, it has scope constraints. Its declaration and assignment are similar to var. The purpose of introducing let is to resolve all issues posed by variables scope, which developers face during development. The properties of let are that They have visibility linked to the block they belong with. We can change their values, but they cannot be redeclared in the same scope, unlike var.

"let" helps us by making it easier to see where variables live in our code and make our code cleaner and easier to read.

**Example of let:-**

let age = 25; // Number

let name = "John"; // String

let developer = true;// Boolean

let location = null; // Null

let task; // undefined

let age= 50;

console.log(age); // SyntaxError: identifier "age" has already been declared.

Copy

**Const:-**

Const is also introduced in ES6. It is a variable type assigned to data whose value cannot and will not be changed throughout the program. Const is more strict as compared to var and let. Const is also limited to the scope in which it operates. We declare const just like var and let. We use const when we are sure a variable will not be redeclared. The characteristic of const and their declarations are block-scoped, and they cannot be updated or redeclared.

**Example:-**

const age = 20;

const job = 'developer';

const name; // SyntaxError: missing initializer

const num = 10;

num = 20; //Compiler Error: Cannot assign to 'num' because it is a constant or read-only property

Copy

**Single line comments in JavaScript:-**

To create a single line comment in JavaScript, we have to place two slashes "//" in front of the code or text that we want the interpreter to ignore. When we place these two slashes, all text will be ignored, until the next line. Single line comments are used for writing small notes in the program

// This is a single line JavaScript comment

Copy

**Multi-line comments in JavaScript:-**

Although a single line comment is quite useful, but when we want to comment the long segment of code, we have to use multiline comment.  Multiline comment begins with /\* and ends with \*/.

/\*

This is a Multiline Comment

This is a Multiline Comment

This is a Multiline Comment

\*/

Copy

**Code index.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Tutorial on Js</title>

</head>

<body>

<h1>This is Js tutorial by Harry</h1>

</body>

<!-- <script src="js/tut2.js"></script> -->

<!-- <script src="js/tut3.js"></script> -->

<!-- <script src="js/tut4.js"></script> -->

<!-- <script src="js/tut5.js"></script> -->

<!-- <script src="js/tut6.js"></script> -->

<!-- <script src="js/tut7.js"></script> -->

<!-- <script src="js/tut8.js"></script> -->

<!-- <script src="js/tut9.js"></script> -->

<!-- <script src="js/tut10.js"></script> -->

<script src="js/tut11.js"></script>

</html>

Copy

**Js code as described/written in the video**

console.log('tut3');

// Variables in js

// var, let, const

var name = 'harry';

var channel;

var marks = 3454;

marks = 0;

channel = 'CodeWithHarry'

console.log(name, channel, marks);

// Rules for creating JavaScript Variables

/\*

1. Cannot start with numbers

2. Can start with letter, numbers, \_ or $

3. Are case sensitive

\*/

var city = 'Delhi';

console.log(city);

const ownersName = 'Hari Ram';

// ownersName = 'Harry'; // Cannot do this (error)

console.log(ownersName);

const fruit = 'Orange';

{

let city = 'Rampur';

city = 'Kolkata';

console.log(city);

}

console.log(city);

const arr1 = [12,23,12,53, 3];

// arr1.push(45);

console.log(arr1)

/\* Most common programming case types:

1. camelCase

2. kebab-case

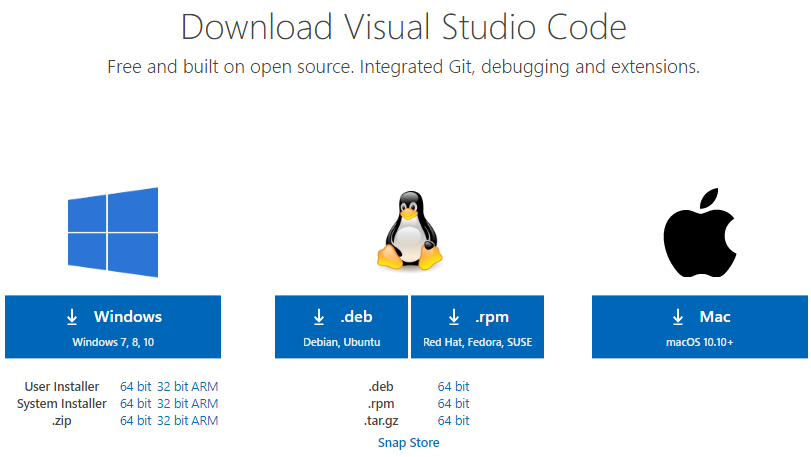
3. snake\_case

4. PascalCase

\*/

**Data Types in JavaScript (Primitive & Reference Types) | JavaScript Tutorial In Hindi #4**

To run a**JS program**, we need an IDE like Visual Studio Code or Codeblocks. For this series, we are using Virtual Studio Code (VS Code). ***Visual Studio Code*** is a fast source code editor and provides the tools that a developer needs for a quick code-build-debug cycle. To download VS Code, click on [***Download Virtual Studio Code***](https://code.visualstudio.com/download) . For guidance, check the tutorial [***Installing VS Code, Extensions & Setup***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-1)



In today's tutorial, we will study about the ***primitive and reference datatype*** and their differences. In JavaScript, a variable stores two types of values: primitive and reference. JavaScript provides six primitive types like Boolean, number, string, undefined, null, symbol, and a reference type as object literals, array, functions, and dates. When we assign a value to a variable, the JavaScript engine will determine whether the value belongs to primitive or reference datatype. We do not have to specify the type of data that whether it is a string or an integer.

**What are Primitive and Reference Types in JavaScript?**

**Primitive types:-**

Primitive data types are numbers, Booleans, strings, undefined, null, and symbol. **Primitive data types** are the **basic** or **common** data types in JavaScript. Following is the table of primitive data types in JavaScript.

| **Primitive Data Type** | **Meaning** |
| --- | --- |
| var x; | undefined |
| var x=undefined; | undefined |
| var x=null; | null type data |
| var x=3; | number. |
| var x=6.5 | number |
| var x="5" | string |
| var x='5' | string |
| var x="Hello World" | string |
| var x=true | Boolean |
| var x=false; | Boolean |

**Here is an example:**

let number = 20

Copy

The number variable stores a number value i.e. 20. Number values are called "primitive values" because they are simple building blocks of JavaScript apps.

let name = 'Harry'

let isMale = true

Copy

String and Booleans are also primitive types so we can also declare variables using var and const.

**Reference Types**

Reference in JavaScript are datatypes based on primitive. Like Objects, Arrays, and Functions. Everything is JavaScript is either an Object or Primitive datatype. When we create an object, that value is not directly assigned to the variable. Instead, a reference to that value is what gets set. Variable knows about the location of the object in memory, not the object itself. Following is the table of the reference data type.

| **Reference Data Type** | **Meaning** |
| --- | --- |
| var y=[ "March", "April", "May" ] | Array |
| var y={ name : "Harry",  age : 22,  gender : "male" } | Object |
| var y=function(){ } | Function on |
| var y=new Date(); | Date |

**Here’s an example:-**

var student = {

name: 'Harry',

age: 20,

} //object

var sports= ['Tennis', 'Cricket']//array

Copy

Here, a student is an object and, therefore, a so-called reference type. The student object holds properties that have primitive values. This does not affect the object being a reference type, though.

The sports array is also a reference type - in this case, it holds a list of strings. A string is a primitive datatype, but it does not affect the array. Arrays are reference types.

**What is the difference between primitive and reference datatype?**

* JavaScript stores the primitive value on the stack because the size of a primitive value is fixed. On the other hand, JavaScript stores the reference value on the heap because the size of the reference value is dynamic.
* One of the most significant differences between primitive data and reference data is that, If the value is primitive, then we manipulate the **actual value**stored in that variable. Whereas, If the value is of reference data type, we can manipulate that object's reference, rather than the actual object. It means a variable that stores an object is **accessed by reference**.

**Code index.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Tutorial on Js</title>

</head>

<body>

<h1>This is Js tutorial by Harry</h1>

</body>

<!-- <script src="js/tut2.js"></script> -->

<!-- <script src="js/tut3.js"></script> -->

<!-- <script src="js/tut4.js"></script> -->

<!-- <script src="js/tut5.js"></script> -->

<!-- <script src="js/tut6.js"></script> -->

<!-- <script src="js/tut7.js"></script> -->

<!-- <script src="js/tut8.js"></script> -->

<!-- <script src="js/tut9.js"></script> -->

<!-- <script src="js/tut10.js"></script> -->

<script src="js/tut11.js"></script>

</html>

Copy

**Js code as described/written in the video**

// Primitive data types

// String

let name = "harry";

console.log("My string is " + name);

console.log("Data type is " + (typeof name));

// Numbers

let marks = 34.4;

console.log("Data type is " + (typeof marks));

// Boolean

let isDriver = true;

console.log("Data type is " + (typeof isDriver));

// Null

let nullVar = null;

console.log("Data type is " + (typeof nullVar));

// Undefined

let undef = undefined;

console.log("Data type is " + (typeof undef));

// Reference Data Types

// Arrays

let myarr = [1, 2, 3, 4, false, "string"];

console.log("Data type is " + (typeof myarr));

// Object Literals

let stMarks = {

harry: 89,

Shubham: 36,

Rohan: 34

}

console.log( typeof stMarks);

function findName() {

}

console.log( typeof findName);

let date = new Date();

console.log( typeof date);

**JavaScript Type Conversion & Coercion | JavaScript Tutorial In Hindi #5**

In this tutorial, we will learn about type conversions and coercion in JavaScript with examples. JavaScript is a programming language used to build dynamic web pages. In this language, we do not have to specify the data type when we declare it. We can assign new data of a different type to the same variable. Under some circumstances, JavaScript will perform automatic type conversion. Typecasting/Type conversion and coercion in simple terms means changing the data type of a value to another data type like an integer to string, Boolean into String, etc.

**JavaScript Type Conversion:-**

When we convert one data type to another data type, this process is called type conversion. In JavaScript, there are two types of type conversion.

**Explicit Conversion:-**

The type conversion that we manually do is known as explicit type conversion. In JavaScript, explicit type conversions are done using the built-in methods like String (), Number(), etc.

* **Number Conversion:-**To convert Boolean values or numeric strings to numbers, we use the Number() i.e. an in-built method. Number() method in case of empty strings and null values return **0.** If a string is an invalid number like having an alphabet in a string, the result will be NaN.

**The following are the rules of the numeric value:**

| **Value** | **Return** |
| --- | --- |
| undefined | NaN |
| null | 0 |
| true and false | 1 and 0 |
| string | Whitespaces from the start and end are removed. If the remaining string is empty,  the result is 0. Otherwise, the number is “read” from the string.  If the string contain any alphabet like 67a90, it will give NaN error. |

**For example:-**

let res;

// string to number

res = Number('100');

console.log(res); // 100

// boolean to number

res = Number(false);

console.log(res); // 0

res = Number(' ')

console.log(res); // 0

res = Number('hello');

console.log(res); // NaN

res = Number(undefined);

console.log(res); // NaN

Copy

* **Boolean Conversion :-**Boolean type conversion happens in logical operations. It also follows the rules, but they are mostly obvious:
* NaN, 0, undefined, null, "" are converting to false
* everything else, including objects, to true

**For Example:-**

res=Boolean(1); // true

res= Boolean(0); // false

res= Boolean("hello"); // true

res= Boolean(""); // false

Copy

* **String Conversion:-**The method String() is used to convert numbers to strings. It can be used on any type of numbers, literals, variables, or expressions. The method toString() does the same.

**Example:-**

let res

let a=90

res= String(a) // returns a string from a number variable a

String(378009) // returns a string from a number literal 378009

a.toString()

(378009).toString()

Copy

* **parseInt and parseFloat:-**The in-built method parseInt() and parseFloat() return numbers from strings that start with numeric data. Here is an examples:

console.log( parseInt('$100 dollars') ); // NaN

console.log( parseInt('+10.25990 kg') ); // 10

console.log( parseFloat(' +10.25 kg') ); // 10.25

console.log( parseFloat('ABC’) ); //NaN

Copy

**Implicit Conversion:-**

Sometimes JavaScript automatically converts one data type to another. This type of conversion is known as implicit conversion.

* **Conversion To String:-**When we add a number into a string, JavaScript converts the number to a string before concatenation. Here is an example:

let res;

res = '3' + 4;

console.log(res) // "34"

res = '9' + true;

console.log(res); // "9true"

res= '0' + null;

console.log(res); // "0null"

Copy

* **Conversion To Number:-**Numeric string used with operators like +, - , / , \* returns number type

let res

res = '4' - '4';

console.log(res); // 0

res = '4' \* 5;

console.log(res); // 20

Copy

* **Boolean Conversion to Number:-**If we use Boolean, true is considered as 1 and false is considered as 0.

let res;

res = '5' - true;

console.log(res); // 4

res = 10 + false;

console.log(res); // 10

Copy

In JavaScript, Null is considered as 0 when used with numbers. Arithmetic operation of undefined with number, boolean or null returns NaN

res = 4 + null; // 4

res = 4 - undefined;// NaN

Copy

**Code index.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Tutorial on Js</title>

</head>

<body>

<h1>This is Js tutorial by Harry</h1>

</body>

<!-- <script src="js/tut2.js"></script> -->

<!-- <script src="js/tut3.js"></script> -->

<!-- <script src="js/tut4.js"></script> -->

<!-- <script src="js/tut5.js"></script> -->

<!-- <script src="js/tut6.js"></script> -->

<!-- <script src="js/tut7.js"></script> -->

<!-- <script src="js/tut8.js"></script> -->

<!-- <script src="js/tut9.js"></script> -->

<!-- <script src="js/tut10.js"></script> -->

<script src="js/tut11.js"></script>

</html>

Copy

**Js code as described/written in the video**

// Type conversion

console.log('Welcome to tut5');

let myVar;

myVar = String(34);

// console.log(myVar, (typeof myVar));

let booleanVar = String(true);

// console.log(booleanVar, (typeof booleanVar));

let date = String(new Date());

// console.log(date, (typeof date));

let arr = String([1,2,3,4,5]);

// console.log(arr.length, (typeof arr));

let i = 75;

// console.log(i.toString())

let stri = Number("3434");

stri = Number("343d4");

stri = Number(false);

stri = Number([1,2,3,4,5,6,7,8,9]);

// console.log(stri, (typeof stri));

let number = parseFloat('34.098');

console.log(number.toFixed(2), (typeof number));

// Type coercion

let mystr = Number("698");

let mynum = 34;

console.log(mystr + mynum);

Copy

**Previous**

**Strings: Properties, Methods & Template Literals in JavaScript | JavaScript Tutorial In Hindi #6**

In today's tutorial, we will study about ***string properties and methods in JavaScript.*** As we know, strings are useful for holding data that can be represented in text form. One of the most popular operations on strings are to check their length, to build and concatenate them using the operator (+), checking for the existence or location of substrings with the indexOf() method, or extracting substrings with the substring() method. In this tutorial, we will also learn about ***JavaScript template literals***that makes it easier to work with the string template.

**Note that:**JavaScript counts positions from zero. 0 is the first position in a string, 1 is the second, and so on.

**JavaScript String Methods:-**

Let's see the JavaScript string methods with examples.

* **charAt(c):**It returns the character at the “c” position within the string.

var myString = 'JavaScript!!!';

console.log(myString.charAt(7));

//output: i

Copy

* **concat(p1, p2):** Combines one or more strings and returns the concatenated string. Remember that the original string is not modified.

var str1="JavaScript"

var str2=str1.concat(" is","awesome")

console.log(str2);

//Output: JavaScript is awesome

Copy

* **indexOf(substr, [start\_from]):** This method Searches and returns the index number of the searched character within the string. If not found, it will return -1 . “Start\_from” is an optional argument specifying the position within string to begin the search. Default is 0.

//indexOf(char/substring)

var str1="Hi, my name is Sam!";

var str2 = str1.indexOf("locate");

//Output: -1

Copy

* **lastIndexOf(substr, [start\_from]):**This method searches and  returns the index number of the searched character within the string. This method will return the index of the last occurrence of a specified text in a string.If the character or string is not found, -1 is returned. "Start\_from" is an optional argument specifying the position within string to begin the search. Default is string.length-1

var myString = 'javascript Node.js';

console.log(myString.lastIndexOf('N'));

//output: 11

Copy

* **slice(start, [end]):** This method returns a substring of the string based on the "start" and "end" index , it will not include the "end" index itself. "End" argument is optional, and if none is specified, the slice includes all characters from "start" to end of string.

var text="programming"

var mystr= text.slice(0,4)

console.log(mystr)

//Output:- "prog"

Copy

* **split(delimiter, limit):** This method Splits a string into substring to the specified delimiter, and returns an array containing each element. The argument “limit” is an integer that lets you specify the maximum number of elements to return.

var txt = "a,b,c,d,e"; // String

txt.split(","); // Split on commas

txt.split(" "); // Split on spaces

Copy

* **substring(from, [to]):** This method will returns the characters in a string between “from” and “to” indexes. It will not include “to” inself. “To” argument is optional, and if omitted, up to the end of the string is assumed.

//substring(from, [to])

var myString = 'javascript Programming';

myString = myString.substring(0,10);

console.log(myString)

//output: javascript

Copy

* **toLowerCase():**This method will Returns the string with all of its characters converted to lowercase.

//toLowerCase()

var myString = 'JAVASCRIPT';

myString = myString.toLowerCase();

console.log(myString)

//output: javascript

Copy

* **toUpperCase():** This method will Returns the string with all of its characters converted to uppercase.

//toUpperCase()

var myString = 'javascript';

myString = myString.toUpperCase();

console.log(myString)

//output: JAVASCRIPT

Copy

* **search(“str”):** The method searches a string for a specified value and returns the position of the match:

var str = "Welcome to programming World!!";

var pos = str.search("programming");

Copy

* **substr() Method:**The method substr() is similar to slice(). The only difference is that the second parameter specifies the length of the extracted part.

var str = "Apple, Banana, Kiwi";

var res = str.substr(7, 6);

// Output: Banana

Copy

**JavaScript template literals:-**

Prior to ES6, we use single quotes (') or double quotes (") to wrap a string literal. At that time, the strings have very limited functionality. To help us in solving more complex problems, ES6 template literals provide the syntax that allows you to work with strings in a much cleaner way. In ES6, we can create a template literal by wrapping the string in backticks as follows:

let simple = `This is an example of

Template literal`;

Copy

Following are the features of javascript template literals:

1. We can write a Multiline string
2. It provides the ability to substitute part of the string for the values of variables or expressions. This feature is also called string interpolation.
3. It provides the ability to transform a string so that it is safe to include in HTML.

let firstName = 'Code With',

lastName = 'Harry';

let greeting = `Welcome to ${firstName} ${lastName}`;

console.log(greeting);

//Welcome to Code With Harry

Copy

**Code index.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Tutorial on Js</title>

</head>

<body>

<h1>This is Js tutorial by Harry</h1>

</body>

<!-- <script src="js/tut2.js"></script> -->

<!-- <script src="js/tut3.js"></script> -->

<!-- <script src="js/tut4.js"></script> -->

<!-- <script src="js/tut5.js"></script> -->

<!-- <script src="js/tut6.js"></script> -->

<!-- <script src="js/tut7.js"></script> -->

<!-- <script src="js/tut8.js"></script> -->

<!-- <script src="js/tut9.js"></script> -->

<!-- <script src="js/tut10.js"></script> -->

<script src="js/tut11.js"></script>

</html>

Copy

**Js code as described/written in the video**

console.log('We are at tut 6');

const name = 'Harry';

const greeting = 'Good Morning';

// console.log(greeting + ' ' + name);

let html;

html = "<h1> this is heading</h1>"+

"<p> this is My para</p>";

html = html.concat('this', ' str2');

console.log(html);

// console.log(html.length);

// console.log(html.toLowerCase());

// console.log(html.toUpperCase());

// console.log(html);

// console.log(html[1]);

// console.log(html.indexOf('<'));

// console.log(html.lastIndexOf('<'));

// console.log(html.charAt(3));

// console.log(html.endsWith('dsfsdfd'));

// console.log(html.includes(' fg'));

// console.log(html.substring(1,6));

// console.log(html.slice(0, 4))

// console.log(html.split('>'));

// console.log(html.replace('this', 'it'));

let fruit1 = 'Orange\'';

let fruit2 = 'Apple';

let myHtml = `Hello ${name}

<h1> This is "my" heading </h1>

<p> You like ${fruit1} and ${fruit2}`;

document.body.innerHTML = myHtml;

**Arrays and Objects In JavaScript | JavaScript Tutorial In Hindi #7**

In today's tutorial, we will study about arrays and objects in JavaScript. If you are familiar with any programming language, then you have a fair idea about arrays. If you do not have any experience with any other programming language, then do not worry. We are going to study about arrays and object in JavaScript in detail in this tutorial.

**Arrays in JavaScript:-**

Arrays are the objects in which we can store multiple values in a single variable. An array stores a sequential collection of elements of fix sizes. There are two syntaxes used for creating an empty array:

let arr = new Array();

let arr = [];

Copy

**Accessing array elements:-**

Arrays are zero-indexed. The first element of an array is store at 0 indexes and the second element store at index 1 and so on. If we use an invalid index number it returns undefined.

let arr = ['first element', 'second element', 'last element']

console.log(arr[0]) // output ' first element'

console.log(arr[1]) // output ' second element'

console.log(arr[arr.length - 1]) // output ' last element'

Copy

**Following are some arrays method:**

* **Length of an Array:-**The property lengthsets or returns the number of elements in an array.

let age = [33,55,67,18]

console.log(age.length)

Copy

* **indexOf():-** This method Search the array for an element and returns its position

let std = ["Mark", "John", "Jonas", "Jack"];

let a = std.indexOf("John");

Copy

* **sort():** This method is used to sort the elements of an array

let age = [33,55,67,18]

let s\_age= age.sort(age)

//18,33,55,67

Copy

* **reverse():** This method is used to reverses the element of an array. The first array element becomes the last and the last becomes the first.

let age = [33,55,67,18]

let r\_age= age.reverse(age)

//18,67,55,33

Copy

* **concat():** This method will returns a new array comprised of this array joined with an other array or value

let alpha = ["a", "b", "c"];

let numeric = [1, 2, 3];

var alphaNumeric = alpha.concat(numeric); // a,b,c,1,2,3

Copy

Here are some methods which are used to modify an array. Remember that, when we modify an array, we are modifying the original array.

* **push():** This method is used to add an item to the end of an Array

let fruits = ["Banana", "Orange", "Apple"];

let len = fruits.push('Mango')

// ["Apple", "Banana", "Orange", "Mango"]

Copy

* **pop():-**This method is used to remove an item from the end of an Array

let last = fruits.pop() // remove Mango (from the end)

["Apple", "Banana", "Orange"]

Copy

* **shift():-**This method is used to remove an item from the beginning of an Array

let first = fruits.shift() // remove Apple from the front

// ["Banana", "Orange"]

Copy

* **unshift():-**This method is used to add an item to the beginning of an Array

let newLength = fruits.unshift('Strawberry') // add to the front

// ["Strawberry", "Banana", "Orange"]

Copy

* **splice():**This method is used to remove an item by index position

let removedItem = fruits.splice(pos, 1) // this is how to remove an item

// ["Strawberry", "Orange"]

Copy

**JavaScript Object:-**

In JavaScript, Object is a non-primitive data type. The object is like other variables, but the only difference is that an object holds multiple values, arrays, functions etc. We create an object with figure brackets {…} with an optional list of *properties*. A property is a "key: value" pair, where the key is a string/ property name, and value can be anything.

**The syntaxs of creating object are following:-**

let user = new Object(); // "object constructor" syntax

let user = {}; // "object literal" syntax

Copy

**Example :-**

let user = {

name: "Harry", // by key "name" store value "Harry"

age: 25, // by key "age" store value 25

language: “JavaScript” // by key "language" store value “ JavaScript”

};

Copy

In the user object, there are three properties:

* The first property has the name "name" and the value "Harry".
* The second property has the name "age" and the value 25.
* The third one has the name "language" and the value "JavaScript".

**The syntax for accessing the property of an object is:**

1. **The dot notation (.):**The syntax of dot notation to access a property of an object is:

objectName.property

Copy

For example, to access the "name" property of the user object, we use the following expression:

user.name

Copy

1. **Array-like notation ( []):**The syntax to access the value of an object’s property via the bracket notation is:

objectName['propertyName'];

Copy

For example:

console.log(user['name']);

console.log(user['age']);

Copy

To change the value of a property, you use the assignment operator. For example:

user.age = 27;

Copy

**Code index.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Tutorial on Js</title>

</head>

<body>

<h1>This is Js tutorial by Harry</h1>

</body>

<!-- <script src="js/tut2.js"></script> -->

<!-- <script src="js/tut3.js"></script> -->

<!-- <script src="js/tut4.js"></script> -->

<!-- <script src="js/tut5.js"></script> -->

<!-- <script src="js/tut6.js"></script> -->

<!-- <script src="js/tut7.js"></script> -->

<!-- <script src="js/tut8.js"></script> -->

<!-- <script src="js/tut9.js"></script> -->

<!-- <script src="js/tut10.js"></script> -->

<script src="js/tut11.js"></script>

</html>

Copy

**Js code as described/written in the video**

console.log('We are in tut7.js and lets discuss about arrays');

let marks = [34, 23, 24, 93 ,73, 25];

const fruits = ['Orange', 'Apple', 'Pineapple'];

const mixed = ['str', 89, [3, 5]];

const arr = new Array(23,123,21, 'Orange');

// console.log(marks);

// console.log(mixed);

// console.log(fruits[1]);

// console.log(arr.length);

// console.log(Array.isArray('dfdf'));

arr[0] = 'harry';

let arrelement = arr[0];

// console.log('element :', arrelement);

// console.log(arr);

let value = marks.indexOf(73);

// console.log(value)

// Mutating or Modifying arrays

// marks.push(3564);

// marks.unshift(1009);

// marks.pop()

// marks.shift()

// marks.splice(2, 3);

// marks.reverse()

let marks2 = [1, 2,3, 7]

marks = marks.concat(marks2);

// console.log(marks);

let myobj = {

'first name': 'harry',

channel: 'CodeWithHarry',

isActive: true,

marks: [1,5,3,6]

}

console.log(myobj)

console.log(myobj['channel'])

console.log(myobj.channel)

**f Else Conditionals & Switches in JavaScript | JavaScript Tutorial In Hindi #8**

In today's tutorial, we will study about ***conditional statements in JavaScript.*** As we know, conditional statements are used to decide the execution flow, based on different conditions. When the condition is true, specific action is performed, and if the condition is false, another action will perform. In JavaScript, we have the if, if-else, else if and switch statement.

We express a condition for if statements using relational operators. The relational operators allow us to compare two values to see whether they are equal, unequal, greater than, or less.

|  |  |
| --- | --- |
| **Conditions** | **Meaning** |
| a == b | a is equal to b |
| a  != b | a is not equal to b |
| a < b | a is less than b |
| a> b | a is greater than b |
| a <= b | a is less than or equal to b |
| a >= b | a is greater than or equal to b |
| a===b | a has the same value and same type as b |

**if condition:-**

The if statement is one of the most popular statement that is used by the programmers. We use if statement when we want to execute a statement in-case the certain condition is satisfied.

**Syntax:**

if(condition expression)

{

// code to be executed

}

Copy

**Example: if condition**

if( 5 > 0){

console.log("5 is greater than 0");

}

if( 5 < 0){

console.log("5 is less than 0");

}

Copy

The first if statement contains 5 > 0 as a conditional expression. The conditional expression 5 > 0 will be evaluated to be true, so the message "5 is greater than 0" will be displayed on the console, whereas the conditional expression in second if statement will be evaluated to be false, so the message "5 is less than 0" will not be displayed.

**else condition:-**

When the 'if condition' evaluates to false, we use the else statement. The else statement must follow **if** or **else if** statement. Multiple else statements at the same time are not allowed.

**Syntax:**

if(condition expression){

//Execute this code.

}

else{

//Execute this code.

}

Copy

**Example: else condition**

let salary = 500;

if( salary > 1000)

{

console.log("My Salary is greater than 1000");

}else{

console.log("My Salary is less than or equal to 1000");

}

Copy

**Output:-**My Salary is less than or equal to 1000

The first if statement contains salary > 1000 as a conditional expression. The conditional expression salary > 1000 will be evaluated to be true, so the message "My salary is greater than 1000" will be displayed on the console, whereas if the conditional expression evaluated to be false, so the message "My salary is less than or equal to 1000" will be displayed.

**else if condition:-**

We use "else if" condition when we want to apply second level conditions after the if statement. Following is the syntax of else if condition.

**Syntax:-**

if(condition expression)

{

//Execute this code block

}

else if(condition expression){

//Execute this code block

}

Copy

**Example: else if condition**

var salary = 500;

if( salary > 1000)

{

console.log("My Salary is greater than 1000");

}

else if(salary < 1000){

console.log("My Salary is less than 1000");

} else{

console.log("My Salary equal to 1000");

}

Copy

The first if statement contains salary > 1000 as a conditional expression. The conditional expression salary > 1000 will be evaluated to be true, so the message "My salary is greater than 1000" will be displayed on the console, whereas if Salary <1000 expression will be evaluated to be true, so the message "My salary is less than 1000" will be displayed. And if both conditions are evaluated to be false, then the last else block's statements will execute.

**JavaScript if else shortcut: ternary operator:-**

JavaScript provides a ternary operator that can be used as a shorthand of the if-else statement. Following is the syntax of the ternary operator.

Condition? expression1: expression2

Copy

If the condition evaluates to be true, the operator returns the value of the expression1. And if the condition evaluates to be false, it returns the value of the expression2.

**Example:-**

5>0? "Greater" : "Smaller";

Copy

If the expression 5 > 0 is evaluated to true, the message "Greater" will display. And if the condition is evaluated to false, the message "Smaller" will display.

**The JavaScript Switch Statement:-**

The switch statement is used to evaluate the expression. The switch statement is a part of JavaScript conditional statements. It is also used to control the flow of program execution. The switch performs different actions based on different conditions. The switch statement is often used with a break or a default keyword.

**Syntax:-**

switch(expression) {

case a:

// code block

break;

case b:

// code block

break;

default:

// code block

}

Copy

The expression in the switch statement is evaluated once. The value of the expression is compared with the values of each case. If the expression has a match, the associated block of code is executed, and if there is no match, the default code block is executed.

**The break Keyword:-**

The break keyword in JavaScript is used to breaks out of the switch block. This keyword will stop the execution of the block. It is not necessary to break every case in a switch block.

***Note:***If we do not use the break statement, the next case will be executed even if the evaluation does not match the case.

**Code index.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Tutorial on Js</title>

</head>

<body>

<h1>This is Js tutorial by Harry</h1>

</body>

<!-- <script src="js/tut2.js"></script> -->

<!-- <script src="js/tut3.js"></script> -->

<!-- <script src="js/tut4.js"></script> -->

<!-- <script src="js/tut5.js"></script> -->

<!-- <script src="js/tut6.js"></script> -->

<!-- <script src="js/tut7.js"></script> -->

<!-- <script src="js/tut8.js"></script> -->

<!-- <script src="js/tut9.js"></script> -->

<!-- <script src="js/tut10.js"></script> -->

<script src="js/tut11.js"></script>

</html>

Copy

**Js code as described/written in the video**

console.log('This is tutorial 8');

const age = 128;

const doesDrive = false;

// const vari = 34;

// if (age!=19){

// console.log('Age is not 19')

// }

// if(age !== 65){

// console.log('Age is not 65')

// }

// else{

// console.log('Age is not 19')

// }

// if (typeof vari !== 'undefined'){

// console.log('Vari is defined');

// }

// else{

// console.log('Vari is not defined');

// }

// if (doesDrive || age>18){

// console.log("You can drive");

// }

// else{

// console.log("You cannot drive");

// }

// console.log(age==45? 'Age is 45': 'Age is not 45');

switch (age) {

case 18:

console.log("You are 18");

break;

case 28:

console.log("You are 28");

break;

case 38:

console.log("You are 38");

break;

default:

console.log("You are unknown age");

break;

}

**For, While & Do while Loops in JavaScript | JavaScript Tutorial In Hindi #9**

There will be times when we want to repeat an action or run some code multiple times. Suppose we want to display the message 100 times. We could do this by simply creating a function display\_Message() and calling it 100 times using copy and paste. No doubt, this will work but, we should not do something like this. Duplicating code is never a good idea. The versatility of the computer lies in its ability to perform the set of instructions repeatedly. The generic solution for repeating code with control is provided in the form of loops. There are four different kinds of loops in JavaScript that we can be used to repeat some chunk of code:

* **for**
* **for…in**
* **forEach**
* **while**
* **do…while**

**The for Loop:-**

The for loop is used when we want to execute the set of statements for a certain number of times. This loop will repeat a block of code as long as a certain condition is met. Its syntax is:

for(initialization; condition; increment) {

// Code to be executed

}

Copy

* ***initialization***:- It is used to initialize the counter variables, for example, let i=0.
* ***Condition***:- The condition is evaluated at the beginning of each iteration. If it evaluates to true, the loop statements execute, and If it evaluates to false, the execution of the loop ends, i.e., i<100.
* ***Increment***:- it updates the loop counter with an incremented value value each time the loop runs for example i++;

***Following is the example of for loop***:

for(var i=1; i<=5; i++) {

console.log(i);

}

Copy

**The for… in Loop:-**

It is a special type of loop, used when we want to iterates over the properties of an object or the elements of an array. The syntax for using the for-in loop is:

for(variable in object) {

// Code to be executed

}

Copy

The loop counter, i.e., *variable* in the for-in loop, is a string, not a number. It contains the name of the current property or the index of the current array element.

The following example will show how to loop through all the properties of a JavaScript object.

