**use9**

**JavaScript**

**Notes**

**INDEX**

**INTRODUCTION**

* JavaScript is client side dynamic pages.
* java is user friendly. A user can change in program on run time.
* JavaScript is an object-based scripting language that is lightweight and cross-platform.
* JavaScript is not compiled but translated. The JavaScript Translator (embedded in browser) is responsible to translate the JavaScript code.

**USES OF JAVASCRIPT**

* JavaScript is used to create interactive websites. It is mainly used for:
* Client-side validation
* java is user friendly. A user can changes in program on run time.
* Dynamic drop-down menus
* Displaying date and time
* Displaying popup windows and dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box)
* Displaying clocks etc.

**Places to put js**

* Between the body tag of html
* Between the head tag of html
* In .js file (external java Script)

Different ways to display data in javascript:

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

**1.document.write()**:

<html>

<body>

<script type="text/javascript">

document.write("JavaScript is a simple language for learners");

</script>

</body>

</html>

* The script tag specifies that we are using JavaScript.
* The text/javascript is the content type that provides information to the browser about the data.
* The document.write() function is used to display dynamic content through JavaScript. We will learn about document object in detail later.

**2.Using innerHTML**

JavaScript can use the document.getElementById(id) method.

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

<!DOCTYPE html>  
<html>  
<body>  
<p id="con"> </p>  
<script>  
document.getElementById("con").innerHTML = “Genext”;  
</script>  
</body>  
</html>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# JavaScript defer

The **defer** is a Boolean value, used to indicate that script is executed after the document has been parsed. It works only with external scripts (i.e., works only when we are specifying the **src** attribute in <script> tag). It declares that the script will not create any content.

**<script** src = "java.js" defer**>**  **</script>**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title> </title>

<script type="text/javascript" src="java.js" defer> </script>

</head>

<body>

<h1 id="abc"> Genext </h1>

</body>

</html>

**3.Usingwindow.alert()**

You can use an alert box to display data:  
  
<script>  
window.alert(‘hello’);  
</script>

**4 Console**

<script>  
console.log(‘hello’); or console.warn(“Hello”);  
</script>

**Variables**

Variables are container for storing the data value.

A JavaScript variable is simply a name of storage location.

* There are some rules while declaring a JavaScript variable (also known as identifiers).
* Name must start with a letter (a to z or A to Z), underscore( \_ ), or dollar( $ ) sign.
* After first letter we can use digits (0 to 9), for example value1.
* JavaScript variables are case sensitive, for example x and X are different variables.
* EXAMPLE:

<script>

var x = 10;

var y = 20;

var z=x+y;

document.write(z);

</script>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title></title>

</head>

<body>

<h1 id="abc"> Hello Genext </h1>

<script type="text/javascript">

var x = prompt("Enter the First Value");

var y = prompt("Enter Second Value");

z = x+y;

document.getElementById('abc').innerHTML = "The C value Is = " + z;

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title></title>

</head>

<body>

<h1 id="abc"> Hello Genext </h1>

<script type="text/javascript">

var x = parseInt(prompt("Enter the First Value"));

var y = parseInt(prompt("Enter Second Value"));

z = x+y;

document.getElementById('abc').innerHTML = "The C value Is = " + z;

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title></title>

</head>

<body>

<h1 id="abc"> Hello Genext </h1>

<script type="text/javascript">

var x = parseInt(prompt("Enter the First Value"));

var y = parseInt(prompt("Enter Second Value"));

z = x+y;

document.write("<br>" +z);

z = x-y;

document.write("<br>" +z);

z = x\*y;

document.write("<br>" +z);

z = x/y;

document.write("<br>" +z);

z = x%y;

document.write("<br>" +z);

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title></title>

</head>

<body>

<h1 id="abc"> Hello Genext </h1>

<script type="text/javascript">

var x = parseInt(prompt("Enter the First Value"));

var y = parseInt(prompt("Enter Second Value"));

document.write("<br>" + "Add is" + (x+y));

document.write("<br>" + "Sub is" + (x-y));

document.write("<br>" + "Mul is" + (x\*y));

document.write("<br>" + "Div is" + (x/y));

document.write("<br>" + "Mod is" + (x%y));

</script>

</body>

</html>

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

The "equal to" operator is written like == in JavaScript. (comparison)

Swapping Using Third Variable

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title></title>

</head>

<body>

<h1 id="abc"> Hello Genext </h1>

<script type="text/javascript">

var x = parseInt(prompt("Enter the First Value"));

var y = parseInt(prompt("Enter Second Value"));

z = x;

x = y;

y = z;

document.write(" After Swap x is = " + x);

document.write("<br> After Swap y is = " + y);

</script>

</body> </html>

Without Using third variable

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title></title>

</head>

<body>

<h1 id="abc"> Hello Genext </h1>

<script type="text/javascript">

var x = parseInt(prompt("Enter the First Value"));

var y = parseInt(prompt("Enter Second Value"));

x = x + y;

y = x - y;

x = x - y;

document.write(" After Swap x is = " + x);

document.write("<br> After Swap y is = " + y);

</script>

</body>

</html>

­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title></title>

</head>

<body>

<h1 id="abc"> Hello Genext </h1>

<script type="text/javascript">

var x = parseInt(prompt("Enter the First Value"));

var y = parseInt(prompt("Enter Second Value"));

x = x+y - (y=x);

document.write(" After Swap x is = " + x);

document.write("<br> After Swap y is = " + y);

</script>

</body> </html>

**JAVASCRIPT DATA TYPE**

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

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

**JavaScript primitive data types**

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

|  |  |
| --- | --- |
| DATA TYPES |  |
| String (COLLECTION OF CHARACTER) | Represent sequence of character.”genext” |
| Number ( 0 to 9) | Represent value.100 |
| Boolean ( true / false ) | Represent value either true |
| Undefined | Represent undefined value |
| Null | Represent null;no value at all |

**JavaScript non-primitive data types:**

The non-primitive data types are as follows:

|  |  |
| --- | --- |
| DATA TYPES |  |
| Object | Represent instance through which we can access member. |
| Array | Represent group of similar values. |
| REGEXP | Represent regular expression. |

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

For example:

var a=40; //holding number

var b="Rahul"; //holding string

* Strings are written inside double or single quotes. Numbers are written without quotes.
* If you put a number in quotes, it will be treated as a text string.
* One Statement, Many Variables.

You can declare many variables in one statement.

Start the statement with var and separate the variables by comma:

var employee = "natasha", job = "faculty", salary = 200;

<script>

var x=10, y=20, z=30;

var employee = "natasha",job = "faculty",salary = 50000;

document.write(employee);

</script>

**JAVASCRIPT OPERATORS**

JavaScript operators are symbols that are used to perform operations on operands.

**Types of Operator:**

* Arithmetic operator ( +,-,\*,/,% )
* Relational/comparison operator (==, >=, <= , !=)
* Logical operator ( &&, ||, ! )
* Assignment operator ( =, +=, -=, /=, \*/ )

* **Arithmetic Operators**

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

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| + | Addition | 30+20=50 |
| - | Subtraction | 30-20=10 |
| \* | Multiplication | 10\*30=300 |
| / | Divide | 30/10=3 |
| % | Modulus(Remainder) | 30%10=0 |
| ++ | Increment | Var a=10;a++;  Now, a=11 |
| -- | Decrement | Var a=10;a--;  Now a=9 |

* **COMPARISON OPERATOR**

**Comparison Operator compares the two operands.**

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| == | Is equal to | 10==20=false |
| === | Identical (equal value and equal type) |  |
| != | Not equal to | 10!=20 true |
| !== | not equal value or not equal type |  |
| > | Greater than | 10>5 true |
| < | Less than | 10<5 false |
| >= | Greater than and equals to | 10>=10 true |
| <= | Less than and equal to | 10<=10 true |

**LOGICAL OPERATOR**

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| && | Logical AND | (10>20&& 10>5)=false |
| || | Logical OR | (10>20 && 10>5)=true |
| ! | Logical NOT | !(10==20) = true |

* **The AND operator (&&) returns true if both expressions are true, otherwise it returns false.**
* **The OR operator (||) returns true if one or both expressions are true, otherwise it returns false.**
* **The NOT operator (!) returns true for false statements and false for true statements.**

* **Assignment Operator**

Assignment operators assign values to JavaScript variables.

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| = | Assign | x=10 |
| += | Add and assign | var a=20; a+=10; Now a = 30 |
| -= | Sub and assign | Var a=10;a-=20;  Now a=-10 |
| \*= | Multiply and assign | Var a=10; a\*=10;  Now a=100 |
| /= | Divide and assign | var a=10; a/=2; Now a = 5 |
| %= | Modulus and assign | var a=11; a%=2; Now a = 1 |

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title></title>

</head>

<body>

<h1 id="abc"> Hello Genext </h1>

<script type="text/javascript">

var a=10,b=2;

var c = a \*\* b;

document.write(c);

</script>

</body>

</html>

**Conditional statements**

The JavaScript if-else statement is used to execute the code whether condition is true or false. There are three forms of if statement in JavaScript.

* If Statement
* If else statement
* if else if statement

**If statement**

It evaluates the content only if expression is true. The signature of JavaScript if statement is given below.

if(expression)

{

//content to be evaluated

}

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title></title>

</head>

<body>

<h1 id="abc"> Hello Genext </h1>

<script type="text/javascript">

var a = 10,b=20;

if(a>b)

{

alert("A is Greater");

}

if(b>a)

{

alert("B IS Greater");

}

if(a==b)

{

alert("Both are Same");

}

</script>

</body>

</html>

Output:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title></title>

</head>

<body>

<h1 id="abc"> Hello Genext </h1>

<script type="text/javascript">

var age = parseInt(prompt("Eneter Age"));

if(age>=18)

{

alert("You are eligible for vote");

}

if(age<18)

{

alert("You are Not Eligible");

}

</script>

</body>

</html>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**JavaScript If...else Statement**

It evaluates the content whether condition is true of false. The syntax of JavaScript if-else statement is given below.

Syntax:

if(expression){

//content to be evaluated if condition is true

}

else{

//content to be evaluated if condition is false

}

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title></title>

</head>

<body>

<h1 id="abc"> Hello Genext </h1>

<script type="text/javascript">

var age = parseInt(prompt("Eneter Age"));

if(age>=18)

{

alert("You are eligible for vote");

}

else

{

alert("You are Not Eligible");

}

</script>

</body>

</html>

**Example:**

<script>

var a=20;

if(a%2==0){

document.write("a is even number");

}

else{

document.write("a is odd number");

}

</script>

**Output:**a is even

**JavaScript If...else if statement**

It evaluates the content only if expression is true from several expressions. The signature of JavaScript if else if statement is given below.

if(expression1){

//content to be evaluated if expression1 is true

}

else if(expression2){

//content to be evaluated if expression2 is true

}

else{

//content to be evaluated if no expression is true

}

**Example:-**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title></title>

</head>

<body>

<h1 id="abc"> Hello Genext </h1>

<script>

var a=parseInt(prompt("Enter No"));

var b=parseInt(prompt("Enter No"));

var c=parseInt(prompt("Enter No"));

if(a>b)

{

if(a>c)

{

printf("A is Greatest");

}

else

{

printf("C is gReatest");

}

}

else

{

if(b>c)

{

document.write("B is Greatest");

}

else

{

document.write("C is Greatest");

}

}

</script>

</body>

</html>

**Example:**

<script>

var a=20;

if(a==10)

{

document.write("a is equal to 10");

}

else if(a==15)

{

document.write("a is equal to 15");

}

else if(a==20){

document.write("a is equal to 20");

}

else{

document.write("a is not equal to 10, 15 or 20");

}

</script>

**Output**:a is equal to 20

**JavaScript Switch**

The JavaScript switch statement is used to execute one code from multiple expressions. It is just like else if statement that we have learned in previous page. But it is convenient than if..else..if because it can be used with numbers, characters etc.

switch(expression)

{

case value1:

code to be executed;

break;

case value2:

code to be executed;

break;

......

default:

code to be executed if above values are not matched;

}

**Example:**

**<script>**

**var grade='A';**

**var result;**

**switch(grade){**

**case 'A':**

**result="A Grade";**

**break;**

**case 'B':**

**result="B Grade";**

**break;**

**case 'C':**

**result="C Grade";**

**break;**

**default:**

**result="No Grade";**

**}**

**document.write(result);**

**</script>**

**Output:** A Grade

**Javascript Loop**

A loop is a control flow statement for specifying iteration, which allows code to be executed repeatedly.

