**JavaScript setTimeout() && setInterval() Method 🕒**

## **JavaScript setTimeout() Method ⏰**

* Executes a function after waiting a specified number of milliseconds.

**Syntax:**

setTimeout(function, milliseconds);

### **Parameters:**

* **function**: Function to be executed.
* **milliseconds**: Number of milliseconds to wait before execution.

### **Example:1**

<button onclick="setTimeout(gfg, 2000);">

Press me

</button>

<script>

function gfg() {

alert('Welcome to Nawjiwan Homes');

}

</script>

**Output:** After a pause of 2 seconds, an alert box will pop up when the button is pressed.

### **Example:2 Displaying a Message After Delay:**

setTimeout(function() {

console.log("Hello after 2 seconds!");

}, 2000)

**Example:3 Displaying a Message After Delay:**

setTimeout(function() {

alert("This will pop up after 3 seconds!");

}, 3000);

**Example:4 Changing Text After Delay:**

setTimeout(function() {

document.getElementById("message").innerText = "Delayed Message";

}, 4000);

**Example:5 Executing a Named Function After Delay:**

function greet() {

console.log("Hello there!");

}

setTimeout(greet, 5000);

**Real-Life Example - Email Submission Notification:**

Imagine a scenario where a user submits a form on a website, and you want to display a notification message after a short delay to confirm that the email has been successfully sent.

// Assume form submission code here

setTimeout(function() {

displayNotification("Email sent successfully!");

}, 3000);

function displayNotification(message) {

// Code to display notification message to the user

console.log(message);

}

In this example, **setTimeout()** is used to schedule the display of a notification message after a 3-second delay, providing a better user experience by confirming the action.

### **Additional Example:**

**Delayed Button Click Response:**

Suppose you have a button on a web page, and you want to change its appearance a few seconds after it's clicked to provide visual feedback to the user.

document.getElementById("myButton").addEventListener("click", function() {

// Button click event handler

console.log("Button clicked!");

setTimeout(function() {

document.getElementById("myButton").style.backgroundColor = "green";

}, 2000);

});

clearTimeout(value);

### clearTimeout() **Function:**

The **clearTimeout()** function is employed to cancel the execution of a scheduled **setTimeout()** function, effectively preventing it from running.

clearTimeout(timeoutID);

### **Parameters:**

* **timeoutID**: The ID returned by the **setTimeout()** function that identifies the timer to be canceled.

### **Example Usage:**

Suppose we schedule a **setTimeout()** function to display a message after 3 seconds, but then decide we want to cancel it before it executes. We can do so using **clearTimeout()**.

// Schedule a setTimeout() function

let timeoutID = setTimeout(function() {

console.log("This message will never be displayed!");

}, 3000);

// Cancel the timeout

clearTimeout(timeoutID);

console.log("Timeout canceled!");

In this example, the message scheduled to be displayed after 3 seconds will never be shown because **clearTimeout()** is called immediately after scheduling the **setTimeout()** function, preventing its execution.

By utilizing **clearTimeout()**, we gain greater control over the execution of asynchronous tasks, enhancing the flexibility and responsiveness of our JavaScript applications.

### **Example 1: Cancelling a Scheduled Message Display**

Suppose we schedule a message to be displayed after 3 seconds, but then decide to cancel it before it executes.

javascriptCopy code

let timeoutID = setTimeout(function() {

console.log("This message will never be displayed!");

}, 3000);

// Cancel the timeout

clearTimeout(timeoutID);

console.log("Timeout canceled!");

### **Example 2: Delayed Alert with Cancellation**

We schedule an alert to pop up after 5 seconds, but provide an option for the user to cancel it before it shows up.

javascriptCopy code

let timeoutID = setTimeout(function() {

alert("Hello! This alert will be canceled if you click OK within 3 seconds.");

}, 5000);

// Cancel the timeout if the user clicks OK before 3 seconds

setTimeout(function() {

clearTimeout(timeoutID);

console.log("Alert canceled!");

}, 3000);

### **Example 3: Cancelling a Delayed Function Call**

We schedule a function to execute after 2 seconds but then cancel it before execution.

javascriptCopy code function delayedFunction() { console.log("This function will never be called!"); } let timeoutID = setTimeout(delayedFunction, 2000); // Cancel the delayed function call clearTimeout(timeoutID); console.log("Delayed function call canceled!");

javascriptCopy code

function delayedFunction() {

console.log("This function will never be called!");

}

let timeoutID = setTimeout(delayedFunction, 2000);

// Cancel the delayed function call

clearTimeout(timeoutID);

console.log("Delayed function call canceled!");

# ****JavaScript setInterval() Method 🔄****

The JavaScript **setInterval()** method is employed to repeatedly execute a specified function or code snippet at fixed intervals. It's particularly useful for creating animations, polling for updates, and implementing periodic tasks.

### **Syntax:**

javascriptCopy code

setInterval(function, milliseconds);

​

#### Parameters:

**function**: The function or code snippet to be executed repeatedly.

**milliseconds**: The interval between each execution, in milliseconds.

**param1**, **param2**, ...: (Optional) Parameters passed to the function each time it is executed.

#### Description:

The **setInterval()** function schedules the execution of the specified function at regular intervals, determined by the specified delay.

It returns an identifier (a unique integer value) that can be used to stop the repeated execution of the function using the **clearInterval()** function.

#### Example Usages:

**Displaying a Message Every Second:**

javascriptCopy code

setInterval(function() {

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

}, 1000);

**Updating Time Every Second:**

javascriptCopy code

setInterval(function() {

let currentTime = new Date();

document.getElementById("clock").innerText = currentTime.toLocaleTimeString();

}, 1000);

**Animating a Carousel Every 3 Seconds:**

javascriptCopy code

let currentIndex = 0;

let images = ["image1.jpg", "image2.jpg", "image3.jpg"];

setInterval(function() {

currentIndex = (currentIndex + 1) % images.length;

document.getElementById("carousel").src = images[currentIndex];

}, 3000);

**Polling for Updates Every 5 Seconds:**

javascriptCopy code

setInterval(function() {

// Code to check for updates from server

console.log("Checking for updates...");

}, 5000);

**Running a Scheduled Task with :**

javascriptCopy code

let greeting = "Hello, ";

let name = "John";

setInterval(function(greeting, name) {

console.log(greeting + name + "!");

}, 2000);

### **Real-Life Example:**

Suppose you are developing a dashboard for a weather application. You want to periodically fetch the latest weather data from an API and update the dashboard to display the current temperature, humidity, and other relevant information.

javascriptCopy code

let timeLeft = 60;

function countdown() {

if (timeLeft > 0) {

console.log(timeLeft + " seconds remaining.");

timeLeft--;

} else {

console.log("Time's up!");

clearTimeout(timeoutID);

}

}

let timeoutID = setInterval(countdown, 1000);

// Cancel the countdown timer if the player exits early

document.getElementById("exitButton").addEventListener("click", function() {

clearInterval(timeoutID);

console.log("Timer canceled by player.");

});

These examples illustrate how clearInterval\*\*()\*\* can be used to manage scheduled tasks effectively, providing control and flexibility in JavaScript programming.