Unit IV

Conditional Statements and Loops - Event Handlers – The Document Objects – Window Object – Math, Number and Data objects

What is a Conditional Statement?

A conditional statement is a statement that you can use to execute a bit of code based on a condition or to do something else if that condition is not met.

Using Conditional Statements

Using if/else Statement Blocks

While using conditional statements, you will see that they are similar to functions in some ways. Most notable are the curly brackets ({}) that surround the sections of code that will be executed given a condition.

The if/else Statement Block Structure in JavaScript

The if/else statement in JavaScript is a control flow structure that allows the execution of different code blocks based on a condition.

Basic Syntax

```
if (condition) {
   // Code to execute if the condition is true
} else {
   // Code to execute if the condition is false
}
```

Example 1: Simple if/else

```
let num = 10;
if (num > 5) {
   console.log("The number is greater than 5.");
} else {
   console.log("The number is 5 or less.");
}
```

Output:

```
The number is greater than 5.
```

Example 2: if/else if/else Structure

If multiple conditions need to be checked, you can use else if:

```
let score = 85;
if (score >= 90) {
   console.log("Grade: A");
} else if (score >= 75) {
   console.log("Grade: B");
} else if (score >= 50) {
   console.log("Grade: C");
} else {
   console.log("Grade: F");
}
```

Output:

```
Grade: B
```

Example 3: Nested if/else

You can nest if/else statements within each other.

```
let age = 20;
let hasID = true;
if (age >= 18) {
   if (hasID) {
      console.log("You can enter.");
   } else {
      console.log("ID is required.");
   }
} else {
   console.log("You are underage.");
}
```

Output:

```
You can enter.
```

Example 4: Using Ternary Operator (Shorter if/else)

The ternary operator (condition? trueCase : falseCase) provides a shorthand way to write if/else statements.

```
let isMember = true;
let discount = isMember ? "10% discount" : "No discount";
console.log(discount);
```

Output:

```
10% discount
```

The switch Statement in JavaScript

The switch statement in JavaScript is used for executing different blocks of code based on different conditions. It is an alternative to multiple if/else if statements and is often used when comparing a single variable against multiple possible values.

Syntax:

```
switch (expression) {
   case value1:
      // Code to execute if expression === value1
      break;
   case value2:
      // Code to execute if expression === value2
      break;
   default:
      // Code to execute if no case matches
}
```

- expression: The value being compared.
- case: Represents different possible values of the expression.
- break: Exits the switch after a match is found to prevent unnecessary execution.
- default: Executes if no case matches (similar to else in if/else).

```
var thename="Fred";
switch (thename) {
case "George" :
```

```
window.alert("George is an OK name");
break;
case "Fred" :
window.alert("Fred is the coolest name!");
window.alert("Hi there, Fred!");
break;
default :
window.alert("Interesting name you have there");
}
```

First, this example declares and assigns a variable named thename; it is given a value of Fred. Next, the switch statement begins, using the variable thename as the basis for comparison. Then, the block is opened with a curly bracket, followed by the first case statement. Written like this, it is saying, "If thename is equal to George then execute the commands after the colon at the end of this line". If thename were equal to George, you would get an alert. Next you see the break statement, which tells the browser to exit the code block and move on to the next line of code after the block. You use the break statement in the switch block to be sure only one of the case sections is executed; otherwise, you run the risk of having all the cases executed following the one that returned true, because, by default, the browser would continue to the next statement rather than exit the block entirely even though it finds one of the cases to be true. To be sure that the browser exits the block, you add the break statement. If you get back to the script, you see that thename is not equal to George, so this case is skipped; however, the next comparison returns true because thename is equal to Fred in the script. Thus, the set of statements in this case block will be executed. Note that two lines of JavaScript code appear before the break statement here. This shows that you could have any number of lines within a case, as long as you remember to end with the break statement. Finally, you see the keyword default. This is used in the event that none of the case statements returns true. If this happens, the default section of code will be executed.

What is a Loop?

A loop is a block of code that allows you to repeat a section of code a certain number of times, perhaps changing certain variable values each time the code is executed.

Why Loops Are Useful

Loops are useful because they allow you to repeat lines of code without retyping them or using cut and paste in your text editor. This not only saves you the time and trouble of repeatedly typing the same lines of code, but also avoids typing errors in the repeated lines. You are also able to change one or more variable values each time the browser passes through the loop, which again saves you the time and trouble of typing a line that is only slightly different than the previous line.

Using Loops

The major loop types are:

- for Loop
- Nested for Loop
- while Loop
- do...while Loop
- for...in Loop
- forEach() Loop
- break Statement
- continue Statement

1. for Loop - The for loop is used when the number of iterations is known in advance.

Syntax:

```
for (initialization; condition; increment/decrement) {
   // Code to execute in each iteration
}
```

Example: Print numbers from 1 to 5

```
for (let i = 1; i <= 5; i++) {
   console.log(i);
}
```

Explanation:

let i = 1: Initializes i to 1.
i <= 5: Loops until i reaches 5.
i++: Increments i by 1 in each iteration.

Outputs:

```
1
2
3
4
5
```

2. Nested for Loop - A for loop inside another for loop is called a nested loop.

Example: Print a 3x3 pattern

```
for (let i = 1; i <= 3; i++) {
   for (let j = 1; j <= 3; j++) {
      console.log(`i = ${i}, j = ${j}`);
   }
}
```

Explanation:

The outer loop (i) runs 3 times.
The inner loop (j) runs 3 times for each i.

Outputs:

```
i = 1, j = 1
i = 1, j = 2
i = 1, j = 3
i = 2, j = 1
i = 2, j = 2
i = 2, j = 3
i = 3, j = 1
i = 3, j = 2
i = 3, j = 3
```

3. while Loop - The while loop executes as long as a condition is true.

Syntax:

```
while (condition) {
   // Code to execute
}
```

Example: Print numbers from 1 to 5

```
let i = 1;
while (i <= 5) {
   console.log(i);
   i++;
}
```

Explanation:

i = 1: Starts with 1.
while (i <= 5): Loops until i > 5.
i++: Increments i by 1.

Outputs:

```
1
2
3
4
5
```

4. do...while Loop - Executes the loop at least once, even if the condition is false.

Syntax:

```
do {
   // Code to execute
} while (condition);
```

Example: Print numbers from 1 to 5

```
let i = 1;
do {
   console.log(i);
   i++;
} while (i <= 5);
```

Explanation:

Executes the block before checking the condition.
If i > 5, it still runs once.

Outputs:

```
1
2
3
4
5
```

5. for...in Loop - Used for iterating over object properties.

Syntax:

```
for (let key in object) {
   // Code to execute
}
```

Example: Iterate over an object

```
let person = { name: "John", age: 25, city: "New York" };
for (let key in person) {
   console.log(`${key}: ${person[key]}`);
}
```

Explanation:

Iterates over person properties.