**Types of Loop:**

**1. For loop**

 Loops through a block of code a number of times.

The syntax of for loop is given below.

for (initialization; condition; increment,decrement)

{

code to be executed

}

**Example:**

**OUTPUT:**

1

2

3

4

2.**While Loop**

The while loop loops through a block of code as long as a specified condition is true.

**Syntax:**

**Initialization;**

while (**condition**)

{

code block to be executed

**inc/dec;**

}

**Example:**

<script>

var i=1;

while (i<=150)

{

document.write(i + "<br/>");

i++;

}

</script>

**Output:**

11

12

13

14

15

3.**Do While Loop**

The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed once, before checking the test expression. Hence, the do...while loop is executed at least once.

**Syntax:**

do{

code to be executed

}while (condition);

**Example:**

<script>

Var i=21;

do{

document.write(i + "<br/>");

i++;

}while (i<=25);

</script>

**Output:**

21

22

23

24

25

* **Javascript Functions**

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

**Advantage of JavaScript function:**

There are mainly two advantages of JavaScript functions.

* **Code reusability**: We can call a function several times so it save coding.
* **Less coding:** It makes our program compact. We don’t need to write many lines of code each time to perform a common task.

**Syntax of declaring function :**

function functionName([arg1, arg2, ...argN]){

//code to be executed

}

**Example:**

**<html>**

**<body>**

**<script>**

**function msg(){**

**alert("hello! Natasha");**

**}**

**</script>**

**<input type="button" onclick="msg()" value="call function"/>**

**</body>**

**</html>**

**JavaScript Function Arguments**

We can call function by passing arguments.

**Example of one argument:**

<html>

<body>

<script>

function getcube(number){

alert(number\*number\*number);

}

</script>

<form>

<input type="button" value="click" onclick="getcube(4)"/>

</form>

</body>

</html>

**Function with Return Value**

We can call function that returns a value and use it in our program. Let’s see the example of function that returns value.

<html>

<body>

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

<script>

var x = myFunction(4, 3);

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

function myFunction(a, b) {

return a + b;

}

</script>

</body>

</html>

**Anonymous Function** is a function that does not have any name associated with it. Normally we use the *function*keyword before the function name to define a function in JavaScript, however, in anonymous functions in JavaScript, we use only the *function*keyword without the function name.

An anonymous function is not accessible after its initial creation, it can only be accessed by a variable it is stored in as a *function as a value*. An anonymous function can also have multiple arguments, but only one expression.

function() {

// Function Body

}

The below examples demonstrate anonymous functions.

**Example 1:**In this example, we define an anonymous function that prints a message to the console. The function is then stored in the *greet*variable. We can call the function by invoking *greet().*

* Javascript

|  |
| --- |
| <script>  var greet = function () {      console.log("Welcome to CrystalItSoft!");  };  greet();  </script> |

**Output:**

Welcome to CrystalItSoft!

**Example 2:**In this example, we pass arguments to the anonymous function.

* Javascript

|  |
| --- |
| <script>  var greet = function (platform) {      console.log("Welcome to ", platform);  };    greet("CrystalItSoft!");  </script> |

**Output:**

Welcome to CrystalItSoft!

As JavaScript supports Higher-Order Functions, we can also pass anonymous functions as parameters into another function.

**Example 3:**In this example, we pass an anonymous function as a callback function to the *[setTimeout()](https://www.geeksforgeeks.org/java-script-settimeout-setinterval-method/)*method. This executes this anonymous function 2000ms later.

* Javascript

|  |
| --- |
| <script>  setTimeout(function () {      console.log("Welcome to CrystalItSoft!");  }, 2000);  </script> |

**Output:**

Welcome to CrystalItSoft!

Another use case of anonymous functions is to invoke the function immediately after initialization, this is also known as [*Self Executing Function*](https://www.geeksforgeeks.org/what-is-the-purpose-of-self-executing-function-in-javascript/)*.*This can be done by adding parenthesis we can immediately execute the anonymous function.

**Example 4:**In this example, we have created a self-executing function.

* Javascript

|  |
| --- |
| <script>  (function () {      console.log("Welcome to CrystalItSoft!");  })();  </script>  **Immediately Invoked Function Expressions or IIFEs**  // Regular Function.  function Greet()  {  console.log("Welcome to GFG!");  };  // Execution of Regular Function.  Greet();  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  // IIFE creation and execution.  (function() { console.log("Welcome to GFG!"); })(); |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

JavaScript Events

The change in the state of an object is known as an **Event**. In html, there are various events which represents that some activity is performed by the user or by the browser. When [javascript](https://www.javatpoint.com/javascript-tutorial) code is included in [HTML](https://www.javatpoint.com/html-tutorial), js react over these events and allow the execution. This process of reacting over the events is called **Event Handling**. Thus, js handles the HTML events via **Event Handlers**.

**For example**, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

Mouse events:

|  |  |  |
| --- | --- | --- |
| **Event Performed** | **Event Handler** | **Description** |
| click | onclick | When mouse click on an element |
| mouseover | onmouseover | When the cursor of the mouse comes over the element |
| mouseout | onmouseout | When the cursor of the mouse leaves an element |
| mousedown | onmousedown | When the mouse button is pressed over the element |
| mouseup | onmouseup | When the mouse button is released over the element |
| mousemove | onmousemove | When the mouse movement takes place. |

Keyboard events:

|  |  |  |
| --- | --- | --- |
| **Event Performed** | **Event Handler** | **Description** |
| Keydown & Keyup | onkeydown & onkeyup | When the user press and then release the key |

Form events:

|  |  |  |
| --- | --- | --- |
| **Event Performed** | **Event Handler** | **Description** |
| focus | onfocus | When the user focuses on an element |
| submit | onsubmit | When the user submits the form |
| blur | onblur | When the focus is away from a form element |
| change | onchange | When the user modifies or changes the value of a form element |

Window/Document events

|  |  |  |
| --- | --- | --- |
| **Event Performed** | **Event Handler** | **Description** |
| load | onload | When the browser finishes the loading of the page |
| unload | onunload | When the visitor leaves the current webpage, the browser unloads it |
| resize | onresize | When the visitor resizes the window of the browser |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**JavaScript Objects**

A java Script object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

Creating Objects in JavaScript

* By object literal
* By creating instance of Object directly (using new keyword)
* JavaScript Object by object literal

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

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

**Example:**

<script>

emp={id:102,name:"Shyam Kumar",salary:40000}

document.write(emp.id+" "+emp.name+" "+emp.salary);

</script>

2) **By creating instance of Object**

The syntax of creating object directly is given below:

Var objectname=new Object ();

Here, new keyword is used to create object.

Example:

<Script>

Var emp=new Object ();

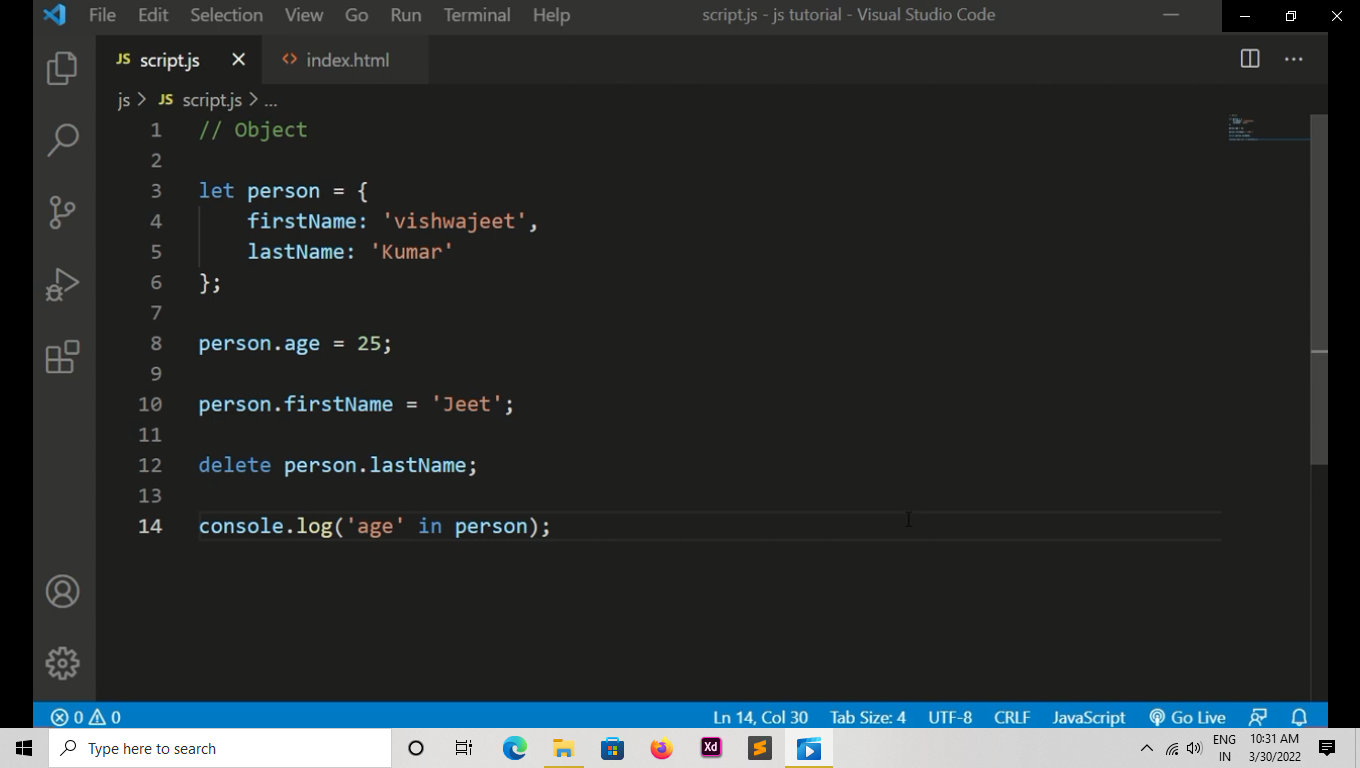
emp.id=101;