// An object with some properties

var person = {name: "Harry", language: "JavaScript", age: 20};

// Loop through all the properties in the object

for(var i in person) {

console.log( i + " = " + person[i]);

}

Copy

**The forEach Loop:-**

In JavaScript, the  forEach is a type of loop that is used for Array method. With the help of forEach loop, we can execute a function on each item within an array. The function can only be used on Arrays, Sets, and Maps.

**Example:-**

const alpha = ['a', 'b', 'c'];

for (let i = 0; i < alpha.length; i++) {

console.log(alpha[i]);

}

Copy

**while loop:-**

A while loop is used when we do not know how many times a certain block of code should execute. It evaluates the expression inside the parenthesis (). If the expression evaluates to true, the code inside the while loop is executed. Every time the expression is re-evaluated, the process continues until the expression evaluates to false. When the expression evaluates to false, the loop stops.

The syntax of the while loop is:

while (expression) {

// body of loop

}

Copy

**For Example:-**

let i = 1, n = 100;

while (i <= n) {

console.log(i);

i += 1;

}

Copy

**do...while loop:-**

In this loop, the body inside the do statement is executed first. Then the condition is evaluated. If the condition evaluates to true, the body of the loop inside the do statement is executed again. This process continues until the condition evaluates to false. Then the loop stops.

The syntax of do...while loop is:

do {

// body of loop

} while(condition)

Copy

**For Example:-**

let i = 1;

let n =100;

do {

console.log(i);

i++;

} while(i <= n);

Copy

**Note**: do...while loop is similar to the while loop. The only difference is that in do…while loop, the body of loop is executed at least once.

**Breaking a Loop:-**

Sometimes, we may want to end our loop before it reaches completion. By using break statement, we can accomplish this task. Here is an example:

for (let i = 0; i < 100; i++) {

console.log(i);

if (i == 90) {

break;

}

}

Copy

When the value of i equals **90**, the break keyword stops the loop.

**Skipping an Iteration:-**

When we want the loop to skip its current iteration and move on to the next one we use “continue” keyword. Here is an example:

let i = 10;

for (let x = 1; x <=i; x++) {

if (x == 5) {

continue;

}}

Copy

As mentioned above, continue will stop the loop when x becomes equal to 5 and move on to the next iteration. In handling errors, continue keyword is also very useful when we want the loop to move on to the next item.

**Code index.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Tutorial on Js</title>

</head>

<body>

<h1>This is Js tutorial by Harry</h1>

</body>

<!-- <script src="js/tut2.js"></script> -->

<!-- <script src="js/tut3.js"></script> -->

<!-- <script src="js/tut4.js"></script> -->

<!-- <script src="js/tut5.js"></script> -->

<!-- <script src="js/tut6.js"></script> -->

<!-- <script src="js/tut7.js"></script> -->

<!-- <script src="js/tut8.js"></script> -->

<!-- <script src="js/tut9.js"></script> -->

<!-- <script src="js/tut10.js"></script> -->

<script src="js/tut11.js"></script>

</html>

Copy

**Js code as described/written in the video**

console.log("We are at tutorial 9");

// console.log(1)

// console.log(2)

// console.log(3)

// General Loops: for, while, do-while

// let a =34;

// a +=1;

// a++;

// console.log(a);

// for(let i=0; i<100;i++){

// console.log(i);

// }

// let j = 110;

// while (j < 10) {

// console.log(j + 1);

// j += 1;

// }

// let k = 0;

// do {

// if(k===5){

// k +=1;

// continue;

// }

// console.log(k + 1);

// k +=1;

// } while (k < 10);

let arr = [2,5,6,4,2,3];

// for (let i = 0; i < arr.length; i++) {

// const element = arr[i];

// console.log(element)

// }

arr.forEach(function(element, index, array){

console.log(element, index, array);

});

// let obj = {

// name: "Rohan Das",

// age: 78,

// type: "Dangerous Programmer",

// os: "Ubuntu"

// }

// for(let key in obj){

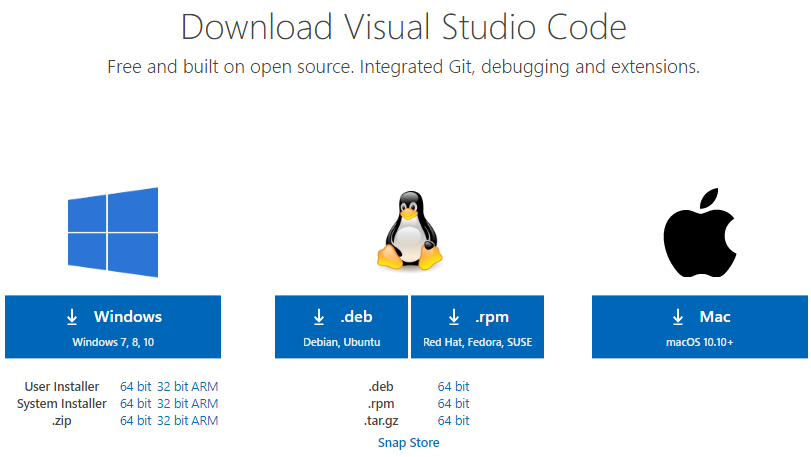
// console.log(`The ${key} of object is ${obj[key]}`)

// }

console.log('done');

**Functions in JavaScript | JavaScript Tutorial In Hindi #10**

For this series, we are using Virtual Studio Code (VS Code). ***Visual Studio Code*** is a fast source code editor and provides the tools that a developer needs for a quick code-build-debug cycle. To download VS Code, click on [***Download Virtual Studio Code***](https://code.visualstudio.com/download) . For guidance, check the tutorial [***Installing VS Code, Extensions & Setup***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-1)



In today's tutorial, we will study about ***functions in JavaScript***. As we know, function is a group of reusable code which can be called anywhere in the program. This eliminates the need to rewrite the same code. Functions allow a programmer to divide a big program into a smaller and manageable function. JavaScript provides functions just like other programming languages. A JavaScript function can be defined using **the function** keyword.

There are mainly two advantages of JavaScript functions.

1. **Code reusability**: We can call a function several times whenever we need it.
2. **Less coding**: It reduces the line of code and makes our program compact.

**Define a Function:-**

There are two different ways to define a function in JavaScript:

**Function Declaration:-**

A **Function Declaration**defines a named function. To create a function declaration, use the function keyword, and then write the function's name. When using function declarations, the function definition is hoisted. Thus the function is allowed to be used before it is defined. The syntax of the function declaration is as follows:

function name(parameters){

//statements

}

Copy

Inside the function body, we can implement the logic. **For example**, the following print\_message () function simply shows a message to the console:

function print\_message(message) {

console.log(message);

}

print\_message("Hello World");

//Output: Hello World

Copy

In the body of the print\_message() function, we call the console.log() function to output a message to the console.

**Function Expressions:-**

A **Function Expressions** defines a named or anonymous function. An anonymous function is a function that does not have any name. Function Expressions are not hoisted, and therefore we cannot call them before they are defined. In the example below, we are setting the anonymous function object equal to a variable. Following is the syntax of function expression :

let variable\_name = function(parameters){

// statements

}

Copy

**Example:-**

let add=function(a, b) {

return a + b;

}

console.log(add(4,3));

//Output: 7

Copy

**Returning a value:-**

JavaScript function returns undefined. See the following example:

function print\_message(message) {

console.log(message);

}

let result = say('Hello World');

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

//Output:

//Hello World

//Result: undefined

Copy

To specify a return value for a function, use the return statement followed by an expression or a value, like this:

return expression;

Let's create a new function called get\_distance:

function get\_Distance(speed, time) {

let dist = speed \* time;

return dist;

}

Copy

To call the get\_Distance function, we can call it as part of initializing a variable:

var myDistance = get\_Distance(8, 5);

Copy

When the getDistance function gets called, it gets evaluated and returns a numerical value that is then assigned to the myDistance variable. That's all there is to it.

**Key Takeaways:-**

In today's tutorial, we have learned that the function is a *subprogram* designed to perform a particular task. Functions are executed when they are called. This is known as *invoking* a function. Arguments can be *passed* into functions and used within the function. Functions *always* return a value. In JavaScript, if no return value is specified, the function will return undefined

**Code index.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Tutorial on Js</title>

</head>

<body>

<h1>This is Js tutorial by Harry</h1>

</body>

<!-- <script src="js/tut2.js"></script> -->

<!-- <script src="js/tut3.js"></script> -->

<!-- <script src="js/tut4.js"></script> -->

<!-- <script src="js/tut5.js"></script> -->

<!-- <script src="js/tut6.js"></script> -->

<!-- <script src="js/tut7.js"></script> -->

<!-- <script src="js/tut8.js"></script> -->

<!-- <script src="js/tut9.js"></script> -->

<!-- <script src="js/tut10.js"></script> -->

<script src="js/tut11.js"></script>

</html>

Copy

**Js code as described/written in the video**

console.log('this is tut 10');

if(1){

let i =234;

console.log(i)

}

console.log(i);

function ui(name)

{

let i = 9;

console.log(i);

return `This is a ${name} ui`;

}

console.log(ui("harry"), i)

// const mygreet = function(name, thank='Thank You'){

// let msg = `Happy Birthday ${name} How I wish I could fly to you right now and be with you on this special day of yours. But remember, my good wishes are always there with you. ${thank}!`;

// return msg;

// }

// let name = 'SkillF';

// let name2 ='Rohan';

// let val = mygreet(name, 'Thanks a lot');

// console.log(val);

// const myobj = {

// name: "SkillF",

// game: function(){

// return "GTA";

// }

// }

// console.log(myobj.game())

// arr = ['fruit', 'vegetable', 'furniture'];

// arr.forEach(function(element, index, array) {

// console.log(element, index)

// });

**Manipulating Websites Using JS Window Object | JavaScript Tutorial In Hindi #11**

We have focused on the basics of JavaScript like arrays, strings, objects, etc. In this tutorial, we will begin to make things happen on the screen, which is one of the main purposes of using JavaScript. Today we will study window object along with its properties and methods.

**The Window Object:-**

The***window object*** represents a window in the browser containing the DOM document. The browser automatically creates a window object. The window is not the object of the JavaScript. It is the object of the browser. A window can be the main window, a frameset, or even a new window created with JavaScript. The developers use the window object to close and open browser window, displaying alert and prompt dialogs.

Window object has many properties and methods. The following are the few properties and methods of window object.

**Window Object Properties:-**

The window object properties are the variables created inside the window object. We can access the properties of window object by using the syntax:

window.propertyname

Copy

where **property name** is the name of the property.

The following are the most popular window object properties:

|  |  |
| --- | --- |
| **Properties** | **Explanation** |
| width | It specifies the initial width of the browser window. |
| height | It specifies the initial height of the browser window. |
| innerWidth | It specifies the initial width of the window content area |
| innerHeight | It specifies the initial height of the window content area |
| outerWidth | It specifies the initial width of the navigator window |
| outerHeight | It specifies the initial height of the navigator window |
| menubar | It specifies whether the window should contain the browser menubar |
| location | It specifies whether the window should contain the browser location/URL box |
| scrollbars | It hides or shows browser horizontal/vertical scrollbars |
| top | Specified the number of pixels from the top of the screen to the new window |

**Window Object Methods:-**

The window object methods are the functions that are created inside the Window Object. These methods are used to perform various actions on the browser window, such as how it displays a message or gets input from the user.

Following are the some of the most commonly used window object methods:

|  |  |
| --- | --- |
| **Method** | **Description** |
| alert() | It will display an alert box with a message on the screen |
| blur() | It will remove focus from the current window |
| close() | It will close the current window |
| confirm() | It will display a dialog box with a message on the screen |
| focus() | It will set the focus to the current window |
| moveTo() | It will move a window to the specified position |
| open() | It will open a new browser window |
| print() | It will print the content of the current window |
| prompt() | It will display a dialog box that prompts the user for input |
| resizeBy() | It will resize the window by the specified pixels |
| resizeTo() | It will resize the window to the specified height and width |
| scrollBy() | It will scroll the document by the specified number of pixels |
| scrollTo() | It will scroll the document to the specified coordinates |
| stop() | It will stop the window from loading |

**Example:-**

In the following example, we will calculate the Width and Height of the Window using window object properties and then show the result on the screen using alert( ). As mentioned in the above table, The window object provides the innerWidth and innerHeight property to determine the width and height of the browser window. Here is an example

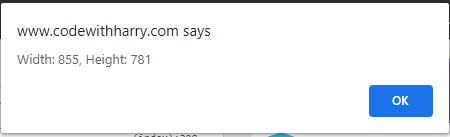
let a = window.innerWidth;

let b = window.innerHeight;

alert("Width: " + a + ", " + "Height: " + b);

Copy

**Result:-**



**Code index.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Tutorial on Js</title>

</head>

<body>

<h1>This is Js tutorial by Harry</h1>

</body>

<!-- <script src="js/tut2.js"></script> -->

<!-- <script src="js/tut3.js"></script> -->

<!-- <script src="js/tut4.js"></script> -->

<!-- <script src="js/tut5.js"></script> -->

<!-- <script src="js/tut6.js"></script> -->

<!-- <script src="js/tut7.js"></script> -->

<!-- <script src="js/tut8.js"></script> -->

<!-- <script src="js/tut9.js"></script> -->

<!-- <script src="js/tut10.js"></script> -->

<script src="js/tut11.js"></script>

</html>

Copy

**Js code as described/written in the video**

console.log('This is tut 11');

let a = window.document;

// alert('hello harry');

// a = prompt('This will destroy your computer. Type your name');

// a = confirm('Are you sure you want to delete this page?');

a = window.innerHeight;

a = innerWidth;

a = scrollY;

a = location.toString();

a = window.history;

console.log(a);

**Understanding DOM & Creating a Website Layout | JavaScript Tutorial In Hindi #12**

In today’s tutorial, we will study about **the Document Object Model**. As we know, every web page resides inside a browser window, which can be considered as an object. A DOM represents the HTML document that is displayed in that window. The Document object has various properties that allow access to and modification of document content. So, let us explore the concept of DOM along with its properties in detail.

**DOM - Document Object Model:-**

We can access the document content and modified it is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy.

**The window object** is at the top of the hierarchy. **The second one is the document object.**The HTML document that gets loaded into a window becomes a document object. The third one is the forms. Everything enclosed in the <form>...</form> tags sets the form object. And the last one is the **Form control elements**. The form object contains all the elements defined for that object, such as input fields, buttons, radio buttons, and checkboxes.

The DOM is a tree-like representation of the contents of a webpage. Tree of “nodes” with different relationships depending on how they are arranged in the HTML document.

<div id="main\_class ">

<div class="myclass1"></div>

<div class="myclass2"></div>

</div>

Copy

**Explanation:**

In the above example, the

 <div class="myclass1"></div> is a “child” of <div id="main\_class"></div> and a sibling to <div class="myclass2"></div>. It is like a family tree. <div id="main\_class"></div> is a **parent**, with its **children** on the next level, each on their own “branch”.

**DOM Properties:-**

With the Document Object Model (DOM), we can create and build documents, add, modify, or delete elements and content. Anything found in an HTML document can be accessed, changed, deleted, or added using the Document Object Model, with a few exceptions. The syntax for accessing the DOM properties is:

document.property\_name

Copy

 Here are some of the most common properties of the document object model.

|  |  |
| --- | --- |
| **Property** | **Description** |
| document.anchors | It will return all <a> elements that have a name attribute |
| document.baseURI | It will return the absolute base URI of the document |
| document.body | It will return the <body> element |
| document.cookie | It will return the document's cookie |
| document.doctype | It will return the document's doctype |
| document.documentElement | It will return the <html> element |
| document.documentMode | It will return the mode used by the browser |
| document.documentURI | It will return the URI of the document |
| document.embeds | It will return all HTML <embed> elements |
| document.forms | It will return all HTML <form> elements |
| document.head | It will return the HTML <head> element |
| document.images | It will return all HTML <img> elements |
| document.implementation | It will return the DOM implementation |
| document.links | It will return all HTML <area> and <a> elements that have a href attribute |
| document.scripts | It will return all HTML <script> elements |
| document.title | It will return the <title> element |
| document.URL | It will return the complete URL of the document |

**Note that:** The DOM is a representation of the various components of the browser and the current Web document that can be accessed or manipulated using JavaScript.

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading"> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this">

<li class="childul">this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul">of my dreams</li>

</ul>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<input type="button" value="submit">

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<script src="js/tut15.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log('Welcome to tut 12.js');

let a = document;

a = document.all;

// a = document.body;

// a = document.forms[0];

// Array.from(a).forEach(function(element){

// console.log(element);

// })

a = document.links[0];

// use document.images and document.scripts and print the list of images and scripts on an html page

console.log(a);

**WebPage Crawler In JavaScript - Exercise 1 | JavaScript Tutorial In Hindi #13**

Are you ready for your JavaScript first exercise? Your task is to create **WebPage Crawler In JavaScript.**

**Problem statement:-**

You have to create a variable, which takes a string that contains the text, which is a link you are interested in.

Your task is to fetch all the links from a given page which contains this text.

For example, you have to fetch the link which contains JavaScript word or text that looks like [**www.codewithharry.com**](http://www.codewithharry.com/)**.** You have to fetch all the links, filter them and then display the links you are interested in on the screen.

You are advised to participate in solving this problem. This task helps you become a good problem solver and enables you to accept the challenge and acquire new skills.

If you like my work, then keep supporting and stay up to date with [**codewithharry**](http://www.codewithharry.com./)**.**

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading"> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this">

<li class="childul">this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul">of my dreams</li>

</ul>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<input type="button" value="submit">

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<script src="js/tut15.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

// Exercise 1

// You have to create a variable which is a string containing the text which is a link you are interested in.

// You have to fetch all the links from a given page which contains this text

// codewithharry.com

// javascript

Copy

**HTML Element Selectors In JavaScript | JavaScript Tutorial In Hindi #14**

If you are reading this tutorial, then you must be familiar with DOM (Document Object Model). If not, then check [***tutorial#12***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-12) to get the concept of DOM. Today's tutorial is about DOM selector, so one must have a fair idea about DOM before moving forward. As we know that the JavaScript is most commonly used to get or modify the content or value of the HTML elements. However, before we perform any action like change style or add animation, our first task is to need to find or select the target HTML element. In today's tutorial, we will see some of the common ways of selecting the elements on a page using different methods.

| **Gets** | **Selector Syntax** | **Method** |
| --- | --- | --- |
| ID | #name | getElementById() |
| Class | .name | getElementsByClassName() |
| Tag | name | getElementsByTagName() |
| Selector (single) |  | querySelector() |
| Selector (all) |  | querySelectorAll() |

DOM Selectors is used to selecting HTML elements within a document using JavaScript. There are two types of a selector, i.e., single element selector and multiple element selector.

**Single Element Selector:-**

It is used to select single HTML elements within a document. Following are some single element selectors:

**Accessing Elements by ID:-**

To access the single element in the DOM is by its unique ID. We can get or modify an element by ID with the **getElementById()**method of the document object. It will return null if no elements with the specified ID exists.

An ID should be unique within a page. However, if more than one element with the specified ID exists, the getElementById() method returns the first element in the source code.

document.getElementById(elementID);

Copy

In order to be accessed by ID, the HTML element must have an id attribute. We have a h1 element with an ID of demo.

<h1 id="demo">Heading 1</h1>

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

x.style.color = "red";

x.className;

x.childNodes;

x.parentNode;

x.innerText = 'JavaScript Tutorial';

x.innerHTML = '<b>Harry is a good boy</b>';

Copy

**Explanation of above Example:-**

* **className:**The className property sets or returns the class name of an element
* **childNodes:** The childNodes property returns a collection of a node's child nodes.
* **parentNode:**The parentNode property returns the parent node of the specified node.
* **innerText:** The innerText property sets or returns the text content of the specified node.
* **innerHTML:**The innerHTML property sets or returns the HTML content of an element.

**querySelector;-**

It will return the first element that matches the specified CSS selector in the document. The querySelector() method only returns the first element that matches the specified selectors.

document.querySelector('#myclass');

document.querySelector('.myclass');

document.querySelector('li');

Copy

The above statement will return the first element that matches the CSS selector #myclass, .myclass and li. We can use all kinds of CSS selectors within the querySelector method.

**Multiple Element Selector:-**

It is used to select multiple HTML elements within a document. There are three ways in which we can select elements in a DOM using multiple element selectors.

* querySelectorAll()
* getElementsByTagName()
* getElementsByClassName()

**querySelectorAll:-**

It will return a list of the document's elements that match the specified group of selectors. The querySelectorAll() method returns all elements in the document as a static NodeList object. The NodeList object represents a collection of nodes. The nodes can be accessed by index numbers. The index starts at 0.

document.querySelectorAll('.heading')

Copy

It will return a list of all elements that matches the specified CSS selector.

document.querySelectorAll("p.animation ");

Copy

It will return all <p> elements in the document with class="animation"

**Accessing Elements by Tag Name:-**

We can also select HTML elements with their tag name by using getElementsByTagName() method. This method returns an array-like object of all child elements with the given tag name. The syntax is:

document.getElementsByTagName(tagname);

**Here is an example:**

<article> My article 1</article>

<article> My article 2</article>

Copy

We can modify every tag in the document with a for and forEach loop. In this example, we are using for loop

let mytag = document.getElementsByTagName('article');

for (let i = 0; i < mytag.length; i++) {

mytag[i].style.border = '1px solid blue';

}

Copy

**Accessing Elements by Class:-**

The class attribute is used to access one or more specific elements in the DOM. The ID should be unique, whereas one or more elements can have the same class. Using ***getElementsByClassName()*** method., we can get all the elements with a given class name. Here is the syntax

document.getElementsByClassName(class\_name);

Copy

We can access more than one element, and in our example we have two elements with a demo class. For Example:

<div class="demo"> class 1</div>

<div class="demo"> class 2</div>

Copy

To find how many elements with class="demo" there are in the document, we use the length property of the HTMLCollection object. Here is an example

document.getElementsByClassName("demo").length;

Copy

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading"> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this">

<li class="childul">this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul">of my dreams</li>

</ul>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<input type="button" value="submit">

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<script src="js/tut15.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log('Welcome to tutorial 14');

/\*

element selectors:

1. Single element selector

2. Multi element selector

\*/

// 1. Single element selector

let element = document.getElementById('myfirst');

// element = element.className;

// element = element.childNodes;

// element = element.parentNode;

element.style.color = 'red';

element.innerText = 'Harry is a good boy';

element.innerHTML = '<b>Harry is a good boy</b>';

// console.log(element.innerText);

let sel = document.querySelector('#myfirst');

sel = document.querySelector('.child');

sel = document.querySelector('div');

sel.style.color = 'green';

console.log(sel)

// 2. Multi element selector

let elems = document.getElementsByClassName('child');

elems = document.getElementsByClassName('container');

elems = document.getElementsByTagName('div');

console.log(elems)

for (let index = 0; index < elems.length; index++) {

const element = elems[index];

console.log(element);

element.style.color = 'blue';

}

// Array.from(elems).forEach(element => {

// console.log(element);

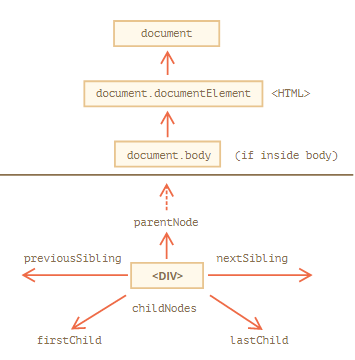
// element.style.color = 'blue';

// });

// console.log(elems[0].getElementsByClassName('child'))

**Children, Parent & Traversing the DOM | JavaScript Tutorial In Hindi #15**

In the previous tutorial, we have learned how to use the built-in methods of the document object to access HTML elements by ID, class, tag name, and query selectors. Often, we want to move through the DOM without specifying every element beforehand. In this tutorial, we will go over how to traverse the DOM using parent, child, and sibling properties. As we know that all the operations on the DOM start with the document object. It is the main entry point, and we can access any node from it. Here is a diagram of links that allow for travel between DOM nodes:



We can traverse the DOM in three directions, downwards, upwards and sideways. Each type of traversal uses a different method.

**Traversing downwards:-**

There are two methods to traverse downwards, the first one is querySelector and the second one is children.

**querySelector or querySelectorAll:-**

To traverse downwards from a specific element, we can use querySelector( ) or querySelectorAll( ). The **querySelector()** returns the first element within the document that matches the specified selector whereas the **querySelectorAll()** returns the NodeList representing a list of the document's elements that match the specified group of selectors.

<div class="add">

<h2 class="add\_\_title">title</h2>

</div>

const component = document.querySelector('.add')

console.log(component)

Copy

https://api.codewithharry.com/media/videoSeriesFiles/courseFiles/javascript-tutorials-in-hindi-15/base64_B7zEGJR.png

**children:-**

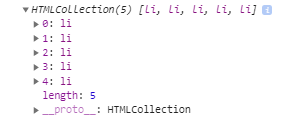
The property that lets you select direct descendants is called children . It selects elements that are immediately nested in another element. The children returns a HTML Collection that updates when children elements are changed.

const items= document.querySelector('.myclass')

const l\_Items = items.children

console.log(l\_Items)

Copy



To return the first and last child of a node, use the ***firstChild***and ***lastChild*** property. The node can be of any node type, including text node, comment node, and element node. Similarly, ***firstElementChild***and ***lastElementChild*** return the first and last child Element node, and the ***childNodes*** returns a live NodeList of all child nodes of any node type of a specified node.

**Selecting a specific child:-**

While traversing the DOM, we can select the nth-item in the list from both NodeLists and HTML Collections. For this, we use the index of the element. Similarly, we do in the case of the array to select a specific element.

const mylist = document.querySelectorAll('li')

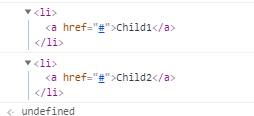
const firstItem = mylist[0]

const secondItem = mylist[1]

console.log(firstItem)

console.log(secondItem)

Copy



**Traversing upwards:-**

There is one method to traverse upwards: parentElement

**parentElement:-**

The property that let us select the parent element is known as parentElement. The parentElement returns *null* if the parent node is not an element node. Following is the example

const mylist = document.querySelectorAll('li')

const firstItem = mylist[0]

const secondItem = mylist[1]

console.log(firstItem.parentElement)

console.log(secondItem.parentElement)

Copy

https://api.codewithharry.com/media/videoSeriesFiles/courseFiles/javascript-tutorials-in-hindi-15/base64_w1LqcZx.png

**Traversing sideways:-**

There are two methods to traverse sideways. One of them is nextElementSibling, and the other one is previousElementSibling.

**nextElementSibling:-**

To select the next element, we use the nextElementSibling. The difference between this property and nextSibling is that nextSibling returns the next sibling node as an element node, a text node or a comment node, while nextElementSibling returns the next sibling node as an element node and ignores the text and comment nodes.

const item1 = document.querySelector('li')

const item2 = item1.nextElementSibling

console.log(item2)

Copy



**previousElementSibling:-**

To select the previous element, we use previousElementSibling. The difference between this property and previousSibling, is that previousSibling returns the previous sibling node as an element node, a text node or a comment node, while previousElementSibling returns the previous sibling node as an element node and ignores the text and comment nodes.

const item5 = document.querySelectorAll('li')[1]

const item6 = item5.previousElementSibling

console.log(item6)

Copy



**Node Type:-**

The **nodeType** property is an integer that identifies what the node is. It differentiate between different kind of nodes from each other, such as elements, text and comments. The syntax is:

var type = node.nodeType;

Copy

It will return an integer which specifies the type of the node.

| **Constant** | **Value** | **Description** |
| --- | --- | --- |
| Node.ELEMENT\_NODE | 1 | An Element node like <h1> or <p>. |
| Node.ATTRIBUTE\_NODE | 2 | An Attribute of an Element. |
| Node.TEXT\_NODE | 3 | The actual Text inside an Element |
| Node.COMMENT\_NODE | 8 | A Comment node |
| Node.DOCUMENT\_NODE | 9 | A Document node. |
| Node.DOCUMENT\_TYPE\_NODE | 10 | A DocumentType(<!DOCTYPE html>) node |
| Node.DOCUMENT\_FRAGMENT\_NODE | 11 | A DocumentFragment node. |

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading"> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this">

<li class="childul">this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul">of my dreams</li>

</ul>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<input type="button" value="submit">

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<script src="js/tut15.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log('Welcome to tutorial 15');

let cont = document.querySelector('.no');

cont = document.querySelector('.container');

let nodeName = cont.childNodes[1].nodeName;

let nodeType = cont.childNodes[1].nodeType;

// console.log(nodeName)

// console.log(nodeType)

// Node types

// 1. Element

// 2. Attribute

// 3. Text Node

// 8. Comment

// 9. document

// 10. docType

// console.log(cont.childNodes);

// console.log(cont.children);

let container = document.querySelector('div.container');

// console.log(container.children[1].children[0].children);

// console.log(container.firstChild);

// console.log(container.firstElementChild);

// console.log(container.lastChild);

// console.log(container.lastElementChild);

// console.log(container.children);

// console.log(container.childElementCount); // Count of child elements

console.log(container.firstElementChild.parentNode);

console.log(container.firstElementChild.nextSibling);

console.log(container.firstElementChild.nextElementSibling);

console.log(container.firstElementChild.nextElementSibling.nextElementSibling);

**Creating, Removing & Replacing Elements | JavaScript Tutorial In Hindi #16**

In the previous lecture, we explored a number of ways to select the HTML element. In this tutorial, we will study how to create new elements and manipulate existing elements on the page, such as removing elements from the page, replacing elements with new elements. This tutorial will help us master all the javaScript techniques that we need to manipulate elements in the DOM.

**createElement():-**

The **document.createElement()** is a method used to create the HTML element. **createElement()**. Following is the syntax**.**

var element= document.createElement("name");

Copy

In the above syntax, the name is passed as a parameter. The name specifies the type of the created element. The document.createElement() returns the newly created element.

The following example uses the document.createElement() to create a new <div> element:

let div = document.createElement('div');

Copy

There is also another method, which is createTextNode(). This method creates a Text Node with the specified text. Use the createElement() method to create an element Node with the specified name. After the Text Node is created, use the *element*.appendChild() method to append it to an element.

var paragraph = document.createElement("P");

var text = document.createTextNode("This is a paragraph.");

Copy

**appendChild():-**

The **appendChild()** method is used to create a text node as the node's last child. Appending in JavaScript is a way to insert content to the end of already existing elements. To append in Javascript, we use theappendChild() method.

**Syntax:**

node.appendChild( node )

**Here is an example:**

var paragraph = document.createElement("P");

var text = document.createTextNode("This is a paragraph.");

paragraph.appendChild(text);

Copy

**Working with Attributes:-**

The attributes are the special words used inside the start tag of an HTML element. JavaScript provides us with several methods for adding, removing or changing an HTML element's attribute. Following are the some methods to set attributes in HTML element.

**setAttribute() :-**

The setAttribute() method is used to set an attribute on the specified element. Using this method, a new attribute is added with the specified name and value. If the attribute already exists on the element, the value is updated. Here is an example:

In this example, we add a href attribute with a value of "https://codewithharry.com/" to an <a> element:

document.getElementById("myAnchor").setAttribute("href", "https://codewithharry.com/");

Copy

**getAttribute() :-**

The getAttribute() method is used to get the current value of a attribute on the specified element. If the attribute does not exist on the element, it will return null. Here is an example:

In this example, we will get the value of the target attribute of an <a> element:

var h = document.getElementById("myAnchor").getAttribute("target");

Copy

**hasAttribute():-**

The hasAttribute() method is used to check an element has a specified attribute or not. The syntax is :

let result = element.hasAttribute(name);

Copy

In the argument, we specifies the name of the attribute that we want to check.

This method returns a Boolean value that indicates if the element has the specified attribute. If the element contains an attribute, it will return true; otherwise, it will return false. Here is an example:

In this example, we find that if the <button> element has an onclick attribute:

var h = document.getElementById("Btn").hasAttribute("onclick");

Copy

**removeAttribute() :-**

The removeAttribute() method is used to remove an attribute from the specified element. The difference between this method and the removeAttributeNode() method is that the removeAttributeNode() method removes the specified Attr *object*, while this method removes the attribute with the specified *name*. The result will be the same. Also this method has no return value, while the removeAttributeNode() method returns the removed attribute as an Attr object.

Remove the href attribute from an <a> element:

document.getElementById("myAnchor").removeAttribute("href");

Copy

**replaceWith():-**

The replaceWith() method replaces this ChildNode in the children list of its parent with a set of Node. Here is an example:

let element1 = document.getElementById("myid1 ");

let element2 = document.createElement("u");

let content = document.createTextNode("Added Content");

element2.appendChild(content);

element1.replaceWith(element2);

Copy

We are getting the element using its id**.**Then we are creating a new elementwhich creates an underlined content and then we adds content to the newly created element. In next step we are adding the content to the element2. And finally we are replacing the old element, with the newly created element(old\_elem.replaceWith(new\_elem);).

**replaceChild():-**

The replaceChild() method replaces a child node with a new node. We can create a new node, or The new node could be an existing node in the document or the new node could be an existing node in the document. In the following example, we will replace an existing node sp2 with the new span element sp1:

parentDiv.replaceChild(sp1, sp2);

Copy

**removeChild():-**

This method removes a specified child node of the specified element. It will return the removed node as a Node object, or *null* if the node does not exist.Remember that theremoved child node is no longer part of the DOM. In the following example, we will find out if a list has any child nodes. If so, remove its second child node.

let list = document.getElementById("myList");

if (list.hasChildNodes()) {

list.removeChild(list.childNodes[1]);

}

Copy

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading"> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this">

<li class="childul">this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul">of my dreams</li>

</ul>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<input type="button" value="submit">

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<script src="js/tut15.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log('this is tut 16');

let element = document.createElement('li');

let text = document.createTextNode('I am a text node');

element.appendChild(text)

// Add a class name to the li element

element.className = 'childul';

element.id = 'createdLi';

element.setAttribute('title', 'mytitle');

// element.innerText = '<b>Hello this is created by harry</b>';

// element.innerHTML = '<b>Hello this is created by harry</b>';

let ul = document.querySelector('ul.this');

ul.appendChild(element);

// console.log(ul)

// console.log(element)

let elem2 = document.createElement('h3');

elem2.id = 'elem2';

elem2.className = 'elem2';

let tnode = document.createTextNode('This is a created node for elem2');

elem2.appendChild(tnode);

element.replaceWith(elem2);

let myul = document.getElementById('myul');

myul.replaceChild(element, document.getElementById('fui'));

myul.removeChild(document.getElementById('lui'));

let pr = elem2.hasAttribute('href');

elem2.removeAttribute('id');

elem2.setAttribute('title', 'myelem2title');

console.log(elem2, pr);

// quick quiz

// create a heading element with text as "Go to CodeWithHarry" and create an a tag outside it with href = "https://www.codewithharry.com"

Copy

**Previous**

**Events & Event Object In JavaScript | JavaScript Tutorial In Hindi #17**

In this tutorial, we discuss some important concepts about events and look how they work in browsers.