Outputs:

```
name: John
age: 25
city: New York
```

6. forEach() Loop - Used for iterating over arrays.

Syntax:

```
array.forEach(function(element, index, array) {
   // Code to execute
});
```

Example: Iterate over an array

```
let numbers = [1, 2, 3, 4, 5];
numbers.forEach(function(num) {
   console.log(num);
});
```

Explanation:

Loops through each element of numbers.

Outputs:

```
1
2
3
4
5
```

7. break Statement - The break statement stops loop execution immediately.

Example: Stop loop when i is 3

```
for (let i = 1; i <= 5; i++) {
   if (i === 3) {
      break;
   }
   console.log(i);
}
```

Explanation:

Stops execution when i is 3.

Outputs:

```
1
2
```

8. continue Statement - The continue statement skips the current iteration and moves to the next.

Example: Skip i = 3

```
for (let i = 1; i <= 5; i++) {
   if (i === 3) {
      continue;
   }
   console.log(i);
}
```

Explanation:

Skips i = 3 and continues the loop.

Outputs:

```
1
2
4
5
```

What is an Event Handler?

An event handler is a predefined JavaScript property of an object that is used to handle an event on a Web page. When events occur, you are able to use JavaScript event handlers to identify them and then perform a specific task or set of tasks. JavaScript enables you to react to an action by the viewer and to make scripts that are interactive, and more useful to you and to the viewer. Event handlers are useful because they enable you to gain access to the events that may occur on the page. For instance, if you wanted to send an alert to the viewer when he or she moves the mouse over a link, you could use the event handler to invoke the JavaScript alert you have coded to react to the event. You are now making things happen based on the actions of the viewer, which enables you to make more-interactive Web pages.

Understanding Event Handler Locations and Uses

To see how event handlers work, you need to know where you can place them in a document and how to use them to add JavaScript code for an event. Event handlers can be used in a number of locations. They can be used directly within HTML elements by adding special attributes to those elements. They can also be used within the <script> and </script> tags or in an external JavaScript file.

Using an Event Handler in an HTML Element

To use an event handler directly in an HTML element, you need to know the keyword for the event handler and where to place the event handler within the HTML code. To give you an example, introduced the onclick event handler, which is used to make something happen when the viewer clicks a specific area of the document. One element that can be clicked is a form button. So, suppose you want to alert the viewer to something when the user clicks a form button. You would write something similar to the following code:

```
<input type="button" value="Click Me!" onclick="JavaScript code here" />
```

To use an event handler, you add it as an additional attribute to an HTML tag. The only difference between an event handler "attribute" and an HTML attribute is that you can add JavaScript code inside an event handler attribute rather than just an attribute value. In the previous code, you would replace the JavaScript code here text with some actual JavaScript code. So, to make an alert pop up when the user clicks the button, you can add the necessary JavaScript code right inside your onclick attribute, as shown in the following example:

```
<body>
<form>
<input type="button" value="Click Me!" onclick="window.alert('Hi!');" />
</form>
</body>
```

When the viewer clicks this plain button, an alert will pop up with a greeting. Notice that the rules on the quote marks apply here. Using the onclick event handler as an attribute requires you to use

double quotes around all of your JavaScript code, so when you need quote marks for the alert, you use single quotes in order to avoid possible errors. Also notice that the alert command ends with a semicolon. This enables you to add additional JavaScript code after the alert, which enables you to perform multiple actions onthe click event rather than just a single JavaScript statement.
You could code in two alerts if you wanted to do so. All you have to do is remember to include the semicolons to separate the alert commands. This will be a little different because all of the code will be on one line rather than separate lines, as you normally see:

```
<body>
<form>
<input type="button" value="Click Me!"
onclick="window.alert('Hi!');window.alert('Bye!');" />
</form>
</body>
```

This example is able to perform two JavaScript statements on the same event by using semicolons to separate them. When using event handlers, you can execute multiple commands this way. It is important, however, to keep everything between the event handler keyword and the ending set of quotes on one line in your text editor; otherwise, a line break in the code could cause it not to run properly or to give a JavaScript error.
If the code you want to use becomes really long, you may wish to put the code in a function instead. The event handler can be used for any JavaScript code, so you can use it to call a function you have defined elsewhere. For example, you could place your two alerts within a function inside an external JavaScript file, and call the function from an event handler in the HTML code. First, code the external JavaScript file as follows:

```
function hi_and_bye() {
window.alert('Hi!');
window.alert('Bye!');
}
```

Next, add the script tags and the event handler to your HTML code:

```
<body>
<form>
<input type="button" value="Click Me!" onclick="hi_and_bye();" />
</form>
<script type="text/javascript" src="js_event_01.js"></script>
</body>
```

Notice how the function is called using the event handler just like a normal function call within a script. This enables you not only to shorten the code within the event handler, but also to reuse the function on another button click or event later in the page instead of writing the two alerts out again. The use of a function can help you quite a bit, especially when the code you want to use becomes extremely long.

Using an Event Handler in the Script Code

You can also use an event handler within the script code (whether using the script tags in the HTML document or using an external JavaScript file). One way to do this is to give the element an id attribute and then use the JavaScript method document.getElementById() to access the element. Once that is done, you can tie an event to the element.

Add the id Attribute

To use the previous script in this way, you will first add an id attribute to the HTML tag for the input button, as shown here:

```
<body>
<form>
<input type="button" value="Click Me!" id="say_hi" />
</form>
<script type="text/javascript" src="js_event_01.js"></script>
</body>
```

Notice that the button input element was given an id of say_hi. You will use this to access the button and tie it to an event in your script.

Access the Element

The document.getElementById() method allows you to access any element in the HTML document that has an id attribute using the value of its id attribute. In order to access the button input element you have been using with an id of say_hi, you could use the following code:

```
document.getElementById("say_hi");
```

This simply tells the browser you want to access the element with the id of say_hi in the HTML document. This method could be used directly in the script to access the element, but oftentimes you will want to assign this expression to a variable to save typing if you use it repeatedly. Thus, you could use the following code:

```
var hi_button = document.getElementById("say_hi");
```

Now, grab the rest of the code from the JavaScript file (js_event_01.js) and add the new line of script to it, as shown here:

```
function hi_and_bye() {
window.alert('Hi!');
window.alert('Bye!');
}
var hi_button = document.getElementById("say_hi");
hi_button.onclick = hi_and_bye;
```

There is now also an additional line of code. The last line of code takes the variable used for the input button element and gives it the onclick event handler by adding it after the variable name and a dot (.). The function hi_and_bye (which displays the two alerts) is assigned to handle the click event on the input button. Thus, when the button is clicked, the viewer will see the two alerts.

1. Abort Event

Triggered when: Media (such as an image or video) loading is stopped before completion.

Example: <img src="invalid_image.jpg" onabort="window.alert('Image loading aborted');" />

Explanation: If the browser fails to load the image (e.g., due to an incorrect URL) and the user stops loading, the onabort event fires, displaying an alert.