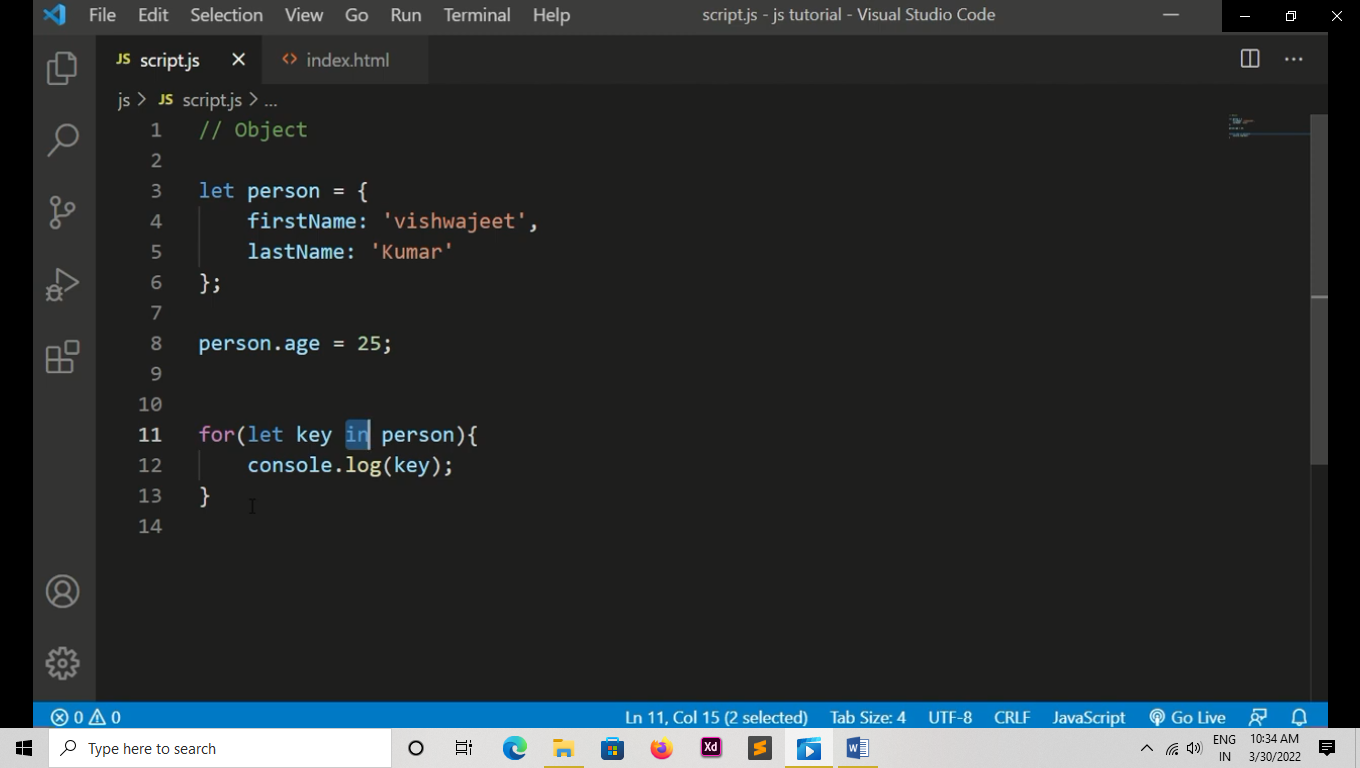
emp.name=”NATASHA";

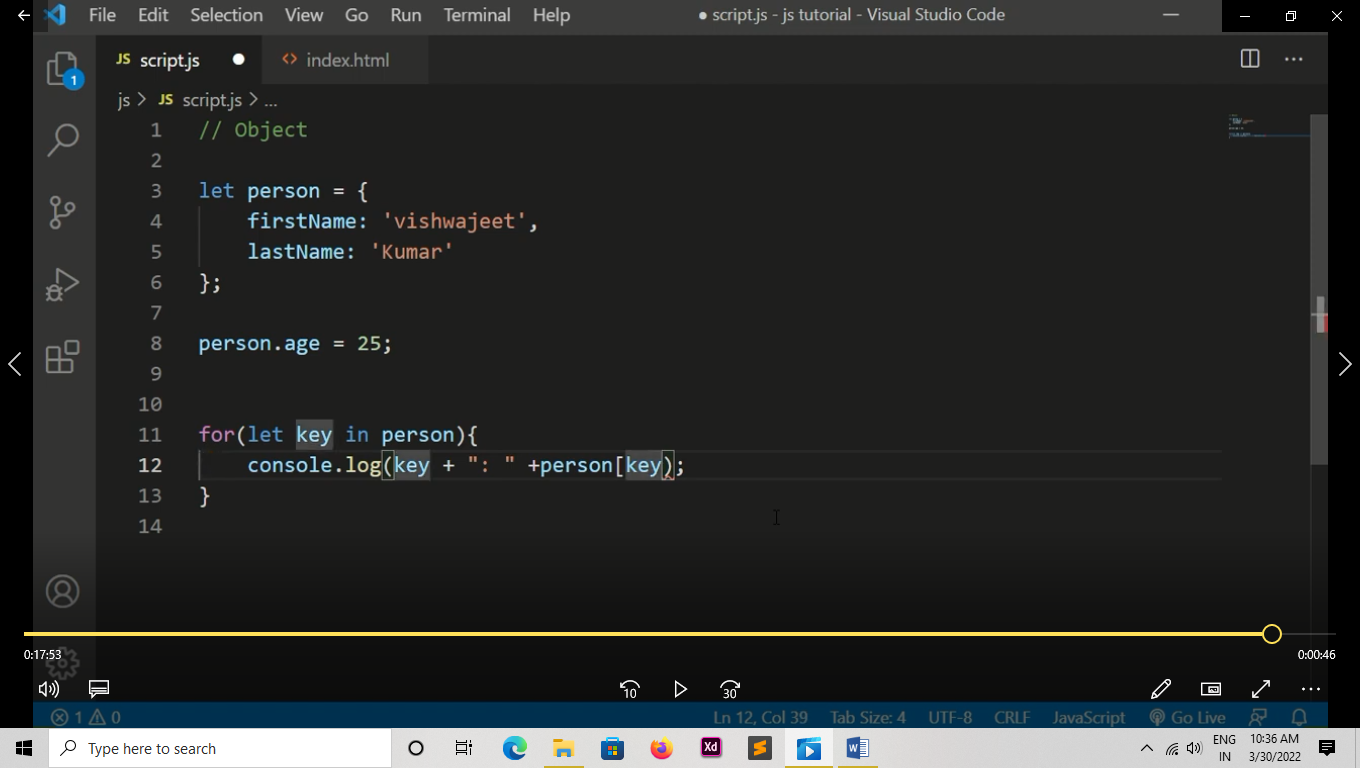
emp.salary=50000;

Document. Write (emp.id+" "+emp.name+" "+emp.salary);

