CCS-403 Course Name: Web Technology Tools Credits: 4

Unit-1: Introduction

Introduction to basic concept, Internet domains, Client IP address, web

client and server, The Phases of Web Site Development, Creating

Internet World Wide Web pages- HTML - Hypertext Markup Language,

Basic HTML Concepts, Lists, Tables, linking documents frames, adding

Graphics and multimedia, HTML Forms.

Unit-2: CSS

Introduction to Style sheet, types of style sheets- Inline, External,

Embedded CSS, color and background, text formatting attributes, CSS

Border, margin properties, Positioning Use of classes in CSS, Div and

span tag, Classes, use of external style sheets.

Unit-3: Java Script

Intro to script, types, intro of JavaScript, JavaScript identifiers,

operators, control & Looping structure, Intro of Array, Array with

methods, form object, Math, String and Date Objects with methods

User defined & Predefined functions, DOM objects, Window Navigator,

, Event handling, Validations On Forms, introduction to cookies.

Unit-4: XML

Intro & features of XML, XML writing elements, attributes etc. XML with

CSS, DSO, XML Namespaces XML DTD, XML Schemas, Writing Simple

sheets using XSLT, SAX & DOM Parsers, SOAP Introduction.

Unit-5: PHP

Introduction, apache/IIS installation, setting and configuration PHP to

work under apache/IIS, writing PHP, data types, variables, constants

operators, arrays, loops, functions PHP Server variables, working with

forms

Prescribed Book

1. Ivan Bayross, “Web enabled commercial application development using

HTML, DHTML, JavaScript, PERL-CGI”, BPB pub., 2nd Ed., 2000

Reference Books

Joe Fawcett,Danny Ayers,Liam R.E. Quin, “Beginning XML” Wrox Press, 5th Ed., 2012

Deitel & Deitel, “XML how to program”, Pearson, 2000

Ivan Bayross , “HTML, DHTML, JavaScript, Perl & CGI” ,BPB pub. 3rd Ed.,2004

**Intro to Script and types:**

* Client-side scripts are programs that run on the user's computer or device, rather than on a web server. They are typically written in languages like JavaScript and or VBScripts, and are used to add interactivity and functionality to web pages.
* One of the main benefits of client-side scripts is that they can make web pages more dynamic and responsive. For example, they can be used to validate form data before it is submitted to a server, or to update the content of a page without requiring a full page refresh. This can make websites feel more interactive and engaging for users.
* Client-side scripts are also useful for implementing features like animations, slide shows, and pop-up windows. They can be used to create custom user interfaces, and to interact with third-party APIs and services.
* Overall, client-side scripts are a key tool for web developers looking to create modern, interactive web applications. They allow developers to add functionality and interactivity to web pages without requiring additional server-side processing or page loads.
* Two types of Client Scripts exists

1. VBScript
2. JavaScript

VBScript:

* + VBScript (Visual Basic Scripting Edition) is a scripting language developed by Microsoft that is based on the Visual Basic programming language. It is primarily used in Microsoft Windows environments for tasks such as automating administrative tasks and controlling ActiveX components.
  + VBScript is a client-side scripting language, which means that it runs on the user's computer or device rather than on a web server. It is commonly used for tasks such as form validation, browser automation, and manipulating the Document Object Model (DOM) of a web page.
  + VBScript is only supported by Internet Explorer and Microsoft Edge browsers, whereas JavaScript is supported by all major web browsers.
  + VBScript is less commonly used in modern web development than JavaScript, as its usage has declined in favor of other scripting languages like JavaScript and PHP. However, it is still used in some legacy systems and applications, particularly in enterprise environments that rely on Microsoft technologies.

**Intro of JavaScript:**

* JavaScript is a high-level, dynamic, and interpreted programming language that is used primarily to create interactive web applications. It was created in 1995 by Brendan Eich while he was working at Netscape Communications Corporation.
* It enables developers to create dynamic and responsive web pages that can respond to user input and update content without requiring a full page refresh. It is commonly used for things like form validation, interactive maps, animations, and more.
* One of the key features of JavaScript is its ability to manipulate the Document Object Model (DOM), which is the structure that represents a web page in the browser.
* JavaScript is also often used in connection with other web development technologies such as HTML and CSS. It can be embedded directly into HTML code using script tags, or it can be loaded from separate JavaScript files.
* As a programming language, JavaScript is known for its flexibility and ease of use. It is loosely typed, which means that variables can be assigned different types of data at different times.
* Overall, JavaScript is a powerful and versatile programming language that is essential for modern web development.