2. Blur Event

Triggered when: An element (like an input field) loses focus.

Example: <input type="text" onblur="window.alert('Input field lost focus');" />

Explanation: When the user clicks on the input field and then clicks somewhere else, the input loses focus, triggering the alert.

3. Change Event

Triggered when: The value of an input field is changed and the element loses focus.

Example: <input type="text" onchange="window.alert('Input value changed');" />
Explanation: If a user types something into the input field and clicks outside it, the onchange event fires and displays an alert.

4. Click Event
Triggered when: An element is clicked.
Example: <button onclick="window.alert('Button clicked');">Click Me</button>
Explanation: When the button is clicked, the onclick event fires and shows an alert.

5. Focus Event
Triggered when: An element (like an input field) gains focus.
Example: <input type="text" onfocus="window.alert('Input field focused');" />
Explanation: When a user clicks inside the input field or navigates to it using the Tab key, the event fires and displays an alert.

6. Keydown Event
Triggered when: A key is pressed down.
Example: <input type="text" onkeydown="window.alert('Key pressed down');" />
Explanation: The alert is triggered as soon as any key is pressed down in the input field.

7. Keypress Event
Triggered when: A key is pressed and held.
Example: <input type="text" onkeypress="window.alert('Key is being pressed');" />
Explanation: Similar to keydown, but this event is now deprecated. It's recommended to use keydown or keyup instead.

8. Keyup Event
Triggered when: A key is released after being pressed.
Example: <input type="text" onkeyup="window.alert('Key released');" />
Explanation: When a user releases a key after pressing it inside the input field, the alert appears.

9. Load Event
Triggered when: A page or media element fully loads.
Example: <body onload="window.alert('Page fully loaded');">
Explanation: The alert appears when the webpage has completely loaded.

10. Mousedown Event
Triggered when: A mouse button is pressed down on an element.
Example: <button onmousedown="window.alert('Mouse button pressed');">Press Me</button>
Explanation: The alert appears when the user presses the mouse button on the button.

11. Mousemove Event
Triggered when: The mouse moves over an element.
Example: <div onmousemove="window.alert('Mouse moved');" style="width:100px; height:100px; background-color:yellow;">
Move here
</div>
Explanation: The alert appears when the user moves their mouse over the yellow box.

12. Mouseover Event
Triggered when: The mouse pointer enters an element.
Example:
<div onmouseover="window.alert('Mouse over the box');" style="width:100px; height:100px; background-color:blue;">

Hover me
</div>
Explanation: When the user moves the mouse pointer over the blue box, the alert appears.
13. Mouseout Event
Triggered when: The mouse pointer leaves an element.
Example:
<div onmouseout="window.alert('Mouse left the box');" style="width:100px; height:100px; background-color:red;">
Leave me
</div>
Explanation: When the user moves the mouse out of the red box, the alert appears.
14. Mouseup Event
Triggered when: A mouse button is released after being pressed.
Example: <button onmouseup="window.alert('Mouse button released');">Release Me</button>
Explanation: The alert appears when the user releases the mouse button after clicking the button.
15. Reset Event
Triggered when: A form is reset.
Example:
<form onreset="window.alert('Form reset');">
<input type="text" />
<input type="reset" value="Reset Form" />
</form>
Explanation: The alert appears when the reset button is clicked.
16. Submit Event
Triggered when: A form is submitted.
Example:
<form onsubmit="window.alert('Form submitted'); return false;">
<input type="text" required />
<input type="submit" value="Submit Form" />
</form>
Explanation: The alert appears when the submit button is clicked. return false; prevents the form from actually submitting.
17. Unload Event
Triggered when: A user leaves the page or navigates away.
Example: <body onunload="window.alert('Page is unloading');">
Explanation: The alert appears when the user closes the browser tab or navigates to another page.
Creating Scripts Using Event Handlers: The Text Box Message
This script demonstrates how to use JavaScript event handlers to display a message in a text box when hovering over a link and clear it when the mouse moves away.
1. HTML Structure (textbox_message.html)
```
<body>
  <a href="message.html" id="msg_link">Get Message</a>
  <br /><br />
  <form>
    <input type="text" id="msg_box" />
  </form>
  <script type="text/javascript" src="textbox_message.js"></script>
</body>
```

- The link (<a>) has an id="msg_link".
- The text box (<input>) has an id="msg_box".
- JavaScript is included via textbox_message.js.

2. JavaScript File (textbox_message.js)

```
var message_text = "Help! I'm in a box!";
var message_link = document.getElementById("msg_link");
var message_box = document.getElementById("msg_box");
message_link.onmouseover = function() {
  message_box.value = message_text;
};
message_link.onmouseout = function() {
  message_box.value = "";
};
```

How it works?

- When hovering over the link, the onmouseover event sets the text box value to "Help! I'm in a box!".
- When the mouse moves away, the onmouseout event clears the text box.

The Button Link

JavaScript allows you to navigate to a different URL using the window.location property.

1. HTML Structure (button_link.html)

```
<form>
<input type="button" value="Go Searching!" id="btn1" />
</form>
<script type="text/javascript" src="button_link.js"></script>
```

- The button (<input>) has an id="btn1".

2. JavaScript File (button_link.js)

```
var web_page1 = "http://www.yahoo.com";
var b1 = document.getElementById("btn1");
b1.onclick = function() {
  window.location = web_page1;
};
```

- When the button is clicked, the browser redirects to http://www.yahoo.com.

3. Multiple Buttons Example

```
<form>
  <input type="button" value="Go Searching!" id="btn1" /><br /><br />
  <input type="button" value="HTML Help" id="btn2" /><br /><br />
  <input type="button" value="JavaScripts" id="btn3" />
</form>
var web_page1 = "http://www.yahoo.com";
var web_page2 = "http://www.pageresource.com";
var web_page3 = "http://www.javascriptcity.com";
document.getElementById("btn1").onclick = function() { window.location = web_page1; };
document.getElementById("btn2").onclick = function() { window.location = web_page2; };
document.getElementById("btn3").onclick = function() { window.location = web_page3; };
```

- Each button redirects to a different website when clicked.

Other Ways to Register Events

In addition to using event handler properties (e.g., onclick), two other methods can register events in JavaScript:

1. addEventListener() Method

Syntax: element.addEventListener('event_type', function_name, useCapture);

Example:

```
var web_page1 = "http://www.yahoo.com";
var b1 = document.getElementById("btn1");
b1.addEventListener('click', function() {
  window.location = web_page1;
}, false);
```

Key Points:

- Uses event type (e.g., 'click') instead of onclick.
- Accepts function names without parentheses or anonymous functions.
- Supports event capturing (true) and bubbling (false).
- To remove an event, use:

element.removeEventListener('event_type', function_name, useCapture);

2. attachEvent() Method

Syntax: element.attachEvent('event_handler', function_name);

Example:

```
var web_page1 = "http://www.yahoo.com";
var b1 = document.getElementById("btn1");
b1.attachEvent('onclick', function() {
  window.location = web_page1;
});
```

Key Points:

- Uses event handler ('onclick') instead of event type ('click').
- Only supports event bubbling (no capturing).
- To remove an event, use:

element.detachEvent('event_handler', function_name);

The Document Objects

Defining the Document Object

The document object is an object that is created by the browser for each new HTML page (document) that is viewed. By doing this, JavaScript gives you access to a number of properties and methods that can affect the document in various ways.

