JavaScript

Introduction

- JavaScript (JS) is a lightweight interpreted programming language used to develop web pages.
- It is most well-known as scripting language for web pages and many non-browser environments such as node.js also use it.
- Scripting language is a programming language that is used to manipulate, customize, and automate the facilities of an existing system.
- JS can change HTML Content, HTML Attribute values, HTML styles, hide and display HTML Elements.
- JS is **case sensitive.**
- JS and Java are completely different languages, both in concept and design.

History

- JS was invented by Brendan Eich in 1995.
- It was developed for NetScape 2 and became the ECMA-262 standard in 1997.
- JavaScript is standardized at ECMA (European Computer Manufacturers Association) to deliver a standardized, international programming language based on JavaScript.
- This standardized version of JavaScript, called ECMAScript, behaves the same way in all applications that support the standard.

JS Where To

- In HTML JS code is inserted between **<script> </script>** tag.

- We can place the **<script> </script>** tag under **<head> </head>** tag or **<body> </body>** tag.
- We can include an external JS file like this: **<script src = myscript.js/>**

JS Output

JS output data can be displayed in following different ways:

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

Example1:

```
<body>
    <h1>Welcome to My JS notes.</h1>
    <p id="js"></p>
    <script>
        document.getElementById("js").innerHTML = "Hello from innerHTML"
    </script>
</body>
```

Output:

Welcome to My JS notes.

Hello from innerHTML

Example2:

```
<body>
    <h4>Welcome to My JS notes.</h4>
    <script>
        document.write("Hello from document.write()")
    </script>
</body>
```

Output:

Welcome to My JS notes.

Hello from document.write()

Note:

Using **document.write()** after an html file is loaded will clear all the existing html.

Example2.1:

```
<body>
    <p>welcome to My <i>document.write()</i></p><br/>
    <p>Using <i>document.write()</i> after an html file is loaded
        <br> will delete all exsiting HTML.</p>
    <button onclick="document.write('Cleared everything')" value="Clear">Clear</button>
</body>
```

Output

Before 'Clear'

welcome to My *document.write()*

Using *document.write()* after an html file is loaded
will delete all exsiting HTML.

Clear

After 'Clear'

Cleared everything

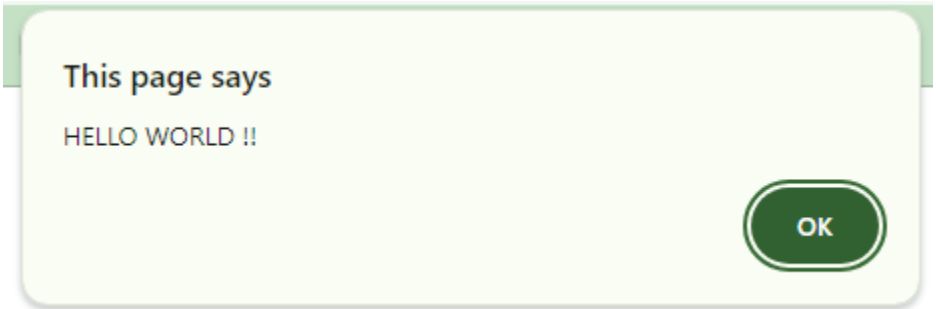
Example3:

```
<body>
    <p>welcome to My <i>window.alert()</i></p><br/>
    <script>
        window.alert("HELLO WORLD !!")
    </script>
</body>
```

Note:

- ➔ Whenever a page is loaded the content under **window.alert()** or **alert()** will be displayed first within a alert box, followed by this the rest content will appear.
- ➔ **window.alert()** and **alert()** works same**.**