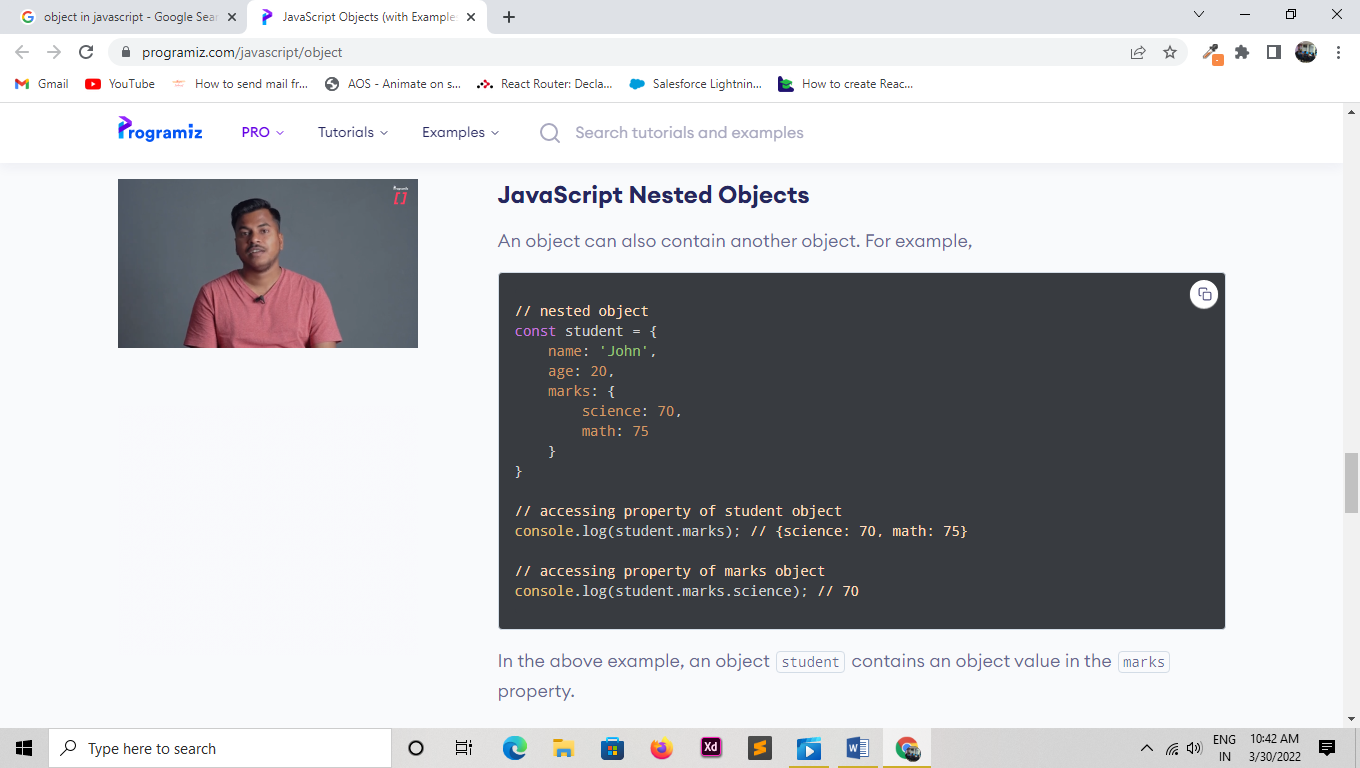
</script>

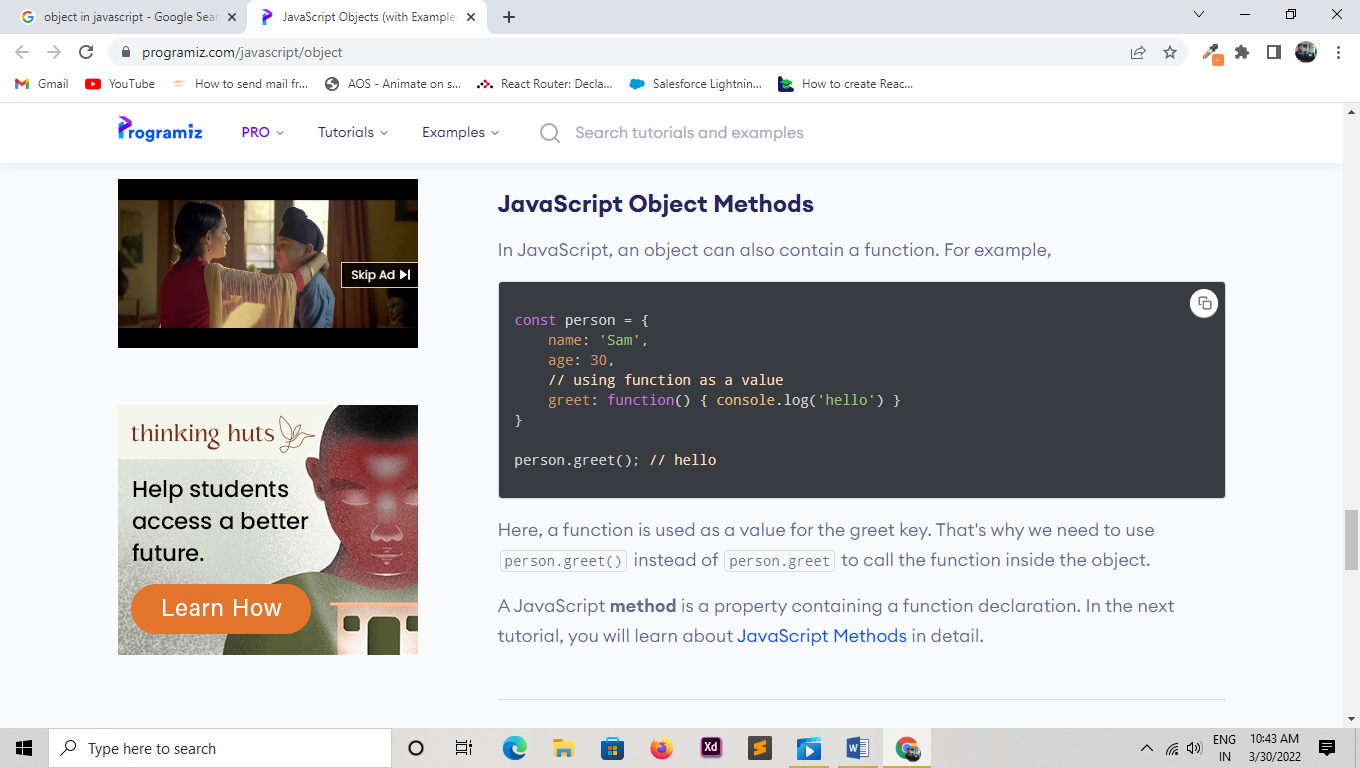




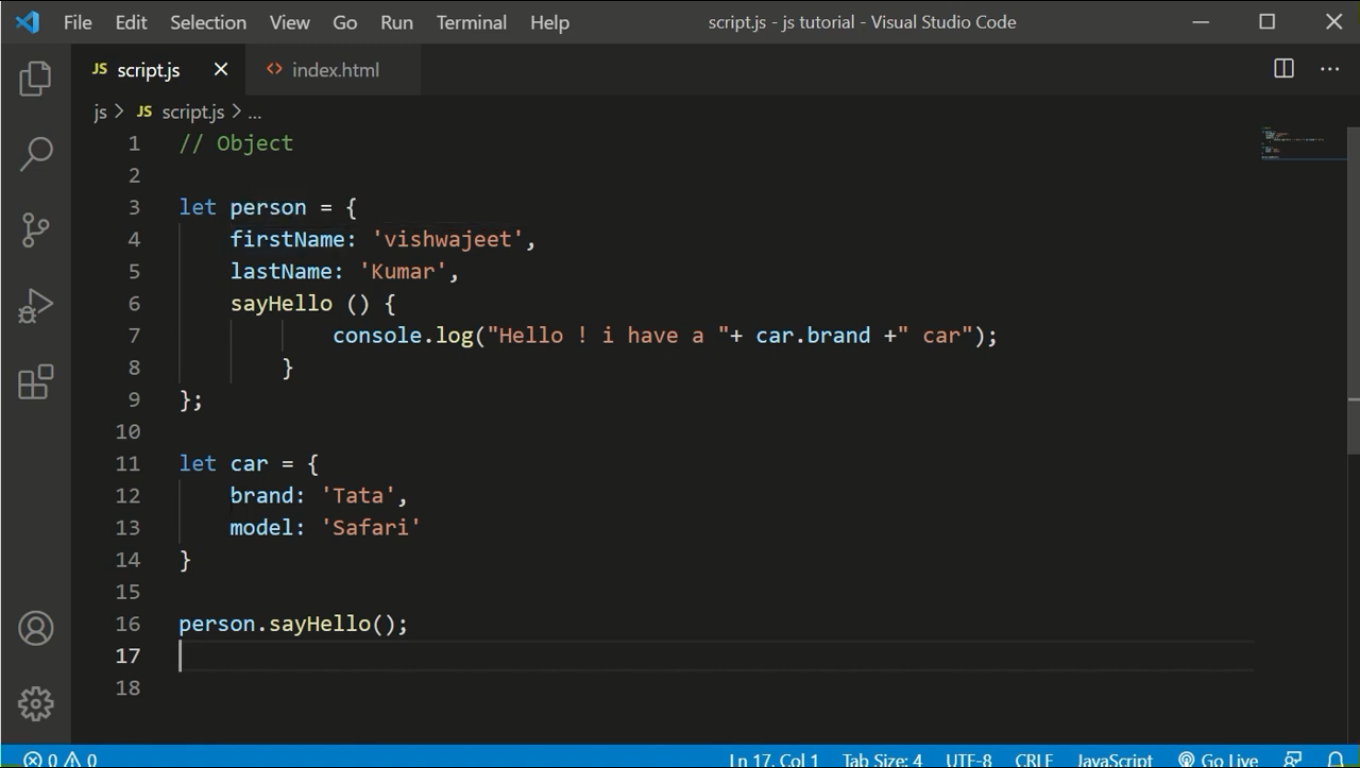


**Nested Object**





**This Key use in Object**



**JavaScript Array**

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

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

1) **JavaScript array literal**

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

Var arrayname=[value1,value2.....valueN];

<script>

var i;

var emp=["Sanjana","kratika","nitya"];

for (i=0;i<emp.length;i++)

{

document.write(emp[i] + "<br/>");

}

</script>

2) **JavaScript Array directly (new keyword)**

The syntax of creating array directly is given below:

Var array name=new Array();

Here, new keyword is used to create instance of array.

**Example:**

<script>

var i;

var emp = new Array();

emp[0] = "Arun";

emp[1] = "Varun";

emp[2] = "John";

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

document.write(emp[i] + "<br>");

}

</script>

Java script array operator

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title></title>

</head>

<body>

<script type="text/javascript">

let emp = ["Akash","Pranav","preshit"," Aditya ", " Kapil ", "Nazim ", " Jitendra ", " Harshit " ];

emp[2] = " Nishant";

emp.push("Preshit");

// Insert element at the end of array

emp.unshift("Abu");

// Insert element at the Begining of array

emp.pop();

// Remove last element

emp.shift();

// remove start element

emp.splice(2,4);

// remove the certain element like (start, end)

for(i=0;i<emp.length;i++)

{

document.write("<br>" + emp[i]);

}

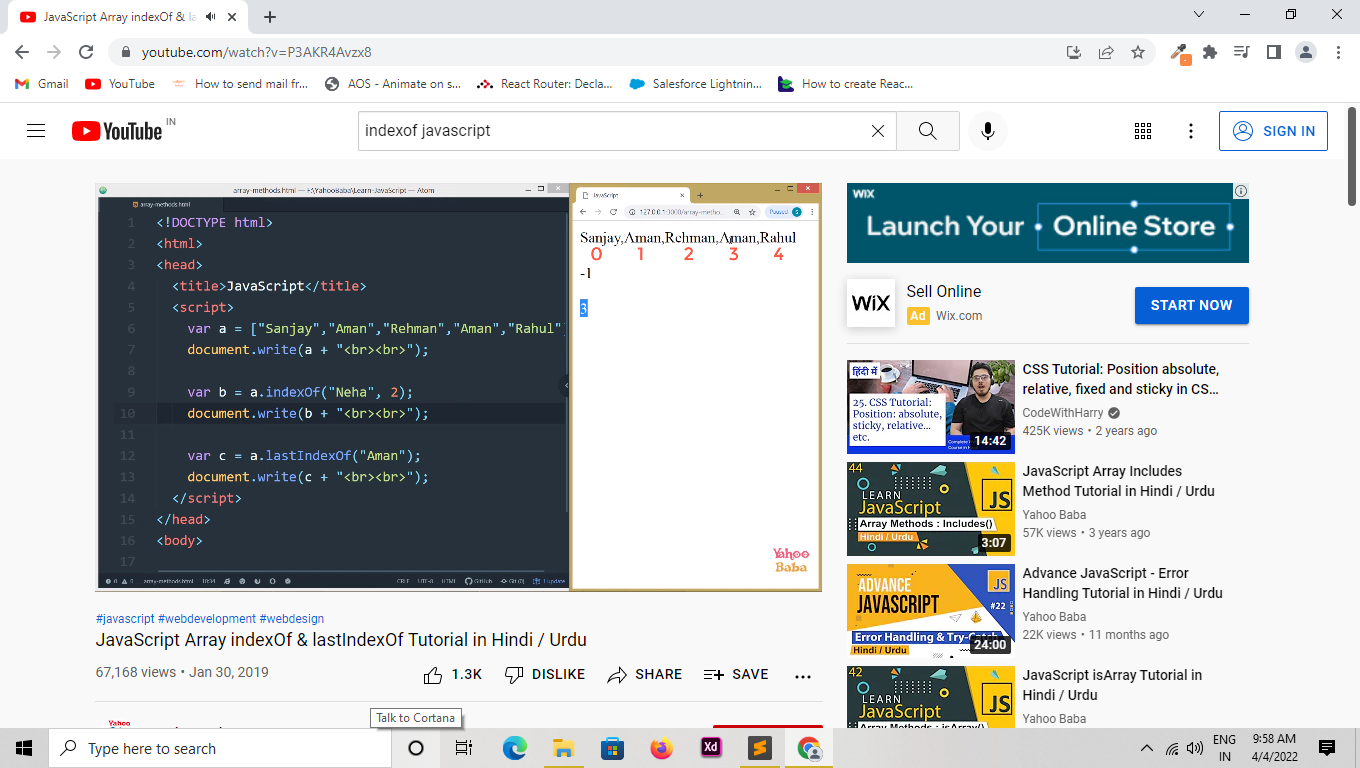
document.write("<br>" + emp.sort[i]);

</script>

</body>

</html>

**INdexOf**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**JavaScript Date Object**

The JavaScript date object can be used to get year, month and day. You can display a timer on the webpage by the help of JavaScript date object.

JavaScript Date Methods

The important methods of date object are as follows:

|  |  |
| --- | --- |
| Method | Description |
| getFullYear() | Return the yr. in 4digit.EX.2017 |
| getMonth() | Return the month in 2 digit.  EX.0to 11. |
| getMonth()+1 | Return th month in 2 digit. (upto 12) |
| getDate() | Returns the date in1-2 digit from 1 to 31 |
| getDay() | Returns the day of week in 1 digit 0 to 6 |
| getHours() | Returns all the element having the given name value |
| getMinutes() | Returns all the element having the given class name. |
| getSeconds() | Returns all the element having the given class name. |
| getMilliseconds | Returns all the element having the given tag name |

**Date example:**

<html>

<body>

Current Date and Time: <span id="txt"></span>

<script>

var today=new Date();

document.getElementById('txt').innerHTML=today;

</script>

</body>

</html>

**JavaScript Current Time Example**

<html>

<body>

Current Time: <span id="txt"></span>

<script>

var today=new Date();

var h=today.getHours();

var m=today.getMinutes();

var s=today.getSeconds();

document.getElementById('txt').innerHTML=h+":"+m+":"+s;

</script>

</body>

</html>

**JavaScript Math Object**

The JavaScript math object provides several constants and methods to perform mathematical operation.

* **Math.sqrt(n)**

The JavaScript math.sqrt(n) method returns the square root of the given number.

**Example**

<!DOCTYPE html>

<html>

<body>

Square Root of 17 is: <span id="p1"></span>

<script>

document.getElementById('p1').innerHTML=Math.sqrt(17);

</script>

</body>

</html>

**OUTPUT:** Square Root of 17 is: 4.123105625617661

**Math.random()**

The JavaScript Math.random() method returns the random number between 0 to 1.

**Example:**

<html>

<body>

Random Number is: <span id="p2"></span>

<script>

document.getElementById('p2').innerHTML=Math.random();

</script>

</body>

</html>

**OUTPUT:**Random Number is: 0.7202219589284733

**Math.pow(m,n)**

The JavaScript math.pow(m,n) method returns the m to the power of n that is mn.