Using the Document Object Model

The Document Object Model (DOM) allows JavaScript (and other scripting languages) to access the structure of the document in the browser. Each document is made up of structured nodes (for example, the body tag would be a node, and any elements within the body element would be child nodes of the body element). With this structure in place, a scripting language can access the elements within the document in a number of ways, allowing for the modification of the elements within the document.

If you had the following HTML code, you could use JavaScript to access its structure:

```
<body>
<h1>My Page</h1>
<img src="myimage.jpg" alt="My Picture" />
</body>
```

The h1 and img elements are both child nodes of the body element. Each element also has its own nodes. The h1 element contains a text node as its child node (the text “My Page”), while the img element contains two attribute nodes (src=“myimage.jpg” and alt=“My Picture”). This type of structure is available throughout the document, so while this is a simple example, much more complex document structure trees could be drawn for most HTML pages. You have already been accessing the DOM using the document.getElementById() method to access elements in the document by their id attribute values. You can also get groups of elements using such methods as getElementsByTagName() or getElementsByClassName(). Accessing the DOM with JavaScript allows you to create more dynamic scripts that can alter elements within the document in reaction to user events. You are also able to create elements and nodes using certain JavaScript methods of the document object (such as createElement() or createTextNode()).

Properties of the document Object in JavaScript

The document object provides various properties to access and manipulate an HTML document. Below is a list of important properties with descriptions and examples.

1. Document Content & Elements

Property	**Description**	**Example**
activeElement	Returns the currently focused element.	console.log(document.activeElement);
body	Returns the <body> element of the document.	console.log(document.body);
head	Returns the <head> element of the document.	console.log(document.head);
documentElement	Returns the root <html> element.	console.log(document.documentElement);
childNodes	Returns a list of child nodes of the document.	console.log(document.childNodes);

2. Document Appearance & Colors

Property	**Description**	**Example**
alinkColor	Sets or returns the color of active links.	document.alinkColor = "red";
bgColor	Sets or returns the background color of the document.	document.bgColor = "lightblue";
fgColor	Sets or returns the foreground text color.	document.fgColor = "black";
linkColor	Sets or returns the color of hyperlinks.	document.linkColor = "blue";
vlinkColor	Sets or returns the color of visited links.	document.vlinkColor = "purple";

3. Document Metadata & Encoding

Property	**Description**	**Example**
characterSet	Returns the character encoding of the document (e.g., "UTF-8").	console.log(document.characterSet);
charset	Returns the character encoding (legacy).	console.log(document.charset);
inputEncoding	Returns the encoding used to parse the document.	console.log(document.inputEncoding);
contentType	Returns the MIME type of the document.	console.log(document.contentType);
defaultCharset	Default character set (browser-dependent).	console.log(document.defaultCharset);

4. Document Structure & Mode

Property	**Description**	**Example**
doctype	Returns the <!DOCTYPE> declaration.	console.log(document.doctype);
compatMode	Returns "CSS1Compat" for standards mode or "BackCompat" for quirks mode.	console.log(document.compatMode);
designMode	Enables or disables editing of the document.	document.designMode = "on";
dir	Sets or returns the text direction ("ltr" or "rtl").	document.dir = "rtl";

5. Document URL & Navigation

Property	**Description**	**Example**
URL	Returns the full URL of the document.	console.log(document.URL);
URLUnencoded	Returns the URL in an unencoded format (IE only).	console.log(document.URLUnencoded);
documentURIObject	Returns the document's URI (Firefox-specific).	console.log(document.documentURIObject);
domain	Returns the domain name of the document.	console.log(document.domain);
location	Returns the window.location object (URL info).	console.log(document.location.href);
protocol	Returns the protocol used ("http:", "https:").	console.log(document.location.protocol);
referrer	Returns the URL of the page that referred the user.	console.log(document.referrer);

6. Document Forms & Elements

Property	**Description**	**Example**
forms	Returns all <form> elements.	console.log(document.forms);
formName	Returns a form element by name (deprecated).	console.log(document.formName);

Property	Description	Example
images	Returns all <img> elements.	console.log(document.images);
links	Returns all <a> and <area> elements with an href.	console.log(document.links);
anchors	Returns all <a> elements with a name attribute.	console.log(document.anchors);

7. Embedded Content & Plugins

Property	Description	Example
embeds	Returns all <embed> elements.	console.log(document.embeds);
plugins	Returns all installed plugins (browser-dependent).	console.log(document.plugins);
applets	Returns all <applet> elements (deprecated).	console.log(document.applets);

8. Document Scripts & Styles

Property	Description	Example
scripts	Returns all <script> elements.	console.log(document.scripts);
styleSheets	Returns all stylesheets used in the document.	console.log(document.styleSheets);

9. Document Events & States

Property	Description	Example
readyState	Returns "loading", "interactive", or "complete".	console.log(document.readyState);
expando	Allows adding custom properties to the document.	document.expando = "Custom Value";
defaultView	Returns the window object associated with the document.	console.log(document.defaultView);
parentWindow	Returns the parent window object (IE only).	console.log(document.parentWindow);

10. File & Modification Properties

Property	Description	Example
lastModified	Returns the last modified date of the document.	console.log(document.lastModified);
fileCreatedDate	Returns the file creation date (browser-specific).	console.log(document.fileCreatedDate);
fileModifiedDate	Returns the file modification date (browser-specific).	console.log(document.fileModifiedDate);

11. Frames & Layers (Deprecated in Modern Browsers)

Property	Description	Example
frames	Returns the window.frames collection.	console.log(document.frames);
layers	Returns all <layer> elements (deprecated).	console.log(document.layers);

12. Miscellaneous Properties

Property	Description	Example
height	Returns the document's height (deprecated).	console.log(document.height);
width	Returns the document's width (deprecated).	console.log(document.width);
uniqueID	Returns a unique identifier for the document (IE only).	console.log(document.uniqueID);
tags	Returns elements by tag name (deprecated).	console.log(document.tags);

Example:

```
// Get document title and URL
console.log("Title:", document.title);
console.log("URL:", document.URL);
// Change the background color
document.bgColor = "lightblue";
// Get all images on the page
console.log("Images:", document.images);
// Check if the document is fully loaded
console.log("Ready State:", document.readyState);
```

Methods of the document Object in JavaScript

The document object provides numerous methods to manipulate the DOM, handle events, and interact with the webpage. Below is a categorized list of these methods along with descriptions and examples.