**What are Events and Event Handlers ?**

Events are actions that happen in the webpage like clicking a button or submit the form. If the user selects a button on a webpage, we might want to respond to that action by displaying an alert on the webpage. Each available event has an **event handler**, which is a block of code that runs when the event occurred. Event handlers are sometimes called **event listeners**. The listener listens out for the event happening, and the handler is the code that is run in response to it happening. There are many ways to assign a handler. Let us see them, starting from the simplest one.

**HTML-attribute:-**

We can set a handler in HTML with an attribute named on<event>. For example, to assign a click handler for an input, we can use onclick, here is an example:

<input value="Click here" onclick="alert('Click here!')" type="button">

Copy

The code inside onclick runs on mouse click. But note that the HTML-attribute is not a convenient place to write a lot of code, so it will better create a JavaScript function and call it there.

**addEventListener():-**

The JavaScript addEventListener() method allows the programmer to set up functions to be called when a specified event happens, such as when a user clicks a button. The purpose of using the addEventListener() method is to attach an event handler to the specified element. While using the addEventListener() method, remember that it attaches an event handler to an element without overwriting existing event handlers. We can add many event handlers to one element. Removing event is as simple as adding an event handler. This task can easily be done by using the removeEventListener() method. Following is the syntax to add an event listener.

**Syntax:-**

element.addEventListener(event, function, useCapture);

Copy

* The first parameter is the type of event like "click" or "mousedown".
* The second parameter is the function we want to call when the event occurs.
* The third parameter is optional. It is a boolean value specifying whether to use event bubbling or event capturing.

**Example:-**

Alert "Event Occurred" when the user clicks on an element:

document.addEventListener("click", function(){ alert("Event Occurred"); });

Copy

Similarly, we can also refer to an external "named" function:

document.addEventListener("click", myfunc);

function myfunc () {

document.getElementById("demo").innerHTML = "Hello World";

}

Copy

**Passing Event as a Parameter:-**

Sometimes we may want to know more information about the event, such as what element was clicked. When an event happens, the browser creates an *event object*, puts details into it and passes it as an argument to the handler. Here is an example that shows event type, element and coordinates of the click.

document.addEventListener("click", myfunc);

function myfunc(event) {

alert(event.type + " at " + event.currentTarget);

alert("Coordinates: " + event.clientX + ":" + event.clientY);};

Copy

Some properties of the event object used in the above program are:

* **type:**This will tell the event type, here it is "click".
* **currentTarget:**It returns the element whose event listeners triggered the event.
* **clientX / event.clientY:**It returns the window-relative coordinates of the cursor, for pointer events.

There are many properties, and these depend on the event type. The keyboard events and pointer events have a different set of properties. Here are some properties of the event object

**Event Properties and Methods:-**

|  |  |
| --- | --- |
| **Property/Method** | **Description** |
| altKey | It will return whether the "ALT" key was pressed when the mouse event was triggered. |
| button | It will return which mouse button was pressed when the mouse event was triggered. |
| charCode | It will return the Unicode character code of the key that triggered the onkeypress event. |
| clientX | It will return the horizontal coordinate of the mouse pointer, relative to the current window, when the mouse event was triggered. |
| clientY | It will return the vertical coordinate of the mouse pointer, relative to the current window, when the mouse event was triggered. |
| code | It will return the code of the key that triggered the event. |
| deltaX | It will return the horizontal scroll amount of a mouse wheel (x-axis). |
| deltaY | It will return the vertical scroll amount of a mouse wheel (y-axis). |
| deltaZ | It will return the scroll amount of a mouse wheel for the z-axis. |
| detail | It will return a number that indicates how many times the mouse was clicked. |
| keyCode | It will return the Unicode character code of the key that triggered the onkeypress event, or the Unicode key code of the key that triggered the onkeydown or onkeyup event. |
| location | It will return the location of a key on the keyboard or device. |
| pageX | It will return the horizontal coordinate of the mouse pointer, relative to the document, when the mouse event was triggered. |
| pageY | It will return the vertical coordinate of the mouse pointer, relative to the document, when the mouse event was triggered. |
| screenX | It will return the horizontal coordinate of the mouse pointer, relative to the screen, when an event was triggered. |
| screenY | It will return the vertical coordinate of the mouse pointer, relative to the screen, when an event was triggered. |
| shiftKey | It will return whether the "SHIFT" key was pressed when an event was triggered. |
| type | It will return the name of the event. |
| which | It will return which mouse button was pressed when the mouse event was triggered. |
| view | It will return a reference to the Window object where the event occurred. |

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading"> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this">

<li class="childul">this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul">of my dreams</li>

</ul>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<input type="button" value="submit">

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<script src="js/tut15.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log("This is tutorial 17 on events");

document.getElementById("heading").addEventListener("click", function(e) {

let variable;

console.log("You have clicked the heading");

variable = e.target;

variable = e.target.className;

variable = Array.from(e.target.classList);

variable = e.target.id;

variable = e.offsetX;

variable = e.offsetY;

variable = e.clientX;

variable = e.clientY;

console.log(variable);

// location.href = '//codewithharry.com'

});

**More on JavaScript Events | JavaScript Tutorial In Hindi #18**

As we have studied before, the JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page. When the user clicks a button, it is called an event. Events are a part of the Document Object Model(DOM), and every HTML element contains a set of events that can trigger the JavaScript Code.

In the previous lecture, we have studied different events like onclick, onsubmit, etc. Now let's explore more events available in JavaScript. There are three types of events, mouse events, input events, and keyboard events. Here are some important events, along with the example.

**Dblclick:-**

The purpose of a double click event is to triggers after two clicks on the same element within a short timeframe. This event is rarely used nowadays. Even when we use it, we should *never* use both an onclick and a dblclick event handler on the same HTML element. When the user double–clicks on an element, a one-click event takes place before the dblclick. This will cause a problem on the webpage. So, keep the clicks and dblclicks well separated to avoid complications.

const myclick = document.querySelector('aside');

myclick.addEventListener('dblclick', function (e) {

console.log(“Double click event occur”);});

Copy

**Mousemove:-**

The mousemove event takes place at an element when a mouse is moved while the cursor's hotspot is inside it. This event works fine, but we should know that it may take quite some system time to process all mousemove events. If we move the mouse one pixel, the mousemove event fires.

const test= document.getElementById('test');

test.addEventListener('mousemove', function (e) {

console.log(“ Mousemove event occur”);});

Copy

**Mouseover and mouseout:-**

For an element when a mouse or other pointing device is used to move the cursor onto the element or one of its child elements, mouseover event is fired. Whereas, for an element when a mouse or other pointing device is used to move the cursor so that it is no longer contained within the element or one of its children, the **mouseout** event is fired.

**Example of Mouseover:-**

const test= document.getElementById('test');

test.addEventListener("mouseover", function( event ) { event.target.style.color = "red";});

Copy

**Example of Mouseout:-**

const test= document.getElementById('test');

test.addEventListener("mouseout", function( event ) {

event.target.style.color = "red";});

Copy

**Mouseenter and mouseleave:-**

The mouseenter event is fired at an element when a mouse initially moved so that its hotspot is within the element at which the event was fired. Whereas the **mouseleave** event is fired at an element when the mouse or other pointing device is moved out of it. These events are similar, but the difference is that the mouseleave event is fired when the pointer has exited the element *and* all of its descendants. In contrast, the mouseout event is fired when the pointer leaves the element *or* leaves one of the elements' descendants even if the pointer is still within the element.

**Example of Mouseenter:-**

const mouseTarget = document.getElementById('mouseTarget');

mouseTarget.addEventListener('mouseenter', function(e) {

mouseTarget.style.border = '5px dotted blue';});

Copy

**Example of Mouseleave:-**

const mouseTarget = document.getElementById('mouseTarget');

mouseTarget.addEventListener('mouseleave', function(e){

mouseTarget.style.border = '1px solid red'; });

Copy

**Mousedown and mouseup:-**

The **mousedown** event is fired at an element when the pointing device like mouse button is pressed while the pointer is inside the element. Whereas, the **mouseup** event is fired at an element when a button on a pointing device is released while the pointer is located inside it.

There is a difference between click and mousedown event. As we know, that click event occurs after a full click action; that is, the mouse button is pressed and released while the pointer remains inside the same element. Whereas, the mousedown event is fired the moment the button is initially pressed.

**Example of mousedown:**

myevent.addEventListener('mousedown', function(e) {

console.log("Mousedown event occur")});

Copy

**Example of mouseup:**

myevent.addEventListener('mouseup', function(e) {

console.log("Mouseup event occur")});

Copy

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading"> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this">

<li class="childul">this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul">of my dreams</li>

</ul>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<input type="button" value="submit">

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<script src="js/tut15.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log('This is tut 18');

// let btn = document.getElementById('btn');

// btn.addEventListener('click', func1);

// // btn.addEventListener('mousedown', func3);

// btn.addEventListener('dblclick', func2);

// function func1(e) {

// console.log("Thanks", e);

// e.preventDefault();

// }

// function func2(e) {

// console.log("Thanks its a double click", e);

// e.preventDefault();

// }

// function func3(e) {

// console.log("Thanks its a mouse down ", e);

// e.preventDefault();

// }

// document.querySelector('.no').addEventListener('mouseenter', function(){

// console.log('You entered no')

// })

// document.querySelector('.no').addEventListener('mouseleave', function(){

// console.log('You exited no')

// })

document.querySelector('.container').addEventListener('mousemove', function(e){

console.log(e.offsetX, e.offsetY);

document.body.style.backgroundColor = `rgb(${e.offsetX}, ${e.offsetX},154)`;

console.log('You triggered mouse move event')

})

**Smart Page Crawler In JS Exercise1: Solution | JavaScript Tutorial In Hindi #19**

This tutorial contains the solution to Exercise 1 i.e. Smart Page Crawler. The problem statement along with the instructions is given below:

**Problem statement:-**

You have to create a variable, which takes a string that contains the text, which is a link you are interested in.

Your task is to fetch all the links from a given page which contains this text.

For example, you have to fetch the link which contains JavaScript word or text that looks like [**www.codewithharry.com**](http://www.codewithharry.com/)**.** You have to fetch all the links, filter them and then display the links you are interested in on the screen.

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading"> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this">

<li class="childul">this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul">of my dreams</li>

</ul>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<input type="button" value="submit">

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<script src="js/tut15.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

let str = "python";

let links = document.links;

console.log(links);

let href;

Array.from(links).forEach(function(element) {

href = element.href;

if (href.includes(str)) {

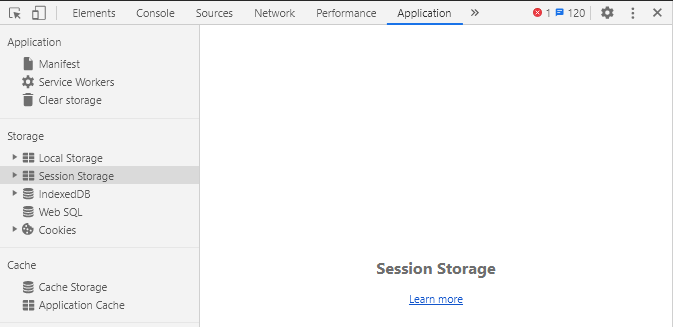
console.log(href);

}

});

**Local & Session storage in JavaScript | JavaScript Tutorial In Hindi #20**

Web applications can store data locally within the user's browser with web storage. Before HTML5, application data stored in cookies, included in every server request. But with the Advent of Html5, we have got various options to store information on the client browser. Previously we were having only cookies, which were very restrictive, and the size of the cookies was very small. But now the web storage is more secure, and large amounts of data can be stored locally, without affecting website performance. We have local storage and session storage. We can access the web storage by right-clicking on the webpage and then selecting option applications. Here we can see the option of local and session storage. Let's talk about all these web storage in detail.



***Figure1: Inspect->Application***

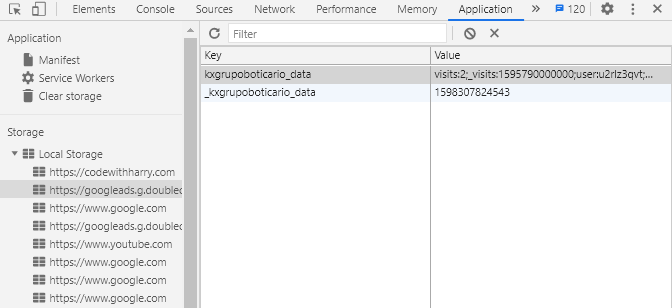
**HTML Web Storage Objects:-**

HTML web storage provides us with two objects for storing data:

* **window.localStorage**– It stores the data with no expiration date
* **window.sessionStorage**-It stores the data for one session. That means the data is lost when the browser tab is closed.

**LocalStorage:-**

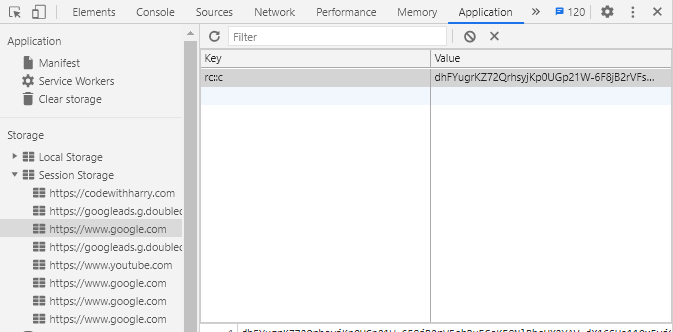
The way to store data on the client's computer is by local storage. The local storage allows us to save the key/value pairs in a web browser, and it stores data with no expiration date. We can access local storage via JavaScript and HTML5. However, the user can clear the browser data to erase all localStorage data.



***Figure 2: Local Storage***

**Session storage:-**

The session storage is used to store data only for a session, meaning that it is stored until the browser (or tab) is closed. Remember that, in session storage, the data is never transferred to the server and can only be read on the client-side. The storage limit is between 5-10MB. By opening multiple windows or tabs with the same URL creates sessionStorage for each tab or window.



***Figure3: Session storage***

Both storage objects provide the same methods and properties like setItem, getItem, removeItem, and clear. The examples in this tutorial are for localStorage, but the same syntax works for sessionStorage.

* **Creating Entries:-**We can create the key/value pair entries with localStorage.setItem, providing a key and a value. Here is an example:

let key = 'Entry\_1';

localStorage.setItem(key, 'Value');

Copy

* **Reading Entries:-**We can read entries with localStorage.getItem. Here is an example:

let myItem = localStorage.getItem(key);

Copy

* **Updating Entries:-**We can update an entry just as we create a new one with setItem, but with a key that already exists. Here is an example:

localStorage.setItem(key, 'New Value');

Copy

* **Deleting Entries:-**We can delete an entry with the removeItem method. Here is an example:

localStorage.removeItem(key);

Copy

* **Clearing Everything:-**We can clear everything that's stored in localStorage. Here is an example:

localStorage.clear();

Copy

* **Storing JSON Objects:-**Only strings can be stored with localStorage or sessionStorage, but we can use **JSON.stringify**to store more complex objects and **JSON.parse** to read them. Here is an example:

// Create item:

let myObj = { name: 'Harry', language: 'JavaScript' };

localStorage.setItem(key, JSON.stringify(myObj));

// Read item:

let item = JSON.parse(localStorage.getItem(key));

Copy

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading"> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this">

<li class="childul">this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul">of my dreams</li>

</ul>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<input type="button" value="submit">

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<script src="js/tut15.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log('This is tut 20');

let impArray = ['adrak', 'pyaz', 'bhindi'];

// // Add a key-value pair inside local Storage

// localStorage.setItem('Name', 'Harry');

// localStorage.setItem('Name2', 'Rohan');

// localStorage.setItem('Sabzi', JSON.stringify(impArray));

// Clears the entire local storage

// localStorage.clear();

// Clear a particular key-value pair

// localStorage.removeItem('Name2');

// Retrieve an item from the local Storage

let name = localStorage.getItem('Name');

name = JSON.parse(localStorage.getItem('Sabzi'));

console.log(name)

// sessionStorage.setItem('sessionName', 'sHarry');

// sessionStorage.setItem('sessionName2', 'sRohan');

// sessionStorage.setItem('sessionSabzi', JSON.stringify(impArray));

Copy

**Previous**

**Creating Editable Div Using JavaScript: Exercise 2 | JavaScript Tutorial In Hindi #21**

Until now, we have explored many new concepts of JavaScript. Now it’s time to solve the exercise. The task you have to do is ***"Creating Editable Div Using JavaScript"***. In this exercise, you have to use an event listener. If you have not watched the tutorial about events and event objects in JavaScript, then check [***tutorial#17***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-17).

**Problem Statement:-**

You have to create a div and inject it into the page, which has the heading. Whenever someone clicks on the div, it should be converted into an editable div. When the user clicks away (use blur event), save the note into the local storage.

If you are not familiar with the concept of local storage, check [***tutorial#20***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-20)**.**You are advised to participate in solving this problem. You cannot learn to code by just watching the code implementation until you start doing it yourself.

Have you solved this task? If yes, then it’s time to check the solution. The solution is discussed in [***tutorial#25***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-25)

If you like my work, then keep supporting and stay up to date with [**codewithharry**](http://www.codewithharry.com./)**.**

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading"> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this">

<li class="childul">this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul">of my dreams</li>

</ul>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<input type="button" value="submit">

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<script src="js/tut15.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log('Tutorial 21 exercise 2 JavaScript');

/\*

You have to create a div and inject it into the page which contains a heading.

whenever someone clicks on the div, it should be converted into an editable item. whenever user clicks away (blur). save the note into the local storage.

\*/

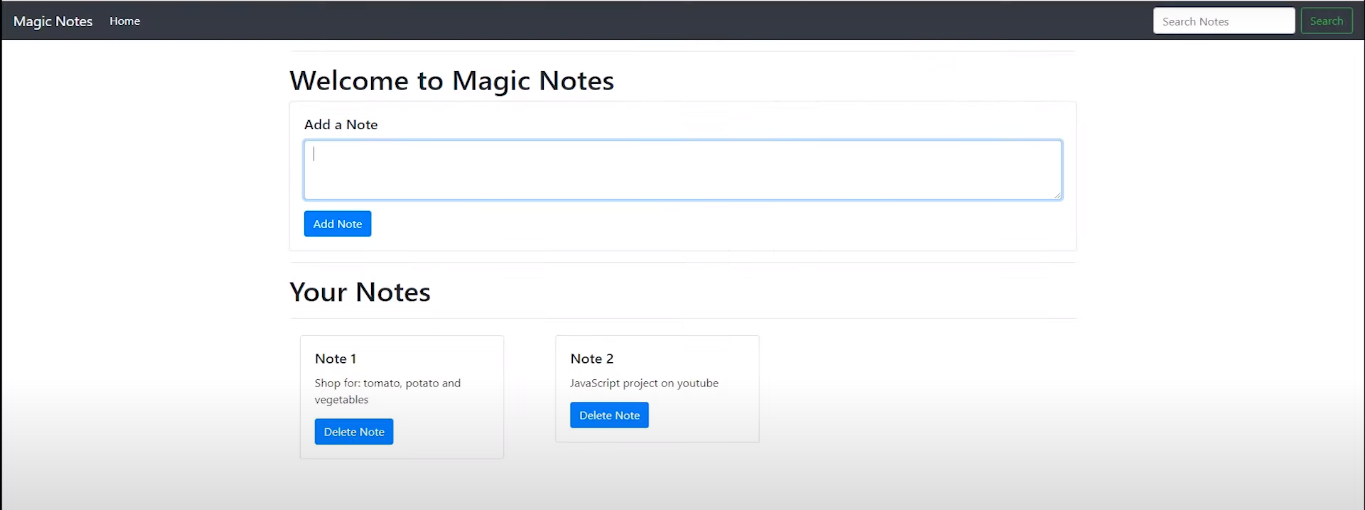
**Project 1: Build a Notes Taking Website Using Pure JavaScript | JavaScript Tutorial In Hindi #22**

Now it’s time to start our first project, which is to build a note taking website*.*Projects snowball your coding skills. Programmers are always learning new things from new projects. The project details are explained below.

**Project Explanation:**

The project we are building is ***“Note Taking Website App Using Pure JavaScript.”***We all have a notes app on our mobiles or laptop. **Note-taking** **websites**are the online equivalent of notebooks, and because they are digital, they can do more for you than paper ever could. You can even build a more advanced level note-taking website, but we will keep it simple for now, as it is our first project!

The website we are going to build in this tutorial looks like the following image:



**Setting Up the Project**

For this project, we will use the VS Code. **Visual Studio Code** is a free and powerful source-code editor that runs on the desktop. If you have not installed the Virtual Studio Code yet, then click on the link below for installation guidance:

***<https://codewithharry.com/videos/javascript-tutorials-in-hindi-1>***

The project setup is really simple. Because we only want to use JavaScript without any frameworks. Start by creating a new project folder in VS Code, and inside that project folder, create two empty new files: *index.html* and *app.js*. The *index.html file* is the entry point for our website and contains the HTML code. In this project, we are also using **Bootstrap**. If you are not familiar with Bootstrap, then do not worry; just follow the tutorial. The file *app.js* will be included in *index.html*.

**Bootstrap:-**

If you are studying web development, you must have come across the word Bootstrap. It is used for applying styling the user interface components. The easiest way to include Bootstrap is to add it from https://getbootstrap.com/docs/4.0/getting-started/introduction/.

 First add the following in the head section:

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

Copy

Then include the following code in the body section, before the closing </body> tag:

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

Copy

**Bootstrap** is bundled with many components that can be used to provide good user experience and user interactions in a web page like cards, navigation bars, dropdowns, icons, buttons, forms and also sizing options for different DOM elements. In this project, we are using bootstrap components for making the front end of website. Then after desiging thewebsite, we will add logic into it using pure javaScript.

So, let’s start coding!!!

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Notes App</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="#">Magic Notes</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent"

aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav mr-auto">

<li class="nav-item active">

<a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>

</li>

</ul>

<form class="form-inline my-2 my-lg-0">

<input class="form-control mr-sm-2" id="searchTxt" type="search" placeholder="Search" aria-label="Search">

<button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>

</form>

</div>

</nav>

<div class="container my-3">

<h1>Welcome To Magic Notes</h1>

<div class="card">

<div class="card-body">

<h5 class="card-title">Add a note</h5>

<div class="form-group">

<textarea class="form-control" id="addTxt" rows="3"></textarea>

</div>

<button class="btn btn-primary" id="addBtn">Add Note</button>

</div>

</div>

<hr>

<h1>Your Notes</h1>

<hr>

<div id="notes" class="row container-fluid"> </div>

</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

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

</body>

</html>

Copy

**JavaScript code as described/written in the video**

console.log("Welcome to notes app. This is app.js");

showNotes();

// If user adds a note, add it to the localStorage

let addBtn = document.getElementById("addBtn");

addBtn.addEventListener("click", function(e) {

let addTxt = document.getElementById("addTxt");

let notes = localStorage.getItem("notes");

if (notes == null) {

notesObj = [];

} else {

notesObj = JSON.parse(notes);

}

notesObj.push(addTxt.value);

localStorage.setItem("notes", JSON.stringify(notesObj));

addTxt.value = "";

// console.log(notesObj);

showNotes();

});

// Function to show elements from localStorage

function showNotes() {

let notes = localStorage.getItem("notes");

if (notes == null) {

notesObj = [];

} else {

notesObj = JSON.parse(notes);

}

let html = "";

notesObj.forEach(function(element, index) {

html += `

<div class="noteCard my-2 mx-2 card" style="width: 18rem;">

<div class="card-body">

<h5 class="card-title">Note ${index + 1}</h5>

<p class="card-text"> ${element}</p>

<button id="${index}"onclick="deleteNote(this.id)" class="btn btn-primary">Delete Note</button>

</div>

</div>`;

});

let notesElm = document.getElementById("notes");

if (notesObj.length != 0) {

notesElm.innerHTML = html;

} else {

notesElm.innerHTML = `Nothing to show! Use "Add a Note" section above to add notes.`;

}

}

// Function to delete a note

function deleteNote(index) {

// console.log("I am deleting", index);

let notes = localStorage.getItem("notes");

if (notes == null) {

notesObj = [];

} else {

notesObj = JSON.parse(notes);

}

notesObj.splice(index, 1);

localStorage.setItem("notes", JSON.stringify(notesObj));

showNotes();

}

let search = document.getElementById('searchTxt');

search.addEventListener("input", function(){

let inputVal = search.value.toLowerCase();

// console.log('Input event fired!', inputVal);

let noteCards = document.getElementsByClassName('noteCard');

Array.from(noteCards).forEach(function(element){

let cardTxt = element.getElementsByTagName("p")[0].innerText;

if(cardTxt.includes(inputVal)){

element.style.display = "block";

}

else{

element.style.display = "none";

}

// console.log(cardTxt);

})

})

/\*

Further Features:

1. Add Title

2. Mark a note as Important

3. Separate notes by user

4. Sync and host to web server

\*/

Copy

**PreviousNext**

CodeWithHarry

Copyright © 2022

**Math Object In JavaScript | JavaScript Tutorial In Hindi #23**

Date and time are the prominent features in computer programming. In JavaScript, suppose our task is to create a website with a calendar, a train schedule, or an interface to set up appointments. These applications need to show relevant times or perform calculations around arrivals and departures or start and end times. To perform all of these tasks, JavaScript comes with the Date object and related methods. In this tutorial, we will go over how to format and use date and time in JavaScript.

**Date Object:-**

The Date object is a built-in datatype into the JavaScript language. The Date object is used to deal with dates and times in JavaScript. Once a Date object is created, many methods allow us to operate on it. Most methods allow us to get and set the month, day, year, hour, minute, and second using either local time or universal time. There are four ways of instantiating a date:

let d = new Date();

let d = new Date(milliseconds);

let d = new Date(dateString);

let d = new Date(year, month, day, hours, minutes, seconds, milliseconds);

Copy

**Here is an example:-**

let mydate = new Date();

mydate.setHours(0);

alert(mydate);

mydate.setHours(0, 0, 0, 0);

alert(mydate);

Copy

There are many methods available to obtain a date in various formats and perform time zone conversions. There are methods to read and alter individual components of the local date and time, such as **getDay()** and **setHours()**. Versions of the same methods also exist to read and manipulate UTC's date and time, such as **getUTCDay()**and **setUTCHours()**. Here is the list of some of the data object methods:

**Date Object Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| getDate() | It will return the day of the month (from 1-31) |
| getDay() | It will return the day of the week (from 0-6) |
| getFullYear() | It will return the year |
| getHours() | It will return the hour (from 0-23) |
| getMilliseconds() | It will return the milliseconds (from 0-999) |
| getMinutes() | It will return the minutes (from 0-59) |
| getMonth() | It will return the month (from 0-11) |
| getSeconds() | It will return the seconds (from 0-59) |
| getTime() | It will return the number of milliseconds since midnight Jan 1 1970, and a specified date |
| getTimezoneOffset() | It will return the time difference between UTC time and local time, in minutes |
| getUTCDate() | It will return the day of the month, according to universal time (from 1-31) |
| getUTCHours() | It will return the hour, according to universal time (from 0-23) |
| getUTCMilliseconds() | It will return the milliseconds, according to universal time (from 0-999) |
| getUTCMinutes() | It will return the minutes, according to universal time (from 0-59) |
| getUTCMonth() | It will return the month, according to universal time (from 0-11) |
| getUTCSeconds() | It will return the seconds, according to universal time (from 0-59) |
| setDate() | It will set the day of the month of a date object |
| setFullYear() | It will set the year of a date object |
| setHours() | It will set the hour of a date object |
| setMilliseconds() | It will set the milliseconds of a date object |
| setMinutes() | It will set the minutes of a date object |
| setMonth() | It will set the month of a date object |
| setUTCHours() | It will set the hour of a date object, according to universal time |
| toDateString() | It will convert the date portion of a Date object into a readable string |
| toLocaleString() | It will convert a Date object to a string, using locale conventions |
| toString() | It will convert a Date object to a string |
| toTimeString() | It will convert the time portion of a Date object to a string |
| toUTCString() | It will convert a Date object to a string, according to universal time |
| UTC() | It will return the number of milliseconds in a date since midnight of January 1, 1970, according to UTC time |
| valueOf() | It will return the primitive value of a Date object |

**Conclusion**

In this tutorial, we learned how to create an instance of the Date object and use its built-in methods. It is very important to know how to work with dates is essential for many common tasks in JavaScript, as this can enable us to do many things from setting up a repeating report to displaying dates and schedules in the correct time zone.

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<script src="js/tut23.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log("Welcome to tutorial number 23");

let x = 3;

let y = 6;

let z;

z = x+y;

z = x-y;

z = x\*y;

z = x/y;

// Exploring the Math object

z = Math;

z = Math.PI;

z = Math.E;

z = Math.round(5.4);

z = Math.ceil(5.3);

z = Math.floor(-5.3);

z = Math.abs(5);

z = Math.sqrt(64);

z = Math.pow(2, 3);

z = Math.min(2, 3, 34,234, 2342,34);

z = Math.max(2, 3, 34,234,34);

z = Math.random();

z = 100\*Math.random()

z = Math.ceil(50 + (100-50)\*Math.random())

// a = (0, 1)

// a100 = a\*100 = (0, 100)

// a10\_100 = 10+a\*(100 - 10)

console.log(z);

// 3

// 2.8

// 2

// 1

// 0

// -1

// -2

// -2.7

// -3

**Date Object: Date & Time In JavaScript | JavaScript Tutorial In Hindi #24**

With the advent of time, mathematical calculations have become an integral part of every programming language. In this article, we will cover the details of the Math object in JavaScript, which helps the user to perform all kinds of mathematical operations.

**The JavaScript Math Object:-**

The Math object is used to perform mathematical tasks. It is a built-in static object, so we do not need to instantiate it. We can access all its properties and methods directly. Math object works with the Number type. Unlike many other global objects, Math is not a constructor. We refer to the constant pi as **Math.PI** and we call the cos function as **Math.cos(x),**where x is the method's argument.

**Note:**Different browsers can give a different result. Even the same JavaScript engine on a different OS or architecture can give different results!

**JavaScript Math Methods:-**

Let's see the list of JavaScript Math methods with descriptions.

|  |  |
| --- | --- |
| **Methods** | **Description** |
| abs() | It will return the absolute value of the given number. |
| acos() | It will return the arccosine of the given number in radians. |
| asin() | It will return the arcsine of the given number in radians. |
| atan() | It will return the arc-tangent of the given number in radians. |
| cbrt() | It will return the cube root of the given number. |
| ceil() | It will return a smallest integer value, greater than or equal to the given number. |
| cos() | It will return the cosine of the given number. |
| cosh() | It will return the hyperbolic cosine of the given number. |
| exp() | It will return the exponential form of the given number. |
| floor() | It will return the largest integer value, lower than or equal to the given number. |
| hypot() | It will return the square root of sum of the squares of given numbers. |
| log() | It will return the natural logarithm of a number. |
| max() | It will return the maximum value of the given numbers. |
| min() | It will return the minimum value of the given numbers. |
| pow() | It will return the value of base to the power of exponent. |
| random() | It will return the random number between 0 (inclusive) and 1 (exclusive). |
| round() | It will return the closest integer value of the given number. |
| sin() | It will return the sine of the given number. |
| sinh() | It will return the hyperbolic sine of the given number. |
| sqrt() | It will return the square root of the given number |
| tan() | It will return the tangent of the given number. |
| tanh() | It will return the hyperbolic tangent of the given number. |
| trunc() | It will return an integer part of the given number. |

**Converting between degrees and radians:-**

The trigonometric functions such as sin(), cos(),tan() expect angles in *radians*.

Since humans tend to think in degrees, and some functions can accept degrees, it is good to keep functions handy that convert between the two. Here is an example:

function degToRad(degrees) {

return degrees \* (Math.PI / 180);

};

function radToDeg(rad) {

return rad / (Math.PI / 180);

};

Copy

**Key Takeaways:-**

The math object is one of the most important features in JavaScript, as it provides various properties and methods which can perform mathematical operations. We can use mathematical constants such as PI and Log values using the properties provided by Math object such as Math.PI. In addition to the above, we can also invoke the methods provided by Math object such as abs(), ceil(), etc. to perform various mathematical operations on multiple variables.

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<script src="js/tut23.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log("Welcome to tutorial 24");

let today = new Date();

// console.log(typeof today);

let otherDate = new Date('8-4-2003 04:54:08');

// otherDate = new Date('June 13 1976');

// otherDate = new Date('09/16/1976');

console.log(otherDate);

let a;

a = otherDate.getDay();

a = otherDate.getDate();

a = otherDate.getMinutes();

// a = otherDate.getSeconds();

// a = otherDate.getHours();

a = otherDate.getTime();

a = otherDate.getMilliseconds();

a = otherDate.getMonth();

console.log(a);

otherDate.setDate(23);

otherDate.setMonth(0);

otherDate.setFullYear(1900);

otherDate.setMinutes(2);

otherDate.setHours(1);

otherDate.setSeconds(3);

console.log(otherDate);

**Creating an Editable Div Exercise 2: Solution | JavaScript Tutorial In Hindi #25**

This tutorial contains the solution to Exercise 2 i.e. ***Creating Editable Div Using JavaScript***. The problem statement along with the instructions is given below:

**Problem Statement:-**

You have to create a div and inject it into the page, which has the heading. Whenever someone clicks on the div, it should be converted into an editable div. When the user clicks away (use blur event), save the note into the local storage.