How to save and execute java script:

To save and execute JavaScript code, you can follow these steps:

Create a new text file using a text editor, such as Notepad, Sublime Text, or Visual Studio Code.

Write your JavaScript code in the text file. For example, you could write a simple "Hello, World!" program:

console.log("Hello, World!");

1. Save the file with a .js extension. For example, you could save the file as hello.js.
2. Open a web browser and create a new HTML file using a text editor.
3. In the HTML file, create a <script> tag and specify the src attribute to the path of the JavaScript file that you just created. For example:

<!DOCTYPE html>

<html>

<head>

<title>My HTML Page</title>

</head>

<body>

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

</body>

</html>

1. Save the HTML file with an .html extension. For example, you could save the file as mypage.html.
2. Open the HTML file in a web browser, and you should see the output of your JavaScript code in the console.

Alternatively, you can run JavaScript code directly in the web browser console without creating an HTML file. To do this, simply open the web browser console (usually by pressing F12 or Ctrl+Shift+J), and type your JavaScript code directly into the console prompt. For example:

console.log("Hello, World!");

Then, press Enter, and you should see the output in the console.

**JavaScript identifiers (Variable Names):**

* JavaScript identifiers are names used to identify variables, functions, objects, and other entities in JavaScript code. Identifiers are case-sensitive and can contain letters, digits, underscores, and dollar signs. The first character of an identifier must be a letter, underscore, or dollar sign (but not a digit).
* Here are some examples of valid JavaScript identifiers:
  + - age
    - firstName
    - \_lastName
    - $totalAmount
    - calculateTotal

And here are some examples of invalid JavaScript identifiers:

* + - 2name (starts with a digit)
    - first-name (contains a hyphen)
    - my$%variable (contains special characters)
* It's important to choose meaningful and descriptive names for your identifiers to make your code easier to read and maintain.
* Here are five simple examples of JavaScript identifiers:
  + - age - This identifier could be used to store the age of a person in a variable.
    - firstName - This identifier could be used to store the first name of a person in a variable.
    - isReady - This identifier could be used to store a boolean value indicating whether a certain task is ready to be executed or not.
    - myFunction - This identifier could be used to name a function that performs a specific task.
    - shoppingCart - This identifier could be used to store an array of items in a shopping cart.

JavaScript supports several datatypes, including:

1. **Primitive Datatypes:** These datatypes are immutable (cannot be changed) and are copied by value.
   * **String:** A sequence of characters used to represent text. For example, "Hello, world!".
   * **Number:** A numeric datatype used to represent integers, floating-point numbers, and NaN (Not a Number). For example, 42 or 3.14.
   * **Boolean:** A logical datatype used to represent true or false values.
   * **Undefined:** A special value that indicates a variable has not been assigned a value.
   * **Null:** A special value that represents the intentional absence of any object value.
2. **Object Datatypes:** These datatypes are mutable (can be changed) and are copied by reference.
   * **Object:** A collection of key-value pairs, where the keys are strings and the values can be any datatype.
   * **Array:** An ordered collection of elements, where each element can be of any datatype and is identified by an index.
   * **Function:** A callable object that performs a specific task.
   * **Date:** A datatype that represents a date and time value.
   * **RegExp:** A datatype that represents a regular expression pattern.

**JavaScript Operators:**

**JavaScript supports various types of operators, including:**

* **Arithmetic Operators:** Used to perform arithmetic operations on numerical values.

Example:

let x = 10;

let y = 5;

console.log(x + y); // Output: 15

console.log(x - y); // Output: 5

console.log(x \* y); // Output: 50

console.log(x / y); // Output: 2

console.log(x % y); // Output: 0

* **Assignment Operators:** Used to assign values to variables.

Example:

let x = 10;

x += 5; // Same as x = x + 5

console.log(x); // Output: 15

* **Comparison Operators:** Used to compare two values and return a boolean value.

Example:

let x = 10;

let y = 5;

console.log(x > y); // Output: true

console.log(x < y); // Output: false

console.log(x >= y); // Output: true

console.log(x <= y); // Output: false

console.log(x === y); // Output: false

console.log(x !== y); // Output: true

* **Logical Operators:** Used to combine two or more conditions and return a boolean value.

Example:

let x = 10;

let y = 5;

console.log(x > 5 && y < 10); // Output: true

console.log(x > 5 || y > 10); // Output: true

console.log(!(x > 5)); // Output: false

* **Bitwise Operators:** Used to perform bitwise operations on binary numbers.