1. Document Creation & Modification

Method	Description	Example
createAttribute(name)	Creates a new attribute node.	let attr = document.createAttribute("class");
createAttributeNS(namespace, name)	Creates a new attribute node with namespace	let attr = document.createAttributeNS(null, "data-value");
createCDATASection(data)	Creates a CDATA section.	let cdata = document.createCDATASection("Some text");
createComment(data)	Creates a comment node.	let comment = document.createComment("This is a comment");

Method	**Description**	**Example**
createDocumentFragment()	Creates a lightweight document fragment.	let fragment = document.createDocumentFragment();
createElement(tagName)	Creates a new HTML element.	let div = document.createElement("div");
createElementNS(namespace, tagName)	Creates an element with namespace	let svg = document.createElementNS("http://www.w3.org/2000/svg", "svg");
createEntityReference(name)	Creates an entity reference	let entityRef = document.createEntityReference("amp");
createEvent(eventType)	Creates an event object.	let event = document.createEvent("MouseEvent");
createEventObject()	Creates an event object.	let evt = document.createEventObject();
createNodeIterator(root, whatToShow, filter, entityReferenceExpansion)	Creates a node iterator.	let iterator = document.createNodeIterator(document.body, NodeFilter.SHOW_ELEMENT, null, false);
createNSResolver(nodeResolver)	Creates an XML namespace resolver.	let resolver = document.createNSResolver(document.documentElement);
createProcessingInstruction(target, data)	Creates a processing instruction node.	let pi = document.createProcessingInstruction("xml-stylesheet", "href='style.css' type='text/css'");
createRange()	Creates a Range object.	let range = document.createRange();
createStyleSheet(url, index)	Creates a new stylesheet.	document.createStyleSheet("style.css");

Method	**Description**	**Example**
createTextNode(text)	Creates a text node.	let text = document.createTextNode("Hello, World!");
createTreeWalker(root, whatToShow, filter, entityReferenceExpansion)	Creates a tree walker object.	let walker = document.createTreeWalker(document.body, NodeFilter.SHOW_ELEMENT, null, false);

2. Event Handling

Method	**Description**	**Example**
attachEvent(event, function)	Attaches an event listener.	document.attachEvent("onclick", function() { alert("Clicked!"); });
detachEvent(event, function)	Removes an attached event.	document.detachEvent("onclick", myFunction);
addEventListener(event, function, useCapture)	Adds an event listener.	document.addEventListener("click", function() { alert("Clicked!"); });
removeEventListener(event, function, useCapture)	Removes an event listener.	document.removeEventListener("click", myFunction);

3. Element Selection & Manipulation

Method	**Description**	**Example**
elementFromPoint(x, y)	Returns the topmost element at the given coordinates.	let elem = document.elementFromPoint(100, 100);
getElementById(id)	Returns an element by its ID.	let elem = document.getElementById("myDiv");
getElementsByClassName(className)	Returns elements by class name.	let elems = document.getElementsByClassName("myClass");
getElementByName(name)	Returns elements by name	let elems = document.getElementsByName("username");

Method	**Description**	**Example**
	(deprecated).	
getElementsByTagName(tagName)	Returns elements by tag name.	let elems = document.getElementsByTagName("p");
getElementsByTagNameNS(namespace, tagName)	Returns elements by tag name within a namespace.	let elems = document.getElementsByTagNameNS("http://www.w3.org/1999/xhtml", "div");

4. Document Editing & Commands

Method	**Description**	**Example**
execCommand(command, showUI, value)	Executes a command (e.g., bold, italic, copy).	document.execCommand("bold");
queryCommandEnabled(command)	Checks if a command is enabled.	console.log(document.queryCommandEnabled("copy"));
queryCommandIndeterm(command)	Checks if a command is in an indeterminate state.	console.log(document.queryCommandIndeterm("bold"));
queryCommandState(command)	Checks if a command is active.	console.log(document.queryCommandState("bold"));
queryCommandSupported(command)	Checks if a command is supported.	console.log(document.queryCommandSupported("copy"));
queryCommandValue(command)	Gets the value of a command.	console.log(document.queryCommandValue("fontName"));

5. Document Focus & Selection

Method	**Description**	**Example**
hasFocus()	Checks if the document has focus.	console.log(document.hasFocus());
getSelection()	Returns the selected text.	let selection = document.getSelection();
setActive()	Sets focus on an element (IE only).	document.setActive();

6. Document Writing & Loading

Method	**Description**	**Example**
write(text)	Writes HTML to the document.	document.write("<h1>Hello</h1>");
writeln(text)	Writes HTML with a newline at the end.	document.writeln("<p>New line</p>");
open(mimeType, replace)	Opens a document for writing.	document.open();
close()	Closes a document opened for writing.	document.close();

7. Document Loading & Processing

Method	**Description**	**Example**
load(url)	Loads a new document.	document.load("data.xml");
mergeAttributes(sourceElement)	Copies attributes from another element.	document.mergeAttributes(document.getElementById("source"));
recalc()	Recalculates document styles.	document.recalc();
releaseCapture()	Releases the mouse capture.	document.releaseCapture();

Example:

```
// Creating a new paragraph element
let para = document.createElement("p");
para.textContent = "This is a new paragraph.";
document.body.appendChild(para);
// Selecting an element by ID
let header = document.getElementById("myHeader");
```

```
console.log(header.innerText);
// Writing into the document
document.write("<h1>Welcome!</h1>");
// Checking if document has focus
console.log(document.hasFocus());
// Executing a command
document.execCommand("bold");
```

Window object

The window object is created for each window that appears on the screen. A window can be the main window, a frame set or individual frame, or even a new window created with JavaScript. It differs from the document object in that the window object contains the document object (as well as many other objects, such as history, navigator, and so on). This object makes available for use in your scripts a number of new properties and methods that are directly under the window object.

Properties of the window Object in JavaScript

The window object represents the browser window and is the global object in JavaScript. It contains several properties that allow interaction with the browser.