If you are not familiar with the concept of local storage, check [***tutorial#20***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-20)**.**You are advised to participate in solving this problem. You cannot learn to code by just watching the code implementation until you start doing it yourself.

Have you solved this task? If yes, then it’s time to check the solution. The solution is discussed in [***tutorial#25***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-25)

If you like my work, then keep supporting and stay up to date with [**codewithharry**](http://www.codewithharry.com./)**.**

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<script src="js/tut23.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log('This is tutorial 25')

/\*

You have to create a div and inject it into the page which contains a heading.

whenever someone clicks on the div, it should be converted into an editable item. whenever user clicks away (blur). save the note into the local storage.

\*/

// Create a new element

let divElem = document.createElement('div');

// Add text to that created element

let val = localStorage.getItem('text');

let text;

if (val==null){

text = document.createTextNode('This is my element. click to edit it');

}

else{

text = document.createTextNode(val);

}

divElem.appendChild(text);

// Give element id, style and class

divElem.setAttribute('id', 'elem');

divElem.setAttribute('class', 'elem');

divElem.setAttribute('style', 'border:2px solid black; width: 154px; margin: 34px; padding:23px;');

// Grab the main container

let container = document.querySelector('.container');

let first = document.getElementById('myfirst');

// Insert the element before element with id first

container.insertBefore(divElem, first);

console.log(divElem, container, first)

// add event listener to the divElem

divElem.addEventListener('click', function () {

let noTextAreas = document.getElementsByClassName('textarea').length;

if(noTextAreas ==0){

let html = elem.innerHTML;

divElem.innerHTML = ` <textarea class="textarea form-control" id="textarea" rows="3"> ${html}</textarea>`;

}

//listen for the blur event on textarea

let textarea = document.getElementById('textarea');

textarea.addEventListener('blur', function() {

elem.innerHTML = textarea.value;

localStorage.setItem('text', textarea.value);

})

});

**Object Literals, Constructors and Object Oriented JavaScript | JavaScript Tutorial In Hindi #27**

Before getting into the detail of JavaScript OOP concepts, let us first understand what Object-oriented programming (OOP) is. The basic idea of object-oriented programming is that objects are used to model real world things that we want to represent inside our programs. We can also describe OOP as the OOP provides a simple way to access functionality that would otherwise be hard or impossible to make use of.

**What are objects?**

Objects can contain related code and data, representing information about the thing we are trying to model, and functionality that we want it to have. Object data is stored neatly inside an object package, making it easy to structure and access.

JavaScript is a powerful programming language that supports Object-Oriented Programming (OOP). In JavaScript, objects created using object literal are **singletons**. This means when we made a change in an object, it affects the object entire the script. Whereas if an object is created using the constructor function, then the change will not affect the object throughout the script. In this tutorial,

// constructor function

function MyCar(){};

// literal notation

var myCar = {};

Copy

**Objects created using object literal:-**

Since these are **singletons**, a change to an object affects the object's entire script. A change in one instance will affect all the instances of the object. Object literal is a comma-separated list of name-value pairs inside the curly braces. The object literals encapsulate the data. This minimizes the use of global variables, which can cause problems when combining the code.

**Following is an example object literal**:

var myCar = {

make: 'Ford',

model: 'Mustang',

year: 2000

};

Copy

However, if we have to create multiple instances of a structure and perform operations based on predefined values, then we should use a function constructor.

**Objects created using the constructor function:-**

The object that is defined with a **function constructor lets** us have multiple instances of that object. If the change is made to one instance, it will not affect other instances. As we know that the constructor is essentially a function that acts as a blueprint for creating objects. A convention for declaring constructors is always to capitalize its function name.

function MyCar (make,model,year){

this.make = make;

this.model= model;

this.year = year;

}

Copy

Now we have our constructor function called **MyCar**. We can **"construct"** our object instances based on that blueprint. Let us create a MyCar instance object called "car1"

// creating objects

let car1 = new MyCar(“Ford”, “Mustang”, 2000);

Copy

**"new" keyword:-**

To create a new object instance, we use the**"new"** keyword to create an object based on a constructor. Adding the "new" keyword is a crucial step when creating an object from a constructor. The new keyword creates a new empty object. It binds property or function, which is declared with this keyword to the new object.

**Conclusion:-**

In this tutorial, we have learned about object literals and constructor functions. The main difference between them is that if we need multiple instances of our object, the object that is defined with a constructor lets us have multiple instances of that object. In comparison, the object literals are basically singletons with variables that are all public.

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<script src="js/tut23.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log("This is tut 27");

// Object Literal for creating objects

let car = {

name: "Maruti 800",

topSpeed: 89,

run: function() {

console.log("car is running");

}

};

// we have already seen constructors like these:

// new Date();

// Creating a constructor for cars

function GeneralCar(givenName, givenSpeed) {

this.name = givenName;

this.topSpeed = givenSpeed;

this.run = function() {

console.log(`${this.name} Is Running`);

};

this.analyze = function() {

console.log(

`This car is slower by ${200 - this.topSpeed} Km/H than Mercedes`

);

};

}

car1 = new GeneralCar("Nissan", 180);

car2 = new GeneralCar("Marutu Alto", 80);

car3 = new GeneralCar("Mercedes", 200);

console.log(car1, car2, car3);

**Object Prototype In javascript | JavaScript Tutorial In Hindi #28**

In this tutorial, we will discuss what prototypes are and how they help JavaScript achieve the concepts of Object-Oriented Programming. In the previous tutorial, we had learned various ways of creating objects in JavaScript. One way is to create an object in JavaScript is using the constructor function.

**Introduction to JavaScript prototype:-**

As we know that all objects in JavaScript are instances of Object. A typical object inherits properties from Object.prototype. The **Object.prototype** object has many built-in methods and properties such as toString(), valueOf(), etc.All objects see the object prototype object changes through prototype chaining unless the properties and methods subject to those changes are overridden further along the prototype chain. This is a very dangerous mechanism to override or extend object behavior.

JavaScript is a prototype-based language. Whenever we create a function using JavaScript, the JavaScript engine adds a *prototype* property inside a function. The prototype**property** is an object where we can attach methods and properties in a prototype object, which enables all the other objects to inherit these methods and properties. As we have studied in tutorial#27, one way to create an object is by using a function constructor.

function MyDetails(name, job, yearOfBirth){

this.name= name;

this.job= job;

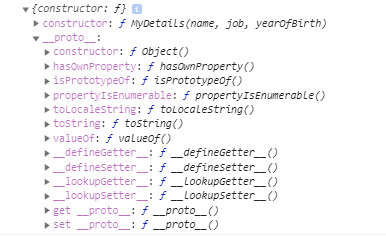
this.yearOfBirth= yearOfBirth;

}

console.log(MyDetails.prototype)

Copy

**Output:-**



The prototype object includes many properties and methods. Here is the explanation of a few prototype object methods.

**Methods and Description:**

* **hasOwnProperty():**It will return a boolean indicating whether an object contains the specified property as a direct property of that object and not inherited through the prototype chain.
* **isPrototypeOf():**It will return a boolean indicating whether the specified object is in the prototype chain of the object, this method is called upon.
* **propertyIsEnumerable():**It will return a boolean that indicates whether the specified property is enumerable or not.
* **toLocaleString():**It will return the string in the local format.
* **toString():**It will return the string.
* **valueOf() :**It will return the primitive value of the specified object.

**Use of Prototype:-**

JavaScript is using the prototype object in two things. First, one is to find properties and methods of an object, and the other is to implement inheritance in JavaScript. Here is an example:

f

function Student() {

this.name = 'Harry';

this.gender = 'Male';

}

Student.prototype.sayHi = function(){

console.log("Hello World!");

};

let std = new Student();

std.toString();

Copy

In the above code, the JavaScript engine checks whether the toString() method is attached to std or not. If it does not find there, it uses std \_proto\_\_ link, which points to the prototype object of Student function. If it still cannot find it there, then it goes up in the hierarchy and check the prototype object of Object function because all the objects are derived from Object in JavaScript, and look for the toString() method. Thus, it finds the toString() method in the prototype object of Object function and so we can call std.toString().

**Summary:-**

This tutorial has covered JavaScript object prototypes, the prototype property, and how it can add methods to constructors, and other related topics. In the next tutorial, we will look at how we can implement inheritance functionality in JavaScript.

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<script src="js/tut23.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log("This is tutorial 28");

// Object literal : Object.prototype

let obj = {

name: "harry",

channel: "Code With Harry",

address: "Mars"

}

function Obj(givenName){

this.name = givenName

}

Obj.prototype.getName = function (){

return this.name;

}

Obj.prototype.setName = function (newName){

this.name = newName;

}

let obj2 = new Obj("Rohan Das");

console.log(obj2);

**Notes App with Title - Exercise 3: Solution | JavaScript Tutorial In Hindi #29**

This tutorial contains the solution to ***Exercise 3 i.e. Improving the Magic Notes Website***. The problem statement along with the instructions is given below:

**Problem Statement:-**

Before working on this exercise, make sure you have watched the [***tutorial#22***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-22), in which we build a note-taking website using pure JavaScript. In this exercise, you have to edit that website so that each note has a title.

**Task:-**

The task you have to do is ***“Improving Magic Notes Website”***. For this, add a title option in the “Add a Note” section. So that when the user adds the note, each note has a title.

 This tutorial is a part of the JavaScript Tutorials for Beginners series. If you have not accessed this series, click on the link below:

***[https://codewithharry.com/videos/javascript-tutorials-in-hindi-1](https://codewithharry.com/videos/javascript-tutorials-in-hindi-1" \t "_blank)***

If you like my work, then keep supporting and stay up to date with***[codewithharry](http://www.codewithharry.com./" \t "_blank).***

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Notes App</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="#">Magic Notes</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent"

aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav mr-auto">

<li class="nav-item active">

<a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>

</li>

</ul>

<form class="form-inline my-2 my-lg-0">

<input class="form-control mr-sm-2" id="searchTxt" type="search" placeholder="Search"

aria-label="Search">

<button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>

</form>

</div>

</nav>

<div class="container my-3">

<h1>Welcome To Magic Notes</h1>

<div class="card">

<div class="card-body">

<div class="form-group">

<h5 class="card-title">Add title</h5>

<input type="text" class="form-control" id="addTitle" aria-describedby="title"

placeholder="Enter title">

</div>

<h5 class="card-title">Add a note</h5>

<div class="form-group">

<textarea class="form-control" id="addTxt" rows="3"></textarea>

</div>

<button class="btn btn-primary" id="addBtn">Add Note</button>

</div>

</div>

<hr>

<h1>Your Notes</h1>

<hr>

<div id="notes" class="row container-fluid"> </div>

</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

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

</body>

</html>

Copy

**JavaScript code as described/written in the video**

console.log("Welcome to notes app. This is app.js");

showNotes();

// If user adds a note, add it to the localStorage

let addBtn = document.getElementById("addBtn");

addBtn.addEventListener("click", function(e) {

let addTxt = document.getElementById("addTxt");

let addTitle = document.getElementById("addTitle");

let notes = localStorage.getItem("notes");

if (notes == null) {

notesObj = [];

} else {

notesObj = JSON.parse(notes);

}

let myObj = {

title: addTitle.value,

text: addTxt.value

}

notesObj.push(myObj);

localStorage.setItem("notes", JSON.stringify(notesObj));

addTxt.value = "";

addTitle.value = "";

// console.log(notesObj);

showNotes();

});

// Function to show elements from localStorage

function showNotes() {

let notes = localStorage.getItem("notes");

if (notes == null) {

notesObj = [];

} else {

notesObj = JSON.parse(notes);

}

let html = "";

notesObj.forEach(function(element, index) {

html += `

<div class="noteCard my-2 mx-2 card" style="width: 18rem;">

<div class="card-body">

<h5 class="card-title">${element.title}</h5>

<p class="card-text"> ${element.text}</p>

<button id="${index}"onclick="deleteNote(this.id)" class="btn btn-primary">Delete Note</button>

</div>

</div>`;

});

let notesElm = document.getElementById("notes");

if (notesObj.length != 0) {

notesElm.innerHTML = html;

} else {

notesElm.innerHTML = `Nothing to show! Use "Add a Note" section above to add notes.`;

}

}

// Function to delete a note

function deleteNote(index) {

// console.log("I am deleting", index);

let notes = localStorage.getItem("notes");

if (notes == null) {

notesObj = [];

} else {

notesObj = JSON.parse(notes);

}

notesObj.splice(index, 1);

localStorage.setItem("notes", JSON.stringify(notesObj));

showNotes();

}

let search = document.getElementById('searchTxt');

search.addEventListener("input", function(){

let inputVal = search.value.toLowerCase();

// console.log('Input event fired!', inputVal);

let noteCards = document.getElementsByClassName('noteCard');

Array.from(noteCards).forEach(function(element){

let cardTxt = element.getElementsByTagName("p")[0].innerText;

if(cardTxt.includes(inputVal)){

element.style.display = "block";

}

else{

element.style.display = "none";

}

// console.log(cardTxt);

})

})

/\*

Further Features:

1. Add Title

2. Mark a note as Important

3. Separate notes by user

4. Sync and host to web server

\*/

Copy

**PreviousNext**

CodeWithHarry

Copyright © 2022 Code

**Prototype Inheritance In JavaScript | JavaScript Tutorial In Hindi #30**

In the previous tutorial, we went over what object is, how to create an object, and how to access and modify object properties. Now in this tutorial, we will learn how prototypes can be used to extend objects. As we know that JavaScript is a **prototype-based language**. The object's properties and methods can be shared through generalized objects that can be cloned and extended. This sharing is known as ***prototypical inheritance.***

Every object in JavaScript has an internal property called prototype. Here is an empty object i.e., let a = {}. In this way, we normally create an object, but note that another way to accomplish this is with the object constructor: let a = new Object(). We have discussed the object in the [***tutorial#27***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-27). Before understanding the prototype inheritance, let us first understand what inheritance is.

**Inheritance:-**

Inheritance let the objects to share each other's properties. In other words, Inheritance is the process by which one object can be based on another.

**Prototype Inheritance:**

When we try to access a property or any object's method, the JavaScript will first search on the object itself, and then it will search the object's prototype if not found. If, after checking both the object and its prototype still no match is found, JavaScript will check the linked object prototype and continue searching until the end of the prototype chain is reached. Object.prototype is at the end of the prototype chain. All the objects inherit the properties and methods of Object. When we try to search beyond the end of the chain, results in null. There are different ways to create an object in JavaScript.

* Object literal
* Function constructor
* The create() method

We have already discussed the object literal and function constructor in ***[tutorial#27](https://codewithharry.com/videos/javascript-tutorials-in-hindi-27)***. Today we will discuss **object.create()** method.

**Object.create():-**

The **Object.create()** method using an existing object as the prototype, creates a new object. The syntax is:

Object.create(proto, [propertiesObject])

Copy

Here is an example:

const myDetail= {

isHuman: true,

printIntro: function() {

console.log(`My name is ${this.name}. Am I human? ${this.isHuman}`);

}};

const myself = Object.create(myDetail);

myself.name = 'Harry'; // "name" is a property set on "me", but not on "person"

myself.printIntro();

// expected output: "My name is Harry. Am I human? true"

Copy

To create inheritance between function constructors, call the parent constructor using call or link the prototype of the child constructor to the parent constructor prototype.

**Using call to chain constructors for an object:-**

The call() allows the method or function belonging to one object to be assigned and called for another object.This method provides a new value of this to the method or function. With call(), we can write a method once and then inherit it in another object without rewriting the new object's method.

**For example**, the constructor for the Factory object is defined with two parameters: name and location. Other functions, Food, invoke Factory, passing this, name, and location. Factory initializes the property's name and location; the specialized functions define the category.

function Factory (name, location) {

this.name = name;

this.location = location;

}

function Food(name, location) {

Factory.call(this, name, location);

this.category = 'food';

}

const myFood = new Food('Nestle', 'UK' );

Copy

**Manually set the constructor:-**

The **constructor** property is used to return a reference to the object constructor function that created the instance object. We mostly use this property to define a **function-constructor** function by calling it with **a new** and prototype-inherits chain. Here is a simple example of how to set the constructor manually:

function Factory() {

/\* ... \*/

}

Factory.prototype.FactoryMethod = function FactoryMethod() {}

function Product1() {

Factory.call(this)

}

Product1.prototype.constructor = Product1

Copy

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<!-- <script src="js/tut23.js"></script> -->

<!-- <script src="js/tut24.js"></script> -->

<!-- <script src="js/tut25.js"></script> -->

<!-- <script src="js/tut27.js"></script> -->

<!-- <script src="js/tut28.js"></script> -->

<script src="js/tut30.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log("This is tutorial 30");

const proto = {

slogan: function () {

return `This company is the best`;

},

changeName: function (newName) {

this.name = newName

}

}

// This creates harry object

let harry = Object.create(proto);

harry.name = "harry";

harry.role = "Programmer";

// harry.changeName("Harry2")

// console.log(harry)

// This also creates harry object

const harry1 = Object.create(proto, {

name: { value: "harry", writable: true },

role: { value: "Programmer" },

});

harry1.changeName("Harry2")

// console.log(harry1)

// Employee constructor

function Employee(name, salary, experience) {

this.name = name;

this.salary = salary;

this.experience = experience;

}

// Slogan

Employee.prototype.slogan = function () {

return `This company is the best. Regards, ${this.name}`;

}

// Create an object from this constructor now

let harryObj = new Employee("Harry", 345099, 87);

console.log(harryObj.slogan())

// Programmer

function Programmer(name, salary, experience, language) {

Employee.call(this, name, salary, experience);

this.language = language;

}

// Inherit the prototype

Programmer.prototype = Object.create(Employee.prototype);

// Manually set the constructor

Programmer.prototype.constructor = Programmer;

let rohan = new Programmer("Rohan", 2, 0, "Javascript");

console.log(rohan);

**ES6 Classes and Inheritance | JavaScript Tutorial In Hindi #31**

In today's tutorial, we will study about ES6 Classes and inheritance. But first, let us revise the concept of OOP in javascript. In a programming language, the objects provide us with an easy way to model data. Suppose we have an object called car*.* The *car*object has **properties**: values that contain data about the car, and the **methods**: functions that define actions that the car can perform. The "objects" and "actions" are the basis of **Object-Oriented Programming**. The notion behind writing the code is that it more closely reflects how we think in reality. In **Object-Oriented Programming**, everything is an object, and any actions we need to perform on data are written as methods of an object. Going back to the *car* object, even in the smallest real-world application, we are going to have multiple *car* objects. If we have to code each one, we may as well go back to pen and paper.

**What are the ES6 Classes?**

**ES6 Classes** is the common JavaScript pattern of getting the class-like inheritance hierarchies using functions and prototypes. They are effectively simple compared to prototype-based OO, offering a convenient declarative form for class patterns that encourage interoperability.

Here is an example of ES6 Classes in javaScript:

class SimpleDate {

constructor(year, month, day) {

this.\_year = year;

this.\_month = month;

this.\_day = day;

}

addMonth(nMonths) {

}

getMonth() {

return this.month;

}

}

Copy

**Constructors:-**

The constructor method's purpose is to initialize an instance to a valid state, and it will be called automatically, so we cannot forget to initialize our objects. The constructor function is an object blueprint. From the above example, the following is the constructor of class SimpleDate:

constructor(year, month, day) {

this.\_year = year;

this.\_month = month;

this.\_day = day;

}

Copy

**Defining Methods:-**

The common practice is to assign methods directly to the prototype instead of in the initialization. But, with classes, this syntax is simplified, and we can add the method to the class. Using the method definition introduced in ES6 JavaScript, defining a method is an even more easy and concise process.

class Car {

constructor(name,year ) {

this.name = name;

this.year = year;

}

greet() {

return `${this.name} says hello.`;

}}

Copy

We will create a new instance of Car using the new keyword and assign some values.

const car1 = new Car('Audi', 2018);

Copy

**Inheritance:-**

Inheritance is one of the most important concepts of object-oriented programming. The significant advantage of using ES6 classes over pre-ES6 functions is that class definitions are cleaner and easy to inherit. Without writing prototype code for an inheritance, class inheritance in JavaScript is easier and similar to other object-oriented languages like Java. Following is the way of applying the inheritance concept in javaScript:

**Extending a Class:-**

The most useful feature of constructor functions and classes is that they can be extended into new object blueprints based on the parent. Before es6, the new constructor functions can be created from the parent using the call() method. However, with the help of ES6 classes, the super keyword is used in place of the call to access the parent functions. We will use the extends keyword to refer to the parent class.

//new class from the parent

class Bike extends Car {

constructor(name,year,speed) {

//constructor with super

super(name, year);

// Adding new property

this.speed = speed;

}}

Copy

Now we create a new Bike instance in the same manner.

const bike1 = new Bike(' Trek', 2019,200);

Copy

**Conclusion:-**

In this tutorial, we learned about the similarities and differences between JavaScript constructor functions and ES6 classes. Being familiar with classes is extremely helpful, as this makes the coding easier.

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<!-- <script src="js/tut23.js"></script> -->

<!-- <script src="js/tut24.js"></script> -->

<!-- <script src="js/tut25.js"></script> -->

<!-- <script src="js/tut27.js"></script> -->

<!-- <script src="js/tut28.js"></script> -->

<!-- <script src="js/tut30.js"></script> -->

<script src="js/tut31.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log("this is Tutorial31.js");

class Employee {

constructor(givenName, givenExperience, givenDivision) {

this.name = givenName;

this.experience = givenExperience;

this.division = givenDivision;

}

slogan(){

return `I am ${this.name} and this company is the best`;

}

joiningYear(){

return 2020 - this.experience;

}

static add(a, b){

return a + b;

}

}

class Programmer extends Employee{

constructor(givenName, givenExperience, givenDivision, language, github){

super(givenName, givenExperience, givenDivision);

this.language = language;

this.github = github;

}

favoriteLanguage(){

if (this.language == 'python'){

return 'Python';

}

else{

return 'JavaScript';

}

}

static multiply(a, b){

return a \* b;

}

}

// harry = new Employee("Harry", 56, "Division");

// console.log(harry.joiningYear());

// console.log(Employee.add(34, 5))

rohan = new Programmer("Rohan", 3, "Lays", "Go", "Rohan420")

console.log(rohan)

console.log(rohan.favoriteLanguage())

console.log(Programmer.multiply(5, 7));

Copy

**Previous**

**Implementing a library class - Exercise 4 | JavaScript Tutorial In Hindi #32**

The task you have to perform is to **“Implement the Library class”.**You have to use the object-oriented programming concept to implement this task. Here is a problem statement.

**Problem Statement:-**

Create a library class and implement the following:

* Create a constructor that takes the book list as an argument.
* **getBookList():** This function should return the list of all book present in the library.
* **issueBook(bookname, user):** This function takes an argument of bookname and user, in bookname, we specify the name of the book, and in user, we specify the name of the user. This function will issue the book to the user.
* **returnBook(bookname):** This function will take one argument, which is the name of the book. The purpose of this function is to remove the book from the users borrowed book data. It is totally up to you how you implement it.

Try to solve this task by yourself. Have you solved this task? If yes, then it is time to check the solution. The solution is discussed in [***tutorial#36***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-36)

If you like my work, then keep supporting and stay up to date with [**codewithharry**](http://www.codewithharry.com./)**.**

**Website.html code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<!-- <script src="js/tut23.js"></script> -->

<!-- <script src="js/tut24.js"></script> -->

<!-- <script src="js/tut25.js"></script> -->

<!-- <script src="js/tut27.js"></script> -->

<!-- <script src="js/tut28.js"></script> -->

<!-- <script src="js/tut30.js"></script> -->

<script src="js/tut31.js"></script>

</html>

Copy

**JavaScript code as described/written in the video**

console.log("This is tutorial 32")

// create a class library and implement the following:

// constructor must take the book list as an argument

// getBookList()

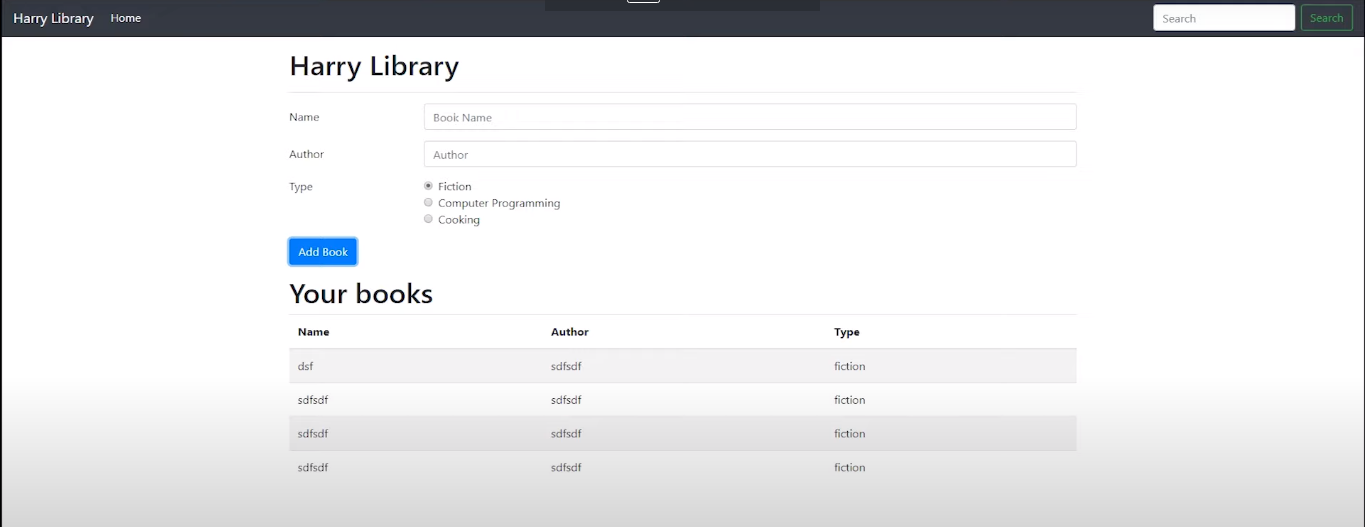
// issueBook(bookname, user)

// returnBook(bookname)

**Project 2: Building a College Library Website | JavaScript Tutorial In Hindi #33**

Let us start our 2nd project, in which we will build the**“College Library Website”**. In this project, we will not only revise the concepts we studied till now, but we will explore some new tips of the VS Code as well.

The website we are going to build in this tutorial looks like the following:



**Setting Up the Project:-**

For this project, we will use the VS Code. **Visual Studio Code** is a free and powerful source-code editor that runs on the desktop. If you have not installed the Visual Studio Code yet, then click on the link below for installation guidance:

***<https://codewithharry.com/videos/javascript-tutorials-in-hindi-1>***

Start by creating a new project folder in VS Code, and inside that project folder, create two empty new files: *index.html* and *indexes6.js*. The *index.html file* contains the HTML code. In this project, we are also using **Bootstrap**.

**Bootstrap:-**

Bootstrap is used for applying styling the user interface components. The easiest way to include Bootstrap is to add it from https://getbootstrap.com/docs/4.0/getting-started/introduction/.

 First add the following in the head section:

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

Copy

Then include the following code in the body section, before the closing </body> tag:

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

Copy

**Bootstrap** also provides us many components that can be used to provide good user experience and user interactions in a web page like cards, navigation bars, dropdowns, icons, buttons, forms and also sizing options for different DOM elements. So, now we are done with project introduction, now let’s start coding!

**Code index.html as described/written in the video**

<!doctype html>

<html lang="en">

<head>

<!-- Required meta tags -->

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<!-- Bootstrap CSS -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title>Welcome to Harry Potter Library</title>

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="#">Harry Library</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent"

aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav mr-auto">

<li class="nav-item active">

<a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>

</li>

</ul>

<form class="form-inline my-2 my-lg-0">

<input class="form-control mr-sm-2" id="searchTxt" type="search" placeholder="Search"

aria-label="Search">

<button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>

</form>

</div>

</nav>

<div id="message"></div>

<div class="container my-3">

<h1>Harry Library</h1>

<hr>

<form id="libraryForm">

<div class="form-group row">

<label for="bookName" class="col-sm-2 col-form-label">Name</label>

<div class="col-sm-10">

<input type="text" class="form-control" id="bookName" placeholder="Book Name">

</div>

</div>

<div class="form-group row">

<label for="Author" class="col-sm-2 col-form-label">Author</label>

<div class="col-sm-10">

<input type="text" class="form-control" id="author" placeholder="Author">

</div>

</div>

<fieldset class="form-group">

<div class="row">

<legend class="col-form-label col-sm-2 pt-0">Type</legend>

<div class="col-sm-10">

<div class="form-check">

<input class="form-check-input" type="radio" name="type" id="fiction" value="fiction"

checked>

<label class="form-check-label" for="fiction">

Fiction

</label>

</div>

<div class="form-check">

<input class="form-check-input" type="radio" name="type" id="programming"

value="programming">

<label class="form-check-label" for="programming">

Computer Programming

</label>

</div>

<div class="form-check disabled">

<input class="form-check-input" type="radio" name="type" id="cooking" value="cooking">

<label class="form-check-label" for="cooking">

Cooking

</label>

</div>

</div>

</div>

</fieldset>

<div class="form-group row">

<div class="col-sm-10">

<button type="submit" class="btn btn-primary">Add Book</button>

</div>

</div>

</form>

<div id="table">

<h1>Your books</h1>

<table class="table table-striped">

<thead>

<tr>

<th scope="col">Name</th>

<th scope="col">Author</th>

<th scope="col">Type</th>

</tr>

</thead>

<tbody id='tableBody'></tbody>

</table>

</div>

</div>

<!-- Optional JavaScript -->

<!-- jQuery first, then Popper.js, then Bootstrap JS -->

<!-- <script src="index.js"></script> -->

<script src="indexes6.js"></script>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

</body>

</html>

Copy

**Code index.js as described/written in the video**

console.log("This is index.js");

// Todos"

// 1. Store all the data to the localStorage

// 2. Give another column as an option to delete the book

// 3. Add a scroll bar to the view

// Constructor

function Book(name, author, type) {

this.name = name;

this.author = author;

this.type = type;

}

// Display Constructor

function Display() {

}

// Add methods to display prototype

Display.prototype.add = function (book) {

console.log("Adding to UI");

tableBody = document.getElementById('tableBody');

let uiString = `<tr>

<td>${book.name}</td>

<td>${book.author}</td>

<td>${book.type}</td>

</tr>`;

tableBody.innerHTML += uiString;

}

// Implement the clear function

Display.prototype.clear = function () {

let libraryForm = document.getElementById('libraryForm');

libraryForm.reset();

}

// Implement the validate function

Display.prototype.validate = function (book) {

if (book.name.length < 2 || book.author.length < 2) {

return false

}

else {

return true;

}

}

Display.prototype.show = function (type, displayMessage) {

let message = document.getElementById('message');

message.innerHTML = `<div class="alert alert-${type} alert-dismissible fade show" role="alert">

<strong>Messge:</strong> ${displayMessage}

<button type="button" class="close" data-dismiss="alert" aria-label="Close">

<span aria-hidden="true">×</span>

</button>

</div>`;

setTimeout(function () {

message.innerHTML = ''

}, 2000);

}

// Add submit event listener to libraryForm

let libraryForm = document.getElementById('libraryForm');

libraryForm.addEventListener('submit', libraryFormSubmit);

function libraryFormSubmit(e) {

console.log('YOu have submitted library form');

let name = document.getElementById('bookName').value;

let author = document.getElementById('author').value;

let type;

let fiction = document.getElementById('fiction');

let programming = document.getElementById('programming');

let cooking = document.getElementById('cooking');

if (fiction.checked) {

type = fiction.value;

}

else if (programming.checked) {

type = programming.value;

}

else if (cooking.checked) {

type = cooking.value;

}

let book = new Book(name, author, type);

console.log(book);

let display = new Display();

if (display.validate(book)) {

display.add(book);

display.clear();

display.show('success', 'Your book has been successfully added')

}

else {

// Show error to the user

display.show('danger', 'Sorry you cannot add this book');

}

e.preventDefault();

}

Copy

**Code indexes6.js as described/written in the video**

console.log('This is ES6 version of Project 2');

class Book {

constructor(name, author, type) {

this.name = name;

this.author = author;

this.type = type;

}

}

class Display {

add(book) {

console.log("Adding to UI");

let tableBody = document.getElementById('tableBody');

let uiString = `<tr>

<td>${book.name}</td>

<td>${book.author}</td>

<td>${book.type}</td>

</tr>`;

tableBody.innerHTML += uiString;

}

clear() {

let libraryForm = document.getElementById('libraryForm');

libraryForm.reset();

}

validate(book) {

if (book.name.length < 2 || book.author.length < 2) {

return false

}

else {

return true;

}

}

show(type, displayMessage) {

let message = document.getElementById('message');

let boldText;

if(type==='success'){

boldText = 'Success';

}

else{

boldText = 'Error!';

}

message.innerHTML = `<div class="alert alert-${type} alert-dismissible fade show" role="alert">

<strong>${boldText}:</strong> ${displayMessage}

<button type="button" class="close" data-dismiss="alert" aria-label="Close">

<span aria-hidden="true">×</span>

</button>

</div>`;

setTimeout(function () {

message.innerHTML = ''

}, 5000);

}

}

// Add submit event listener to libraryForm

let libraryForm = document.getElementById('libraryForm');

libraryForm.addEventListener('submit', libraryFormSubmit);

function libraryFormSubmit(e) {

console.log('YOu have submitted library form');

let name = document.getElementById('bookName').value;

let author = document.getElementById('author').value;

let type;

let fiction = document.getElementById('fiction');

let programming = document.getElementById('programming');

let cooking = document.getElementById('cooking');

if (fiction.checked) {

type = fiction.value;

}

else if (programming.checked) {

type = programming.value;

}

else if (cooking.checked) {

type = cooking.value;

}

let book = new Book(name, author, type);

console.log(book);

let display = new Display();

if (display.validate(book)) {

display.add(book);

display.clear();

display.show('success', 'Your book has been successfully added')

}

else {

// Show error to the user

display.show('danger', 'Sorry you cannot add this book');

}

e.preventDefault();

}

**What is Asynchronous Programming? | JavaScript Tutorial In Hindi #34**

In today's tutorial, we will explore the concept of asynchronous programming in JavaScript, and see why it is important, and how we can use it to effectively handle potential blocking operations such as fetching resources from a server. Asynchronous programming is a fairly advanced JavaScript topic, and you are advised to watch the previous tutorials first before going through this.