**Example:**

<html>

<body>

3 to the power of 4 is: <span id="p3"></span>

<script>

document.getElementById('p3').innerHTML=Math.pow(3,4);

</script>

</body>

</html>

**OUTPUT: 3 to the power of 4 is: 81**

**Math.floor(n)**

The JavaScript math.floor(n) method returns the lowest integer for the given number. For example 3 for 3.7, 5 for 5.9 etc.

**Example:**

<!DOCTYPE html>

<html>

<body>

Floor of 4.6 is: <span id="p4"></span>

<script>

document.getElementById('p4').innerHTML=Math.floor(4.6);

</script>

</body>

</html>

**Math.ceil(n)**

The JavaScript math.ceil(n) method returns the largest integer for the given number. For example 4 for 3.7, 6 for 5.9 etc.

<!DOCTYPE html>

<html>

<body>

Ceil of 4.6 is: <span id="p5"></span>

<script>

document.getElementById('p5').innerHTML=Math.ceil(4.6);

</script>

</body>

</html>

**Math.round(n)**

The JavaScript math.round(n) method returns the rounded integer nearest for the given number. If fractional part is equal or greater than 0.5, it goes to upper value 1 otherwise lower value 0. For example 4 for 3.7, 3 for 3.3, 6 for 5.9 etc.

**Example:**

**<!**DOCTYPE html>

<html>

<body>

Round of 4.3 is: <span id="p6"></span><br>

Round of 4.7 is: <span id="p7"></span>

<script>

document.getElementById('p6').innerHTML=Math.round(4.3);

document.getElementById('p7').innerHTML=Math.round(4.7);

</script>

</body>

</html>

**Output:**

**Round of 4.3 is: 4**

**Round of 4.7 is: 5**

**Math.abs(n)**

The JavaScript math.abs(n) method returns the absolute value for the given number. For example 4 for -4, 6.6 for -6.6 etc.

**Example:**

<!DOCTYPE html>

<html>

<body>

Absolute value of -4 is: <span id="p8"></span>

<script>

document.getElementById('p8').innerHTML=Math.abs(-4);

</script>

</body>

</html>

**Window Object**

The window object represents a window in browser. An object of window is created automatically by the browser.

Window is the object of browser, it is not the object of javascript. The javascript objects are string, array, date etc.

**Methods of window object**

The important methods of window object are as follows:

|  |  |
| --- | --- |
| Method | Description |
| Alert() | Display a alert box containing msg with ok button. |
| Confirm() | displays the confirm dialog box containing message with ok and cancel button. |
| Prompt() | displays a dialog box to get input from the user. |
| Open() | Open the new window |
| Close() | Close the current window |
| setTimeout() | performs action after specified time like calling function, evaluating expressions etc. |

**Example of alert() in javascript**

It displays alert dialog box. It has message and ok button.

<script type="text/javascript">

function abc(){

alert("Hello Alert Box");

}

</script>

<input type="button" value="click" onclick="abc()"/>

**Example of confirm() in javascript**

It displays the confirm dialog box. It has message with ok and cancel buttons.

<script type="text/javascript">

function abc(){

var v= confirm("Are u sure?");

if(v==true){

alert("ok");

}

else{

alert("cancel");

}

}

</script>

<input type="button" value="delete" onclick="abc()"/>

**Example of prompt()**

It displays prompt dialog box for input. It has message and textfield.

<script type="text/javascript">

function msg(){

var name= prompt("Who are you?");

alert("I am "+name);

}

</script>

<input type="button" value="click" onclick="msg()"/>

**Example of open()**

It displays the content in a new window.

<script type="text/javascript">

function msg(){

open("http://www.genextcomputer.com/");

}

</script>

<input type="button" value="genext" onclick="msg()"/>

**Example of setTimeout()**

It performs its task after the given milliseconds.

<script type="text/javascript">

function msg(){

setTimeout(

function(){

alert("Welcome to Javatpoint after 2 seconds")

},2000);

}

</script>

<input type="button" value="click" onclick="msg()"/>

**VALIDATION**

It is important to validate the form submitted by the user because it can have inappropriate values. So validation is must.

The JavaScript provides you the facility the validate the form on the client side so processing will be fast than server-side validation.

In this example, we are going to validate the name and password. The name can’t be empty and password can’t be less than 6 characters long.

**<html>**

**<body>**

**<script>**

**function validateform(){**

**var name=document.myform.name.value;**

**var password=document.myform.password.value;**

**if (name==null || name==""){**

**alert("Name can't be blank");**

**return false;**

**}else if(password.length<6){**

**alert("Password must be at least 6 characters long.");**

**return false;**

**}**

**}**

**</script>**

**<body>**

**<form name="myform" method="post" action="#" onsubmit="return validateform()" >**

**Name: <input type="text" name="name"><br/>**

**Password: <input type="password" name="password"><br/>**

**<input type="submit" value="register">**

**</form>**

**</body>**

**</html>**

**JavaScript Retype Password Validation**

**<!DOCTYPE html>**

**<html>**

**<head>**

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

**function matchpass(){**

**var firstpassword=document.f1.password.value;**

**var secondpassword=document.f1.password2.value;**

**if(firstpassword==secondpassword){**

**return true;**

**}**

**else{**

**alert("password must be same!");**

**return false;**

**}**

**}**

**</script>**

**</head>**

**<body>**

**<form name="f1" action="http://www.javatpoint.com/javascriptpages/valid.jsp" onsubmit="return matchpass()">**

**Password:<input type="password" name="password" /><br/>**

**Re-enter Password:<input type="password" name="password2"/><br/>**

**<input type="submit">**

**</form>**

**</body>**

**</html>**

## JavaScript Accessors (Getters and Setters)

// Create an object:  
const person = {  
  firstName: "Ashwin",  
  lastName: "Kumar",  
  language: "en",  
  get lang() {  
    return this.language;  
  }  
};  
  
// Display data from the object using a getter:  
document.getElementById("demo").innerHTML = person.lang;

const person = {  
  firstName: "John",  
  lastName: "Doe",  
  language: "",  
  set lang(lang) {  
    this.language = lang;  
  }  
};  
  
// Set an object property using a setter:  
person.lang = "en";  
  
// Display data from the object:  
document.getElementById("demo").innerHTML = person.language;

## JavaScript Function or Getter?

// Normal function

const person = {  
  firstName: "John",  
  lastName: "Doe",  
  fullName: function() {  
    return this.firstName + " " + this.lastName;  
  }  
};  
  
// Display data from the object using a method:  
document.getElementById("demo").innerHTML = person.fullName();

// Get Function

const person = {  
  firstName: "John",  
  lastName: "Doe",  
  get fullName() {  
    return this.firstName + " " + this.lastName;  
  }  
};  
  
// Display data from the object using a getter:  
document.getElementById("demo").innerHTML = person.fullName;

**Constructor Object In Javascript**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Object Constructors</h2>

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

<script>

// Constructor function for Person objects

function Person(first, last, age, eye) {

this.firstName = first;

this.lastName = last;

this.age = age;

this.eyeColor = eye;

}

// Create a Person object

const myFather = new Person("crystal", "Genext", 50, "blue");

// Display age

document.getElementById("demo").innerHTML =

"My father is " + myFather.age + ".";

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Object Constructors</h2>

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

<script>

// Constructor function for Person objects

function Person(first, last, age, eye) {

this.firstName = first;

this.lastName = last;

this.age = age;

this.eyeColor = eye;

this.nationality = "English";

}

// Create 2 Person objects

const myFather = new Person("John", "Doe", 50, "blue");

const myMother = new Person("Sally", "Rally", 48, "green");

// Display nationality

document.getElementById("demo").innerHTML =

"My father is " + myFather.nationality + ". My mother is " + myMother.nationality;

</script>

</body>

</html>

## Using the ****prototype**** Property

The JavaScript prototype property allows you to add new properties to object constructors:

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Objects</h2>

<p>The prototype property allows you to add new methods to objects constructors:</p>

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

<script>

function Person(first, last, age, eye) {

this.firstName = first;

this.lastName = last;

this.age = age;

this.eyeColor = eye;

}

Person.prototype.nationality = "English";

const myFather = new Person("John", "Doe", 50, "blue");

document.getElementById("demo").innerHTML =

"The nationality of my father is " + myFather.nationality;

</script>

</body>

</html>

**Dom In java script**

<!DOCTYPE html>

<html>

<body>

<h1>The Document Object</h1>

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

<p>Change the text of the first element with class="example":</p>

<div class="example">Element1</div>

<div class="example">Element2</div>

<script>

const collection = document.getElementsByClassName("example");

collection[0].innerHTML = "Hello World!";

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<style>

.example {

border: 1px solid black;

padding 8px;

}

</style>

</head>

<body>

<h1>The Document Object</h1>

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

<p>Change the background color of all elements with class="example":</p>

<div class="example">

A div with class="example"

</div>

<br>

<div class="example">

A div with class="example"

</div>

<p class="example">

A p element with class="example".

</p>

<p>A <span class="example">span</span> element with class="example".</p>

<script>

const collection = document.getElementsByClassName("example");

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

collection[i].style.backgroundColor = "red";

}

</script>

</body>

</html>

GetelementByTagName

<!DOCTYPE html>

<html>

<body>

<h1>The Document Object</h1>

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

<p>This is a p element</p>

<p>This is also a p element.</p>

<p>This is also a p element.

Change the background color of all p elements in this document:</p>

<script>

const collection = document.getElementsByTagName("P");

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

collection[i].style.backgroundColor = "red";

}

</script>

</body>

</html>

**Create Element**

<!DOCTYPE html>

<html>

<body>

<h1>The Document Object</h1>

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

<p>Create a p element with some text:</p>

<script>

// Create element:

const para = document.createElement("p");

para.innerText = "This is a paragraph.";

// Append to body:

document.body.appendChild(para);

</script>

</body>

</html>

**QuerySelector**

<!DOCTYPE html>

<html>

<body>

<h1>The Document Object</h1>

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

<p>Change the text of the element with id="demo":</p>

<p id="demo">This is a p element with id="demo".</p>

<script>

document.querySelector("#demo").innerHTML = "Hello World!";

</script>

</body>

</html>

**//Create Element and Append Element**

<!DOCTYPE html>

<html>

<body>

<h1>The Element Object</h1>

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

<ul id="myList">

<li>Genext</li>

<li>IICA</li>

</ul>

<p>Click "Append" to append an item to the end of the list:</p>

<button onclick="myFunction()">Append</button>

<script>

function myFunction() {

// Create an "li" node:

const node = document.createElement("li");

// Create a text node:

const textnode = document.createTextNode("Crystal");

// Append the text node to the "li" node:

node.appendChild(textnode);

// Append the "li" node to the list:

document.getElementById("myList").appendChild(node);

}

</script>

</body>

</html>

**Move item one list to other list**

<!DOCTYPE html>

<html>

<body>

<h1>The Element Object</h1>

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

<ul id="myList1">

<li>IICA</li>

<li>RCA</li>

</ul>

<ul id="myList2">

<li>Genext</li>

<li>Crystal</li>

</ul>

<p>Click "Move" to move an item from one list to another:</p>

<button onclick="myFunction()">Move</button>

<script>

function myFunction() {

const node = document.getElementById("myList2").lastElementChild;

document.getElementById("myList1").appendChild(node);

}

</script>

</body>

</html>

**Insert Before**

<!DOCTYPE html>

<html>

<body>

<h1>The Element Object</h1>

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

<ul id="myList">

<li>Genext</li>

<li>Crystal</li>

</ul>

<script>

// Create a "li" element:

const newNode = document.createElement("li");

// Create a text node:

const textNode = document.createTextNode("Rca");