1. Window Size and Position

Property	**Description**	**Example**
innerWidth	Returns the width of the window's viewport.	console.log(window.innerWidth);
innerHeight	Returns the height of the window's viewport.	console.log(window.innerHeight);
outerWidth	Returns the width of the entire browser window.	console.log(window.outerWidth);
outerHeight	Returns the height of the entire browser window.	console.log(window.outerHeight);
screenX / screenLeft	Returns the horizontal position of the window relative to the screen.	console.log(window.screenX);
screenY / screenTop	Returns the vertical position of the window relative to the screen.	console.log(window.screenY);

2. Document and Location Information

Property	**Description**	**Example**
document	Refers to the document object (DOM).	console.log(window.document.title);
location	Returns the current URL and allows navigation.	console.log(window.location.href);
history	Provides access to the browser's history.	console.log(window.history.length);
navigator	Provides information about the browser.	console.log(window.navigator.userAgent);

3. Screen Information

Property	Description	Example
screen	Provides information about the user’s screen.	console.log(window.screen.width);
screen.width	Screen width in pixels.	console.log(window.screen.width);
screen.height	Screen height in pixels.	console.log(window.screen.height);
screen.availWidth	Available screen width (excluding taskbars).	console.log(window.screen.availWidth);
screen.availHeight	Available screen height.	console.log(window.screen.availHeight);
screen.colorDepth	Returns the color depth of the screen.	console.log(window.screen.colorDepth);

4. Window and Browser Status

Property	Description	Example
closed	Returns true if the window is closed.	console.log(window.closed);
frames	Returns an array of frames in the window.	console.log(window.frames.length);
length	Returns the number of frames in the window.	console.log(window.length);
name	Gets or sets the name of the window.	window.name = "MyWindow";
opener	Returns a reference to the window that opened this window.	console.log(window.opener);
parent	Returns the parent window (for iframes).	console.log(window.parent);
top	Returns the topmost parent window.	console.log(window.top);

5. Timers and User Interaction

Property	Description	Example
setTimeout()	Calls a function after a specified delay.	setTimeout(() => console.log("Hello"), 2000);
setInterval()	Calls a function repeatedly at specified intervals.	setInterval(() => console.log("Tick"), 1000);
clearTimeout()	Cancels a timeout set with setTimeout().	clearTimeout(myTimeout);
clearInterval()	Cancels an interval set with setInterval().	clearInterval(myInterval);

6. Miscellaneous Properties

Property	Description	Example
localStorage	Provides access to local storage.	window.localStorage.setItem("key", "value");

Property	Description	Example
sessionStorage	Provides access to session storage.	window.sessionStorage.setItem("key", "value");
console	Provides access to the browser console.	console.log("Hello!");
alert()	Displays an alert box.	window.alert("Hello!");
confirm()	Displays a confirmation box.	window.confirm("Are you sure?");
prompt()	Displays a prompt box.	window.prompt("Enter your name:");

Example:

```
console.log("Window Width: " + window.innerWidth);
console.log("Window Height: " + window.innerHeight);
console.log("Current URL: " + window.location.href);
console.log("Browser User Agent: " + window.navigator.userAgent);
```

Methods of the window Object in JavaScript

The window object provides various methods to interact with the browser. Below is a categorized list of commonly used methods:

1. Window Control Methods

Method	Description	Example
open()	Opens a new browser window or tab.	window.open("https://google.com", "_blank");
close()	Closes the current window (only if opened via script).	window.close();
moveTo(x, y)	Moves the window to a specific position on the screen.	window.moveTo(100, 100);
resizeTo(width, height)	Resizes the window to specified dimensions.	window.resizeTo(800, 600);

2. Navigation Methods

Method	Description	Example
location.assign(url)	Loads a new document.	window.location.assign("https://example.com");
location.replace(url)	Replaces the current page (doesn't store history).	window.location.replace("https://example.com");
location.reload(force)	Reloads the current page (true forces reload from server).	window.location.reload(true);
history.back()	Navigates to the previous page.	window.history.back();
history.forward()	Navigates to the next page.	window.history.forward();

Method	Description	Example
history.go(n)	Navigates n steps in history (-1 for back, 1 for forward).	window.history.go(-1);

3. Dialog Box Methods

Method	Description	Example
alert(message)	Displays an alert box with a message.	window.alert("Hello, World!");
confirm(message)	Displays a confirmation box with OK/Cancel buttons.	let result = window.confirm("Are you sure?");
prompt(message, default)	Displays a prompt box for user input.	let name = window.prompt("Enter your name:", "Guest");

4. Timer Methods

Method	Description	Example
setTimeout(function, delay)	Executes a function after a delay (in milliseconds).	setTimeout(() => alert("Hello!"), 2000);
clearTimeout(id)	Cancels a timeout set with setTimeout().	clearTimeout(myTimeout);
setInterval(function, delay)	Repeats execution of a function at set intervals.	setInterval(() => console.log("Tick"), 1000);
clearInterval(id)	Cancels an interval set with setInterval().	clearInterval(myInterval);

5. Screen and Window Size Methods

Method	Description	Example
scrollTo(x, y)	Scrolls to a specific position.	window.scrollTo(0, 100);
scrollBy(x, y)	Scrolls the window relative to its current position.	window.scrollBy(0, 50);
getComputedStyle(element)	Gets the computed styles of an element.	let styles = window.getComputedStyle(document.body);

6. Storage Methods

Method	Description	Example
localStorage.setItem(key, value)	Stores data in local storage.	window.localStorage.setItem("username", "John");

Method	Description	Example
localStorage.getItem(key)	Retrieves data from local storage.	let user = window.localStorage.getItem("username");
localStorage.removeItem(key)	Removes an item from local storage.	window.localStorage.removeItem("username");
sessionStorage.setItem(key, value)	Stores data for the session.	window.sessionStorage.setItem("token", "abc123");
sessionStorage.getItem(key)	Retrieves session data.	let token = window.sessionStorage.getItem("token");
sessionStorage.removeItem(key)	Removes an item from session storage.	window.sessionStorage.removeItem("token");

7. Event Listener Methods

Method	Description	Example
addEventListener(event, function)	Adds an event listener to an element.	window.addEventListener("resize", () => console.log("Resized!"));
removeEventListener(event, function)	Removes an event listener.	window.removeEventListener("resize", resizeFunction);

Example:

```
// Open a new window
let newWindow = window.open("https://example.com", "_blank", "width=500,height=500");
// Show an alert
window.alert("Hello!");
// Reload the page
window.location.reload();
// Set a timeout to log a message after 2 seconds
let timer = setTimeout(() => console.log("Timeout executed"), 2000);
// Cancel the timeout
clearTimeout(timer);
// Add an event listener for window resize
window.addEventListener("resize", () => console.log("Window resized!"));
```

What is the Math Object?

The Math object is a predefined JavaScript object. The Math object is used for mathematical purposes to give you the values of certain mathematical constants or to perform certain operations when you use a method function.

Properties of the Math Object in JavaScript

The Math object in JavaScript provides several built-in properties to perform mathematical operations. These properties return important mathematical constants.

List of Math Properties

Property	**Description**	**Value**
Math.E	Euler's number (base of natural logarithm)	≈ 2.718
Math.PI	Ratio of a circle's circumference to its diameter	≈ 3.14159
Math.SQRT2	Square root of 2	≈ 1.414
Math.SQRT1_2	Square root of 1/2	≈ 0.707
Math.LN2	Natural logarithm of 2	≈ 0.693
Math.LN10	Natural logarithm of 10	≈ 2.302
Math.LOG2E	Base-2 logarithm of E	≈ 1.442
Math.LOG10E	Base-10 logarithm of E	≈ 0.434

Example:

```
console.log(Math.E);       // Output: 2.718
console.log(Math.PI);      // Output: 3.141592653589793
console.log(Math.SQRT2);    // Output: 1.4142135623730951
console.log(Math.LN10);     // Output: 2.302585092994046
console.log(Math.LOG2E);    // Output: 1.4426950408889634
```

Methods of the Math Object in JavaScript

The Math object in JavaScript provides several built-in methods for performing mathematical operations. These methods help in rounding numbers, generating random values, finding square roots, and more.