As we know that the JavaScript is single-threaded and can only handle one operation at a time. As there is a single execution thread for our program to run, a question arises that how to execute a long-running operation without blocking the thread of execution? Well, welcome to asynchronous programming.

**What is the difference between synchronous and asynchronous programming?**

In ***synchronous* programming**, one thing happens at a time. When we call a function that performs a long-running action, it returns a result when the action has finished. This stops the program for the time the action takes. In contrast, ***asynchronous* programming** allows multiple things to happen at the same time. When we start an action, the program continues to run. When the action finishes, the program is informed and gets the result.

Let's compare synchronous and asynchronous programming using an example: a program that fetches two resources from the network and then combines results.

In synchronous programming, where the request function returns only after it has done its work. To perform this task, we make the requests one after the other. Here the drawback is that the second request will be started only when the first has finished. Suppose the time taken by the first request is 12 seconds, and the time taken by the second request is 13 seconds, so the total time taken will be the sum of the two response times.

In asynchronous programming, the functions that perform a slow action takes an extra argument, a *callback function*. The action is started, and when it finishes, the callback function is called with the result.

For example, the setTimeout function waits a given number of milliseconds (a second is a thousand milliseconds) and then calls a function.

setTimeout(() => console.log("Tick"), 500);

Copy

Waiting is useful when doing something like updating an animation or checking whether something takes longer than a given amount of time.

Performing multiple asynchronous actions in a row using callbacks means that we have to keep passing new functions to handle the program's continuation after the actions.

**Summary:-**

Asynchronous code does not have to wait; the program can continue to run. The asynchronous programming makes it possible to express waiting for long-running actions without stopping the program during these actions. JavaScript usually implements this style of programming using callbacks.

Asynchronous programming is made easier by using promises, objects that represent actions that might complete in the future, and async functions, which allow us to write an asynchronous program as if it were synchronous.

**Code website.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<!-- <script src="js/tut23.js"></script> -->

<!-- <script src="js/tut24.js"></script> -->

<!-- <script src="js/tut25.js"></script> -->

<!-- <script src="js/tut27.js"></script> -->

<!-- <script src="js/tut28.js"></script> -->

<!-- <script src="js/tut30.js"></script> -->

<!-- <script src="js/tut31.js"></script> -->

<!-- <script src="js/tut32.js"></script> -->

<script src="js/tut34.js"></script>

</html>

Copy

**Code tut34.js as described/written in the video**

console.log("This is tutorial 34");

setTimeout(() => {

for (let index = 0; index < 4005; index++) {

const element = index;

console.log("This is index number" + index);

}

}, 100);

console.log('done printing');

Copy

**Previous**

**Ajax tutorial in hindi | JavaScript Tutorial In Hindi #35**

In this tutorial, we will learn what Ajax is and how it works. If you want to learn **AJAX** quickly, then you are at right place. In this tutorial, we will not only learn AJAX theoretically but also learn how to implement the AJAX concept in programming.

Ajax stands for **A**synchronous **J**avaScript **A**nd **XML**. Ajax loads the data from the server and updating the parts of a web page selectively without reloading the whole page.

**Introduction of AJAX:-**

AJAX is a technique for creating faster, and more interactive web applications with the help of XML, HTML, CSS, and JavaScript. It is a web browser technology which is independent of web server software.

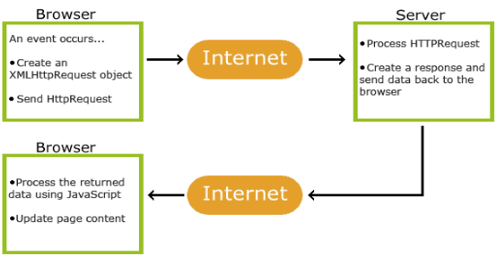
AJAX use of the built-in browser **XMLHttpRequest**(XHR) objects to send and receive information to and from a web server asynchronously, in the background, without blocking the page or interfering with the user's experience.

Ajax uses XHTML for the content, CSS for designing, along with Document Object Model and JavaScript for dynamic content display. Before AJAX technology, the web applications transmit information to and from the server using synchronous requests. It this we fill out a form, hit submit, and get directed to a new page with new information from the server. But with AJAX, when we hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen.

We can guess the popularity of AJAX, such that we hardly find an application that doesn't use Ajax to some extent. The example of Ajax-driven online applications are Gmail, Google Maps, YouTube, Facebook, and so many other applications.

**How Does AJAX Work?**

JavaScript and XML combine to make asynchronous updating happen through the use of something called an XMLHttpRequest (XHR) object. When the user visits a web page which is designed using AJAX technology, and a prescribed event occurs (a button, or fills out a form) the JavaScript creates an XMLHttpRequest (XHR) object, which then transfers data in an XML format between a web browser and a web. The XMLHttpRequest(XHR) object sends a request for updated page data to the web server, the server process the request, a response is created at server-side and sent back to the browser, which then uses JavaScript to process the response and display it on the screen as updated content.



**Summary:-**

The JavaScript automates the updating process, the request for updated content is formatted in XML to make it understandable, and JavaScript again refreshes the relevant content for the user viewing the page. Whereas the AJAX technique ignores extraneous page data and only handles requests for updated information and the updated information itself. This shows the AJAX's effectiveness, as it makes the websites and applications that use AJAX faster and more responsive to users.

**Code ajax.html as described/written in the video**

<!doctype html>

<html lang="en">

<head>

<!-- Required meta tags -->

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<!-- Bootstrap CSS -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title>Ajax in one video - CodeWithHarry</title>

</head>

<body>

<h1>Ajax tutorial</h1>

<button type="button" id="fetchBtn" class="btn btn-primary">Fetch Data</button>

<button type="button" id="popBtn" class="btn btn-secondary">Populate</button>

<div class="container">

<h1>Employee list</h1>

<ul id="list">

</ul>

</div>

<!-- Optional JavaScript -->

<!-- jQuery first, then Popper.js, then Bootstrap JS -->

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

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

</body>

</html>

Copy

**Code ajax.js as described/written in the video**

console.log("Ajax tutorial in one video");

let fetchBtn = document.getElementById('fetchBtn');

fetchBtn.addEventListener('click', buttonClickHandler)

function buttonClickHandler() {

console.log('You have clicked the fetchBtn');

// Instantiate an xhr object

const xhr = new XMLHttpRequest();

// Open the object

// xhr.open('GET', 'https://jsonplaceholder.typicode.com/todos/1', true);

// USE THIS FOR POST REQUEST

xhr.open('POST', 'http://dummy.restapiexample.com/api/v1/create', true);

xhr.getResponseHeader('Content-type', 'application/json');

// What to do on progress (optional)

xhr.onprogress = function(){

console.log('On progress');

}

// xhr.onreadystatechange = function () {

// console.log('ready state is ', xhr.readyState);

// }

// What to do when response is ready

xhr.onload = function () {

if(this.status === 200){

console.log(this.responseText)

}

else{

console.log("Some error occured")

}

}

// send the request

params = `{"name":"test34sad545","salary":"123","age":"23"}`;

xhr.send(params);

console.log("We are done!");

}

let popBtn = document.getElementById('popBtn');

popBtn.addEventListener('click', popHandler);

function popHandler() {

console.log('You have clicked the pop handler');

// Instantiate an xhr object

const xhr = new XMLHttpRequest();

// Open the object

xhr.open('GET', 'http://dummy.restapiexample.com/api/v1/employees', true);

// What to do when response is ready

xhr.onload = function () {

if(this.status === 200){

let obj = JSON.parse(this.responseText);

console.log(obj);

let list = document.getElementById('list');

str = "";

for (key in obj)

{

str += `<li>${obj[key].employee\_name} </li>`;

}

list.innerHTML = str;

}

else{

console.log("Some error occured")

}

}

// send the request

xhr.send();

console.log("We are done fetching employees!");

}

Copy

**PreviousNext**

CodeWithHarry

C

**Library Class Implementation Exercise 4: Solution | JavaScript Tutorial In Hindi #36**

This tutorial contains the solution to Exercise 4, i.e. Implementing a library class. The problem statement, along with the instructions, is given below:

**Problem Statement:-**

Create a library class and implement the following:

* Create a constructor that takes the book list as an argument.
* **getBookList():** This function should return the list of all book present in the library.
* **issueBook(bookname, user):** This function takes an argument of bookname and user, in bookname, we specify the name of the book, and in user, we specify the name of the user. This function will issue the book to the user.
* **returnBook(bookname):** This function will take one argument, which is the name of the book. The purpose of this function is to remove the book from the users borrowed book data. It is totally up to you how you implement it.

**Code ajax.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Tutorial on Js</title>

</head>

<body>

<h1>This is Js tutorial by Harry</h1>

</body>

<!-- <script src="js/tut2.js"></script> -->

<!-- <script src="js/tut3.js"></script> -->

<!-- <script src="js/tut4.js"></script> -->

<!-- <script src="js/tut5.js"></script> -->

<!-- <script src="js/tut6.js"></script> -->

<!-- <script src="js/tut7.js"></script> -->

<!-- <script src="js/tut8.js"></script> -->

<!-- <script src="js/tut9.js"></script> -->

<!-- <script src="js/tut10.js"></script> -->

<!-- <script src="js/tut11.js"></script> -->

<script src="js/tut36.js"></script>

</html>

Copy

**Code ajax.js as described/written in the video**

console.log("This is tutorial 32 - solution")

// create a class library and implement the following:

// constructor must take the book list as an argument

// getBookList()

// issueBook(bookname, user)

// returnBook(bookname)

class Library {

constructor(bookList){

this.bookList = bookList;

this.issuedBooks = {};

}

getBookList(){

this.bookList.forEach(element => {

console.log(element);

});

}

issueBook(bookname, user){

if (this.issuedBooks[bookname] ==undefined){

this.issuedBooks[bookname] = user;

}

else{

console.log("This book is already issued!");

}

}

returnBook(bookname){

delete this.issuedBooks[bookname];

}

}

**Callback functions in javascript | JavaScript Tutorial In Hindi #37**

In the previous lecture, we have discussed what functions do and how to use them. But now the question is, what is a**callback functio**n? Callback functions are one of the most important parts of JavaScript, and once we understand how it works, we will become much better in JavaScript. So, in this tutorial, we will understand what callback functions are and how to use them in JavaScript.

**What is a Callback?**

*A callback is a function that is executed after another function has finished executing.* *As we have studied earlier that functions are objects. Functions can take functions as arguments and can be returned by other functions. Functions that take another function as an argument are called*higher-order functions*. So, the callback function can also be defined as any function that is passed as an argument is called a****callback function****.*

**Why do we need Callbacks?**

As we know, instead of waiting for a response, JavaScript will keep executing while listening for other events. Here is an example:

function fun1(){

console.log(“Code With Harry”);

}

function fun2(){

console.log(“JavaScript Tutorial”);

}

first();

second();

Copy

The function fun1 is executed first, and the function fun2 is executed second. JavaScript runs code from top to bottom. However, there are some cases that code must run after something else happens and also not using a top-down approach.

Callbacks are used to make sure that a function is not going to execute before a task is completed but will execute right after the task has completed. It helps us develop asynchronous JavaScript code and keeps us safe from future errors.

In JavaScript, the way to create a callback function is to pass it as a parameter to another function, and then to call it back right after something has happened or some task is completed.

**How to create a Callback:-**

Suppose we want to log a message to the console but it should be there after 5 seconds.

function myMessage (str) {

setTimeout(() => {

// script to download the picture here

console.log(`Code With Harry`);

}, 5000);

}

Copy

There is a built-in method called **“setTimeout”**, which calls a function after a given period of time. So here, the myMessage function is being called after 5 seconds. **(1 second = 1000 milliseconds)**. In other words, the myMessage function is being called after something happened (after 5 seconds passed for this example), but not before. So, the myMessage function is an example of a callback function.

Here is an example of callback functions in JavaScript:

function addition(x, y , callback){

setTimeout(() => {

document.write(`The sum of ${x} and ${y} is ${x+y}.`);

callback();

}, 5000); }

// display() function is called just after the ending of addition() function

function display(){

document.write('Numbers added!');

}

// Calling addition() function

addition(5,5,display);

Copy

Here are the two functions – addition(x, y, callback) and display(). Here addition() is called after 5 seconds with the display() function, i.e. passed in as the third argument to the addition function along with two numbers x and y. As a result, the addition() is invoked with 1, 2 and the display() which is the callback. The addition() prints the addition of the two numbers(x+y), and as soon as that is done, the callback function is executed.

**Output:-**

The sum of 5 and 5 is 10.

Numbers added!

Copy

**Code ajax.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

<div class="container">

<h1>Student list</h1>

<ul id="students"></ul>

<button id="myBtn" class="btn btn-primary">Your Button</button>

<button class="btn btn-primary">Fetch Data</button>

<div id="content"></div>

</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<!-- <script src="js/tut23.js"></script> -->

<!-- <script src="js/tut24.js"></script> -->

<!-- <script src="js/tut25.js"></script> -->

<!-- <script src="js/tut27.js"></script> -->

<!-- <script src="js/tut28.js"></script> -->

<!-- <script src="js/tut30.js"></script> -->

<!-- <script src="js/tut31.js"></script> -->

<!-- <script src="js/tut32.js"></script> -->

<!-- <script src="js/tut34.js"></script> -->

<!-- <script src="js/tut37.js"></script> -->

<!-- <script src="js/tut39.js"></script> -->

<!-- <script src="js/tut39b.js"></script> -->

<!-- <script src="js/tut41.js"></script> -->

<!-- <script src="js/tut43.js"></script> -->

<script src="js/tut44.js"></script>

</html>

Copy

**Code ajax.js as described/written in the video**

console.log("This is tutorial 37");

// Pretend that this response is coming from the server

const students = [

{name: "harry", subject: "JavaScript"},

{name: "Rohan", subject: "Machine Learning"}

]

function enrollStudent(student, callback){

setTimeout(function() {

students.push(student);

console.log("Student has been enrolled");

callback();

}, 1000);

}

function getStudents(){

setTimeout(function() {

let str = "";

students.forEach(function(student){

str += `<li> ${student.name}</li>`

});

document.getElementById('students').innerHTML = str;

console.log("Students have been fetched");

}, 5000);

}

let newStudent = {name:"Sunny", subject:"Python"}

enrollStudent(newStudent, getStudents);

// getStudents();

**Using a Dictionary API Exercise 5 | JavaScript Tutorial In Hindi #38**

Today’s task is to print some data using a Dictionary API. Search words API from Google and in the Try It section, you will find a response. Copy the response and consider that the response is coming from the API.

Here is a link of wordsapi: <https://www.wordsapi.com/>. Here are the instructions for the task:

**Instructions:-**

* You have pretend to use a word API which will contain an object. Your task is to print definition from all the results of that word API.
* You have to take input from an input tag.
* Print the result in the DOM.
* If you are using bootstrap then its a plus

Have you solved this task? If yes, then it is time to check the solution. The solution is discussed in ***[tutorial#45](https://codewithharry.com/videos/javascript-tutorials-in-hindi-45)***

If you like my work, then keep supporting and stay up to date with **[codewithharry](http://www.codewithharry.com./).**

**Code as described/written in the video**

// You have pretend to use a word api which will contain an object and you hve to print definition from all the results of that word api.

// YOu have to take input from an input tag.

// You have to print it in the dom

// If you are using bootstrap then its a plus

Copy

**PreviousNext**

CodeWithHarry

Copyright © 20

**Promises Basics, Promise.then() & Promise.catch() | JavaScript Tutorial In Hindi #39**

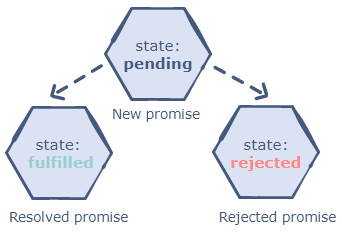
Promises is a way through which we can deal with asynchronous operations in JavaScript. To understand this tutorial, check out my tutorial about [***JavaScript Callbacks***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-37). As we know that **callback functions** were initially used to handle asynchronous operations. However, callbacks functions were limited in terms of functionality and often led to confusing code, so, **promises** were introduced to deal with these problems. Many people struggle with understanding how Promises work especially beginners, so in this tutorial, we will try to understand promises by making it as simple as we can. In this tutorial, you will find an introduction to what Promises are, explanations of terms like resolve, reject along with coding examples.

**What is a Promise?**

A promise in JavaScript is similar to a promises we do in real life. When we make a promise, it is a guarantee that in future, we are going to do something. A promise has two possible outcomes: it will be kept when the time comes, or it will not. Similarly, in JavaScript, when we define a promise, either it will be resolved when the time comes, or it will get rejected. According to [MDN](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise), “the Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.”

A promise has three states:

* **pending**: It is the initial state.
* **Fulfilled**: It indicates that the promised operation was successful.
* **Rejected**: It indicates that the promised operation was unsuccessful.



**The constructor syntax for a promise object is:**

let myPromise = new Promise(function(resolve, reject) {

// code here

});

Copy

When new Promise is created, the function passed into it runs automatically. It contains the producing code which produces the result. Its arguments resolve and reject. Here is an example of simple promise.

var promise = new Promise(function(resolve, reject) {

const x = "geeksforgeeks";

const y = "geeksforgeeks"

if(x === y) {

resolve();

} else {

reject();

} });

promise.then(function(){

console.log('Success, You are a GEEK');}).catch(function () {

console.log('Some error has occurred');});

Copy

A promise is created using a constructor that takes a call back function with two arguments resolve and reject in line 1. If the task is successful(x===y), the promise is resolved. If the task is unsuccessful(x is not equal to y), then the promise is rejected. The then() method is called if the promise is resolved, and the catch() method is called when the promise is rejected or if an error occurred during the code execution.

Promises are the ideal choice for asynchronous programming. Promises can handle multiple asynchronous operations easily and are better at error handling as compared to callbacks and events.

**Benefits of Promises:-**

1. It improves the code readability
2. It is better in the handling of asynchronous operations
3. It has a better flow of control definition in asynchronous logic
4. It is better in error handling

**Wrap Up:-**

So, in this tutorial, we have learned how we create a Promise in JavaScript and use it for the resolved and rejected cases. If you are a beginner, understanding how promises they work might take time.

**Code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

<div class="container">

<h1>Student list</h1>

<ul id="students"></ul>

<button id="myBtn" class="btn btn-primary">Your Button</button>

<button class="btn btn-primary">Fetch Data</button>

<div id="content"></div>

</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<!-- <script src="js/tut23.js"></script> -->

<!-- <script src="js/tut24.js"></script> -->

<!-- <script src="js/tut25.js"></script> -->

<!-- <script src="js/tut27.js"></script> -->

<!-- <script src="js/tut28.js"></script> -->

<!-- <script src="js/tut30.js"></script> -->

<!-- <script src="js/tut31.js"></script> -->

<!-- <script src="js/tut32.js"></script> -->

<!-- <script src="js/tut34.js"></script> -->

<!-- <script src="js/tut37.js"></script> -->

<!-- <script src="js/tut39.js"></script> -->

<!-- <script src="js/tut39b.js"></script> -->

<!-- <script src="js/tut41.js"></script> -->

<!-- <script src="js/tut43.js"></script> -->

<script src="js/tut44.js"></script>

</html>

Copy

**Code as described/written in the video**

// Promise: Best video on promises

// -resolve

// -reject

// -pending

function func1() {

return new Promise(function (resolve, reject) {

setTimeout(() => {

const error = true;

if (!error) {

console.log('Function: Your promise has been resolved')

resolve();

}

else {

console.log('Function: Your promise has not been resolved')

reject('Sorry not fulfilled');

}

}, 2000);

})

}

func1().then(function(){

console.log("Harry: Thanks for resolving")

}).catch(function(error){

console.log("Harry: Very bad bro. Reason: " + error)

})

Copy

**Code tut 39b.js as described/written in the video**

console.log("This is tutorial 37");

// Pretend that this response is coming from the server

const students = [

{ name: "harry", subject: "JavaScript" },

{ name: "Rohan", subject: "Machine Learning" }

]

function enrollStudent(student) {

return new Promise(function (resolve, reject) {

setTimeout(function () {

students.push(student);

console.log("Student has been enrolled");

const error = false;

if (!error) {

resolve();

}

else {

reject();

}

}, 1000);

})

}

function getStudents() {

setTimeout(function () {

let str = "";

students.forEach(function (student) {

str += `<li> ${student.name}</li>`

});

document.getElementById('students').innerHTML = str;

console.log("Students have been fetched");

}, 5000);

}

let newStudent = { name: "Sunny", subject: "Python" }

enrollStudent(newStudent).then(getStudents).catch(function () {

console.log("Some error occured");

});

// getStudents();

// function inside then is ran as - resolve()

// function inside catch is ran as - reject()

Copy

**PreviousNext**

CodeWithHarry

**Project 3: Creating a News Website | JavaScript Tutorial In Hindi #40**

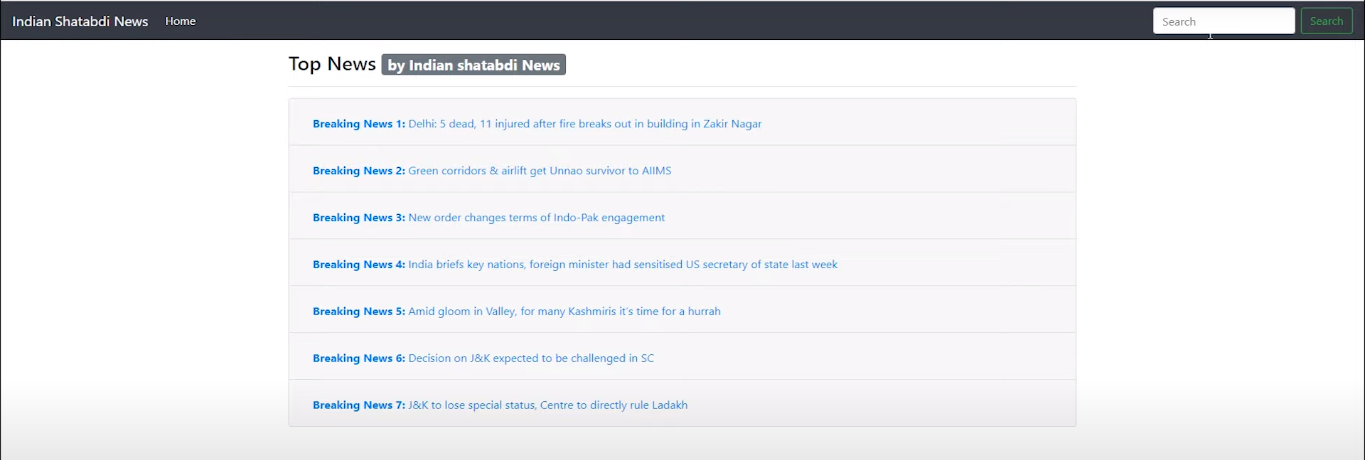
Let us start our 3rd project, in which we will build the **“News Website”.** This website will use news API. The ***News API*** is a simple and easy-to-***use*** API that returns JSON metadata for headlines and articles live all over the web right now. This website will show live news to the user, which will be displayed with the help of news API.

**To get the News API, follow the instructions:**

1. Click on <https://newsapi.org/>
2. Click on “ Get API Key”
3. Register for API Key

Now you will get your API Key, which you will use in the project. Don’t use the API key, which is used in the project. Get your own API key. The procedure to use News API in the project is practically performed in the project. Just follow the tutorial.

The News Website we are going to build in this tutorial looks like the following:



**Setting Up the Project:-**

The IDEwill use the for this project is VS Code. **Visual Studio Code** is a free and powerful source-code editor that runs on the desktop. If you have not installed the Visual Studio Code yet, then click on the link below for installation guidance:

***<https://codewithharry.com/videos/javascript-tutorials-in-hindi-1>***

Start by creating a new project folder in VS Code, and inside that project folder, create two empty new files: *index.html* and *inddel.js*. The *index.html file* contains the HTML code. In this project, we are also using **Bootstrap**.

**Bootstrap:-**

Bootstrap is used for applying styling the user interface components. The easiest way to include Bootstrap is to add it from https://getbootstrap.com/docs/4.0/getting-started/introduction/.

 First add the following in the head section:

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

Copy

Then include the following code in the body section, before the closing </body> tag:

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

Copy

Now we are done with project introduction, now let’s start coding!

**Code index.html as described/written in the video**

<!doctype html>

<html lang="en">

<head>

<!-- Required meta tags -->

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<!-- Bootstrap CSS -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title>Welcome To Indian Shatabdi News</title>

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="#">Indian Shatabdi News</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent"

aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav mr-auto">

<li class="nav-item active">

<a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>

</li>

</ul>

<form class="form-inline my-2 my-lg-0">

<input class="form-control mr-sm-2" type="search" placeholder="Search" aria-label="Search">

<button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>

</form>

</div>

</nav>

<div class="container my-3">

<h3>Top News <span class="badge badge-secondary">by Indian shatabdi News</span></h3>

<hr>

<div class="accordion" id="newsAccordion"></div>

</div>

<!-- Optional JavaScript -->

<!-- jQuery first, then Popper.js, then Bootstrap JS -->

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

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

</body>

</html>

Copy

**Code index.js as described/written in the video**

console.log("This is my index js file");

// Initialize the news api parameters

let source = 'the-times-of-india';

let apiKey = 'd093053d72bc40248998159804e0e67d'

// Grab the news container

let newsAccordion = document.getElementById('newsAccordion');

// Create an ajax get request

const xhr = new XMLHttpRequest();

xhr.open('GET', `https://newsapi.org/v2/top-headlines?sources=${source}&apiKey=${apiKey}`, true);

// What to do when response is ready

xhr.onload = function () {

if (this.status === 200) {

let json = JSON.parse(this.responseText);

let articles = json.articles;

console.log(articles);

let newsHtml = "";

articles.forEach(function(element, index) {

// console.log(element, index)

let news = `<div class="card">

<div class="card-header" id="heading${index}">

<h2 class="mb-0">

<button class="btn btn-link collapsed" type="button" data-toggle="collapse" data-target="#collapse${index}"

aria-expanded="false" aria-controls="collapse${index}">

<b>Breaking News ${index+1}:</b> ${element["title"]}

</button>

</h2>

</div>

<div id="collapse${index}" class="collapse" aria-labelledby="heading${index}" data-parent="#newsAccordion">

<div class="card-body"> ${element["content"]}. <a href="${element['url']}" target="\_blank" >Read more here</a> </div>

</div>

</div>`;

newsHtml += news;

});

newsAccordion.innerHTML = newsHtml;

}

else {

console.log("Some error occured")

}

}

xhr.send()

Copy

**PreviousNext**

CodeWithHarry

Copyright © 202

**Arrow functions in Javascript | JavaScript Tutorial In Hindi #41**

In this tutorial, we will learn about how to use the JavaScript **arrow function** to write more concise code for function expressions. Before learning about arrow function expressions, first, let us briefly review traditional JavaScript functions in order to better show the difference of arrow functions from traditional JavaScript functions.

As we have studied in the previous tutorial, the function is a group of reusable code which can be called anywhere in the program. This eliminates the need to rewrite the same code. The function declaration is a named function written with the function keyword. To get more detailed knowledge about functions, check the [***tutorial#10***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-10).

 Here is an example of a multiply function that returns the product of two parameters:

function mul(a, b) {

return a \* b

}//calling function

mul(3,3) //returns 9

Copy

**What are Arrow Functions?**

One of the most famous features in modern JavaScript is the arrow function.ES6 arrow functions provide us an alternative way to write a more concise and shorter syntax compared to the traditional function expression. Here is an syntax of arrow functions:

**let myfunc = (arg1, arg2, ...argN) => expression**

Let’s see another example of traditional function expression that adds two numbers:

let addition = function(x,y) {

return x + y;

}

console.log(addition(10,10)); // returns 20

Copy

Now lets see an example of arrow function which is equivalent to the above addition() function expression:

let addition = (x,y) => x + y;

console.log(addition(10,10)); // 20;

Copy

In the above example, the arrow function has only one expression x + y so it returns the result of the expression (10+10=20).

Here is an another example of arrow functions that will print “ Hello World”:

greet = () => {

return "Hello World!";

}

Copy

If the function has one statement, and the statement returns a value or string, we can remove the brackets *and* the return keyword:

greet = () => "Hello World!";

Copy

**Limitations of Arrow Functions:-**

An **arrow function expression** is an alternative to a traditional function expression, but there are some limitations:

* Arrow functions do not have its own bindings to this or super, and should not be used as methods.
* It is not suitable for the call, apply and bind methods, which generally rely on establishing a scope.
* Arrow functions cannot be used as constructors.

Arrow functions are a powerful addition to ES6, but we have to be careful while using them. There are some places where arrow functions are not usable, and this can cause difficulty for us to track errors, especially if we do not understand how they really work. Arrow functions are the best choice when working with closures or callbacks, but it is not a good choice when working with object methods or constructors.

**Summary:-**

Arrow functions are handy for one-liners. They come in two flavors:

1. Without curly braces: (...args) => expression, where at the right side is an expression. The function evaluates the expression and returns the result.
2. With curly braces: (...args) => { body }, the brackets allow us to write multiple statements inside the function, but in such scenario, we need an explicit return to return something.

**Code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div>

<div class="container">

<h1>Student list</h1>

<ul id="students"></ul>

<button id="myBtn" class="btn btn-primary">Your Button</button>

<button class="btn btn-primary">Fetch Data</button>

<div id="content"></div>

</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<!-- <script src="js/tut23.js"></script> -->

<!-- <script src="js/tut24.js"></script> -->

<!-- <script src="js/tut25.js"></script> -->

<!-- <script src="js/tut27.js"></script> -->

<!-- <script src="js/tut28.js"></script> -->

<!-- <script src="js/tut30.js"></script> -->

<!-- <script src="js/tut31.js"></script> -->

<!-- <script src="js/tut32.js"></script> -->

<!-- <script src="js/tut34.js"></script> -->

<!-- <script src="js/tut37.js"></script> -->

<!-- <script src="js/tut39.js"></script> -->

<!-- <script src="js/tut39b.js"></script> -->

<!-- <script src="js/tut41.js"></script> -->

<!-- <script src="js/tut43.js"></script> -->

<script src="js/tut44.js"></script>

</html>

Copy

**Code as described/written in the video**

console.log('this is tutorial 41');

// ARROW FUNCTIONS

// Creating a regular function

// const harry = function (){

// console.log("This is the best person ever")

// }

// Converting it to an arrow function

// const harry = ()=>{

// console.log("This is the best person ever")

// }

// harry();

// function returning something

// const greet = function(){

// return "Good Morning";

// }

// One liners do not require braces/return

// one line will automatically return

// const greet = () => "Good Morning";

// const greet = () => ({name: "harry"});

// Single parameters do not need parenthesis

// but you will have to put parenthesis if there are multiple paramteres

const greet = name => "Good Morning " + name + ending;

console.log(greet('Harry'))

**Using JavaScript Objects Exercise 5: Solution | JavaScript Tutorial In Hindi #45**

This tutorial contains the solution to Exercise 5, i.e.**Using a Dictionary API**. The problem statement, along with the instructions, is given below:

**Task:-**

Search words API from Google and in the Try It section, you will find a response. Copy the response and consider that the response is coming from the API.