// Append text node to "li" element:

newNode.appendChild(textNode);

// Insert before existing child:

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

list.insertBefore(newNode, list.children[1]);

</script>

</body>

</html>

**Remove child element**

<!DOCTYPE html>

<html>

<body>

<h1>The Element Object</h1>

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

<p>Click "Remove" to remove the first item from the list:</p>

<button onclick="myFunction()">Remove</button>

<ul id="myList">

<li>Genext</li>

<li>IICA</li>

<li>Crystal</li>

</ul>

<script>

function myFunction() {

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

list.removeChild(list.firstElementChild);

}

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

<h1>The Element Object</h1>

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

<p>Click "Remove" to remove the first child (index 0)</p>

<button onclick="myFunction()">Remove</button>

<ul id="myList">

<li>Genext</li>

<li>RCA</li>

<li>IICA</li>

</ul>

<script>

function myFunction() {

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

if (list.hasChildNodes()) {

list.removeChild(list.children[2]);

}

}

</script>

</body>

</html>

**Clone Element**

<!DOCTYPE html>

<html>

<body>

<h1>The Element Object</h1>

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

<button onclick="myFunction()">Copy</button>

<p>Click "Copy" to copy the "demo" element, including all its attributes and child elements, and append it to the document.</p>

<div id="demo" style="border:1px solid black;background-color:pink">

<p style="color:red; padding:10px">A p element</p>

<p style="color:green; padding:10px ">A p element</p>

<p style="color:blue; padding:10px ">A p element</p>

</div>

<script>

function myFunction() {

const node = document.getElementById("demo");

const clone = node.cloneNode(true);

document.body.appendChild(clone);

}

</script>

</body>

</html>

# HTML DOM Element replaceChild()

<!DOCTYPE html>

<html>

<body>

<h1>The Element Object</h1>

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

<ul id="myList">

<li>Genext</li>

<li>Crystal</li>

<li>IICA</li>

</ul>

<p>Click "Replace" to replace the first item in the the list.</p>

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

<p>This example replaces the Text node "Coffee" with a Text node "Water". Replacing the entire li element is also an alternative.</p>

<script>

function myFunction() {

// Select first child element:

const element = document.getElementById("myList").children[0];

// Create a new text node:

const newNode = document.createTextNode("Water");

// Replace the text node:

element.replaceChild(newNode, element.childNodes[0]);

}

</script>

</body>

</html>

# HTML DOM Element insertAdjacentElement()

<!DOCTYPE html>

<html>

<body>

<h1>The Element Object</h1>

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

<button onclick="myFunction()">Move</button>

<p>Click "Move" to insert the red span after the header.</p>

<span id="mySpan" style="color:red">My span</span>

<h2 id="myH2">My Header</h2>

<script>

function myFunction() {

const span = document.getElementById("mySpan");

const h2 = document.getElementById("myH2");

h2.insertAdjacentElement("afterend", span);

}

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

<h1>The Element Object</h1>

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

<button onclick="myFunction()">Move</button>

<p>Click "Move" to insert the red span before the header:</p>

<h2 id="myH2">My Header</h2>

<p><span style="color:red">My span</span></p>

<script>

function myFunction() {

const span = document.getElementsByTagName("span")[0];

const h2 = document.getElementById("myH2");

h2.insertAdjacentElement("beforebegin", span);

}

</script>

</body>

</html>

**Dom Events**

<!DOCTYPE html>

<html>

<head>

<title>JS Mouse Events - Button Demo</title>

</head>

<body>

<button id="btn">Click me with any mouse button: left, right, middle, ...</button>

<p id="message"></p>

<script>

let btn = document.querySelector('#btn');

// disable context menu when right-mouse clicked

btn.addEventListener('contextmenu', (e) => {

e.preventDefault();

});

// show the mouse event message

btn.addEventListener('mouseup', (e) => {

let msg = document.querySelector('#message');

switch (e.button) {

case 0:

msg.textContent = 'Left mouse button clicked.';

break;

case 1:

msg.textContent = 'Middle mouse button clicked.';

break;

case 2:

msg.textContent = 'Right mouse button clicked.';

break;

default:

msg.textContent = `Unknown mouse button code: ${event.button}`;

}

});

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript HTML Events</h2>

<h2 onclick="this.innerHTML='Ooops!'">Click on this text!</h2>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

<h1 id="h4" onmouseover="handle(event)">Mouse Over Here !!</h1>

<h1 id="h2" onclick="handle(event)">Click Here !!</h1>

<h1 id="h3" ondblclick="handle(event)">Double Click Here !!</h1>

<script type="text/javascript">

function handle(e) {

alert(e.type);

}

</script>

</body>

</html>

<https://www.w3schools.com/jsref/obj_mouseevent.asp>

online show all events.

## Adding the addEventListener Method

You can also add event listeners using a method called addEventListener. The **EventTarget.addEventListener()** method adds the specified EventListener-compatible object to the list of event listeners for the specified event type on the **EventTarget** on which it is called.

<!DOCTYPE html>

<html>

<body>

<button id="eventBtn">Try it</button>

<p id="display"> </p>

<script>

document.getElementById("eventBtn").addEventListener("click", myFunction);

function myFunction() {

document.getElementById("display").innerHTML = "Its Worked !!";

}

</script>

</body>

</html>

**Remove eventlistner**

<!DOCTYPE html>

<html>

<head>

<meta charset=utf-8 />

<title>Events in JavaScript: Removing event listeners</title>

</head>

<body>

<button id="element">Click Me</button>

<script>

var element = document.getElementById('element');

function callback() {

alert('Hello once');

element.removeEventListener('click', callback);

}

// Add listener

element.addEventListener('click', callback);

</script>

</body>

</html>

**Event with anonymous function**

<!DOCTYPE html>

<html>

<head>

<meta charset=utf-8 />

<title>Events in JavaScript: Removing event listeners</title>

</head>

<body>

<button id="element">Click Me</button>

<script>

var user = {

firstname: ‘genext’,

greeting: function(){

alert('My name is ' + this.firstname);

}

};

// Call the method with the correct

// context inside an anonymouse function

element.addEventListener('click', function() {

user.greeting();

});

</script>

</body>

</html>

**Bind Function**

<!DOCTYPE html>

<html>

<head>

<meta charset=utf-8 />

<title>Events in JavaScript: Preserving callback context using bind</title>

</head>

<body>

<button id="element">Click Me</button>

<script>

var element = document.getElementById('element');

var user = {

firstname: 'Bob',

greeting: function(){

alert('My name is ' + this.firstname);

}

};

// Overwrite the original function with

// a new function with its execution

// context 'bound' to the user object

user.greeting = user.greeting.bind(user);

// Add the bound function as the callback

element.addEventListener('click', user.greeting);

</script>

</body>

</html>

**Page Load event**

<!DOCTYPE html>

<html>

<body onload="myFunction()">

<h1>Hello World!</h1>

<script>

function myFunction() {

alert("Page is loaded");

}

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

<img src="pic.jpg" onload="loadImage()" width="100" height="132">

<script>

function loadImage() {

alert("Image is loaded");

}

</script>

</body>

</html>

**Scroll Event**

<!DOCTYPE html>

<html>

<head>

<style>

div {

border: 1px solid black;

width: 200px;

height: 100px;

overflow: scroll;

}

</style>

</head>

<body>

<h1>The onscroll Event</h1>

<p>Try the scrollbar in div.</p>

<div onscroll="myFunction()">In my younger and more vulnerable years my father gave me some advice that I've been turning over in my mind ever since.

<br><br>

'Whenever you feel like criticizing anyone,' he told me, just remember that all the people in this world haven't had the advantages that you've had.'</div>

<p>Scrolled <span id="demo">0</span> times.</p>

<script>

let x = 0;

function myFunction() {

document.getElementById("demo").innerHTML = x += 1;

}

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<style>

.test {

background-color: yellow;

}

</style>

<body style="font-size:20px;height:1500px">

<h1>The onscroll Event</h1>

<p id="myP" style="position:fixed">

If you scroll 50 pixels down from the top of this page, the class "test" is added to this paragraph.

Scroll up again to remove the class.</p>

<script>

window.onscroll = function() {myFunction()};

function myFunction() {

if (document.documentElement.scrollTop > 50) {

document.getElementById("myP").className = "test";

} else {

document.getElementById("myP").className = "";

}

}

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<style>

.slideUp {

animation-name: slideUp;

animation-name: slideUp;

animation-duration: 1s;

animation-duration: 1s;

visibility: visible;

}

@keyframes slideUp {

0% {

opacity: 0;

transform: translateY(70%);

}

100% {

opacity: 1;

transform: translateY(0%);

}

}

body {height:1500px;}

.col-1 {float:left}

.col-2 {float:left;padding-left:25px;}

img {width:180px;height:100px;visibility:hidden;}

hr {margin-top:400px;}

</style>

</head>

<body>

<h1>The onscroll Event</h1>

<p>Scroll down this page.</p>

<p>When you have scrolled 350px from the top, an image will slide in.</p>

<hr>

<div class="col-1">

<img id="myImg" src="img\_pulpit.jpg" width="304" height="228">

</div>

<div class="col-2">

Just some text..

</div>

<script>

window.onscroll = function() {myFunction()};

function myFunction() {

if (document.documentElement.scrollTop > 350) {

document.getElementById("myImg").className = "slideUp";

}

}

</script>

</body>

</html>

**See Input Events Online**

<https://www.w3schools.com/js/js_events_examples.asp>

## Bubbling

The bubbling principle is simple.

**When an event happens on an element, it first runs the handlers on it, then on its parent, then all the way up on other ancestors.**

<!doctype html>

<body>

<style>

  body \*

{

    margin: 10px;

    border: 1px solid blue;

  }

</style>

<form onclick="alert('form')">FORM

  <div onclick="alert('div')">DIV

    <p onclick="alert('p')">P</p>

  </div>

</form>

</body>

## Capturing

<!doctype html>

<body>

<style>

  body \* {

    margin: 10px;

    border: 1px solid blue;

  }

</style>

<form>FORM

  <div>DIV

    <p>P</p>

  </div>

</form>

<script>

  for(let elem of document.querySelectorAll('\*')) {

    elem.addEventListener("click", e => alert(`Capturing: ${elem.tagName}`), true);

    elem.addEventListener("click", e => alert(`Bubbling: ${elem.tagName}`));