1. Rounding Methods

Method	**Description**	**Example**	**Output**
Math.round(x)	Rounds x to the nearest integer	Math.round(4.6)	5
Math.floor(x)	Rounds x down to the nearest integer	Math.floor(4.9)	4
Math.ceil(x)	Rounds x up to the nearest integer	Math.ceil(4.2)	5
Math.trunc(x)	Removes the decimal part (returns the integer part)	Math.trunc(4.9)	4

2. Power & Square Root Methods

Method	**Description**	**Example**	**Output**
Math.pow(x, y)	Returns x raised to the power y	Math.pow(2, 3)	8
Math.sqrt(x)	Returns the square root of x	Math.sqrt(16)	4
Math.cbrt(x)	Returns the cube root of x	Math.cbrt(27)	3

3. Absolute & Sign Methods

Method	**Description**	**Example**	**Output**
Math.abs(x)	Returns the absolute value of x	Math.abs(-10)	10
Math.sign(x)	Returns 1 for positive, -1 for negative, and 0 for zero	Math.sign(-5)	-1

4. Trigonometric Methods

Method	Description	Example	Output
Math.sin(x)	Returns the sine of x (in radians)	Math.sin(Math.PI/2)	1
Math.cos(x)	Returns the cosine of x (in radians)	Math.cos(0)	1
Math.tan(x)	Returns the tangent of x (in radians)	Math.tan(Math.PI/4)	1

Inverse Trigonometric Functions

Method	Description	Example
Math.asin(x)	Returns the arcsine of x	Math.asin(0.5)
Math.acos(x)	Returns the arccosine of x	Math.acos(1)
Math.atan(x)	Returns the arctangent of x	Math.atan(1)
Math.atan2(y, x)	Returns the arctangent of y/x	Math.atan2(1, 1)

5. Logarithmic & Exponential Methods

Method	Description	Example	Output
Math.log(x)	Returns the natural logarithm (base e) of x	Math.log(10)	2.302
Math.log2(x)	Returns the base-2 logarithm of x	Math.log2(8)	3
Math.log10(x)	Returns the base-10 logarithm of x	Math.log10(1000)	3
Math.exp(x)	Returns e raised to the power x	Math.exp(1)	2.718

6. Random Number Methods

Method	Description	Example	Output
Math.random()	Returns a random number between 0 and 1	Math.random()	0.675
Math.random() * (max - min) + min	Returns a random number in a range	Math.random() * (10 - 5) + 5	Between 5 and 10

7. Min & Max Methods

Method	Description	Example	Output
Math.min(x, y, z, ...)	Returns the smallest value	Math.min(3, 7, 1, 9)	1
Math.max(x, y, z, ...)	Returns the largest value	Math.max(3, 7, 1, 9)	9

Example:

```
console.log(Math.round(4.6));  // 5
console.log(Math.sqrt(25));    // 5
console.log(Math.abs(-10));    // 10
console.log(Math.sin(Math.PI/2)); // 1
console.log(Math.random());    // Random number between 0 and 1
console.log(Math.max(10, 20, 30)); // 30
```

Understanding the Number Object

The Number object is another predefined JavaScript object that offers several useful properties and methods for use with numbers.

Properties of the Number Object in JavaScript

The Number object in JavaScript has several useful properties that provide constant values related to numeric operations. These properties help in working with numbers, especially for handling limits, precision, and special values like infinity and NaN.

1. Number.EPSILON

- Represents the smallest difference between two representable floating-point numbers.
- Useful for comparing floating-point numbers.

Value: ≈ 2.220446049250313e-16

Example: console.log(Number.EPSILON); // 2.220446049250313e-16

2. Number.MAX_VALUE

- Represents the largest positive finite number in JavaScript.

Value: ≈ 1.7976931348623157e+308

Example: console.log(Number.MAX_VALUE); // 1.7976931348623157e+308

3. Number.MIN_VALUE

- Represents the smallest positive number (closest to zero) in JavaScript.

Value: ≈ 5e-324

Example: console.log(Number.MIN_VALUE); // 5e-324

4. Number.MAX_SAFE_INTEGER

- Represents the largest exact integer that can be safely represented in JavaScript (2^53 - 1).

Value: 9007199254740991

Example: console.log(Number.MAX_SAFE_INTEGER); // 9007199254740991

5. Number.MIN_SAFE_INTEGER

- Represents the smallest exact integer that can be safely represented in JavaScript (-(2^53 - 1)).

Value: -9007199254740991

Example: console.log(Number.MIN_SAFE_INTEGER); // -9007199254740991

6. Number.POSITIVE_INFINITY

- Represents positive infinity (Infinity).
- A value greater than Number.MAX_VALUE is treated as Infinity.

Example: console.log(Number.POSITIVE_INFINITY); // Infinity
console.log(1 / 0); // Infinity

7. Number.NEGATIVE_INFINITY

- Represents negative infinity (-Infinity).
- A value smaller than Number.MIN_VALUE is treated as -Infinity.

Example:

```
console.log(Number.NEGATIVE_INFINITY); // -Infinity
console.log(-1 / 0); // -Infinity
```

8. Number.NaN (Not-a-Number)

- Represents a value that is not a valid number.
- Used when a mathematical operation fails to return a valid number.

Example:

```
console.log(Number.NaN); // NaN
console.log(0 / 0); // NaN
console.log(Math.sqrt(-1)); // NaN
```

9. constructor Property

- Refers to the constructor function that creates Number objects.
- By default, it returns the built-in Number function.

Example:

```
let num = new Number(42);
console.log(num.constructor); // [Function: Number]
```

Usage: Useful when checking the constructor of an object to verify its type.

10. prototype Property

- Represents the prototype object of Number, from which all Number instances inherit properties and methods.
- Used to add custom methods to all Number objects.

Example:

```
Number.prototype.square = function() {
  return this * this;
};
let num1 = new Number(5);
console.log(num1.square()); // 25
```

Usage: Helpful for extending the functionality of all Number instances. However, modifying built-in prototypes is not recommended as it can cause conflicts with other scripts.

Methods of the Number Object in JavaScript

The Number object in JavaScript provides several built-in methods to manipulate and format numbers.

1. toString(radix)

- Converts a number to a string representation in a specified base (radix).
- radix can be from 2 to 36 (default is 10).

Example:

```
let num = 255;
console.log(num.toString());     // "255" (default: base 10)
console.log(num.toString(2));    // "11111111" (binary)
console.log(num.toString(16));   // "ff" (hexadecimal)
```

2. toFixed(digits)

- Formats a number with a fixed number of decimal places.
- digits specifies the number of decimal places.