Here is a link of wordsapi: <https://www.wordsapi.com/>. Here are the instructions for the task:

**Instructions:-**

* You have pretend to use a word API which will contain an object. Your task is to print definition from all the results of that word API.
* You have to take input from an input tag.
* Print the result in the DOM.
* If you are using bootstrap then its a plus

Have you solved this task? If yes, then it is time to check the solution. The solution is discussed in ***[tutorial#45](https://codewithharry.com/videos/javascript-tutorials-in-hindi-45)***

If you like my work, then keep supporting and stay up to date with **[codewithharry](http://www.codewithharry.com./).**

**Code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<!-- <br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div> -->

<div class="container">

<h1>Student list</h1>

<ul id="students"></ul>

<button id="myBtn" class="btn btn-primary">Your Button</button>

<button class="btn btn-primary">Fetch Data</button>

<div id="content"></div>

</div>

<div class="container my-3">

<h1>Pull the results by clicking the button below:</h1>

<button class="btn btn-primary" id="meanings">Get meanings</button>

<div class="my-2">

<ul id="defs"></ul>

</div>

</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<!-- <script src="js/tut23.js"></script> -->

<!-- <script src="js/tut24.js"></script> -->

<!-- <script src="js/tut25.js"></script> -->

<!-- <script src="js/tut27.js"></script> -->

<!-- <script src="js/tut28.js"></script> -->

<!-- <script src="js/tut30.js"></script> -->

<!-- <script src="js/tut31.js"></script> -->

<!-- <script src="js/tut32.js"></script> -->

<!-- <script src="js/tut34.js"></script> -->

<!-- <script src="js/tut37.js"></script> -->

<!-- <script src="js/tut39.js"></script> -->

<!-- <script src="js/tut39b.js"></script> -->

<!-- <script src="js/tut41.js"></script> -->

<!-- <script src="js/tut43.js"></script> -->

<!-- <script src="js/tut44.js"></script> -->

<script src="js/tut 45.js"></script>

</html>

Copy

**Code as described/written in the video**

console.log("This is tutorial 45");

const myJson = `{

"word": "example",

"results": [

{

"definition": "a representative form or pattern",

"partOfSpeech": "noun",

"synonyms": [

"model"

],

"typeOf": [

"representation",

"internal representation",

"mental representation"

],

"hasTypes": [

"prefiguration",

"archetype",

"epitome",

"guide",

"holotype",

"image",

"loadstar",

"lodestar",

"microcosm",

"original",

"paradigm",

"pilot",

"prototype",

"template",

"templet",

"type specimen"

],

"derivation": [

"exemplify"

],

"examples": [

"I profited from his example"

]

},

{

"definition": "something to be imitated",

"partOfSpeech": "noun",

"synonyms": [

"exemplar",

"good example",

"model"

],

"typeOf": [

"ideal"

],

"hasTypes": [

"pacemaker",

"pattern",

"beauty",

"prodigy",

"beaut",

"pacesetter"

],

"derivation": [

"exemplify",

"exemplary"

]

},

{

"definition": "an occurrence of something",

"partOfSpeech": "noun",

"synonyms": [

"case",

"instance"

],

"typeOf": [

"happening",

"natural event",

"occurrence",

"occurrent"

],

"hasTypes": [

"clip",

"mortification",

"piece",

"time",

"humiliation",

"bit"

],

"derivation": [

"exemplify"

],

"examples": [

"but there is always the famous example of the Smiths"

]

},

{

"definition": "an item of information that is typical of a class or group",

"partOfSpeech": "noun",

"synonyms": [

"illustration",

"instance",

"representative"

],

"typeOf": [

"information"

],

"hasTypes": [

"excuse",

"apology",

"specimen",

"case in point",

"sample",

"exception",

"quintessence",

"precedent"

],

"derivation": [

"exemplify",

"exemplary"

],

"examples": [

"this patient provides a typical example of the syndrome",

"there is an example on page 10"

]

},

{

"definition": "punishment intended as a warning to others",

"partOfSpeech": "noun",

"synonyms": [

"deterrent example",

"lesson",

"object lesson"

],

"typeOf": [

"monition",

"admonition",

"word of advice",

"warning"

],

"derivation": [

"exemplary"

],

"examples": [

"they decided to make an example of him"

]

},

{

"definition": "a task performed or problem solved in order to develop skill or understanding",

"partOfSpeech": "noun",

"synonyms": [

"exercise"

],

"typeOf": [

"lesson"

],

"examples": [

"you must work the examples at the end of each chapter in the textbook"

]

}

],

"syllables": {

"count": 3,

"list": [

"ex",

"am",

"ple"

]

},

"pronunciation": {

"all": "ɪɡ'zæmpəl"

},

"frequency": 4.67

}`;

const myObj = JSON.parse(myJson);

console.log('The object is :', myObj);

console.log('The results in the object are :', myObj['results']);

let meanings = document.getElementById('meanings')

meanings.addEventListener('click', ()=>{

console.log('someone clicked meanings');

populate();

})

function populate() {

let results = myObj['results'];

let html = "";

results.forEach(element => {

html += `<li class="list-group-item list-group-item-light">One of the definitions of example is ${element.definition} </li>`;

});

let defs = document.getElementById('defs');

defs.innerHTML = html;

}

Copy

**PreviousNext**

CodeWithHarry

Cop

**JavaScript Regular Expressions - Basic Functions | JavaScript Tutorial In Hindi #46**

When we first see the Regular Expressions, they may seem like a random string. While they might look awkward because they have confusing syntax, they are also extremely useful. This tutorial acts as an introduction to using regular expressions in JavaScript. In this tutorial, we will cover some useful ways to use them. As we know, regular expressions are notably hard to read as they gain in complexity, but it is necessary for the developer to have some knowledge of regular expressions to know what is being tested.

**What are Regular Expressions ?**

Regular expressions are the patterns that are used to match character combinations in strings. Regular expressions are a powerful way of doing search and replace in strings. It is a small language which is a part of many programming languages like JavaScript, Python, and Java. Regular expression allows us to check a string of characters like a password for patterns, to see if the set password matches with the pattern defined by that regular expression. In this tutorial, we will create a regular expression by using forward slashes ( / ) to enclose the pattern.

**Syntax:-**

/*pattern*/*modifiers*;

**Example:-**

let str = /Code with Harry/i;

Copy

Here **/**Code with Harry**/i** is a regular expression.**“**Code with Harry” is a pattern and “**i**” is a modifier that modifies the search to be case-insensitive. If we write **/**Code with Harry**/g, here “g”**performs a global match that will find all matches rather than stopping after the first match.

**Regular Expressions Methods:-**

**Regular expressions** are used with the RegExp methods like test() and exec() and with the string methods match() , replace() , search() , and split() . These methods are explained in detail below with examples.

**exec():-**

This method will executes a search for a match in a string. It returns an array of information or null on a mismatch. Here is an example:

let obj = /h/.exec("Code with harry");

Copy

**test():-**

This method tests for a match in a string. It returns true or false. Here is an example:

let str = /Code/;

str.test("Code with harry!");

Copy

**match():-**

This method will return an array containing all of the matches, including capturing groups, or null if no match is found. Here is an example:

let str = "JavaScipt tutorial from begineer to advance level";

let result = str.match(/ial/);

Copy

**search():-**

This method will tests for a match in a string. It returns the index of the match, or -1 if the search fails. Here is an example:

let str = 'Code with harry';

let reg = /od/;

// search if the pattern is in string variable

let result = str.search(reg);

console.log(result);

Copy

**replace():-**

This method will executes a search for a match in a string, and replaces the matched substring with a replacement substring. Here is an example:

let str = "Code with abc!";

let result = str.replace("abc", "Harry");

Copy

**split():-**

This method uses a regular expression or a fixed string to break a string into an array of substrings. Here is an example:

// splitting strings into array elements

const test = /[\s,]+/;

let res = 'Hello world! '.split(re);

console.log(result); // ["I", "am", "learning", "JavaScript", "RegEx"]

Copy

**Code website.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<!-- <br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div> -->

<div class="container">

<h1>Student list</h1>

<ul id="students"></ul>

<button id="myBtn" class="btn btn-primary">Your Button</button>

<button class="btn btn-primary">Fetch Data</button>

<div id="content"></div>

</div>

<div class="container my-3">

<h1>Pull the results by clicking the button below:</h1>

<button class="btn btn-primary" id="meanings">Get meanings</button>

<div class="my-2">

<ul id="defs"></ul>

</div>

</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<!-- <script src="js/tut23.js"></script> -->

<!-- <script src="js/tut24.js"></script> -->

<!-- <script src="js/tut25.js"></script> -->

<!-- <script src="js/tut27.js"></script> -->

<!-- <script src="js/tut28.js"></script> -->

<!-- <script src="js/tut30.js"></script> -->

<!-- <script src="js/tut31.js"></script> -->

<!-- <script src="js/tut32.js"></script> -->

<!-- <script src="js/tut34.js"></script> -->

<!-- <script src="js/tut37.js"></script> -->

<!-- <script src="js/tut39.js"></script> -->

<!-- <script src="js/tut39b.js"></script> -->

<!-- <script src="js/tut41.js"></script> -->

<!-- <script src="js/tut43.js"></script> -->

<!-- <script src="js/tut44.js"></script> -->

<!-- <script src="js/tut 45.js"></script> -->

<script src="js/tut46.js"></script>

</html>

Copy

**Code tut46.js as described/written in the video**

console.log("This is tutorial 46");

let reg = /harry/; // This is a regular expression literal in js

reg = /harry/g; // g means global

// reg = /harry/i; // i means case insensitive

// console.log(reg);

// console.log(reg.source);

let s = "This is great code with harry and also harry bhai";

// Functions to match expressions

// 1. exec() - This function will return an array for match or null for no match

let result = reg.exec(s);

// result = reg.exec(s);

// console.log(result);

// result = reg.exec(s);

// console.log(result); ---> We can use multiple exec with global flag

// if (result) {

// console.log(result);

// console.log(result.input);

// console.log(result.index);

// }

// 2. test() - Returns true or false

let result2 = reg.test(s);

// console.log(result2); --> This will only print true if the "reg" is there in the string "s"

// 3. match() - It will return an array of results or null

// let result3 = reg.match(s) ---> This is wrong!!

let result3 = s.match(reg) // ---> This is right

// console.log(result3);

// 4. search() - Returns index of first match else -1

// let result4 = reg.search(s) ---> This is wrong!!

let result4 = s.search(reg) // ---> This is right

// console.log(result4);

// 5. replace() - Returns new replaced string with all the replacements (if no flag is given, first match will be replaced)

let result5 = s.replace(reg, 'SHUBHAM');

console.log(result5);

Copy

**PreviousNext**

CodeWithHarry

Copyri

**Regular Expressions - Metacharacters in JavaScript | JavaScript Tutorial In Hindi #47**

In the previous tutorial, we studied about the regular expression and it’s methods. If you have not watch the tutorial on basics of regular expression, check [***tutorial#46***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-46). In today’s tutorial, we explore **metacharacters in JavaScript**. This tutorial gets us up to speed quickly with regular expressions.

As we know, Regular expressions are the patterns that is used to match character combinations in strings. *Regular Expression*, or *regex* in short, is extremely and amazingly powerful in searching and manipulating text strings. Regex is supported in all the scripting languages (such as Python, PHP, and JavaScript); as well as general purpose programming languages such as Java. Getting started with regex may not be easy due to its confusing syntax, but it is certainly worth the investment of our time.

**Metacharacters in Regular Expressions:-**

Metacharacters are the building blocks of regular expressions. Characters in Regular expression are understood to be either a metacharacter with a special meaning or a regular character with a literal meaning. Following are some common metacharacters in regukar expressions.

| **Character** | **Explanation** | **Example** |
| --- | --- | --- |
| **.** | This metacharacter allows any character | matches any character |
| **+** | This character allows one or more preceding term | /ah+x/ matches ""ahx" or "ahhhhhhhx" |
| **\*** | This character allows zero or more preceding term | /ah\*x/ matches "aax", "ahx", or "ahhhhx" |
| **?** | This character allows zero or one preceding term | /ah?x/ matches "aax" and "ahx" but not "ahhx" |
| **\.** | This character allows a period(.) in the text | /etc\./ matches "etc." |
| **\/** | This character allows forward slash in the text. | /.+\/.+/ matches "home/my" |
| **\\*** | This character allows asterik in the text. | /my\\*name/ matches "my\*name" |
| **\+** | This character allows Plus sign in the text. | /.+ \+ .+/ matches "five + four" |
| **\?** | This character allows question mark in the text. | /.+\?/ matches "really?" |
| **^** | The string should begin with the specified word written after this character. | /^Bye/ matches "Bye" in "Bye Harry" |
| **$** | The string should end with the specified word written before this character. | /Harry$/ matches "Harry" in "and Harry" |

**Examples of Metacharacters:-**

Following are the some examples of how we can use metacharacters in JavaScript.

* Do a global search for "m.d" in a string:

let reg = /m.d/g;

let str = "He's mad!"; //matches

Copy

* Do a global search for at least one "e":

let reg = /o+/g;

let str = "Codeeeee!";//matches

Copy

* Do a global search for a "5", followed by zero or more "0" characters:

let reg = /50\*/g;

let str = "5, 500 or 5000?"; //matches

Copy

* Do a global search for "javaScript" at the end of a string:

let reg = /javaScript$/g;

let str = "Welcome to the tutorial of javaScript"; //matches

Copy

* Do a global search for "javaScript" at the beginning of a string:

let reg = /^javaScript/g;

let str = "javaScript supports OOP";//matches

Copy

* Do a global search for "code" followed by " harry":

let reg = /code(?= harry)/g;

let str = "code with harry"; //matches

Copy

* Do a global, case insensitive search for "code" not followed by "JavaScript":

let reg = /code(?! JavaScript/gi;

let str = "Code JavaScript"; //does not match

Copy

**Code website.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<!-- <br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div> -->

<div class="container">

<h1>Student list</h1>

<ul id="students"></ul>

<button id="myBtn" class="btn btn-primary">Your Button</button>

<button class="btn btn-primary">Fetch Data</button>

<div id="content"></div>

</div>

<div class="container my-3">

<h1>Pull the results by clicking the button below:</h1>

<button class="btn btn-primary" id="meanings">Get meanings</button>

<div class="my-2">

<ul id="defs"></ul>

</div>

</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<!-- <script src="js/tut23.js"></script> -->

<!-- <script src="js/tut24.js"></script> -->

<!-- <script src="js/tut25.js"></script> -->

<!-- <script src="js/tut27.js"></script> -->

<!-- <script src="js/tut28.js"></script> -->

<!-- <script src="js/tut30.js"></script> -->

<!-- <script src="js/tut31.js"></script> -->

<!-- <script src="js/tut32.js"></script> -->

<!-- <script src="js/tut34.js"></script> -->

<!-- <script src="js/tut37.js"></script> -->

<!-- <script src="js/tut39.js"></script> -->

<!-- <script src="js/tut39b.js"></script> -->

<!-- <script src="js/tut41.js"></script> -->

<!-- <script src="js/tut43.js"></script> -->

<!-- <script src="js/tut44.js"></script> -->

<!-- <script src="js/tut 45.js"></script> -->

<script src="js/tut46.js"></script>

</html>

Copy

**Code tut46.js as described/written in the video**

console.log('This is tutorial 47');

let regex = /harrsdfgy/;

// Lets look into some metacharacter symbols

regex = /^harrdc/; // ^ means expression will match if string starts with

regex = /harry$/; // $ at the end of the string means "string ends with"

regex = /h.rry/; //matches any one character

regex = /h\*rry/; //matches any 0 or more characters

regex = /ha?rryi?t/; //? after character means that character is optional

regex = /h\\*rry/

let str = "h\*rry means codewith"; //

let result = regex.exec(str);

console.log("The result from exec is ", result);

if(regex.test(str)){

console.log(`The string ${str} matches the expression ${regex.source}`);

}

else{

console.log(`The string ${str} does not match the expression ${regex.source}`);

}

Copy

**PreviousNext**

CodeWithHarry

Copyrigh

**Regular Expressions - Character sets | JavaScript Tutorial In Hindi #48**

A **regular expression** is a sequence of characters that define a *search pattern*. Usually, such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings or input validation. In the previous tutorial, we have studied the methods and metacharacters of the regular expression. If you have not watched the tutorial yet, then check [***tutorial#46***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-46) and [***tutorial#47***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-47). In today's tutorial, we will learn how to use regular expressions to work with sets of characters. Unlike the dot(.), which matches any single character, character sets enable us to match specific characters and character ranges.

Several characters or character classes inside square brackets […] mean "search for any character among given." Brackets and sets are used to find a range of characters.

**Sets:-**

For instance, [aeio] means any of the 4 characters: 'a', 'e' , 'i', or 'o'. That is called a *set*. Sets can be used in a regular expressions along with regular characters:

**Ranges:-**

Square brackets may also contain *character ranges*. For instance, [a-z] is a character in range from a to z, and [0-9] is a digit from 0 to 9.

**Excluding ranges:-**

Besides normal ranges, there are "excluding" ranges that look like [^…]. They are denoted by a caret character ^ at the start and match any character *except the given ones*.

**For Example:**

* [^aeiou] – any character except 'a', 'e', 'i' 'o' or'u'.
* [^0-5] – any character except a digit, the same as \D.
* [^\s] – any non-space character, same as \S.

The following are the some common character set example.

**[abcdef]:-**

This expression will find any character between the brackets. Here is an example:

Do a global search for the characters "D" and "o" in a string:

let reg = /[Do]/gi;

let str = "Do I Know You?";

Copy

**[^abcdef]:-**

This expression will find any character NOT between the brackets. Here is an example:

Do a global search for characters that are NOT inside the brackets [abc]:

let str = "Is this all there is?";

let reg = /[^abc]/g;

Copy

**[0-9]:-**

This expression will find any character between the brackets (any digit). Here is an example:

Do a global search for the numbers 1, 2, 3,4 and 5 in a string:

let str = "123456789";

let reg = /[1-5]/g;

Copy

**[^0-9]:-**

This expression will find any character NOT between the brackets (any non-digit).Here is an example:

Do a global search for the numbers that are NOT 1 to 5 in a string.

let str = "123456789";

let reg = /[^1-5]/g;

Copy

**(a|b):-**

This expression will find any of the alternatives specified. Here is an example:

Do a global search to find any of the specified alternatives (0|3|2):

let str = "01234567890123456789";

let reg = /(0|3|2)/g;

Copy

**Quantifier:-**

Quantifier matches a number of instances of a character, group, or character class in a string. It is appended to a character and specifies how many characters we need. Following is the table of quantifiers, along with its description.

| **Quantifier** | **Description** |
| --- | --- |
| \* | It matches zero or more times. |
| + | It matches one or more times. |
| ? | It matches zero or one time. |
| { x } | It matches exactly x times. |
| {x ,} | It matches at least x times. |
| { x, y } | It matches from x to y times. |

**Code website.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<!-- <br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div> -->

<div class="container">

<h1>Student list</h1>

<ul id="students"></ul>

<button id="myBtn" class="btn btn-primary">Your Button</button>

<button class="btn btn-primary">Fetch Data</button>

<div id="content"></div>

</div>

<div class="container my-3">

<h1>Pull the results by clicking the button below:</h1>

<button class="btn btn-primary" id="meanings">Get meanings</button>

<div class="my-2">

<ul id="defs"></ul>

</div>

</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<!-- <script src="js/tut23.js"></script> -->

<!-- <script src="js/tut24.js"></script> -->

<!-- <script src="js/tut25.js"></script> -->

<!-- <script src="js/tut27.js"></script> -->

<!-- <script src="js/tut28.js"></script> -->

<!-- <script src="js/tut30.js"></script> -->

<!-- <script src="js/tut31.js"></script> -->

<!-- <script src="js/tut32.js"></script> -->

<!-- <script src="js/tut34.js"></script> -->

<!-- <script src="js/tut37.js"></script> -->

<!-- <script src="js/tut39.js"></script> -->

<!-- <script src="js/tut39b.js"></script> -->

<!-- <script src="js/tut41.js"></script> -->

<!-- <script src="js/tut43.js"></script> -->

<!-- <script src="js/tut44.js"></script> -->

<!-- <script src="js/tut 45.js"></script> -->

<!-- <script src="js/tut46.js"></script> -->

<!-- <script src="js/tut47.js"></script> -->

<!-- <script src="js/tut48.js"></script> -->

<script src="js/tut49.js"></script>

</html>

Copy

**Code as described/written in the video**

console.log("This is tutorial 48");

// Regular Expressions:

// Basic functions

// Metacharacter Symbols

// const regex = /^h/i;

// Character Sets - We use []

let regex = /h[a-z]rry/; // can be any character from a to z

regex = /h[aty]rry/; // can be an a, t or y

regex = /h[^aty]rry/; // cannot be any of a, t or y

regex = /h[^aty]rr[yYu]/; // cannot be any of a, t or y + can be a u or y

regex = /h[a-zA-Z]rr[yu0-9][0-9]/; // we can have as many character sets as we want

// Quantifiers - We use {}

regex = /har{2}y/; // r can occur exactly 2 times

regex = /har{0,2}y/; // r can occur exactly 0 to 2 (0 or 1 or 2) times

// Groupings - We use paranthesis for groupings ()

regex = /(har){2}([0-9]r){3}/

// const str = "hirry9 bhai";

str = "harrry bhai"

str = "harhar1r4r5r bhai";

let result = regex.exec(str);

console.log("The result from exec is ", result);

if(regex.test(str)){

console.log(`The string ${str} matches the expression ${regex.source}`);

}

else{

console.log(`The string ${str} does not match the expression ${regex.source}`);

}

**Shorthand character classes (Regular Expressions) | JavaScript Tutorial In Hindi #49**

As we know, the character ranges are very useful while writing a regular expression. However, they can be hectic to write out every time we want to match common ranges, such as those that designate alphabetical characters or digits. To get rid of this pain, ***shorthand character classes*** represent common ranges, and they make writing regular expressions much simpler. As we have seen from the previous tutorial examples about character set of regular expressions, certain character classes, such as digits [0-9] or characters [0-9A-Za-z\_], are used in most regex patterns.

**These shorthand classes include:**

* **\w:**  This is the **"word character"** class that represents the regex range [A-Za-z0-9\_], and it will match a *single uppercase character, lower-case character, digit, or underscore*
* **\d:** This is the **"digit character"**class represents the regex range [0-9], and it will match the single-digit character
* **\s:** This is the **"whitespace character"**class that represents the regex range, matching a single space, carriage return, tab, line break, form feed, or vertical tab

Along with shorthand character classes( \w, \d, and \s), we can also use the *negated shorthand character classes*. These negated shorthands will match any character that is NOT in the regular shorthand classes.

**These negated shorthand classes include:**

* **\W:** the "non-word character" class represents the regex range [^A-Za-z0-9\_], matching any character that is not included in the range represented by \w
* **\D**: the "non-digit character" class represents the regex range [^0-9], matching any character that is not included in the range represented by \d
* **\S**: the “non-whitespace character” class represents the regex range [^ \t\r\n\f\v], matching any character that is not included in the range represented by \s

Here is the table of shorthand character classes with examples. This table contains all the information explained above.

**Character Classes (Shorthand)**

| **Character** | **DESCRIPTION** | **Example** |
| --- | --- | --- |
| \d | This character means any digit character; functionally equivalent to [0-9] or [[:digit:]] | \d matches 1,56,77 12, 123, etc., but not 1b7 or aabb6. One or more of any digit characters. |
| \D | This character means any non-digit character; functionally equivalent to [^0-9] or [^[:digit:]] | \D matches a, ab, abcd, ab&, but not 1. |
| \w | This character means any "word" character. That is, any alphanumeric character and its functionally equivalent to [\_A-Za-z0-9] or [\_[:alnum:]] | \w matches a, ab, a1,abc123 but not !&. One or more upper- or lower-case letters or digits, but not punctuation or other special symbols/characters. |
| \W | This character means any non-alphanumeric character; functionally equivalent to [^\_A-Za-z0-9] or [^\_[:alnum:]] | \W matches \*, &, but not race or y1. One or more of any character but upper- or lower-case letters and digits. |
| \s | This character means any white space character; space, new line, tab, non-breaking space, etc.; functionally equivalent to [[:space]] | vegetable\s matches "vegetable" followed by any non-white space character. |
| \S | This character means any non-whitespace character; anything other than space, newline, tab, non-breaking space, etc.; functionally equivalent to [^[: space]] | vegetable\S matches "vegetable" followed by any non-whitespace character. |

**Assertions:-**

An Assertion is a regular expression that will succeed if a match is found and fails if a match is not found. Assertion consists of Anchors and Lookarounds.

* **^:**The symbol "^" matches at the beginning of the string.
* **$:**The symbol "$" matches only at the end of the string.
* **\b:**The character "\b" matches only at a word boundary.
* **\B:**the character "\B" Matches only if not at a word boundary.
* **(?=«pattern»):**This is a positive lookahead. It matches the regular expression with the pattern only if the pattern matches what comes next. The pattern is used only to look ahead but otherwise ignored.
* **(?!«pattern»):**This is the negative lookahead: It matches the regular expression with the pattern only if the pattern does not match what comes next. The pattern is used only to look ahead but otherwise ignored.

**Code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<!-- <br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div> -->

<div class="container">

<h1>Student list</h1>

<ul id="students"></ul>

<button id="myBtn" class="btn btn-primary">Your Button</button>

<button class="btn btn-primary">Fetch Data</button>

<div id="content"></div>

</div>

<div class="container my-3">

<h1>Pull the results by clicking the button below:</h1>

<button class="btn btn-primary" id="meanings">Get meanings</button>

<div class="my-2">

<ul id="defs"></ul>

</div>

</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<!-- <script src="js/tut23.js"></script> -->

<!-- <script src="js/tut24.js"></script> -->

<!-- <script src="js/tut25.js"></script> -->

<!-- <script src="js/tut27.js"></script> -->

<!-- <script src="js/tut28.js"></script> -->

<!-- <script src="js/tut30.js"></script> -->

<!-- <script src="js/tut31.js"></script> -->

<!-- <script src="js/tut32.js"></script> -->

<!-- <script src="js/tut34.js"></script> -->

<!-- <script src="js/tut37.js"></script> -->

<!-- <script src="js/tut39.js"></script> -->

<!-- <script src="js/tut39b.js"></script> -->

<!-- <script src="js/tut41.js"></script> -->

<!-- <script src="js/tut43.js"></script> -->

<!-- <script src="js/tut44.js"></script> -->

<!-- <script src="js/tut 45.js"></script> -->

<!-- <script src="js/tut46.js"></script> -->

<!-- <script src="js/tut47.js"></script> -->

<!-- <script src="js/tut48.js"></script> -->

<script src="js/tut49.js"></script>

</html>

Copy

**Code as described/written in the video**

console.log("This is tutorial 49");

// Character classes

let regex = /\war/; //word character - \_ or alphabet or numbers

regex = /\w+d1r/; // \w+ means one or more word characters

regex = /\Wbhai/; // Non word character

regex = /\W+bhai/; // \W+ means more than one Non word character

regex = /number \d999/; // \d means digit

regex = /number \d+/; // \d+ means more than one digit

regex = /\D999/; // \D means non digit

regex = /\D+999/; // \D+ means more than one non digit

regex = /\ska number/; // Match whitespace character

regex = /\s+ka number/; // \s+ means match one or more than one whitespace characters

regex = /\Ska number/; // Match non whitespace character

regex = /\S+ka number/; // Match one or more than one non whitespace character

regex = /4r5r\b/; // word boundary

// Assertions

regex = /h(?=y)/;

regex = /h(?!y)/;

str = "harh7rd1r4r5ry%%$@bhai hdrryika number 899999harry9999";

let result = regex.exec(str);

console.log("The result from exec is ", result);

if(regex.test(str)){

console.log(`The string ${str} matches the expression ${regex.source}`);

}

else{

console.log(`The string ${str} does not match the expression ${regex.source}`);

}

Copy

**PreviousNext**

CodeWithHarry

Copyright © 202

# Project 4: Form Validation using Regular Expressions | JavaScript Tutorial In Hindi #50

Until now, we have three projects. Now it’s time for the 4th project, which no doubt, is one of the most important project. The project we are going to build is “Form Validation using Regular Expressions”. Validation of form input is something that should be taken seriously because nothing worse than garbage data will be submitted to a site which uses data from forms without proper validation.

The website we are going to build in this tutorial looks like the following:

### https://api.codewithharry.com/media/videoSeriesFiles/courseFiles/javascript-tutorials-in-hindi-50/base64.png

#### Setting Up the Project:-

For this project, we will use the VS Code. **Visual Studio Code** is a free and powerful source-code editor that runs on the desktop. If you have not installed the Visual Studio Code yet, then click on the link below for installation guidance:

***<https://codewithharry.com/videos/javascript-tutorials-in-hindi-1>***

***Here are the project setup steps:***

1. Create a new project folder in VS Code.
2. Inside that project folder, create two empty new files: project4.html and project4.js.

We are also using **Bootstrap**. If you are not familiar with Bootstrap, then do not worry; just follow the tutorial.

#### Bootstrap:-

Bootstrap is used for applying styling the user interface components. The easiest way to include Bootstrap is to add it from https://getbootstrap.com/docs/4.0/getting-started/introduction/.

 First add the following in the head section:

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

Copy

Then include the following code in the body section, before the closing </body> tag:

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

Copy

**Bootstrap** is bundled with many components that can be used to provide good user experience and user interactions on a web page. In this project, we are using bootstrap components for making the front end of the website. So, let’s start coding!!!

### 

#### Code as described/written in the video

<!doctype html>

<html lang="en">

<head>

<!-- Required meta tags -->

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<!-- Bootstrap CSS -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title> Welcome to Dragon travels</title>

</head>

<body>

<div id="success" class="alert alert-success alert-dismissible fade" role="alert">

<strong>Success!</strong> Your travel request has been successfully submitted

<button type="button" class="close" data-dismiss="alert" aria-label="Close">

<span aria-hidden="true">×</span>

</button>

</div>

<div id="failure" class="alert alert-danger alert-dismissible fade" role="alert">

<strong>Error!</strong> Your travel request has not been sent due to errors

<button type="button" class="close" data-dismiss="alert" aria-label="Close">

<span aria-hidden="true">×</span>

</button>

</div>

<div class="container">

<h1>Dragon Travel desk</h1>

<form>

<div class="form-group">

<label for="name">Username</label>

<input type="text" class="form-control" id="name" placeholder="Enter your username">

<small id="namevalid" class="form-text text-muted invalid-feedback">

Your username must be 2-10 characters long and should not start with a number

</small>

</div>

<div class="form-group">

<label for="email">Email address</label>

<input type="text" class="form-control" id="email" placeholder="Enter your email">

<small id="emailvalid" class="form-text text-muted invalid-feedback">

Your email must be a valid email

</small>

</div>

<div class="form-group">

<label for="car">Select your car</label>

<select class="form-control" id="car">

<option>Omni</option>

<option>Maruti 800</option>

<option>Ford Titanium</option>

<option>Audi A4</option>

</select>

</div>

<div class="form-group">

<label for="address">Enter your address</label>

<textarea class="form-control" name="address" id="address" rows="3"></textarea>

</div>

<div class="form-group">

<label for="address">Enter your phone number</label>

<input class="form-control" name="phone" id="phone" rows="3">

<small id="emailvalid" class="form-text text-muted invalid-feedback">

Your phone number must be 10 digit long

</small>

</div>

<div class="form-group">

<label for="exampleFormControlTextarea1">Any other message...</label>

<textarea class="form-control" id="exampleFormControlTextarea1" rows="3"></textarea>

</div>

<button id="submit" class="btn btn-primary">Submit</button>

</form>

</div>

<!-- Optional JavaScript -->

<!-- jQuery first, then Popper.js, then Bootstrap JS -->

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

<script src="js/project4.js"></script>

</body>

</html>

Copy

#### Code as described/written in the video

console.log("This is project 4");

const name = document.getElementById('name');

const email = document.getElementById('email');

const phone = document.getElementById('phone');

let validEmail = false;

let validPhone = false;

let validUser = false;

$('#failure').hide();

$('#success').hide();

// console.log(name, email, phone);

name.addEventListener('blur', ()=>{

console.log("name is blurred");

// Validate name here

let regex = /^[a-zA-Z]([0-9a-zA-Z]){2,10}$/;

let str = name.value;

console.log(regex, str);

if(regex.test(str)){

console.log('Your name is valid');

name.classList.remove('is-invalid');

validUser = true;

}

else{

console.log('Your name is not valid');

name.classList.add('is-invalid');

validUser = false;

}

})

email.addEventListener('blur', ()=>{

console.log("email is blurred");

// Validate email here

let regex = /^([\_\-\.0-9a-zA-Z]+)@([\_\-\.0-9a-zA-Z]+)\.([a-zA-Z]){2,7}$/;

let str = email.value;

console.log(regex, str);

if(regex.test(str)){

console.log('Your email is valid');

email.classList.remove('is-invalid');

validEmail = true;

}

else{

console.log('Your email is not valid');

email.classList.add('is-invalid');

validEmail = false;

}

})

phone.addEventListener('blur', ()=>{

console.log("phone is blurred");

// Validate phone here

let regex = /^([0-9]){10}$/;

let str = phone.value;

console.log(regex, str);

if(regex.test(str)){

console.log('Your phone is valid');

phone.classList.remove('is-invalid');

validPhone = true;

}

else{

console.log('Your phone is not valid');

phone.classList.add('is-invalid');

validPhone = false;

}

})

let submit = document.getElementById('submit');

submit.addEventListener('click', (e)=>{

e.preventDefault();

console.log('You clicked on submit');

console.log(validEmail, validUser, validPhone);

// Submit your form here

if(validEmail && validUser && validPhone){

let failure = document.getElementById('failure');

console.log('Phone, email and user are valid. Submitting the form');

let success = document.getElementById('success');

success.classList.add('show');

// failure.classList.remove('show');

// $('#failure').alert('close');

$('#failure').hide();

$('#success').show();

}

else{

console.log('One of Phone, email or user are not valid. Hence not submitting the form. Please correct the errors and try again');

let failure = document.getElementById('failure');

failure.classList.add('show');

// success.classList.remove('show');

// $('#success').alert('hide');

$('#success').hide();

$('#failure').show();

}

})

Copy

**PreviousNext**

CodeWithHarry

**Iterators in JavaScript | JavaScript Tutorial In Hindi #51**

We are going to explore the concept of **“Iterators”** in this tutorial. As its name implies, iterators are a new way to *loop*over any collection in JavaScript. Today we will conceptually understand iterators along with an example. Iterators were introduced in ES6. It is one of the popular JavaScript concepts since they are widely useful and used in various places. So, let us start with the theory.

**What is an iterator?**

It is an object that allows us to traverse over a list or collection. Iterators' purpose is to define the sequences and implement the iterator protocol that returns an object by using a next() method that contains the **value** and **done**.

* **done:** It is a boolean value indicating whether any more elements in the sequence could be iterated upon.
* **value:** It is the current element of the sequence.

*So, we can define iterators as an****“object that knows how to access items from a collection one at a time, while keeping track of its current position within that sequence.”***

Suppose we have an array, and it contains five numbers, i.e., [1,2,3,4,5]. As we know, the Iterator object has a next() method that returns the next item in the sequence. So, when we write next(), well get the element of the array. The next() method returns an object with two properties: value and done. If there are elements present in the sequence that could be iterated, then the value contains the next element and done is set to false:

**{ value: 'next value', done: false }**

If we call the next() method after the last value has been returned, then the next() returns the result object as follows:

**{done: true: value: undefined}**

Here the value of the done property, which is true, indicates that there is no more value to return, and the value of the property is set to undefined.