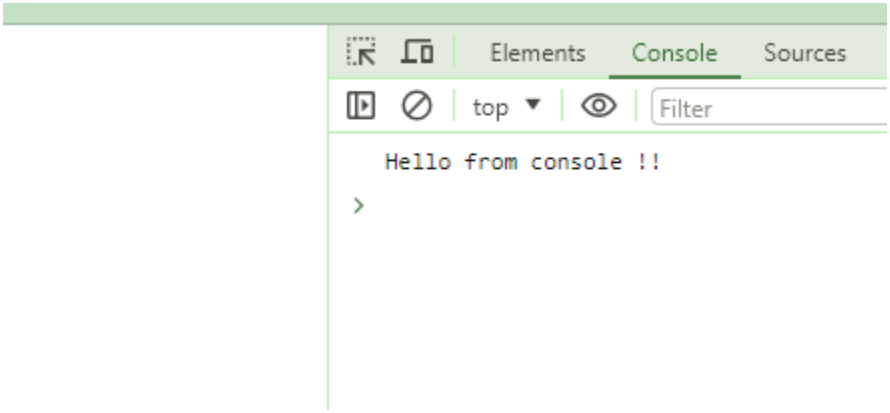
Output:



Example4:

```
<body>
    <script>
        console.log("Hello from console !!")
    </script>
</body>
```

Output:



Note:

- **console.log ()** is mostly used for debugging purposes.
- To view the output of cosole.log:
- -> browser -> right click -> inspect -> console

JS Variables:

- Variables are used to store data.
- JS Variables can be declared in 4 ways:
    1. Automatically
    2. Using var
    3. Using let
    4. Using const

Automatically:

➔ These variables are undeclared variables.
➔ They are automatically declared when first used.
➔ **Example:** id = 30959;

Using var:

➔ The **var** keyword was used in JS from 1995 to 2015.
➔ In 2015 the keywords **let** and **const** were added.
➔ The **var** is mostly preferred to use in code written for old browsers.
➔ Variables declared using **var** keyword has **Global Scope.**
➔ Variables declared using **var** keyword have no **block scope.**
➔ Variables declared using **var** keyword can be **redeclared**.
➔ **Example:** var id = 30959, age = 21;

Using let:

- ➔ The **let** keyword is used when we cannot use const.
- ➔ Variables declared with **let** keyword have **block scope** and cannot be **redeclared & reassigned** in the **same scope.**
- ➔ These variables must be declared before use.
- ➔ **Example:** let name = “kalyan”;

Using const:

- ➔ The **const** keyword in JS is used if we want the values not to be changed.
- ➔ The **const** keyword in JS is used if we want the type of variables to be fixed.
- ➔ Variables declared with **const** keyword have **block scope** and cannot be **redeclared & reassigned** in the **same scope.**
- ➔ These variables must be declared before use.
- ➔ **Example:** const id = 30959;

JS Operators:

- ➔ There are different types of operators in JS:
- ➔ **Arithmetic: +, -, *, **, /, %, ++, --**
- ➔ **Assignment: =, +=, -=, *=, **=, /=, %=**
- ➔ **Comparison: ==, ===, !=, !==, >, <, >=, <=**
- ➔ **String: + (concatenation), +=**
- ➔ **Logical: &&, ||, !**
- ➔ **Bitwise: &, |, ~, ^, <<, >>, >>> (unsigned right shift)**
- ➔ **Ternary: ?**
- ➔ **Type: typeof, instanceof**

Note:

- ➔ The **===** comparison operator checks the equal value and equal type.
- ➔ The **instanceof** operator returns true if an object is an instance of an object type and the **** is the **exponential** operator.

JS Datatypes:

- ➔ Datatypes define the type of data that can be stored in a variable.
- ➔ JS has 8 Datatypes:
  1. **String**
  2. **Number**
  3. **Boolean**
  4. **BigInt**
  5. **Undefined**
  6. **Null**
  7. **Symbol**
  8. **Object** **->** can contain an **object, array, date.**

String:

- ➔ In JS to represent a string variable we can use both single quotes **(‘ ‘)** as well as double quotes **(“ “).**
- ➔ When we are working with strings and we want to enclose some text with special symbols like **(‘ ‘, “ “, /)** we need to use escape character **(\).**

  Example:

```
<body>
    <p id="name"></p>
    <p id="fn"></p>
    <p id="nn"></p>
    <script>
        let name = "kalyan"
        let fullName = "'R' Venkat Kalyan"
        let newLine = "In JS we use \\n to give a line break"
        document.getElementById("name").innerHTML= "My Name is "+name
        document.getElementById("fn").innerHTML = fullName
        document.getElementById("nn").innerHTML = newLine
    </script>
</body>
```