Example:

```
let num = 5.6789;
console.log(num.toFixed(2));  // "5.68"
console.log(num.toFixed(0));  // "6"
```

3. toExponential(digits)

- Converts a number to scientific notation.
- digits specifies the number of decimal places (optional).

Example:

```
let num = 12345;
console.log(num.toExponential());    // "1.2345e+4"
console.log(num.toExponential(2));   // "1.23e+4"
```

4. toPrecision(digits)

- Formats a number to a specified total number of significant digits.

Example:

```
let num = 5.6789;
console.log(num.toPrecision(3));  // "5.68"
console.log(num.toPrecision(2));  // "5.7"
console.log(num.toPrecision(1));  // "6"
```

5. valueOf()

- Returns the primitive value of a Number object.

Example:

```
let num = new Number(10);
console.log(num.valueOf());  // 10
```

6. Number.isInteger(value)

- Checks if a value is an integer.

Example:

```
console.log(Number.isInteger(10));    // true
console.log(Number.isInteger(10.5));  // false
console.log(Number.isInteger("10"));  // false
```

7. Number.isFinite(value)

- Checks if a value is a finite number (not Infinity, -Infinity, or NaN).

Example:

```
console.log(Number.isFinite(100));        // true
console.log(Number.isFinite(Infinity));   // false
console.log(Number.isFinite(NaN));        // false
console.log(Number.isFinite("100"));      // false
```

8. Number.isNaN(value)

- Checks if a value is NaN (Not-a-Number).

Example:

```
console.log(Number.isNaN(NaN));      // true
console.log(Number.isNaN(10));       // false
console.log(Number.isNaN("hello"));  // false
```

9. Number.parseFloat(value)

- Converts a string into a floating-point number.

Example:

```
console.log(Number.parseFloat("10.99"));   // 10.99
console.log(Number.parseFloat("10px"));    // 10
console.log(Number.parseFloat("abc"));     // NaN
```

10. Number.parseInt(value, radix)

- Converts a string into an integer with an optional radix.

Example:

```
console.log(Number.parseInt("100"));      // 100 (default: base 10)
console.log(Number.parseInt("100", 2));   // 4 (binary to decimal)
console.log(Number.parseInt("10px"));     // 10
console.log(Number.parseInt("abc"));      // NaN
```

Using the Date Object

The Date object is another predefined JavaScript object. It enables you to set certain time values and to get certain time values that you can use in your scripts. The Date object in JavaScript has two important properties related to its structure:

1. constructor Property

The constructor property returns the function that created the instance of the Date object.
For Date objects, the constructor is the built-in Date function itself.

Example:

```
let date = new Date();
console.log(date.constructor);  // Outputs: [Function: Date]
console.log(date.constructor === Date);  // true
```

2. prototype Property

The prototype property is an object that allows adding new methods or properties to all Date instances.
This property is available on Date, not on individual Date instances.
Any new methods or properties added to Date.prototype will be inherited by all Date objects.

Example: Adding a Custom Method to Date.prototype

```
Date.prototype.getYearMonth = function() {
   return this.getFullYear() + "-" + (this.getMonth() + 1);
};
let myDate = new Date();
console.log(myDate.getYearMonth());  // Example output: "2025-3"
```

Methods of the Date Object in JavaScript

The Date object provides various methods to work with dates and times. These methods can be classified into different categories:

1. Creating and Manipulating Dates

Method	**Description**	**Example**
new Date()	Creates a new date object with the current date and time.	let date = new Date();
new Date(value)	Creates a date from a timestamp (milliseconds since Jan 1, 1970).	let date = new Date(1700000000000);
new Date(year, month, day, hours, minutes, seconds, ms)	Creates a date with specific components.	let date = new Date(2025, 2, 22, 10, 30, 15);
Date.now()	Returns the current timestamp (milliseconds since Jan 1, 1970).	console.log(Date.now());

2. Getting Date Components

Method	**Description**	**Example Output**
getFullYear()	Gets the four-digit year.	date.getFullYear(); // 2025
getMonth()	Gets the month (0-11).	date.getMonth(); // 2 (March)
getDate()	Gets the day of the month (1-31).	date.getDate(); // 22
getDay()	Gets the day of the week (0-6, where 0 is Sunday).	date.getDay(); // 6 (Saturday)

Method	Description	Example Output
getHours()	Gets the hour (0-23).	date.getHours(); // 10
getMinutes()	Gets the minutes (0-59).	date.getMinutes(); // 30
getSeconds()	Gets the seconds (0-59).	date.getSeconds(); // 15
getMilliseconds()	Gets the milliseconds (0-999).	date.getMilliseconds(); // 500
getTime()	Gets the timestamp (milliseconds since Jan 1, 1970).	date.getTime(); // 1745347200000

3. Setting Date Components

Method	Description	Example
setFullYear(year, month, day)	Sets the year. Optionally, you can set the month and day.	date.setFullYear(2030, 5, 10);
setMonth(month)	Sets the month (0-11).	date.setMonth(4); // May
setDate(day)	Sets the day of the month (1-31).	date.setDate(15);
setHours(hour)	Sets the hour (0-23).	date.setHours(14);
setMinutes(min)	Sets the minutes (0-59).	date.setMinutes(45);
setSeconds(sec)	Sets the seconds (0-59).	date.setSeconds(30);
setMilliseconds(ms)	Sets the milliseconds (0-999).	date.setMilliseconds(200);
setTime(timestamp)	Sets the date using a timestamp.	date.setTime(1745347200000);

4. Converting Dates to Strings

Method	Description	Example Output
toDateString()	Returns the date in a human-readable format.	"Sat Mar 22 2025"
toTimeString()	Returns the time portion of the date.	"10:30:15 GMT+0000"
toISOString()	Converts to an ISO 8601 string format.	"2025-03-22T10:30:15.000Z"
toUTCString()	Converts to a UTC string.	"Sat, 22 Mar 2025 10:30:15 GMT"
toLocaleDateString()	Returns the date in a localized format.	"3/22/2025"
toLocaleTimeString()	Returns the time in a localized format.	"10:30:15 AM"

5. Date Comparison Methods

Method	Description	Example
valueOf()	Returns the primitive value of the Date object (same as getTime()).	console.log(date.valueOf());
getTimezoneOffset()	Returns the difference in minutes between UTC and local time.	console.log(date.getTimezoneOffset());

Example:

```
let date = new Date();
// Get components
console.log(date.getFullYear());  // 2025
console.log(date.getMonth());  // 2 (March)
console.log(date.getDate());  // 22
// Set components
date.setFullYear(2030);
date.setMonth(5);  // June
console.log(date.toDateString());  // "Fri Jun 22 2030"
// Convert to string
console.log(date.toISOString());  // "2030-06-22T10:30:15.000Z"
```