**Here is a simple example of an iterator:**

function myIterable() {

let counter = 0;

return {

next:function(){

if (counter < 5) {

counter++;

return { done: false, value: counter };

} else {

return { done: true, value: undefined };

}

}

}

}

Copy

**Code Explanation:-**

The above code executes five steps, with the counter incrementing (counter++) every run. First, we return the value 1, then the value 2, and so on till we get the last element 5then we indicate that the end of the iteration has been reached, and the value becomes equal to undefined.

**Summary:-**

In this tutorial, we learned about what iterator is and how we can use it. In JavaScript, an **iterator** is an object which defines a sequence and a return value upon the end of the sequence. An iterator implements the Iterator protocol by having a next() method that returns an object with two properties. As an iterator moves over the data structure and provides the elements sequentially, the object returned by the iterator contains a value and a done property.

**Code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<!-- <br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div> -->

<div class="container">

<h1>Student list</h1>

<ul id="students"></ul>

<button id="myBtn" class="btn btn-primary">Your Button</button>

<button class="btn btn-primary">Fetch Data</button>

<div id="content"></div>

</div>

<div class="container my-3">

<h1>Pull the results by clicking the button below:</h1>

<button class="btn btn-primary" id="meanings">Get meanings</button>

<div class="my-2">

<ul id="defs"></ul>

</div>

</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<!-- <script src="js/tut23.js"></script> -->

<!-- <script src="js/tut24.js"></script> -->

<!-- <script src="js/tut25.js"></script> -->

<!-- <script src="js/tut27.js"></script> -->

<!-- <script src="js/tut28.js"></script> -->

<!-- <script src="js/tut30.js"></script> -->

<!-- <script src="js/tut31.js"></script> -->

<!-- <script src="js/tut32.js"></script> -->

<!-- <script src="js/tut34.js"></script> -->

<!-- <script src="js/tut37.js"></script> -->

<!-- <script src="js/tut39.js"></script> -->

<!-- <script src="js/tut39b.js"></script> -->

<!-- <script src="js/tut41.js"></script> -->

<!-- <script src="js/tut43.js"></script> -->

<!-- <script src="js/tut44.js"></script> -->

<!-- <script src="js/tut 45.js"></script> -->

<!-- <script src="js/tut46.js"></script> -->

<!-- <script src="js/tut47.js"></script> -->

<!-- <script src="js/tut48.js"></script> -->

<!-- <script src="js/tut49.js"></script> -->

<!-- <script src="js/tut51.js"></script> -->

<script src="js/tut52.js"></script>

</html>

Copy

**Code as described/written in the video**

console.log('The file is tutorial 51');

// Iterators

function fruitsIterator(values) {

let nextIndex = 0;

// we will return an object

return {

next: function () {

if (nextIndex < values.length) {

// We will return below object

return {

value: values[nextIndex++],

done: false

}

}

else {

// We will return below object with only done

return {

done: true

}

}

}

}

}

const myArray = ['Apples', 'Grapes', 'Oranges', 'Bhindi'];

console.log("My array is ", myArray)

// Using the iterator

const fruits = fruitsIterator(myArray);

console.log(fruits.next().value)

console.log(fruits.next().value)

console.log(fruits.next().value)

console.log(fruits.next().value)

console.log(fruits.next().value)

Copy

**PreviousNext**

CodeWithHarry

Cop

**Creating an Alarm App: Exercise 6 | JavaScript Tutorial In Hindi #52**

It’s time for 6th exercise. Following is the problem statement:

**Problem Statement:-**

The task you have to do is to create an **alarm clock in JavaScript**(Use your creativity). The Alarm app should allow the user to set the alarm for a certain time.

To add the audio, search “how to play audio in js” on google, most probably you will find the code from stack overflow. Or you can use the following code:

var audio = new Audio('https://interactive-examples.mdn.mozilla.net/media/examples/t-rex-roar.mp3');

audio.play();

Copy

I strongly encourage you to solve this exercise, as this is a great way to become a pro in programming. The more you code, the more you learn. Have you solved this exercise, if yes, then check ***[tutorial#54](https://codewithharry.com/videos/javascript-tutorials-in-hindi-54)*** for the solution. This exercise is a part of the javaScript series for beginners, to access the playlist, check on the link below:

<https://codewithharry.com/videos/javascript-tutorials-in-hindi-1>

If you like my work, then keep following and stay up to date with ***[codewithharry](https://codewithharry.com/)***.

**Code as described/written in the video**

console.log('This is tutorial 52 file');

// You have to crate an alarm clock in javascript (Use your creativity)

// Allow user to set alarm for a certain time

var audio = new Audio('https://interactive-examples.mdn.mozilla.net/media/examples/t-rex-roar.mp3');

audio.play();

Copy

**PreviousNext**

CodeWithHarry

**Generators in Javascript | JavaScript Tutorial In Hindi #53**

In today's tutorial, we will study the **generator in JavaScript**, along with examples. A generator is a special kind of function that was introduced in ES6. In JavaScript, once we execute a function, it has to be executed entirely. But, generator functions enable us to create functions that another code can enter multiple times. Nothing from outside of the generator function can make it pause. Generator function pauses itself when it runs into a yield expression. Once the execution reaches the yield expression, the generator cannot continue execution on its own. Something from *outside* has to *continue*its execution.

Another important difference between generators and normal functions is that generator functions can produce multiple values during its execution. Hence, they can generate a sequence of values, not all at once, but on a per request basis. At every request, the generator function gives us a value until it reaches the end of its execution. Once that happens, *the done*flag will be set to *true*. Here is the syntax of defining the generator function:

**Syntax:-**

We already know how to declare a normal function. So, the syntax of declaring the generator function is quite similar to traditional functions. We declare a generator function by using the**\*** ( asterisk ) operator after the *function* keyword:

function\* myGenerator(){

//code

}

Copy

**Yield:-**

The **yield** keyword pauses the **generator** function execution, and the value of the expression following the **yield** keyword is returned to the **generator's** caller. It acts as a **generator**-based version of the return keyword. In the following example, to pause the generator's execution, and we use the statement *yield*.:

function\* awesomeGenerator(){

yield 'Hello World' // We pause the execution here

console.log('We are back again') // When we resume, we are here

}

Copy

**next() method:-**

A generator gives us the *next()* method, which is used to resume the execution. This method returns an object with two properties. These are value and done:

{

value: [ next value ],

done: [ true if we reach the end, else false]

}

Copy

**Here is an example:**

function\* myGenerator() {

yield 1;

yield 2;

}

let iterator = myGenerator();

let result;

do {

result = iterator.next();

console.log(result);

} while (!result.done);

//Output:-

//{value: 1, done: false}

//{value: 2, done: false}

//{value: undefined, done: true}

Copy

**Code Explanation:-**

As we can see from the above code, when we first call the generator function, it returns us *iterator*object. When we call the next() for the first time, it starts executing the generator function, and it *yields* the first value: 1. Furthermore, calling it for the second time gives us the second and last value 2. Finally, the third call returns no value, and the *done* is set to true, which means that we have finished iterating through the generator.

**Code as described/written in the video**

console.log("this is tutorial 53");

// Generators in JavaScript

// 1 - 1B

function\* numbersGen(){

let i = 0;

// yield 1;

// yield 2;

// yield 3;

// yield 4;

while(true){

yield i++;

// yield (i++).toString();

}

}

const gen = numbersGen();

console.log(gen.next().value);

console.log(gen.next().value);

Copy

**PreviousNext**

CodeWithHarry

Co

**Alarm Clock In JavaScript Exercise 6: Solution | JavaScript Tutorial In Hindi #54**

This tutorial contains the solution to Exercise 6, i.e. ***Creating an Alarm App***. The problem statement, along with the instructions, is given below:

**Problem Statement:-**

The task you have to do is to create an alarm clock in JavaScript (Use your creativity). The Alarm app should allow the user to set the alarm for a certain time.

To add the audio, search “how to play audio in js” on google, most probably you will find the code from stack overflow. Or you can use the following code:

var audio = new Audio('https://interactive-examples.mdn.mozilla.net/media/examples/t-rex-roar.mp3');

audio.play();

Copy

**Code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title>Document</title>

</head>

<body>

<div class="container">

<h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>

<div id="myfirst" class="child red good" id="first">child 1

<ul class="this" id='myul'>

<li class="childul" id='fui'>this</li>

<li class="childul">is</li>

<li class="childul">a</li>

<li class="childul">list </li>

<li class="childul" id='lui'>of my dreams</li>

</div>

<div class="child">child 2</div>

<div class="child red">child 3</div>

<div class="child">child 4</div>

<form action="none.html" method="post">

<a href="//codewithharry.com">Go to Code With Harry</a>

<br>

<br>

Search this website: <input type="text" name="Hello" id="">

<button id="btn">Submit form</button>

<!-- <input type="button" id='btn' value="submit"> -->

</form>

</div>

<!-- <br>

<div class="no">this is a dummy div1</div>

<div class="no">this is a dummy div2</div>

<div class="no">this is a dummy div3</div> -->

<div class="container">

<h1>Student list</h1>

<ul id="students"></ul>

<button id="myBtn" class="btn btn-primary">Your Button</button>

<button class="btn btn-primary">Fetch Data</button>

<div id="content"></div>

</div>

<div class="container my-3">

<h1>Option to set an alarm</h1>

<form>

<div class="form-group">

<label for="alarm">Set Alarm</label>

<input type="text" class="form-control" id="alarm" name="alarm"

placeholder="Enter the time in yyyy-mm-dd hh:mm:ss">

<small id="alarmHelp" class="form-text text-muted">We'll never share your alarm with anyone else.</small>

</div>

<button id="alarmSubmit" type="submit" class="btn btn-primary">Set Alarm</button>

</form>

<div class="my-2">

<ul id="defs"></ul>

</div>

</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

</body>

<!-- <script src="js/tut12.js"></script> -->

<!-- <script src="js/tut14.js"></script> -->

<!-- <script src="js/tut15.js"></script> -->

<!-- <script src="js/tut16.js"></script> -->

<!-- <script src="js/tut17.js"></script> -->

<!-- <script src="js/tut18.js"></script> -->

<!-- <script src="js/tut20.js"></script> -->

<!-- <script src="js/tut21.js"></script> -->

<!-- <script src="js/tut23.js"></script> -->

<!-- <script src="js/tut24.js"></script> -->

<!-- <script src="js/tut25.js"></script> -->

<!-- <script src="js/tut27.js"></script> -->

<!-- <script src="js/tut28.js"></script> -->

<!-- <script src="js/tut30.js"></script> -->

<!-- <script src="js/tut31.js"></script> -->

<!-- <script src="js/tut32.js"></script> -->

<!-- <script src="js/tut34.js"></script> -->

<!-- <script src="js/tut37.js"></script> -->

<!-- <script src="js/tut39.js"></script> -->

<!-- <script src="js/tut39b.js"></script> -->

<!-- <script src="js/tut41.js"></script> -->

<!-- <script src="js/tut43.js"></script> -->

<!-- <script src="js/tut44.js"></script> -->

<!-- <script src="js/tut 45.js"></script> -->

<!-- <script src="js/tut46.js"></script> -->

<!-- <script src="js/tut47.js"></script> -->

<!-- <script src="js/tut48.js"></script> -->

<!-- <script src="js/tut49.js"></script> -->

<!-- <script src="js/tut51.js"></script> -->

<!-- <script src="js/tut52.js"></script> -->

<!-- <script src="js/tut53.js"></script> -->

<script src="js/tut54.js"></script>

</html>

Copy

**JavaScript Code as described/written in the video**

console.log("This is tutorial 54");

const alarmSubmit = document.getElementById('alarmSubmit');

// Add an event listener to the submit button

alarmSubmit.addEventListener('click', setAlarm);

var audio = new Audio('https://interactive-examples.mdn.mozilla.net/media/examples/t-rex-roar.mp3');

// function to play the alarm ring tone

function ringBell() {

audio.play();

}

// This function will run whenever alarm is set from the UI

function setAlarm(e) {

e.preventDefault();

const alarm = document.getElementById('alarm');

alarmDate = new Date(alarm.value);

console.log(`Setting Alarm for ${alarmDate}...`);

now = new Date();

let timeToAlarm = alarmDate - now;

console.log(timeToAlarm);

if(timeToAlarm>=0){

setTimeout(() => {

console.log("Ringing now")

ringBell();

}, timeToAlarm);

}

}

Copy

**PreviousNext**

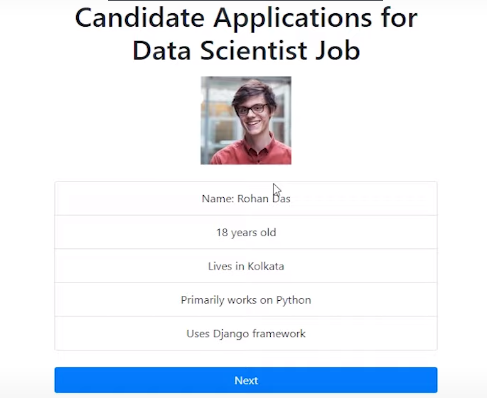
CodeWithHarry

Copy

**Project 5: Creating a CV Screener In JavaScript | JavaScript Tutorial In Hindi #55**

Let us start our 4th project, in which we will build the**“CV Screener”**in JavaScript. CV screening is the most common type of assessment. CV screening is the process of shortlisting candidates from their CVs based on their education, skills. In this project, we will generate random data, and to get the images of random people, we will use random user API. Here is the link: [***https://randomuser.me/***](https://randomuser.me/)

The website we are going to build in this tutorial looks like the following:



**Setting Up the Project:-**

For this project, we will use the VS Code. **Visual Studio Code** is a free and powerful source-code editor that runs on the desktop. If you have not installed the Visual Studio Code yet, then click on the link below for installation guidance:

***<https://codewithharry.com/videos/javascript-tutorials-in-hindi-1>***

1. Create a new project folder in VS Code.
2. Inside that project folder, create two empty new files: *index.html* and *index.js*.

The *index.html file* contains the HTML code. In this project, we are also using **Bootstrap**.

**Bootstrap:-**

Bootstrap is used for applying styling the user interface components. The easiest way to include Bootstrap is to add it from https://getbootstrap.com/docs/4.0/getting-started/introduction/.

 First add the following in the head section:

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

Copy

Then include the following code in the body section, before the closing </body> tag:

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

Copy

**Bootstrap** also provides us many components that can be used to provide good user experience and user interactions in a web page like cards, navigation bars, dropdowns, icons, buttons, forms and also sizing options for different DOM elements.

Now we are done with introduction, let’s start coding.

**Code as described/written in the video**

<!doctype html>

<html lang="en">

<head>

<!-- Required meta tags -->

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<!-- Bootstrap CSS -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title>Welcome to CV Screener</title>

</head>

<body>

<div class="container">

<div class="row">

<div class="col-md-6 mx-auto text-center">

<h1 class="my-3">Candidate Applications for Data Scientist Job</h1>

<div id="image"></div>

<br>

<div id="profile"></div>

<br>

<button class="btn btn-primary btn-block" id="next"> Next</button>

</div>

</div>

</div>

<!-- Optional JavaScript -->

<!-- jQuery first, then Popper.js, then Bootstrap JS -->

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>

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

</body>

</html>

Copy

**Code as described/written in the video**

console.log("This is my index.js");

// Data is an array of objects which contains information about the candidates

const data = [

{

name: 'Rohan Das',

age: 18,

city: 'Kolkata',

language: 'Python',

framework: 'Django',

image: 'https://randomuser.me/api/portraits/men/51.jpg'

},

{

name: 'Shubham Sharma',

age: 28,

city: 'Bangalore',

language: 'JavaScript',

framework: 'Angular',

image: 'https://randomuser.me/api/portraits/men/54.jpg'

},

{

name: 'Camella Cabello',

age: 18,

city: 'Kolkata',

language: 'Python',

framework: 'Django',

image: 'https://randomuser.me/api/portraits/women/55.jpg'

},

{

name: 'Aishwariya Rai',

age: 45,

city: 'Mumbai',

language: 'Python',

framework: 'Flask',

image: 'https://randomuser.me/api/portraits/women/57.jpg'

},

{

name: 'Rohit Sharma',

age: 34,

city: 'Jharkhand',

language: 'Go',

framework: 'Go Framework',

image: 'https://randomuser.me/api/portraits/men/61.jpg'

}

]

// CV Iterator

function cvIterator(profiles){

let nextIndex=0;

return {

next: function(){

return nextIndex<profiles.length ?

{value: profiles[nextIndex++], done: false} :

{done: true}

}

};

}

const candidates = cvIterator(data);

nextCV();

// Button listener for next button

const next = document.getElementById('next');

next.addEventListener('click', nextCV);

function nextCV(){

const currentCandidate = candidates.next().value;

let image = document.getElementById('image');

let profile = document.getElementById('profile');

if(currentCandidate != undefined){

image.innerHTML = `<img src='${currentCandidate.image}'>`;

profile.innerHTML = `<ul class="list-group">

<li class="list-group-item">Name: ${currentCandidate.name}</li>

<li class="list-group-item">${currentCandidate.age} years old</li>

<li class="list-group-item">Lives in ${currentCandidate.city}</li>

<li class="list-group-item">Primarily works on ${currentCandidate.language}</li>

<li class="list-group-item">Uses ${currentCandidate.framework} framework</li>

</ul>`;

}

else{

alert('End of candidate applications');

window.location.reload();

}

}

Copy

**PreviousNext**

# https://codewithharry.com/img/logo-blue.pngCodeWithHarryFor.. of Loop vs For.. in Loop (When to use which?) | JavaScript Tutorial In Hindi #56

As we have studied loops in the previous lecture, we know that loops are used in programming to automate repetitive tasks. JavaScript works in very different ways. It provides us with several ways of doing something. There are several loop statements in JavaScript like while, do-while, forEach. However, in most cases, everyone tends to use for statements because we usually deal with looping scenarios which consist of some conditions with counters. In addition to the **for** loop, there are two other types of iteration methods of for, which are: **for..in** and **for..of**. The main topic of today's tutorial is for..in and for..of loops. Before getting in the detail of these loops, let us revise the for loop.

**for loop:-**

It is the most basic loop in every programming language. The for loop syntax takes three parameters; a variable declaration, an expression that will evaluate before each iteration, and the expression to be evaluated at the end of each iteration. Here is the syntax of for loop:

for (initialization; condition; final expression) {

// code to be executed

}

Copy

For example, the for loop will console.log each item in an array.

const myarray = ['abcd', 'efgh', 'i', 'eeb12'];

for (let i = 0; i < myarray.length; i++) {

console.log(array[i]);

} // Result: abcd,efgh,I,eeb12

Copy

**The for..in loop:-**

**The for..in loop** is an iteration method for iterating over "enumerable" properties of an object. This loop applies to all objects that have these properties. Here enumerable means an array or object or strings. If we are using a for…in loop over an object, it will give us the value to each key in the object. Here is the syntax of the **for..in loop:**

for (variable in enumerable) {

// do stuff

}

Copy

Here is an example to loop through and console.log all the values in the object,

const obj = {

first: 134,

second: 255,

third: 367,

fourth: 433

}

for (const key in obj) {

console.log( obj[key] )

} // Result: 134, 255, 367, 433

Copy

For…in loops do not have any specific order for execution. If we want the guarantee order on an object. Then specify the index in the objects or put the keys in an array. As we know, a character in a string has an index. Therefore, similar to Arrays, the indexes are enumerable properties that happens to be integers. Here is an example of a for..in loop with strings:

const str = 'Lets Code';

for (let index in str) {

console.log(str[index])

} //Result: L, e, t, s C, o, d, e

Copy

**For…Of Loop:-**

To iterate over objects like arrays and strings, we can use the for...of statement. This statement is a newer feature of ECMAScript 6. The **for..of** loop does not work with objects because they are not "iterable". This iteration method is a more reliable way of looping through an Array in sequence. In this example of a for...of loop, we will print each item in the array to the console.

let students = [ "Mark", "Harry", "Joe" ];

// Print out each type of shark

for (let std of students) {

console.log(std);

} //Result: Mark, Harry, Joe

Copy

**Comparison:-**

In the following table, we are comparing for..in loop and for..of loop.

|  | **for..in** | **for..of** |
| --- | --- | --- |
| Applies to | Enumerable Properties | Iterable Collections |
| Objects | Yes | No |
| Arrays | Yes, but not advised | Yes |
| Strings | Yes, but not advised | Yes |

**Which loop should we use? and When?**

Each type of loop is useful in a different scenario:

* If indexes are needed while accessing an array of indexes related logical stuff are there, the for loop is a good choice.
* If there is a need to access keys/properties regardless of the sequence, use *a for-in*loop.
* If there is a need to iterate through items of an iterable, then the *for-of*loop is the right choice.

**Code as described/written in the video**

console.log("This is tutorial 56 on for-in and for-of loops");

let people = ["Harry", "Rohan", "SkillF", "Shubham", "Vikrant"];

// \*\*\*\*\*\*\*\*\*\*For in loop\*\*\*\*\*\*\*

// Traditional for loop:

// for (let index = 0; index < people.length; index++) {

// const element = people[index];

// console.log(element);

// }

// 1. ITERATING AN OBJECT

let obj = {

name: "Harry",

language: "JavaScript",

hobbies: "Android app development"

}

// console.log(obj);

// 1.1 Iterating an object using Traditional for loop:

// for (let index = 0; index < Object.keys(obj).length; index++) {

// const element = obj[Object.keys(obj)[index]];

// console.log(element);

// }

// 1.2 Iterating an object using for-in loop:

for (let key in obj){

console.log(obj[key]);

}

// 2. ITERATING A STRING

// We can use for in with strings to loop through all the characters

myString = "This is my string";

for (let char in myString){

console.log(myString[char]);

}

// Quiz: Use traditional for loop to iterate through the same string

// \*\*\*\*\*\*\*\*\*\*For of loop\*\*\*\*\*\*\*\*\*\*\*

console.log("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*For of loop starts here\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

people = ["Harry", "Rohan", "SkillF", "Shubham", "Vikrant"];

for(let name of people){

console.log(name);

}

for(let name of myString){

console.log(name);

}

Copy

**PreviousNext**

CodeWithHar

**JavaScript Symbols | JavaScript Tutorial In Hindi #59**

In today’s tutorial, we will study about the **symbols** which are the newest JavaScript primitive. It brings a few benefits to the JavaScript language and is very useful when used as object properties. Before symbols were introduced in ES6, JavaScript used seven types of data, which is grouped into two categories. Primitives, including the string, number, boolean, null, and undefined data types and another is the Objects, which includes arrays, functions, and regular JS objects.

**What is the symbol?**

A symbol is a primitive which cannot be recreated. It was introduced in ECMAScript 2015. It is a very peculiar data type. Once we create a symbol, its value is kept private and for internal use. All that remains after the creation of the symbol is the symbol reference. We can create a symbol by calling the Symbol() global factory function:

const mySymbol = Symbol()

Copy

Every time we invoke Symbol(), we get a new and unique symbol, which is different from all other symbols:

const sym1 = Symbol();

const sym2 = Symbol();

console.log(s1 === s2); // false

Copy

We can pass a parameter to Symbol(), and that is used as the symbol *description*, useful just for debugging purposes:

console.log(Symbol()) //Symbol()

console.log(Symbol('Some Test')) //Symbol(Some Test)

Copy

**Examples:**

const mysymbol = Symbol()

const person = {

[mysymbol]: 'John'

}

person[mysymbol]

const mysymbol2 = Symbol()

person[mysymbol2] = () => 'Hello World'

console.log(person[mysymbol2]())

//Output: Hello World

Copy

**Purpose of the symbol:-**

In ES6, symbols were added to the primitives group, just like all other primitive, they are also immutable and does not have methods of their own. The main purpose of symbols was to provide globally unique values that were kept private and for internal use only.

**Use Case: Symbols as keys of non-public properties**

When there are inheritance hierarchies in JavaScript, we have two kinds of properties. First one is created via classes, and the second is a purely prototypal approach:

* **Public properties**: They are seen by clients of the code.
* **Private** **properties**: They are used internally within the pieces that make up the inheritance hierarchy such as classes, objects.

For usability, public properties usually have string keys. However, in the case of private properties with string keys, accidental name clashes can become a problem. Therefore, symbols are the right choice. Here is an example:

Suppose that there are two companies, A and B, developed by two different people. Both of them need to add a property to work on an object, but, they both end up naming their property id by coincidence. This coincidence leads to one of the companies overwriting the data stored in id. Before ES6, when the object key could only be of the string type, this scenario was a real possibility. String type object keys posed the danger of name collision and led to the overwriting of data/values. In such a scenario, symbols have provides a solution to this problem.

let student = { name: "Harry" };

let id\_companyA = Symbol("id");

student[id\_companyA] = "ID assigned by company A";

let id\_companyB = Symbol("id");

student[id\_companyB] = "ID assigned by company B";

console.log(student)

//Output: {name: "Harry", Symbol(id): "ID assigned by company A", Symbol(id): "ID assigned by company B"}

Copy

**Conclusion:-**

Symbols in JavaScript can provide uniqueness to objects. It is worthwhile for all developer to have a basic understanding of them and their various use-cases.

**Code as described/written in the video**

console.log("This is tutorial 59")

// Symbols

const sym1 = Symbol('My identifier');

const sym2 = Symbol('My identifier');

// console.log('Symbol is ', sym1);

// console.log('Type of Symbol is ', typeof sym1);

console.log(sym1 === sym2);

const a = "this is";

const b = "this is";

console.log(a === b);

console.log(null === null);

console.log(undefined === undefined);

const k1 = Symbol('identifier for k1');

const k2 = Symbol('for k2');

myObj = {};

myObj[k1] = "Harry";

myObj[k2] = "Rohan";

myObj["name"] = "Good boy"

myObj[4] = "Good int"

console.log(myObj);

console.log(myObj[k1]);

console.log(myObj[k2]);

console.log(myObj.k2); // !! ALERT !!: cannot do this to get Rohan because it is same as myObj["k2"]

// Symbols are ignored in for in loop

for(key in myObj){

console.log(key, myObj[key])

}

console.log(JSON.stringify(myObj));

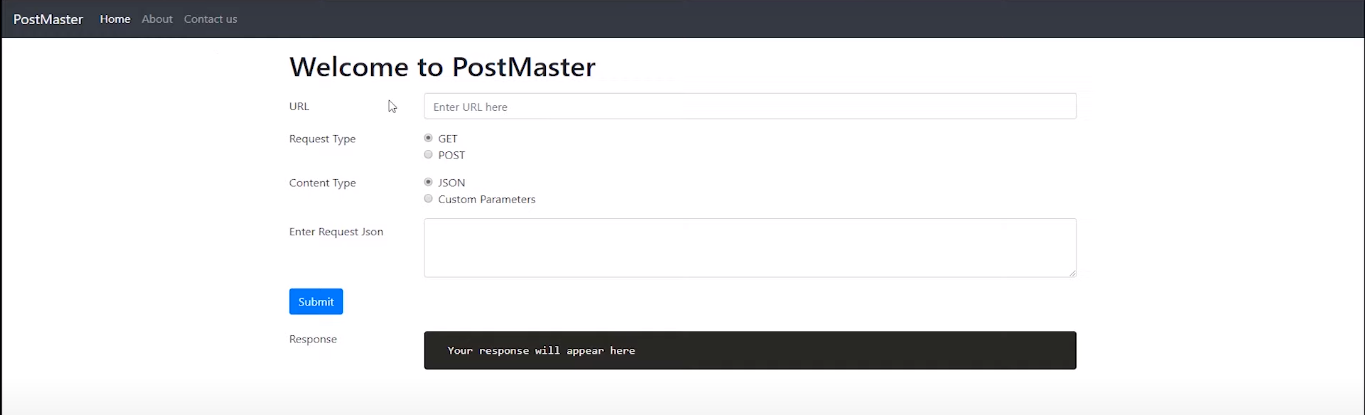
**Project 6: PostMan Clone - Creating a Get/Post Request Website | JavaScript Tutorial In Hindi #63**

Now it’s time for the 6th project. The project we are doing to do is creating a “PostMan Clone”. The project details are given below:

**Project Explanation:-**

**Postman** is a popular application programming interface that makes it easy for developers to create, share, test and document APIs. This is done by allowing the users to create and save simple and complex HTTP/s requests, as well as read their responses. It can make various types of HTTP requests like GET, POST, PUT, PATCH, saving environments for later use, converting the **API** to code for various languages like JavaScript, Python.

The website we are going to build in this tutorial looks like the following:



**Setting Up the Project:-**

For this project, we will use the VS Code. If you have not installed the Visual Studio Code yet, then click on the link below for installation guidance:

***<https://codewithharry.com/videos/javascript-tutorials-in-hindi-1>***

For project setup, start by creating a new project folder in VS Code, and inside that project folder, create two empty new files of HTML (filename.html) and JavaScript (filename.js). The *Html file* is the entry point for our website and contains the HTML code. In this project, we are also using **Bootstrap**.

**Bootstrap:-**

Bootstrap is used for applying styling the user interface components. The easiest way to include Bootstrap is to add it from https://getbootstrap.com/docs/4.0/getting-started/introduction/.

First add the following in the head section:

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

Copy

Then include the following code in the body section, before the closing </body> tag:

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

Copy

**Bootstrap** is bundled with many components that can be used to provide good user experience and user interactions in a web page. In this project, we are using bootstrap components for making the front end of website. So, let’s start coding!!!

**Code as described/written in the video**

<!doctype html>

<html lang="en">

<head>

<!-- Required meta tags -->

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<link rel="stylesheet" href="prism.css">

<!-- Bootstrap CSS -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<style>

#responsePre{

max-height: 500px;

}

</style>

<title>PostMaster by CodeWithHarry!</title>

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="#">PostMaster</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarText"

aria-controls="navbarText" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarText">

<ul class="navbar-nav mr-auto">

<li class="nav-item active">

<a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">About</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Contact us</a>

</li>

</ul>

</div>

</nav>

<div class="container">

<h1 class="my-3">Welcome to PostMaster</h1>

</div>

<div class="container">

<!-- URL box -->

<div class="form-group row">

<label for="url" class="col-sm-2 col-form-label">URL</label>

<div class="col-sm-10">

<input type="text" class="form-control" id="url" placeholder="Enter URL here">

</div>

</div>

<!-- Request type box -->

<fieldset class="form-group">

<div class="row">

<legend class="col-form-label col-sm-2 pt-0">Request Type</legend>

<div class="col-sm-10">

<div class="form-check">

<input class="form-check-input" type="radio" name="requestType" id="get" value="GET" checked>

<label class="form-check-label" for="get">

GET

</label>

</div>

<div class="form-check">

<input class="form-check-input" type="radio" name="requestType" id="post" value="POST">

<label class="form-check-label" for="post">

POST

</label>

</div>

</div>

</div>

</fieldset>

<!-- Content type box -->

<fieldset class="form-group">

<div class="row">

<legend class="col-form-label col-sm-2 pt-0">Content Type</legend>

<div class="col-sm-10">

<div class="form-check">

<input class="form-check-input" type="radio" name="contentType" id="jsonRadio" value="json"

checked>

<label class="form-check-label" for="json">

JSON

</label>

</div>

<div class="form-check">

<input class="form-check-input" type="radio" name="contentType" id="paramsRadio" value="params">

<label class="form-check-label" for="params">

Custom Parameters

</label>

</div>

</div>

</div>

</fieldset>

<!-- Parameters box - This will hide on clicking json option in content type -->

<div id="parametersBox">

<div class="form-row">

<label for="url" class="col-sm-2 col-form-label">Parameter 1</label>

<div class="col-md-4">

<input type="text" class="form-control" id="parameterKey1" placeholder="Enter Parameter 1 Key">

</div>

<div class="col-md-4">

<input type="text" class="form-control" id="parameterValue1" placeholder="Enter Parameter 1 Value">

</div>

<button id="addParam" class="btn btn-primary">+</button>

</div>

<div id="params"></div>

</div>

<!-- Json Request box - This will hide on clicking parameters option in content type -->

<div class="my-3" id="requestJsonBox">

<div class="form-group row">

<label for="requestJsonText" class="col-sm-2 col-form-label">Enter Request Json</label>

<div class="col-sm-10">

<textarea class="form-control" id="requestJsonText" rows="3"></textarea>

</div>

</div>

</div>

<!-- Submit button which will trigger fetch api -->

<div class="form-group row my-2">

<div class="col-sm-10">

<button id="submit" class="btn btn-primary">Submit</button>

</div>

</div>

<div class="my-3" id="responseJsonBox">

<div class="form-group row">

<label for="responseJsonText" class="col-sm-2 col-form-label">Response</label>

<div class="col-sm-10">

<!-- <textarea class="form-control" id="responseJsonText" rows="3">Your response will appear here</textarea> -->

<pre id='responsePre' class="language-javascript"> <code id='responsePrism' class="language-javascript"> Your response will appear here </code> </pre>

</div>

</div>

</div>

</div>

<!-- Optional JavaScript -->

<!-- jQuery first, then Popper.js, then Bootstrap JS -->

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

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

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

</body>

</html>

Copy

**Code as described/written in the video**

console.log('This is my project 6 from JavaScript course');

// Utility functions:

// 1. Utility function to get DOM element from string

function getElementFromString(string) {

let div = document.createElement('div');

div.innerHTML = string;

return div.firstElementChild;

}

// Initialize no of parameters

let addedParamCount = 0;

// Hide the parameters box initially

let parametersBox = document.getElementById('parametersBox');

parametersBox.style.display = 'none';

// If the user clicks on params box, hide the json box

let paramsRadio = document.getElementById('paramsRadio');

paramsRadio.addEventListener('click', () => {

document.getElementById('requestJsonBox').style.display = 'none';

document.getElementById('parametersBox').style.display = 'block';

})

// If the user clicks on json box, hide the params box

let jsonRadio = document.getElementById('jsonRadio');

jsonRadio.addEventListener('click', () => {

document.getElementById('requestJsonBox').style.display = 'block';

document.getElementById('parametersBox').style.display = 'none';

})

// If the user clicks on + button, add more parameters

let addParam = document.getElementById('addParam');

addParam.addEventListener('click', () => {

let params = document.getElementById('params');

let string = `<div class="form-row my-2">

<label for="url" class="col-sm-2 col-form-label">Parameter ${addedParamCount + 2}</label>

<div class="col-md-4">

<input type="text" class="form-control" id="parameterKey${addedParamCount + 2}" placeholder="Enter Parameter ${addedParamCount + 2} Key">

</div>

<div class="col-md-4">

<input type="text" class="form-control" id="parameterValue${addedParamCount + 2}" placeholder="Enter Parameter ${addedParamCount + 2} Value">

</div>

<button class="btn btn-primary deleteParam"> - </button>

</div>`;

// Convert the element string to DOM node

let paramElement = getElementFromString(string);

params.appendChild(paramElement);

// Add an event listener to remove the parameter on clicking - button

let deleteParam = document.getElementsByClassName('deleteParam');

for (item of deleteParam) {

item.addEventListener('click', (e) => {

// TODO: add a confirmation box to confirm parameter deletion

e.target.parentElement.remove();

})

}

addedParamCount++;

})

// If the user clicks on submit button

let submit = document.getElementById('submit');

submit.addEventListener('click', () => {

// Show please wait in the response box to request patience from the user

// document.getElementById('responseJsonText').value = "Please wait.. Fetching response...";

document.getElementById('responsePrism').innerHTML = "Please wait.. Fetching response...";

// Fetch all the values user has entered

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

let requestType = document.querySelector("input[name='requestType']:checked").value;

let contentType = document.querySelector("input[name='contentType']:checked").value;

// If user has used params option instead of json, collect all the parameters in an object

if (contentType == 'params') {

data = {};

for (let i = 0; i < addedParamCount + 1; i++) {

if (document.getElementById('parameterKey' + (i + 1)) != undefined) {

let key = document.getElementById('parameterKey' + (i + 1)).value;

let value = document.getElementById('parameterValue' + (i + 1)).value;

data[key] = value;

}

}

data = JSON.stringify(data);

}

else {

data = document.getElementById('requestJsonText').value;

}

// Log all the values in the console for debugging

console.log('URL is ', url);

console.log('requestType is ', requestType);

console.log('contentType is ', contentType);

console.log('data is ', data);

// if the request type is get, invoke fetch api to create a post request

if (requestType=='GET'){

fetch(url, {

method: 'GET',

})

.then(response=> response.text())

.then((text) =>{

// document.getElementById('responseJsonText').value = text;

document.getElementById('responsePrism').innerHTML = text;

Prism.highlightAll();

});

}

else{

fetch(url, {

method: 'POST',

body: data,

headers: {

"Content-type": "application/json; charset=UTF-8"

}

})

.then(response=> response.text())

.then((text) =>{

// document.getElementById('responseJsonText').value = text;

document.getElementById('responsePrism').innerHTML = text;

Prism.highlightAll();

});

}

});

Copy

**PreviousNext**

CodeWithHarry

Copyright © 2

**Drag & Drop Elements with JavaScript and HTML | JavaScript Tutorial In Hindi #64**

Are you trying to add the drag & drop functionality to your website, but you don't know exactly how to do it? Well, worry not, this tutorial explains how to add this functionality to the website. In this tutorial, we will build a drag-and-drop example using HTML and JavaScript to use the event handlers.