Example:

let x = 5; // Binary representation: 0101

let y = 3; // Binary representation: 0011

console.log(x & y); // Output: 1 (Binary representation: 0001)

console.log(x | y); // Output: 7 (Binary representation: 0111)

console.log(x ^ y); // Output: 6 (Binary representation: 0110)

console.log(~x); // Output: -6 (Binary representation: 1010)

console.log(x << 1); // Output: 10 (Binary representation: 1010)

console.log(x >> 1); // Output: 2 (Binary representation: 0010)

console.log(x >>> 1); // Output: 2 (Binary representation: 0010)

* **Unary Operators:** Used to operate on a single operand.

Example:

let x = 10;

console.log(-x); // Output: -10

console.log(++x); // Output: 11

console.log(x++); // Output: 11

console.log(x); // Output: 12

console.log(typeof x); // Output: "number"

**Control & Looping structure of JavaScript:**

**Control structures and looping structures are essential programming constructs used to control the flow of execution of a program. In JavaScript, there are several control structures and looping structures available, including:**

1. **Conditional Statements (if/else):** The if statement is used to execute a block of code if a particular condition is true. The else statement is used to execute a block of code if the condition is false.

Example:

let x = 10;

if (x > 5) {

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

} else {

console.log("x is less than or equal to 5");

}

The if statement has various variants as:

* + **if-else statement:** This variant of the if statement is used to execute different blocks of code depending on whether a condition is true or false.

Example:

let x = 10;

if (x > 5) {

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

} else {

console.log("x is less than or equal to 5");

}

Output

x is greater than 5

* + **else-if statement:** This variant of the if statement is used to check multiple conditions and execute different blocks of code depending on which condition is true.

Example:

let x = 10;

if (x > 15) {

console.log("x is greater than 15");

} else if (x > 5) {

console.log("x is greater than 5 but less than or equal to 15");

} else {

console.log("x is less than or equal to 5");

}

Output

x is greater than 5 but less than or equal to 15

* + **Nested if statement:** This variant of the if statement is used to execute another if statement inside the body of an existing if statement.

Example:

let x = 10;

let y = 5;

if (x > 5) {

if (y > 2) {

console.log("Both x and y are greater than 5");

} else {

console.log("x is greater than 5 but y is less than or equal to 2");

}

} else {

console.log("x is less than or equal to 5");

}

Output

Both x and y are greater than 5

1. **Switch Statements: The switch statement is used to execute different blocks of code** based on different cases.

Example:

let day = 3;

switch (day) {

case 1:

console.log("Monday");

break;

case 2:

console.log("Tuesday");

break;

case 3:

console.log("Wednesday");

break;

default:

console.log("Invalid day");

break;

}

**Loops (for, while, do-while):** Loops are used to execute a block of code repeatedly until a certain condition is met.

* **The for loop** is used to iterate over a range of values. It has three parts: initialization, condition, and increment.

Example:

for (let i = 0; i < 5; i++) {

console.log(i);

}

* **The while loop** is used to execute a block of code repeatedly while a particular condition is true.

Example:

let i = 0;

while (i < 5) {

console.log(i);

i++;

}

* **The do-while loop** is similar to the while loop, but it always executes the block of code at least once before checking the condition.

Example:

let i = 0;

do {

console.log(i);

i++;

} while (i < 5);

* **Break and Continue Statements:** The break statement is used to exit a loop or switch statement, while the continue statement is used to skip to the next iteration of a loop.

Example:

for (let i = 0; i < 5; i++) {

if (i === 3) {

break;

}

console.log(i);

}

Output

0

1

2

**Intro of Array and Array with methods:**

In JavaScript, an array is a collection of values of any type, including strings, numbers, and other arrays. Arrays in JavaScript are a special type of object, and can be created using the built-in Array constructor or using array literal syntax.