Output:

```
My Name is kalyan

'R' Venkat Kalyan

In JS we use \n to give a line break
```

JS String Methods:

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

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

See Also:

[String Search Methods](#)
[String Templates](#)

[String indexOf()](#)
[String lastIndexOf()](#)
[String search()](#)

Number:

- ➔ JS has only one type of number.
- ➔ Numbers can be written with or without decimal points.
- ➔ In JS Integers are accurate up to 15 digits.
- ➔ **Max Range: +-9007199254740991 or +-2^53-1.**
- ➔ In JS floating arithmetic is not accurate. (2.1 + 3.1)

JS Number Methods:

Method	Description
toString()	Returns a number as a string
toExponential()	Returns a number written in exponential notation
toFixed()	Returns a number written with a number of decimals
toPrecision()	Returns a number written with a specified length
valueOf()	Returns a number as a number

BigInt:

- ➔ In JS **bigint** variables are used to store big int values that are too large to represent in number datatype.
- ➔ In JS to create a **bigint** append a n at end of value or call **BigInt().**
- ➔ In JS we cannot perform any **arithmetic operations** by using a **combination** of **bigint variable** and a **number varible**.
- ➔ **Example:**
    - *let x = 99999999999999999**n**;*
    - *let y = **BigInt**(9999999999999999);*

Objects:

- ➔ Objects are variables too, but objects can contain many values.
- ➔ The values are written as **name:value** pairs.
- ➔ In JS objects use named indexes.
- ➔ We can access object properties in two ways:
    - **objectName.propertyName**
    - **objectName[propertyName]**

Example:

```
<body>
    <p id="fn"></p>
    <p id="mn"></p>
    <p id = "ln"></p>
    <script>
        const details = {fn: "Reddy", mn: "Venkat", ln: "Kalyan"}
        document.getElementById("fn").innerHTML = details.fn
        document.getElementById("mn").innerHTML = details["mn"]
        document.getElementById("ln").innerHTML = details.ln
    </script>
</body>
```

Output:

Reddy

Venkat

Kalyan

Arrays:

- ➔ Arrays are used to store a list of items.
- ➔ In JS Arrays use numbered indexes.
- ➔ In JS arrays can be created in two ways:
    - const myArr = [1,2,3,4]
    - const myArr = new Array(1,2,3,4)

Array Methods:

Array length
Array toString()
Array at()
Array join()
Array pop()
Array push()

Loops in JS:

JS supports the following loops:

- `for` - loops through a block of code a number of times
- `for/in` - loops through the properties of an object
- `for/of` - loops through the values of an iterable object
- `while` - loops through a block of code while a specified condition is true
- `do/while` - also loops through a block of code while a specified condition is true

JS Hoisting:

➔ Hoisting is JS's default behavior of moving declarations to top.
➔ In JS a variable can be declared after it is used using **var** keyword.
➔ Variables declared with **let** and **const** keyword cannot be used before declaration.

Example:

```
x= 10;          // hoisting

var x;
```

JS Use Strict:

➔ The **"use strict"** directive doesn't allow a variable to be used before declaration.
➔ Strict mode is declared by adding **"use strict";** to the beginning of a script.

Example:

```
"use strict";

x = 10 ;        // error due to strict mode

var x = 10;     // executes without error
```

JS JSON:

- ➔ JSON is a format for storing and transporting data.
- ➔ JSON is often used when data is sent from a server to a web page.
- ➔ JSON stands for **JavaScript Object Notation.**
- ➔ JSON is a lightweight data interchange format.
- ➔ JSON is **language independent**.
- ➔ JSON Rules:
    - Data is always in name/value pairs, separated by commas.
    - Curly braces hold objects and square braces hold arrays.