**Prerequisites:**To complete this tutorial, we need basic HTML and JavaScript knowledge.

As we know, drag and drop is a very common feature. It is when we "grab" an object and drag it to a different location. With JavaScript event handlers, we can turn any element into a draggable item or an item that can be dropped into. Drag and drop is a great interface solution. Taking something and drag and drop it is a simple way to do many things, from copying and moving documents as in file managers or dropping items into a cart.

**Drag Events:-**

A typical drag operation begins when a user selects a draggable element, drags the element to a droppable element, and then releases the dragged element. During drag operations, several event types are fired, and some events might fire many times, such as the drag and dragover events. Here are some drag events

| **Event** | **Description** |
| --- | --- |
| drag | It is fired when a *dragged item*is dragged. |
| dragend | It is fired when a drag operation ends, such as releasing a mouse button or hitting the Esc key. |
| dragenter | It is fired when a dragged item enters a valid drop target. |
| dragexit | It is fired when an element is no longer the drag operation's immediate selection target. |
| dragleave | It is fired when a dragged item leaves a valid drop target. |
| dragover | It is fired when a dragged item is being dragged over a valid drop target, every few hundred milliseconds. |
| dragstart | It is fired when the user starts dragging an item. |
| drop | It is fired when an item is dropped on a valid drop target. |

**Code index.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Drag and Drop - Js Tutorial 64</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<div class="whiteBox">

<div class="imgBox" draggable="true"></div>

</div>

<div class="whiteBox"></div>

<div class="whiteBox"></div>

<div class="whiteBox"></div>

</body>

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

</html>

Copy

**Code index.js as described/written in the video**

console.log('This is drag and drop utility');

const imgBox = document.querySelector('.imgBox');

const whiteBoxes = document.getElementsByClassName('whiteBox');

// Event listeners for draggable element imgBox

imgBox.addEventListener('dragstart', (e) => {

console.log('DragStart has been triggered');

e.target.className += ' hold';

setTimeout(() => {

e.target.className = 'hide';

}, 0);

});

imgBox.addEventListener('dragend', (e) => {

console.log('DragEnd has been triggered');

e.target.className = 'imgBox';

});

for (whiteBox of whiteBoxes) {

whiteBox.addEventListener('dragover', (e) => {

e.preventDefault();

console.log('DragOver has been triggered');

});

whiteBox.addEventListener('dragenter', (e) => {

console.log('DragEnter has been triggered');

e.target.className += ' dashed';

})

whiteBox.addEventListener('dragleave', (e) => {

console.log('DragLeave has been triggered');

e.target.className = 'whiteBox'

})

whiteBox.addEventListener('drop', (e) => {

console.log('Drop has been triggered');

e.target.append(imgBox);

})

}

Copy

**Code style.css as described/written in the video**

body{

background: darkslateblue;

}

.imgBox{

background-image: url('photo.png');

position: relative;

top: 7px;

left: 5px;

height: 145px;

width: 145px;

cursor: pointer;

}

.whiteBox{

display: inline-block;

height: 155px;

width: 155px;

margin: 10px;

background-color: white;

border: 3px black solid;

}

.hold{

border: solid red 4px;

}

.dashed{

background: rgb(214, 206, 206);

border-style: dashed;

}

.hide{

display: none;

}

Copy

**PreviousNext**

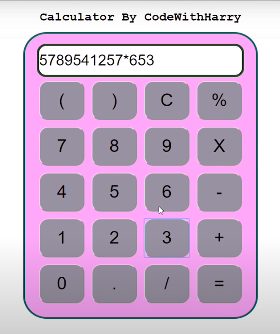
# https://codewithharry.com/img/logo-blue.pngCodeWithHarryProject 7: Creating a Calculator using HTML, CSS & JS | JavaScript Tutorial In Hindi #65

The project we are going to do is to create a ***calculator using HTML, CSS, and JavaScript***.The prerequisites of this project are the basic knowledge of HTML CSS and javaScript. Here are the project details:

**Project Explanation:-**

As we know that a calculator is a machine which allows us to do math operations more easily. For **example**, most calculators will add, subtract, multiply, and divide. Some also do square roots, and more complex calculators can help with calculus and draw function graphs. Calculators are found everywhere. You can even build a more advanced level calculator, but we will keep it simple.

The website we are going to build in this tutorial looks like the following:



**Setting Up the Project:-**

For this project, we will use the VS Code. **Visual Studio Code** is a free and powerful source-code editor that runs on the desktop. If you have not installed the Visual Studio Code yet, then click on the link below for installation guidance:

***<https://codewithharry.com/videos/javascript-tutorials-in-hindi-1>***

The project setup is really simple. Because we only want to use JavaScript without any frameworks. Start by creating a new project folder in VS Code, and inside that project folder, create three empty new files*of HTML*, CSS and javaScript. The *Html file* is the entry point for our website and contains the HTML code. And the CSS file designs of the content of the HTML. Whereas the javascript add functionality in the calculator.

This project belongs to the javaScript series for beginners. To access the complete playlist, click on the link <https://codewithharry.com/videos/javascript-tutorials-in-hindi-1>

**Code index.html as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="style.css">

<title>Calculator</title>

</head>

<body>

<div class="container">

<h1>Calculator By CodeWithHarry</h1>

<div class="calculator">

<input type="text" name="screen" id="screen">

<table>

<tr>

<td><button>(</button></td>

<td><button>)</button></td>

<td><button>C</button></td>

<td><button>%</button></td>

</tr>

<tr>

<td><button>7</button></td>

<td><button>8</button></td>

<td><button>9</button></td>

<td><button>X</button></td>

</tr>

<tr>

<td><button>4</button></td>

<td><button>5</button></td>

<td><button>6</button></td>

<td><button>-</button></td>

</tr>

<tr>

<td><button>1</button></td>

<td><button>2</button></td>

<td><button>3</button></td>

<td><button>+</button></td>

</tr>

<tr>

<td><button>0</button></td>

<td><button>.</button></td>

<td><button>/</button></td>

<td><button>=</button></td>

</tr>

</table>

</div>

</div>

</body>

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

</html>

Copy

**Code style.css as described/written in the video**

.container{

text-align: center;

margin-top:23px

}

table{

margin: auto;

}

input{

border-radius: 21px;

border: 5px solid #244624;

font-size:34px;

height: 65px;

width: 456px;

}

button{

border-radius: 20px;

font-size: 40px;

background: #978fa0;

width: 102px;

height: 90px;

margin: 6px;

}

.calculator{

border: 4px solid #13695d;

background-color: #ff99f7;

padding: 23px;

border-radius: 53px;

display: inline-block;

}

h1{

font-size: 28px;

font-family: 'Courier New', Courier, monospace;

}

Copy

**Code index.js as described/written in the video**

let screen = document.getElementById('screen');

buttons = document.querySelectorAll('button');

let screenValue = '';

for (item of buttons) {

item.addEventListener('click', (e) => {

buttonText = e.target.innerText;

console.log('Button text is ', buttonText);

if (buttonText == 'X') {

buttonText = '\*';

screenValue += buttonText;

screen.value = screenValue;

}

else if (buttonText == 'C') {

screenValue = "";

screen.value = screenValue;

}

else if (buttonText == '=') {

screen.value = eval(screenValue);

}

else {

screenValue += buttonText;

screen.value = screenValue;

}

})

}

Copy

**PreviousNext**

CodeWithHarry

# Fetch api in Javascript | JavaScript Tutorial In Hindi #42

In this tutorial, we will study about fetch API. The Fetch API is a promise-based API of JavaScript for making asynchronous HTTP requests in the browser similar to XMLHttpRequest (XHR). Unlike XHR, the fetch API is a simple and clean API that uses promises to provides more powerful features to fetch resources from the server. Fetch API is standardized now and is supported by all modern browsers except IE. The fetch() method only has one mandatory argument: the URL of the resource that we want to fetch.

#### How to use fetch API:-

To use a Fetch API, just pass the URL, the path to the resource we want to fetch, to fetch() method. Here is the syntax:

fetch('/js/users.json')

.then(response => {

// handle response data

})

.catch(err => {

// handle errors

});

Copy

Pass the path of the resource that we want to retrieve as a parameter to fetch(). We cannot block the user interface by waiting until the request finishes. That is why to fetch() returns a Promise, an object which represents a future result. We are using the then method to wait for the server's response. The catch() method is optional. Its purpose is to intercepts errors if the request fails to complete due to network failure or any other reason.

Now let us see how we can extract the JSON from that response once the request completes:

fetch('URL here')

.then(res => res.json())

.then(json => console.log(json));

Copy

We start the request by calling fetch(). When the promise is fulfilled, it returns a response object, which exposes a json method. Within the first then(), we can call this json method to return the response body as JSON.

#### GET Request:-

GET requests are widely used methods in APIs and websites. The purpose of this method is to retrieve data from the server at the specified resource. The Fetch API uses the GET method for asynchronous requests. Here is an example of get request:

fetch('https://api.github.com/orgs/nodejs')

.then(response => response.json())

.then(data => {

console.log(data) // Prints result from `response.json()` in getRequest

})

.catch(error => console.error(error))

Copy

As we know, the fetch() method returns a promise. The response returned by the promise is a stream object, which means that it returns another promise when we call the json() method. Call to json() method indicates that we are expecting a JSON response. If we want an XML response, use the text() method

#### POST Request:-

The purpose of the post request is to send the data to the server and creates a new resource. The resource post request creates subordinate to some other parent resource. When a new resource is posted to the parent, the API service will automatically associate the new resource by assigning it an ID. All we need to do is set the method and body parameters in the fetch() options:

let data = {

first\_name: 'John',

last\_name: 'Adams',

job\_title: 'Software Engineer'

};

const options = {

method: 'POST',

body: JSON.stringify(data),

headers: {

'Content-Type': 'application/json'

}

}

fetch('https://reqres.in/api/users', options)

.then(res => res.json())

.then(res => console.log(res));

Copy

### Code file tut42.js as described in the video

console.log('This is my tutorial 42');

// Button with id myBtn

let myBtn = document.getElementById("myBtn");

// div with id content

let content = document.getElementById("content");

// function getData(){

// console.log("Started getData")

// url = "harry.txt";

// fetch(url).then((response)=>{

// console.log("Inside first then")

// return response.text();

// }).then((data)=>{

// console.log("Inside second then")

// console.log(data);

// })

// }

function getData(){

console.log("Started getData")

url = "https://api.github.com/users";

fetch(url).then((response)=>{

console.log("Inside first then")

return response.json();

}).then((data)=>{

console.log("Inside second then")

console.log(data);

})

}

function postData(){

url = "http://dummy.restapiexample.com/api/v1/create";

data = '{"name":"harglry347485945","salary":"123","age":"23"}'

params = {

method:'post',

headers: {

'Content-Type': 'application/json'

},

body: data

}

fetch(url, params).then(response=> response.json())

.then(data => console.log(data)

)

}

// console.log("Before running getData")

// getData()

// console.log("After running getData")

postData()

Copy

**PreviousNext**

CodeWithHarry

Copyright © 2022 C

# Async/Await in Javascript | JavaScript Tutorial In Hindi #43

In this tutorial, we will learn about what JavaScript Async/Await keywords do with examples. Promises provide us an easier way to deal with asynchronous programming. If you have not watched the tutorial on promises, then check [***tutorial#39***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-39). Now let us move towards our main topic. **Async/await functions**, which is a new addition with ES2017, that help us even more in allowing us to write completely synchronous-looking code while performing asynchronous tasks behind the scenes.

The functionality we can achieve using async functions can also be achieved by combining promises with generators, but async functions give us what we need without any extra complex code. As the async/await keywords were introduced in the newer version of JavaScript (ES8). Some older browsers may not support the use of async/await.

#### The Async Keyword:-

We use the async keyword with a function to represent that the function is asynchronous. The async function always returns a promise.

Here is the syntax of async function is:

async function name(param1, param2, ...paramN) {

// statements

}

Copy

* **name** - This is the name of the function
* **param** – This is the parameters that are passed to the function

#### Example: Async function

// async function example

async function func() {

console.log('Async/Await tutorial.');

return Promise.resolve(1);

}

func();

Copy

In the above example, the async keyword is used to represent that the function is asynchronous. Since this func() function returns a promise, we can use the chaining method then() like this:

async function func() {

console.log('Async/Await tutorial.');

return Promise.resolve(1);

}

func().then(function(res) {

console.log(res)

});

Copy

**Output:-**

Async/Await tutorial.

1

#### The await keyword:-

The await keyword is used to wait for the asynchronous operation. This keyword is used inside the async function. Here is the syntax to use await is:

let result = await promise;

Copy

The await pauses the async function until the promise returns a result value.

When we want to call this function, we prepend await, and the calling code will stop until the promise is resolved or rejected. Here is another example:

async function func1() {

let promise = new Promise((resolve, reject) => {

setTimeout(() => resolve("Done with Async/Await!"), 1000)

});

let result = await promise; // wait until the promise resolves

console.log(result);

}

f();

Copy

#### Benefits of using an async function:-

* Debugging using promises sometimes is very hard because the debugger will not step over the asynchronous code. But the Async/await makes this very easy because it is just like synchronous code to the compiler.
* As we can see from the example above, the code looks very simple compared to the code using plain promises, with chaining and callback functions.
* Error handling is simpler in async functions.

So, in this tutorial, we have learned about async and await keywords. Async/await provides a nice, simplified way to write async code that is easy to read and maintain.

#### Here is a quick review of what we have studied in this lecture:

* An async function is a function that is declared with the asynckeyword**.**Async functions are the instances of AsyncFunction constructor, and the await keyword is permitted within them.
* The async and await keywords enable the asynchronous, promise-based behavior to be written cleaner.

### Code file tut43.js as described in the video

console.log("This is tutorial 43");

async function harry(){

console.log('Inside harry function');

const response = await fetch('https://api.github.com/users');

console.log('before response');

const users = await response.json();

console.log('users resolved')

return users;

// return "harry";

}

console.log("Before calling harry")

let a = harry();

console.log("After calling harry")

console.log(a);

a.then(data => console.log(data))

console.log("Last line of this js file")

Copy

**PreviousNext**

CodeWithHarry

Copyright © 202

# Error Handling & Try Catch in Javascript | JavaScript Tutorial In Hindi #44

In this tutorial, we will learn about the try, catch, throw, and, finally statements that handle JavaScript's exceptions.No matter how great we are at programming, sometimes our code has errors. They may occur because of unexpected user input and for a thousand other reasons. Usually, a code immediately stops executing in case of an error, printing it to console. But we can solve this problem by using the try..catch statement. It allows us to "catch" errors so the script can do something more reasonable instead of stopping. Before we learn about error handling statements, let us see about the types of errors in programming.

#### Types of Errors:-

There can two types of errors:

* **Syntax Error**: This is the error in the syntax. For example, if we write console.lo('JS');, the above program throws a syntax error. The spelling of the log is a mistake in the above code.
* **Runtime Error**: The runtime error occurs during the execution of the program. For example, calling an invalid function or a variable.

Now,let's see how we can handle these exceptions.

#### What is Try Catch in JavaScript?

Just like other programming languages, JavaScript also has exception handling capabilities. JavaScript implements the try-catch statements as well as the throw operator to handle exceptions. Here is the basic syntax for try…catch:

try{//some code that has an error

}

catch (e) {

//this will run if the code in the try block errors}

Copy

With these statements, in JavaScript, we can also add a throw or a finally clause. Let us see what role they play.

* **throw:** This is a block of code nested within the try statement and allows the programmer to write their own error that they want to handle
* **finally:** This block of code runs once all the other statements have run

#### try/throw/catch:-

The throw operator generates an error. We can define and throw their own custom errors. When the throw statement is executed, the statements present after it does not execute. The control will directly pass to the catch block. In the following example, we create our own error ("This is a new error") in the throw block. Then try the code which throws an error which should be caught by the catch block.

try{

throw new Error ('This is a new error')

}

catch(error){

console.log(error)

console.log("End of try-catch block")

}

Copy

#### try/catch/throw/finally:-

Finally is an optional block of statements that is executed after the execution of try and catch statements. It does not matter that any exception is thrown or not, finally block code will definitely execute if it is present. In this example, we will see how to use the finally statement with the other three statements. In this example, we do not show the entire error. We just logged that the error has been handled in the catch block.

try{

console.log("This statement works")

throw new Error('This statement throws an error')

}

catch(error){

console.log("Error has been handled")

}

finally{

console.log("Everything has been handled")

}

Copy

### Code file tut44.js as described in the video

console.log("This is tutorial 44");

// Pretend this is coming from a server as response

let a = "Harry bhai";

a = undefined;

if (a !=undefined){

throw new Error('This is not undefined');

}

else{

console.log('This is undefined');

}

try {

null.console

console.log("We are inside try block");

functionHarry();

} catch(error) {

console.log(error)

console.log("Are you okay?");

console.log(error.name);

console.log(error.message);

} finally {

console.log("Finally we will run this")

}

Copy

**PreviousNext**

CodeWithHar

# JavaScript Maps | JavaScript Tutorial In Hindi #57

Today we will study **maps in JavaScript**, which was introduced in ES2015. Before ES6, JavaScript did not have a data structure for dictionaries. Programmers use objects as dictionaries from strings to arbitrary values. ES6 introduces Maps, which are dictionaries from arbitrary values to arbitrary values. In this tutorial, we will learn how to work with the JavaScript Map object and its useful methods like get(), set(), clear() etc., to manipulate entries in the map.

#### What are Maps in JavaScript ?

Maps are a new object type that allows us to store collections of key-value pairs. Map keys can be of any type, even objects or functions. In the case of maps, it is also easy to get the size of a map, whereas, it is not as straightforward in case of objects. Along with that, in maps, we can iterate in the order in which the values were added. Here is the syntax of maps in JavaScript:

**Syntax:**

let mymap= new Map([iterable])

Copy

**Parameter:**

iterable - It is any iterable object whose values are stored as key, value pair. Providing parameters is optional. If the parameter is not specified, then a new map is created Empty.

**Returns:**A new Map object

#### Methods and Properties of Maps:-

**size:** It returns the number of elements or the key-value pairs in the map.

**set():** This method adds the key and value to the Map Object. Here is the syntax:

mymap1.set(x, y);

Copy

"x" is the key of the element to be added to the Map and "y" is the value of the element to be added to the Map. It returns a Map object

**has():**This method returns a boolean value depending on whether the specified key is present or not. Here is the syntax:

mymap1.has(x);

Copy

Here "x" is the key of the element to checked. It will returntrue if the element with the specified key is present or else returns false.

**get():**This method returns the value of the corresponding key. Here is the syntax:

map1.get(x);

Copy

Here "x" is the key, whose value is to be returned. It will return the value associated with the key if it is present in Map, otherwise returns undefined

**delete():**This method deletes both the key as well as a value from the map. Here is the syntax:

map1.delete(k);

Copy

Here "k" is the key which is to be deleted from the map. It will return true if the value is found and deleted from the map; otherwise, it returns false

**clear():**This method will remove all the elements from the Map object. Here is the syntax:

map1.clear();

Copy

It requires no parameters and return undefined.

#### Example:-

In this example, we initialize a map from an array that contains arrays of three values:

const students = [

['1', 'Harry'],

['2', 'Joe'],

['3', 'Peter'],

];

let myMap = new Map(students);

myMap.get('2'); //Output: "Joe"

Copy

### Code file tut57.js as described in the video

console.log("This is tutorial 57");

// Maps in JavaScript: We can use any type of key or value

const myMap = new Map();

const key1 = 'myStr', key2 = {}, key3 = function () { };

// Setting map values

myMap.set(key1, 'This is a string');

myMap.set(key2, 'This is a blank obj');

myMap.set(key3, 'This is an empty function');

console.log(myMap);

// Getting the values from a Map

let value1 = myMap.get(key3);

console.log(value1);

// Get the size of the map

console.log(myMap.size);

// You can loop using for..of to get keys and values

for (let [key, value] of myMap) {

console.log(key, value);

}

// Get only keys

for (let key of myMap.keys()) {

console.log('key is ', key);

}

// Get only values

for (let value of myMap.values()) {

console.log('value is ', value);

}

// You can loop through a map using for each loop

myMap.forEach((value, key)=>{

console.log('Key is ', key);

console.log('Value is ', value);

})

// Converting map to an array

let myArray = Array.from(myMap);

console.log('Map to array is ', myArray);

// Converting map keys to an array

let myKeysArray = Array.from(myMap.keys());

console.log('Map to keys array is ', myKeysArray);

// Converting map values to an array

let myValuesArray = Array.from(myMap.values());

console.log('Map to values array is ', myValuesArray);

Copy

**PreviousNext**

CodeWithHa

# JavaScript Sets | JavaScript Tutorial In Hindi #58

In today’s tutorial, we will discuss another advance topic of JavaScipt. With the introduction of ECMAScript 6, came the introduction of a new built-in JavaScript object type: the Set. So, let's start learning about this new concept.

#### What is set in JavaScript?

A set is a collection of unique items. In set, no element can be repeated. The set in ES6 are ordered, elements of the set can be iterated in the insertion order. The power of the set is that it can store any types of values, whether primitive or objects.

**Syntax:**

let myset=new Set([iterable]);

Copy

In the syntax, iterable is an object whose all elements are added to the new set created. In set, the parameter is optional. If the parameter is not specified or null is passed then a new set created is empty. It returns a new set object.

#### Methods of the set:-

**add()**: This method adds the new element with a specified value at the end of the Set object. Here is the syntax:

set1.add(x);

Copy

Here “x” is a value to be added to the set. And its returns set object.

**delete()**: This method deletes an element with the specified value from the Set object. Here is the syntax:

set1.delete(x);

Copy

Here “x” is a value to be deleted from the set. This method returns true if the value is successfully deleted from the set else returns false.

**clear()**: This method removes all the element from the set. Here is the syntax:

set1.clear();

Copy

In this method, no parameter is needed, and it returns undefined.

**entries()**: This method returns an iterator object which contains an array having the entries of the set, in the insertion order. Here is the syntax:

set1.entries();

Copy

This method does not need any parameter. It returns an iterator object that contains an array of [value, value] for every element of the set, in the insertion order.

**has()**: This method returns true if the specified value is present in the Set object. Here is the syntax:

set1.has(x);

Copy

Here “x” is the value to be searched in the Set. This method returns true if the value is present else it returns false.

**values()**: This method returns all the values from the Set in the same insertion order. Here is the syntax:

set1.values();

Copy

No parameters are needed in this method. An iterator object is returned that contains all the values of the set in the same order as they are inserted.

**keys()** – This method returns all the values from the Set in the insertion order. keys() is similar to the values() in case of Sets. Here is the syntax:

set1.keys();

Copy

No parameters are needed in this method. An iterator object returned that contains all the values of the set in the same order as they are inserted.

#### Example:-

Suppose have students coming, and we like to remember everyone. But repeated students should not lead to duplicates. A student must be “counted” only once. For this, we will use set.

let set = new Set();

let john = { name: "John" };

let harry = { name: "harry" };

let cavin = { name: "cavin" };

// some users come multiple times

set.add(john);

set.add(harry);

set.add(cavin);

set.add(john);

set.add(harry);

// set keeps only unique values

for (let user of set) {

console.log(user.name)}

Copy

### Code file tut58.js as described in the video

console.log('This is tutorial 58');

// Set stores unique values

const mySet = new Set(); // Initialize an empty set

console.log('The set looks like :',mySet);

// Adding values to this set

mySet.add('this');

mySet.add('My name');

mySet.add('this');

mySet.add('that');

mySet.add(34);

mySet.add(true);

mySet.add(false);

mySet.add('that2');

console.log('The set looks like this now:',mySet);

// Use a constructor to initialize the set

let mySet2 = new Set([1, 45, 'this', false, {a:4, b:8}, 'this']);

console.log('New set:',mySet2);

console.log(mySet.size); // Get the size of the set

console.log(mySet.has(346)); // Check whether set has 346 or not

console.log('Before removal', mySet.has('that2'));

mySet.delete('that2'); // Remove an element from the set

console.log('After removal', mySet.has('that2'));

// Iterating a set

// for(let item of mySet){

// console.log('Item is :', item );

// }

mySet.forEach((item)=>{

console.log('Item is :', item );

})

// Quiz: Can you use Array.from(mySet) to convert set into an array?

Copy

**PreviousNext**

CodeWithHarry

C

# JavaScript Destructuring | JavaScript Tutorial In Hindi #60

In this tutorial, we will explore the**destructuring** that is used to simplify the JavaScript programs. Everyone knows that JavaScript is a programming language that is used across different platforms ranging from web browsers, mobile devices, web servers, etc. Although this language has not changed a lot through the years when compared to other programming languages, but there are a few recent changes that we should know, some of the new features introduced in ES6, one of them is "destructuring" which is the main focus of this tutorial.

#### What is Destructuring?

Destructuring is breaking down a complex structure into simpler parts. In JavaScript, the complex structure is usually an array or an object. With the destructuring syntax, we can extract smaller fragments from arrays and objects. The destructuring syntax is also used for variable declaration or variable assignment.

#### Object Destructuring:-

The **object destructuring** is a useful feature of javascript to extract properties from objects and bind them to variables. An object destructuring is also capable of extracting multiple properties in one statement and can access properties from nested objects. It sets a default value if the property does not exist. Let us take a look at what problem JavaScript destructuring solves. Sometimes, we need top-level variables like:

const person = {

first: 'John',

last: 'Addison',

country: 'UK',

twitter: '@john\_adison'};

const first = person.first;

const last = person.last;

Copy

We get this repetitive code over and over again, where we need to make a variable from something that is inside of an object or inside of an array. Here instead of creating multiple variables, we destructure them in a single line like:

const { first, last } = person;

Copy

This is the new destructuring syntax. The above code says: give us a variable **first** and **last**, and take it from the person object.

console.log(first); // John

console.log(last); // Addison

Copy

Similarly, if we also wanted the country, we would add the country into our code, like:

const { first, last, country} = person;

Copy

#### Array Destructuring:-

In array destructuring, we use an array literal on the left of an assignment expression. Each variable name on the array literal maps to the corresponding item at the same index on the destructured array. Here is a simple example:

var arr = ["Hello", "World"]

// destructuring assignment

var [first, second] = arr;

console.log(first); // Hello

console.log(second); // World

Copy

**Output:-**

Hello

World

In the above example, the left-hand side of the destructuring assignment is for defining what values are required to unpack from sourced variable. By using the **rest operator (…)** in array destructuring, we can put all the remaining elements of an array in a new array. Here is an example.

var colors = ["Violet", "Indigo", "Blue", "Green"];

// destructuring assignment

var [a,b,...c] = colors;

console.log(a);

console.log(b);

console.log(ac);

Copy

**Output:-**

Violet

Indigo

[ 'Blue', 'Green']

#### Summary:-

The destructuring is a powerful feature that lets us extract properties from an object/array and bind these values to variables. The unique thing about destructuring is its concise syntax and ability to extract multiple variables in one statement.

### Code file tut60.js as described in the video

console.log('This is tutorial 60');

// Destructuring in JavaScript

let a, b;

[a, b] = [34, 564];

// console.log(a, b);

[a, b, c, ...d] = [1,2,3,4,5,6,7,8,9,10, 11, 12, 13];

// console.log(a)

// console.log(b)

// console.log(c)

// console.log(d)

// Array Destructuring

({a, b, c, ...d} = {a: 34, b:345, c:67, d:45, e:34})

// console.log(a, b, c, d)

const fruits = ['Apple', 'Bananas', 'Mangoes'];

[a, b, c] = fruits;

// console.log(a, b, c)

// Object Destructuring

const laptop ={

model: "HP Pavilion",

age: "23 days",

gender: "Male",

net: 1233,

start: function (){console.log('started');}

}

const {model, age, gender, net, start} = laptop;

console.log(model, age, gender, net, start);

start()

Copy

**PreviousNext**

CodeWithHarry

Cop

# JavaScript Wall Clock: Exercise 6 | JavaScript Tutorial In Hindi #61

The task you have to perform is very interesting. To complete this task, you have to use the date and time functions in javaScript. If you have not watched my tutorial on Date Object, then click on the link below:

<https://codewithharry.com/videos/javascript-tutorials-in-hindi-24>

#### Problem Statement:-

You have to create a wall clock using JavaScript. The clock should display at the top of the website. Here is the image of how your clock should look like:

https://api.codewithharry.com/media/videoSeriesFiles/courseFiles/javascript-tutorials-in-hindi-61/base64.png

This exercise belongs to the JavaScript series for beginners. To access the complete playlist, click on the link:<https://codewithharry.com/videos/javascript-tutorials-in-hindi-1> .Solving exercises will help you become a good problem solver and enables you to accept the challenge and acquire new skills. For the solution of this exercise, check the [***tutorial#62***](https://codewithharry.com/videos/javascript-tutorials-in-hindi-62)

If you like my work, then keep supporting and stay up to date with [**codewithharry**](http://www.codewithharry.com./)**.**

# Creating a Clock: Exercise 6 Solution | JavaScript Tutorial In Hindi #62

This tutorial contains the solution to Exercise 7, i.e. **Creating an Wall Clock**. The problem statement, along with the instructions, is given below:

#### ****Problem Statement:-****

You have to create a wall clock using JavaScript. The clock should display at the top of the website. Here is the image of how your clock should look like:

https://api.codewithharry.com/media/videoSeriesFiles/courseFiles/javascript-tutorials-in-hindi-62/base64.png

### Code file clock.html as described in the video

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Alarm Clock</title>

</head>

<body onload="updateClock(); setInterval('updateClock()', 1000)">

<div id="clock">

</div>

</body>

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

</html>

Copy

### Code file clock.js as described in the video

console.log("This is clock.js")

function updateClock(){

// Get the current date

let currentTime = new Date();

// Extract hour, minute and seconds from the date

let currentHour = currentTime.getHours();

let currentMinutes = currentTime.getMinutes();

let currentSeconds = currentTime.getSeconds();

// Pad 0 if minute or second is less than 10 (single digit)

currentMinutes = (currentMinutes < 10 ? "0": "") + currentMinutes;

currentSeconds = (currentSeconds < 10 ? "0": "") + currentSeconds;

// Convert railway clock to AM/PM clock

currentHour = (currentHour>12) ? currentHour - 12 : currentHour;

currentHour = (currentHour==0) ? 12 : currentHour;

// Choose AM/PM as per the time of the day

let timeOfDay = (currentHour < 12 ) ? "AM" : "PM";

// Prepare the time string from hours, minutes and seconds

let currentTimeStr = currentHour + ":" + currentMinutes

+ ":" + currentSeconds + " " + timeOfDay;

// Insert the time string inside the DOM

document.getElementById("clock").innerHTML = currentTimeStr;

}

Copy

**PreviousNext**

CodeWithHarry