Creating an array: Arrays can be created using either the Array constructor or using array literal syntax. Here's an example of creating an array using both methods:

// Using the Array constructor

let myArray = new Array(1, 2, 3, 4, 5);

// Using array literal syntax

let myOtherArray = [1, 2, 3, 4, 5];

Accessing elements: Elements in an array can be accessed using their index, which starts at 0 for the first element. Here's an example of accessing the third element in an array:

let myArray = [1, 2, 3, 4, 5];

let thirdElement = myArray[2]; // returns 3

Modifying elements: Elements in an array can be modified by assigning a new value to their index. Here's an example of modifying the fourth element in an array:

let myArray = [1, 2, 3, 4, 5];

myArray[3] = "four"; // changes the fourth element to "four"

Adding elements: Elements can be added to an array using the push() method, which adds elements to the end of the array. Here's an example of adding a new element to an array:

let myArray = [1, 2, 3, 4, 5];

myArray.push(6); // adds 6 to the end of the array

Removing elements: Elements can be removed from an array using the pop() method, which removes the last element from the array. Here's an example of removing the last element from an array:

let myArray = [1, 2, 3, 4, 5];

myArray.pop(); // removes the last element (5) from the array

Array length: The length of an array can be found using the length property. Here's an example of finding the length of an array:

let myArray = [1, 2, 3, 4, 5];

let arrayLength = myArray.length; // returns 5

Multidimensional arrays: Arrays in JavaScript can be multidimensional, meaning that they can contain other arrays as elements. Here's an example of creating a multidimensional array:

let myArray = [[1, 2], [3, 4], [5, 6]];

let secondElement = myArray[1]; // returns [3, 4]

let thirdElementFirstSubArray = myArray[2][0]; // returns 5

Array methods: (These are inbuilt functions associated with arrays)

push(): Adds one or more elements to the end of an array and returns the new length of the array.

let myArray = [1, 2, 3];

myArray.push(4); // myArray is now [1, 2, 3, 4]

pop(): Removes the last element from an array and returns the removed element.

let myArray = [1, 2, 3];

let removedElement = myArray.pop(); // myArray is now [1, 2] and removedElement is 3

slice(): Returns a new array that contains a portion of an existing array. The original array is not modified.

let myArray = [1, 2, 3, 4, 5];

let subArray = myArray.slice(1, 4);

// subArray is [2, 3, 4] and myArray is still [1, 2, 3, 4, 5]

concat(): Returns a new array that combines two or more arrays.

**Form object:**

* In JavaScript, the form object is used to represent an HTML form element.
* It provides a way to access and manipulate form elements, such as input fields, buttons, and dropdown lists, using JavaScript code.
* The form object is created when an HTML form element is defined in a web page. The form object is automatically generated by the browser and can be accessed using the document.forms collection or by referencing the form element directly by its name or ID attribute.
* The form object in JavaScript provides a powerful way to interact with HTML forms in a web page. It allows you to access and manipulate form elements using JavaScript code, and provides a range of methods and properties for this purpose.

Here are some of the properties and methods of the form object:

**Form Properties:**

**form.name:** Returns the value of the name attribute of the form element.

**form.elements:** Returns a collection of all the form elements contained within the form. This includes input fields, buttons, dropdown lists, etc.

**form.length:** Returns the number of form elements contained within the form.

**form.method:** Returns the HTTP method used to submit the form (either "GET" or "POST").

**form.action:** Returns the URL to which the form data will be submitted when the form is submitted.

**Form Methods:**

**form.submit():** Submits the form to the server. This can be triggered programmatically using JavaScript.

**form.reset():** Resets the form to its initial state. This clears all the input fields and resets the dropdown lists, checkboxes, and radio buttons.

**form.addEventListener():** Adds an event listener to the form element. This allows you to listen for events such as form submission or form reset.

**form.checkValidity():** Checks if all the form elements are valid. This method returns true if all the elements are valid and false if any element is invalid.

**form.reportValidity():** Displays error messages for any invalid form elements. This method displays error messages next to the invalid elements and returns false if any element is invalid.

**Math, String and Date Objects with methods:**

Math Object:

* JavaScript provides several built-in objects to perform operations on numbers.
* The Math object provides mathematical operations in JavaScript.
* Here are some commonly used methods of the Math object:

**Math.PI: R**eturns the value of Pi (approximately 3.14159).

**Math.round(x):** Returns the nearest integer of a number x.

**Math.floor(x):** Returns the largest integer less than or equal to a number x.

**Math.ceil(x):** Returns the smallest integer greater than or equal to a number x.

**Math.pow(x, y):** Returns the value of x to the power of y.

**Math.sqrt(x):** Returns the square root of a number x.

**Math.random():** Returns a random number between 0 (inclusive) and 1 (exclusive).

**Example:**

**Example 1: Generating a random number**

The Math.random() method returns a random number between 0 and 1 (exclusive). You can use this method to generate a random number within a specific range by multiplying the result by the range and adding the minimum value. Here's an example of generating a random number between 1 and 100:

const min = 1;

const max = 100;

const randomNum = Math.floor(Math.random() \* (max - min + 1)) + min;

console.log(randomNum);

// Output: A random number between 1 and 100

In this example, Math.floor() is used to round down the result of Math.random() \* (max - min + 1) to the nearest integer. The + min at the end ensures that the random number falls within the range of 1 to 100.