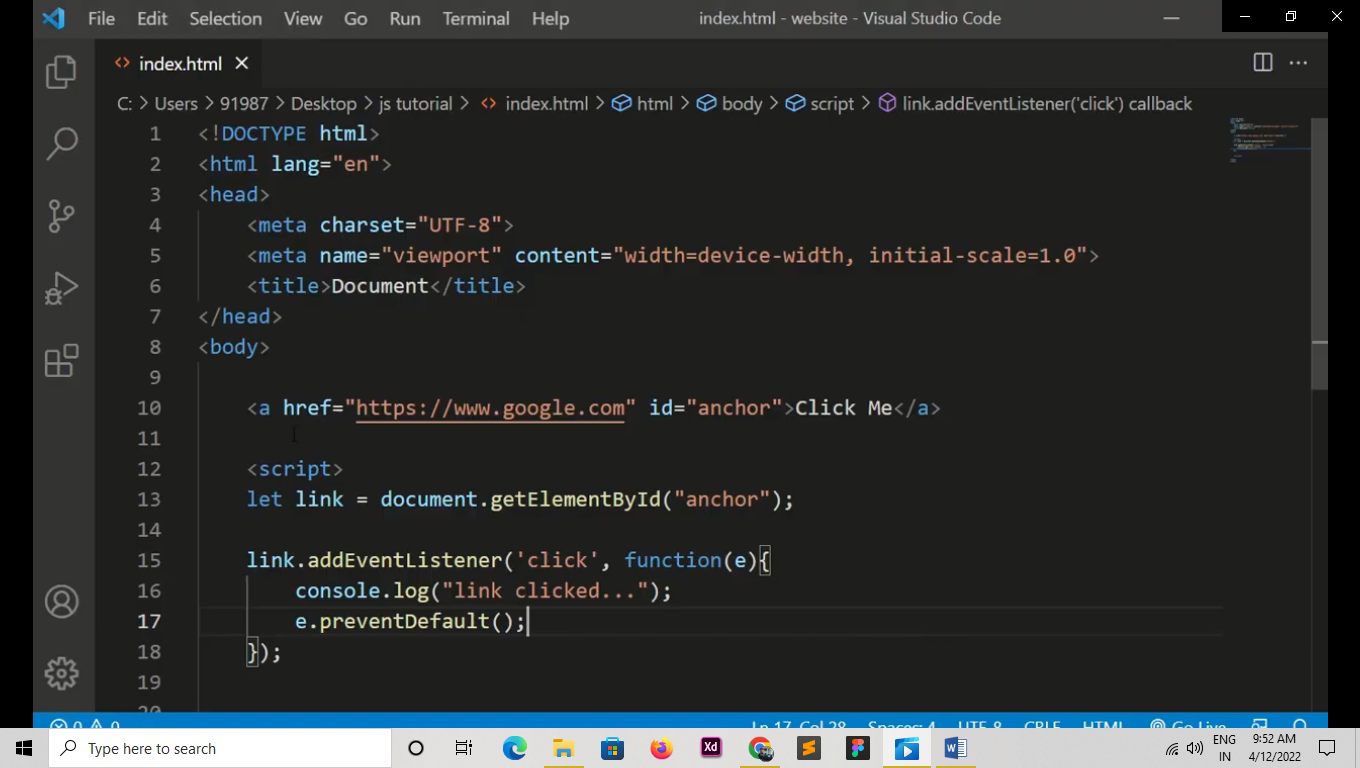
  }

</script>

</body>

**preventDefault**

The preventDefault() method cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur.



<!DOCTYPE html>

<html>

<body>

Try to check this box: <input type="checkbox" id="myCheckbox">

<p>Toggling a checkbox is the default action of clicking on a checkbox. The preventDefault() method prevents this from happening.</p>

<script>

document.getElementById("myCheckbox").addEventListener("click", function(event){

event.preventDefault()

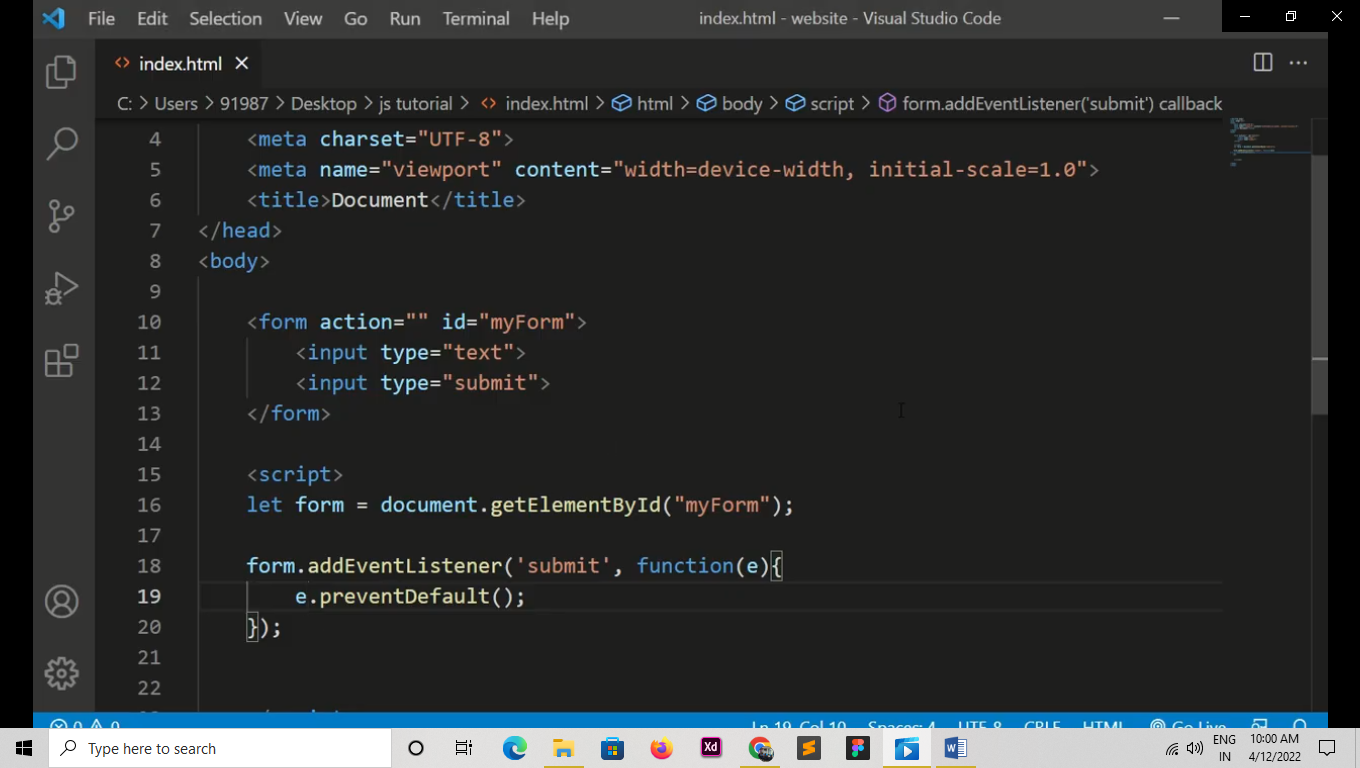
});

</script>

</body>

</html>

**Form Not Submit Bacause you we write preventDefault:**



**Default parameter**

<!DOCTYPE html>

<html>

<body>

<p>Setting a default value to a function parameter (y = 2).</p>

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

<script>

function myFunction(x, y = 2) {

return x \* y;

}

document.getElementById("demo").innerHTML = myFunction(4);

</script>

</body>

</html>

**ES6 (Ecma Script)**

**Rest parameter**

**Rest parameter** is an improved way to handle function parameter, allowing us to more easily handle various input as parameters in a function. The rest parameter syntax allows us to represent an indefinite number of arguments as an array. With the help of a rest parameter a function can be called with any number of arguments, no matter how it was defined. Rest parameter is added in ES2015 or ES6 which improved the ability to handle parameter.

// es6 rest parameter

function fun(...input){

let sum = 0;

for(let i of input){

sum+=i;

}

return sum;

}

console.log(fun(1,2)); //3

console.log(fun(1,2,3)); //6

console.log(fun(1,2,3,4,5)); //15

**Spread operator** :- allows an iterable to expand in places where 0+ arguments are expected. It is mostly used in the variable array where there is more than 1 values are expected. It allows us the privilege to obtain a list of parameters from an array. Syntax of Spread operator is same as [Rest parameter](https://www.geeksforgeeks.org/javascript-rest-operator/) but it works completely opposite of it.

// normal array concat() method

let arr = [1,2,3];

let arr2 = [4,5];

arr = arr.concat(arr2);

console.log(arr); // [ 1, 2, 3, 4, 5 ]

// spread operator doing the concat job

let arr = [1,2,3];

let arr2 = [4,5];

arr = [...arr,...arr2];

console.log(arr); // [ 1, 2, 3, 4, 5 ]

**For of**

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Statements</h1>

<h2>The for...of Loop</h2>

<p>Iterate (loop) over the values of an array:</p>

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

<script>

let text = "";

const branch = [‘crystal’, ‘genext’, ‘iica’];

for (let x of branch) {

text += x + " ";

}

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

</script>

</body>

</html>

**Template Literals** use back-ticks (``) rather than the quotes ("") to define a string: **just like pre tag in html**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Template Literals</h2>

<p>Template literals use back-ticks (``) to define a string:</p>

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

<p>Template literals are not supported in Internet Explorer.</p>

<script>

let text = `Hello world!`;

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

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Template Literals</h2>

<p>Template literals allows multiline strings:</p>

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

<p>Template literals are not supported in Internet Explorer.</p>

<script>

let text =

`web deign

And

Development institute

In indore.`;

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

</script>

</body>

</html>

**Destructuring** Assignment is a JavaScript expression that allows to **unpack values** from arrays, or properties from objects, into distinct variables data can be extracted from arrays, objects, nested objects and **assigning to variables**. In Destructuring Assignment on the left-hand side defined that which value should be unpacked from the sourced variable.

<script>

var names = ["alpha", "beta", "gamma", "delta"];

var [firstName, secondName] = names;

console.log(firstName); //"alpha"

console.log(secondName); //"beta"

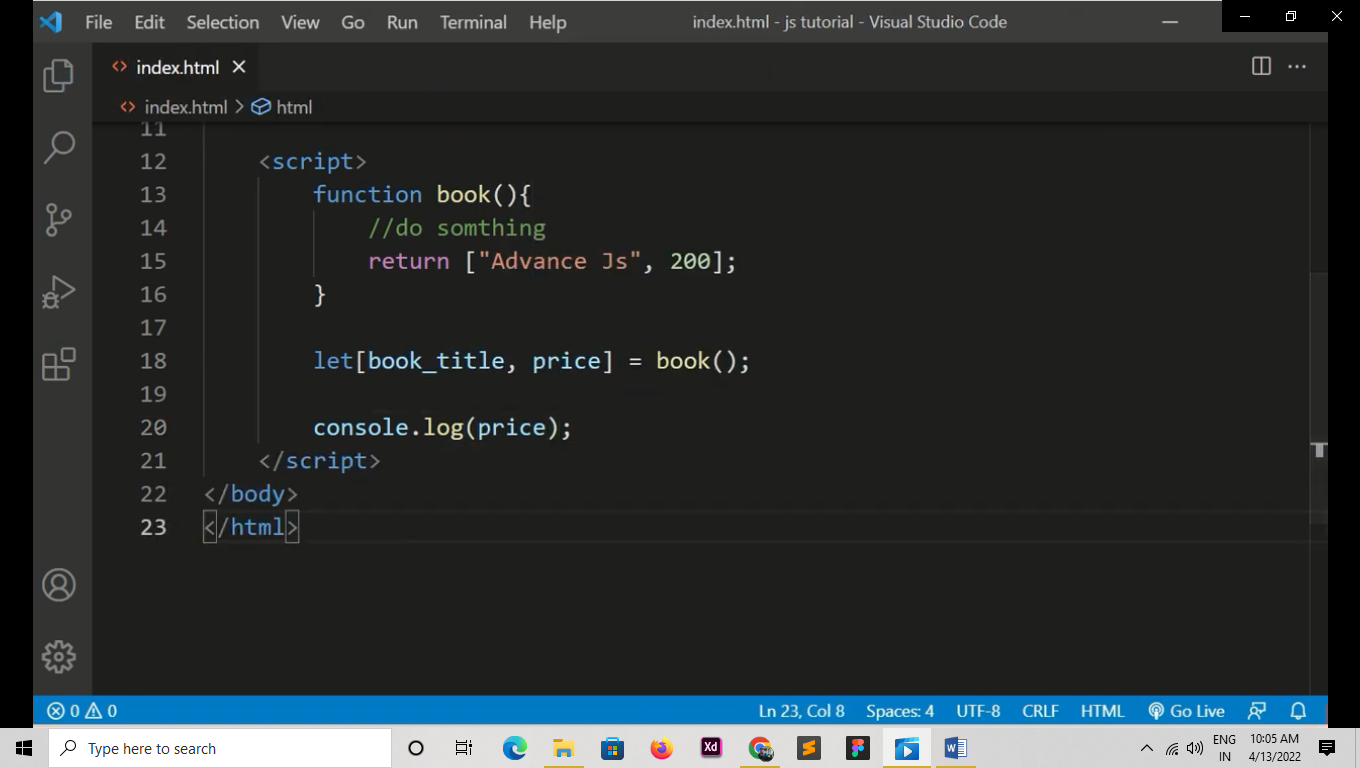
//Both of the procedure are same

var [firstName, secondName] = ["alpha", "beta", "gamma", "delta"];