Example:

```
{
"employees":[
  {"firstName":"John", "lastName":"Doe"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Peter", "lastName":"Jones"}
]
}
```

JS Class:

- ➔ In JS a class is a template for objects.
- ➔ In JS we can create a class by using the **class** keyword.
- ➔ To create a constructor in class we need to use the **constructor** keyword.
- ➔ The **constructor** method is executed when a new object is created.
- ➔ It is used to initialize object properties.
- ➔ If you do not define a constructor method, JS declares an empty constructor method.

Example:

```
<script>
    class myDetails{
        constructor(name){
            this.name = name
            alert(this.name)
        }
    }
    const md = new myDetails("kalyan");
</script>
```

Getters & Setters

➔ In JS we can create getters and setters using **get** & **set** keywords.

Example:

```
class Car {
  constructor(brand) {
    this._carname = brand;
  }
  get carname() {
    return this._carname;
  }
  set carname(x) {
    this._carname = x;
  }
}

const myCar = new Car("Ford");

document.getElementById("demo").innerHTML = myCar.carname;
```

➔ **Hoisting is not supported by classes.**

JS HTML DOM:

- ➔ With HTML DOM, JS can access and change all the elements of an HTML document.
- ➔ When a web page is loaded in a browser, the browser creates a **Document Object Model (DOM).**
- ➔ The DOM is a **W3C (World Wide Web Consortium)** standard and DOM defines a standard for accessing documents.
- ➔ The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

Finding HTML Elements

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

Changing HTML Elements

Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element
Method	**Description**
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element

Adding Events Handlers

Method	Description
document.getElementById(*id*).onclick = function() {*code*}	Adding event handler code to an onclick event

Adding and Deleting Elements

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

JS DOM EventListener:

- ➔ The **addEventListener()** in JS attaches an event handler to the specified element.
- ➔ The **addEventListener()** attaches an event handler without overriding existing events.

Example:

```
<input id = "age" placeholder = "enter age"></input><br>
<button id="myBtn">Check Vote Eligibility</button>
<script>
document.getElementById("myBtn").addEventListener("click", myFunction);
function myFunction() {
  let val = parseInt(document.getElementById("age").value)
  if(val < 18)
  alert("NOT ELIGIBLE")
  else
  alert("ELIGIBLE")
}
</script>
```

- ➔ In this example an event is created for button to validate age and this event triggers **myFunction()** to check eligibility and displays a **alert().**

JS BOM:

- ➔ The **Browser Object Model (BOM)** allows JS to **talk to** the browser.
- ➔ The object used by **BOM** is the **window.**
- ➔ The **window** object is supported by all the browsers, and it represents a browser window.

- `window.open()` - open a new window
- `window.close()` - close the current window
- `window.moveTo()` - move the current window
- `window.resizeTo()` - resize the current window

Window Screen

The `window.screen` object can be written without the window prefix.

Properties:

- `screen.width`
- `screen.height`
- `screen.availWidth`
- `screen.availHeight`
- `screen.colorDepth`
- `screen.pixelDepth`

Window History

The `window.history` object can be written without the window prefix.

To protect the privacy of the users, there are limitations to how JavaScript can access this object.

Some methods:

- `history.back()` - same as clicking back in the browser
- `history.forward()` - same as clicking forward in the browser

Window Location

The `window.location` object can be written without the window prefix.

Some examples:

- `window.location.href` returns the href (URL) of the current page
- `window.location.hostname` returns the domain name of the web host
- `window.location.pathname` returns the path and filename of the current page
- `window.location.protocol` returns the web protocol used (http: or https:)
- `window.location.assign()` loads a new document

JS Popup Boxes:

JS has three kind of popup boxes:

alert ("Displayable Message") -> only we will get ok button.

confirm ("Displayable Message") -> we will get ok & cancel button.

Prompt("Displayable Message", "Default Message") -> ok & cancel button.

JS Cookies:

- ➔ Cookies help us to store information user information in web browsers.
- ➔ Cookies are data stored in small text files on our devices.
- ➔ When a web server has sent a web page to the browser, the server shuts down and it forgets all information. To solve this problem cookies are used to store data and are saved in name-value pairs.
- ➔ Using JS we can create, read, and delete cookies with the **document.cookie** property.

Example:

document.cookie = "username=Kalyan";