**Example 2: Rounding a number**

The Math.round() method returns the nearest integer of a number. You can use this method to round a number to a specific number of decimal places by multiplying the number by a power of 10, rounding it, and then dividing it by the same power of 10. Here's an example of rounding a number to two decimal places:

const num = 3.14159;

const roundedNum = Math.round(num \* 100) / 100;

console.log(roundedNum);

// Output: 3.14

In this example, num \* 100 multiplies the number by 100, giving you a whole number that can be rounded using Math.round(). The resulting number is then divided by 100 to get the original number rounded to two decimal places.

String Object:

JavaScript provides several built-in objects to perform operations on numbers, strings, and dates.

The String object represents a sequence of characters. Here are some commonly used methods of the String object:

**String.length:** Returns the number of characters in a string.

**String.charAt(index):** Returns the character at a specified index in a string.

**String.concat(str1, str2, ..., strN):** Combines two or more strings into one.

**String.indexOf(searchValue, fromIndex):** Returns the index of the first occurrence of a specified value in a string, starting from a specified index.

**String.substring(startIndex, endIndex):** Returns a portion of a string between a specified start index and end index.

**String.toUpperCase():** Converts all characters in a string to uppercase.

**String.toLowerCase():** Converts all characters in a string to lowercase.

Example 1: Finding the length of a string

The String.length property returns the number of characters in a string. You can use this property to find the length of a string. Here's an example:

const str = "Hello, world!";

const strLength = str.length;

console.log(strLength);

// Output: 13

In above example, str.length returns the number of characters in the string "Hello, world!", which is 13.

Example 2: Converting a string to uppercase

The String.toUpperCase() method converts all characters in a string to uppercase. You can use this method to convert a string to uppercase. Here's an example:

const str = "Hello, world!";

const strUppercase = str.toUpperCase();

console.log(strUppercase);

// Output: "HELLO, WORLD!"

In above example, str.toUpperCase() returns a new string with all characters in the original string "Hello, world!" converted to uppercase. The original string is not modified.

Date Object:

The Date object represents a date and time value in JavaScript. Here are some commonly used methods of the Date object:

**Date.now():** Returns the current timestamp in milliseconds.

**Date.parse(dateString):** Parses a date string and returns the timestamp in milliseconds.

**Date.getFullYear():** Returns the year of a date.

**Date.getMonth():** Returns the month of a date (0-11).

**Date.getDate():** Returns the day of the month of a date (1-31).

**Date.getDay():** Returns the day of the week of a date (0-6, where Sunday is 0).

**Date.getHours():** Returns the hours of a date (0-23).

**Date.getMinutes():** Returns the minutes of a date (0-59).

**Date.getSeconds():** Returns the seconds of a date (0-59).

**Date.getMilliseconds():** Returns the milliseconds of a date (0-999).

**Date.toDateString():** Returns a string representation of the date in the format "Day Mon DD YYYY".

Example 1: Getting the current date and time

The Date() constructor with no arguments returns the current date and time. You can use this constructor to get the current date and time. Here's an example:

const currentDate = new Date();

console.log(currentDate);

// Output: The current date and time in your local time zone

In this example, new Date() creates a new Date object with the current date and time.

Example 2: Formatting a date as a string

The Date.toLocaleDateString() method returns a string representation of a date using the user's locale settings. You can use this method to format a date as a string. Here's an example:

const date = new Date('2023-03-08');

const options = { weekday: 'long', year: 'numeric', month: 'long', day: 'numeric' };

const formattedDate = date.toLocaleDateString('en-US', options);

console.log(formattedDate);

// Output: "Wednesday, March 8, 2023"

In this example, new Date('2023-03-08') creates a new Date object representing March 8, 2023. The options object specifies how to format the date as a string using the toLocaleDateString() method. The resulting string is formatted according to the en-US locale, which uses the "Wednesday, March 8, 2023" format.

**User defined & Predefined functions:**

In JavaScript, functions can be classified into two types: user-defined functions and predefined functions.