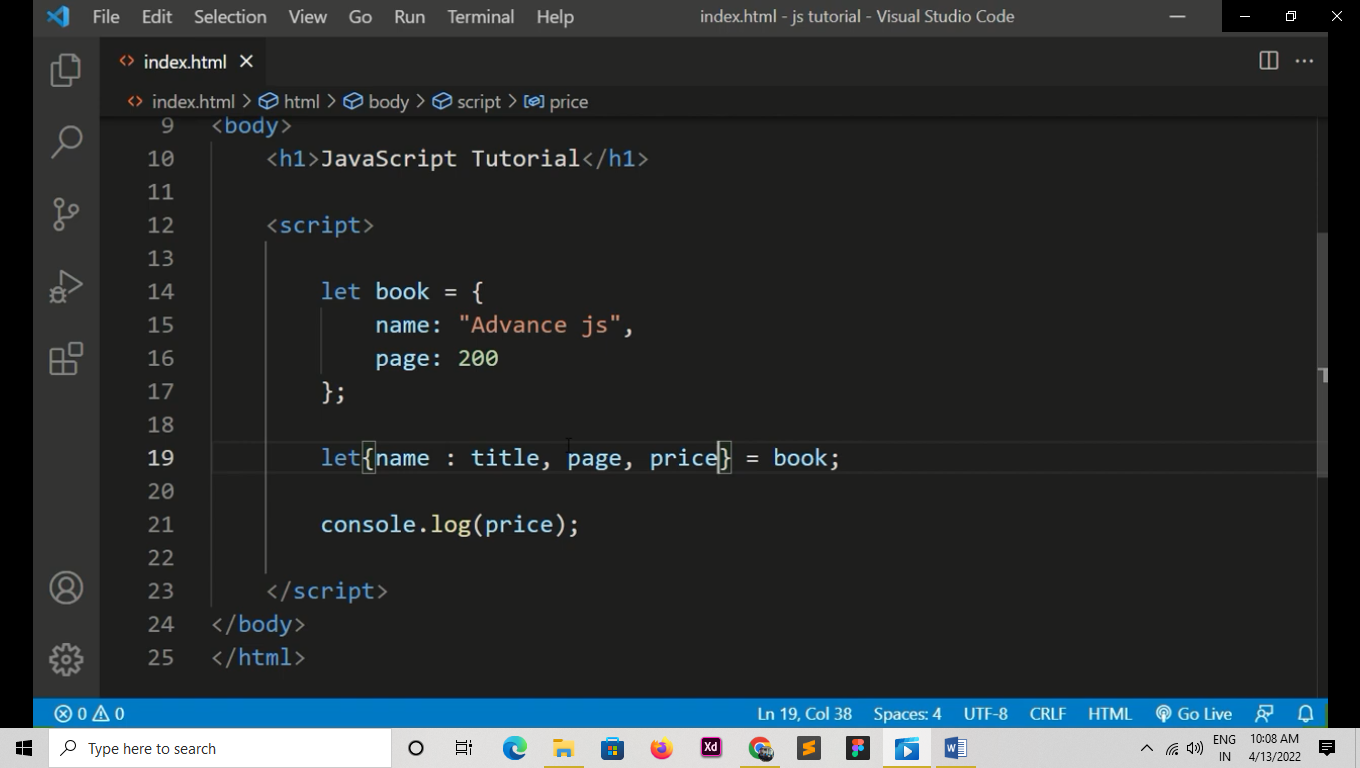
console.log(firstName); //"alpha"

console.log(secondName); //"beta

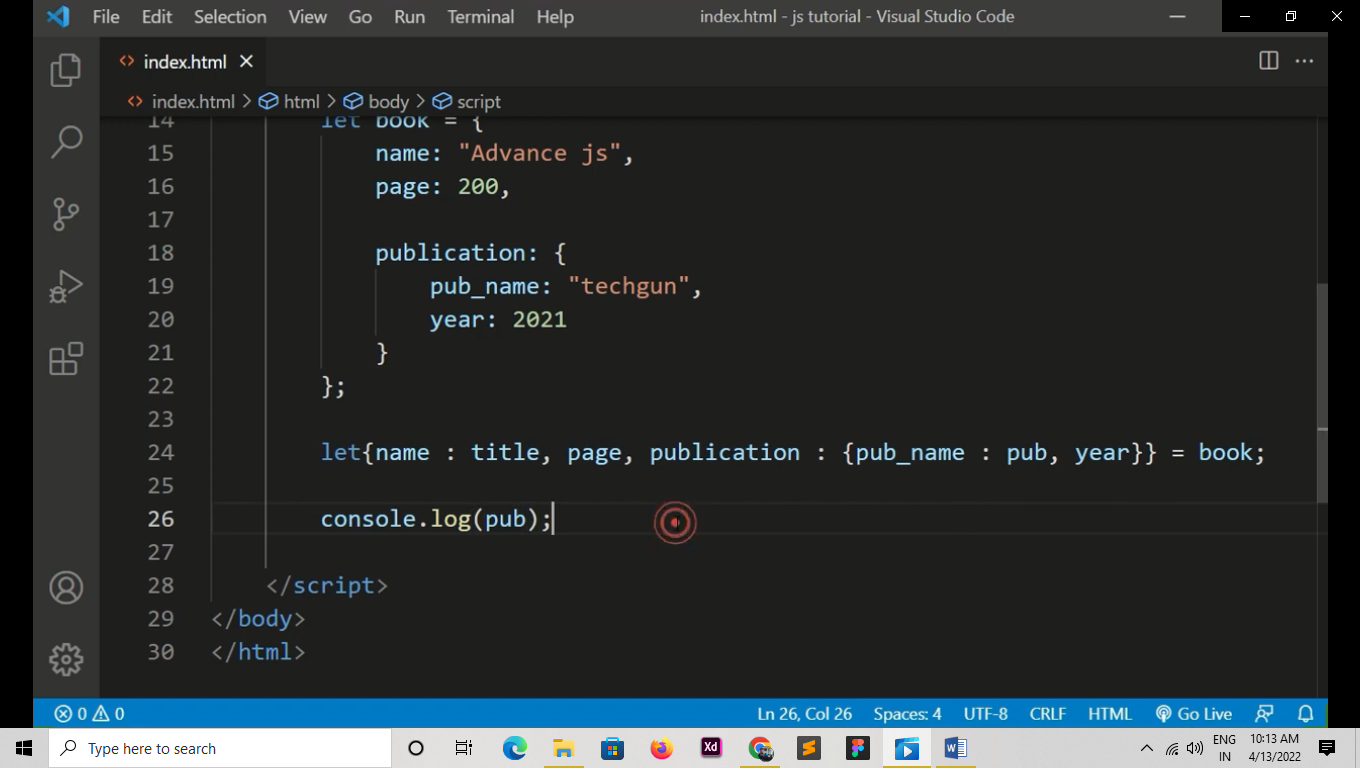
</script>



**Object destruturing**



**Nested Object destructuring:-**



# Introduction to Object Oriented Programming in JavaScript

“How Object-Oriented Programming is implemented in JavaScript? How they differ from other languages? Can you implement Inheritance in JavaScript and so on…”

There are certain features or mechanisms which makes a Language Object-Oriented like:

* **Object**
* **Classes**
* **Encapsulation**
* **Inheritance**

The characteristics of an Object are called as Property, in Object-Oriented Programming and the actions are called methods. An Object is an **instance** of a class. Objects are everywhere in JavaScript almost every element is an Object whether it is a function, array, and string.

//Defining object

let person = {

first\_name:'Mukul',

last\_name: 'Latiyan',

//method

getFunction : function(){

return (`The name of the person is

${person.first\_name} ${person.last\_name}`)

},

//object within object

phone\_number : {

mobile:'12345',

landline:'6789'

}

}

console.log(person.getFunction());

console.log(person.phone\_number.landline);

* Using an **Object Constructor:**

//using a constructor

function person(first\_name,last\_name){

this.first\_name = first\_name;

this.last\_name = last\_name;

}

//creating new instances of person object

let person1 = new person('Mukul','Latiyan');

let person2 = new person('Rahul','Avasthi');

console.log(person1.first\_name);

console.log(`${person2.first\_name} ${person2.last\_name}`);

**Classes**– Classes are **blueprint** of an Object. A class can have many Object, because class is a **template** while Object are **instances** of the class or the concrete implementation.   
Before we move further into implementation, we should know unlike other Object Oriented Language there is **no classes in JavaScript** we have only Object. To be more precise, JavaScript is a prototype based object oriented language, which means it doesn’t have classes rather it define behaviors using constructor function and then reuse it using the prototype.

// Defining class using es6

class Vehicle {

constructor(name, maker, engine) {

this.name = name;

this.maker = maker;

this.engine = engine;

}

getDetails(){

return (`The name of the bike is ${this.name}.`)

}

}

// Making object with the help of the constructor

let bike1 = new Vehicle('Hayabusa', 'Suzuki', '1340cc');

let bike2 = new Vehicle('Ninja', 'Kawasaki', '998cc');

console.log(bike1.name); // Hayabusa

console.log(bike2.maker); // Kawasaki

console.log(bike1.getDetails());

**Encapsulation** – The process of **wrapping property and function**within a **single unit** is known as encapsulation.

//encapsulation example

class person{

constructor(name,id){

this.name = name;

this.id = id;

}

add\_Address(add){

this.add = add;

}

getDetails(){

console.log(`Name is ${this.name},Address is: ${this.add}`);

}

}

let person1 = new person('Mukul',21);

person1.add\_Address('Delhi');

person1.getDetails();

**Abstraction**  :- Sometimes encapsulation refers to **hiding of data** or **data Abstraction** which means representing essential features hiding the background detail. Most of the OOP languages provide access modifiers to restrict the scope of a variable, but their are no such access modifiers in JavaScript but their are certain way by which we can restrict the scope of variable within the Class/Object.

// Abstraction example

function person(fname,lname){

let firstname = fname;

let lastname = lname;

let getDetails\_noaccess = function(){

return (`First name is: ${firstname} Last

name is: ${lastname}`);

}

this.getDetails\_access = function(){

return (`First name is: ${firstname}, Last

name is: ${lastname}`);

}

}

let person1 = new person('Mukul','Latiyan');

console.log(person1.firstname);

console.log(person1.getDetails\_noaccess);

console.log(person1.getDetails\_access());

**Inheritance**– It is a concept in which some property and methods of an Object is being used by another Object. Unlike most of the OOP languages where classes inherit classes, JavaScript Object inherits Object i.e. certain features (property and methods)of one object can be reused by other Objects.

//Inheritance example

class person{

constructor(name){

this.name = name;

}

//method to return the string

toString(){

return (`Name of person: ${this.name}`);

}

}

class student extends person{

constructor(name,id){

//super keyword to for calling above class constructor

super(name);

this.id = id;

}

toString(){

return (`${super.toString()},Student ID: ${this.id}`);

}

}

let student1 = new student('Mukul',22);

console.log(student1.toString());