**User-defined functions are functions that** are defined by the user to perform a specific task. These functions can be called multiple times within the program and can take arguments and return values. Here's an example of a user-defined function:

function sum(a, b) {

return a + b;

}

const result = sum(3, 5);

console.log(result); // Output: 8

In this example, sum() is a user-defined function that takes two arguments a and b and returns their sum. The function is then called with the arguments 3 and 5, and the returned value 8 is assigned to the result variable.

**Predefined functions**, also known as built-in functions, are functions that are already defined in the JavaScript language and are available for use without having to define them. These functions are part of the core JavaScript language and can be used to perform various tasks. Here are some examples of predefined functions:

const str = "Hello, world!";

const strLength = str.length;

console.log(strLength); // Output: 13

const num = 3.14159;

const roundedNum = Math.round(num \* 100) / 100;

console.log(roundedNum); // Output: 3.14

const currentDate = new Date();

console.log(currentDate); // Output: The current date and time in your local time zone

In these examples, str.length, Math.round(), and new Date() are predefined functions that are used to perform specific tasks. The str.length property is used to get the length of a string, Math.round() is used to round a number, and new Date() is used to get the current date and time.

**DOM objects**

In JavaScript, the Document Object Model (DOM) is a programming interface for web documents. The DOM represents the web page as a tree structure where each node represents an element, attribute, or text on the page. JavaScript can manipulate the DOM tree to dynamically change the content and style of the web page.

DOM objects are JavaScript objects that represent elements, attributes, and text nodes in the DOM tree. These objects can be accessed and manipulated using JavaScript code. Here are some commonly used DOM objects:

1. **document**: The document object represents the web page itself and provides methods for accessing and manipulating the DOM tree.
2. **element**: The element object represents an HTML element in the DOM tree, such as a <div> or <p> tag. Elements can be accessed using the document.getElementById() method or by using methods such as document.querySelector() or document.querySelectorAll().
3. **attribute**: The attribute object represents an HTML attribute of an element, such as src or href. Attributes can be accessed and modified using the element.getAttribute() and element.setAttribute() methods.
4. **text**: The text object represents the text content of an element in the DOM tree. Text nodes can be accessed and modified using the element.textContent property.

**Example of JavaScript DOM Object:**

Here's an example of using DOM objects to dynamically change the content and style of a web page:

<!DOCTYPE html>

<html>

<head>

<style>

#myButton {

background-color: blue;

color: white;

font-size: 16px;

padding: 10px 20px;

border-radius: 5px;

}

</style>

</head>

<body>

<h1>Welcome to my web page!</h1>

<p>Click the button below to change the color of this paragraph.</p>

<button id="myButton">Change color</button>

<script>

const button = document.getElementById('myButton');

const paragraph = document.querySelector('p');

button.addEventListener('click', () => {

paragraph.style.color = 'red';

});

</script>

</body>

</html>

In this example, the document.getElementById() method is used to get a reference to the myButton button element. The document.querySelector() method is used to get a reference to the first <p> element on the page. An event listener is added to the button using the addEventListener() method, which listens for a click event and changes the color of the paragraph text using the style.color property.

**Window Navigator**

In JavaScript, the Window Navigator is an object that represents the web browser's navigation bar. It provides information about the user's browser, such as the browser name, version, and platform.

The Navigator object is a property of the window object, and can be accessed using the window.navigator property. Here are some commonly used properties and methods of the Navigator object:

1. **navigator.userAgent**: This property returns the user agent string of the browser, which is a string that identifies the browser, its version, and the operating system.
2. **navigator.platform**: This property returns the platform on which the browser is running, such as "Win32" or "MacIntel".
3. **navigator.language**: This property returns the language of the browser, as defined by the user's operating system.
4. **navigator.cookieEnabled**: This property returns a boolean value indicating whether cookies are enabled in the browser.
5. **navigator.geolocation**: This property returns a Geolocation object that can be used to get the user's current location.

**Example of windows Navigator in JavaScript:**

Here's an example of using the Navigator object to get information about the user's browser:

const userAgent = navigator.userAgent;

const platform = navigator.platform;

const language = navigator.language;

const cookiesEnabled = navigator.cookieEnabled;

console.log(`User agent: ${userAgent}`);

console.log(`Platform: ${platform}`);

console.log(`Language: ${language}`);

console.log(`Cookies enabled: ${cookiesEnabled}`);

In this example, the navigator.userAgent, navigator.platform, navigator.language, and navigator.cookieEnabled properties are used to get information about the user's browser. The values of these properties are then logged to the console.

Note that not all properties and methods of the Navigator object are supported in all browsers, and some may require the user's permission to access (such as the navigator.geolocation property).