# JavaScript Objects

## Real Life Objects, Properties, and Methods

In real life, a car is an **object**.

A car has **properties** like weight and color, and **methods** like start and stop:

|  |  |  |
| --- | --- | --- |
| **Object** | **Properties** | **Methods** |
|  | car.name = Fiat  car.model = 500  car.weight = 850kg  car.color = white | car.start()  car.drive()  car.brake()  car.stop() |

All cars have the same **properties**, but the property **values** differ from car to car.

All cars have the same **methods**, but the methods are performed **at different times**.

## JavaScript Objects

You have already learned that JavaScript variables are containers for data values.

This code assigns a **simple value** (Fiat) to a **variable** named car:

let car = "Fiat";

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

const car = {type:"Fiat", model:"500", color:"white"};

The values are written as **name:value** pairs (name and value separated by a colon).

It is a common practice to declare objects with the const keyword.

## Object Definition

You define (and create) a JavaScript object with an object literal:

### **Example**

const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

Spaces and line breaks are not important. An object definition can span multiple lines:

### **Example**

const person = {  
  firstName: "John",  
  lastName: "Doe",  
  age: 50,  
  eyeColor: "blue"  
};

## Object Properties

The **name:values** pairs in JavaScript objects are called **properties**:

|  |  |
| --- | --- |
| **Property** | **Property Value** |
| firstName | John |
| lastName | Doe |
| Age | 50 |
| eyeColor | Blue |

## Accessing Object Properties

You can access object properties in two ways:

*objectName.propertyName*

or

*objectName["propertyName"]*

### **Example1**

person.lastName;

### **Example2**

person["lastName"];

JavaScript objects are containers for **named values**called properties.

## Object Methods

Objects can also have **methods**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

|  |  |
| --- | --- |
| **Property** | **Property Value** |
| firstName | John |
| lastName | Doe |
| age | 50 |
| eyeColor | Blue |
| fullName | function() {return this.firstName + " " + this.lastName;} |

A method is a function stored as a property.

### **Example**

const person = {  
  firstName: "John",  
  lastName : "Doe",  
  id       : 5566,  
  fullName : function() {  
    return this.firstName + " " + this.lastName;  
  }  
};

In the example above, this refers to the **person object**.

I.E. **this.firstName** means the **firstName** property of **this**.

I.E. **this.firstName** means the **firstName** property of **person**.

## What is this?

In JavaScript, the this keyword refers to an **object**.

**Which** object depends on how this is being invoked (used or called).

The this keyword refers to different objects depending on how it is used:

|  |
| --- |
| In an object method, this refers to the **object**. |
| Alone, this refers to the **global object**. |
| In a function, this refers to the **global object**. |
| In a function, in strict mode, this is undefined. |
| In an event, this refers to the **element** that received the event. |
| Methods like call(), apply(), and bind() can refer this to **any object**. |

## Note

this is not a variable. It is a keyword. You cannot change the value of this.

## See Also:

## The **this** Keyword

In a function definition, this refers to the "owner" of the function.

In the example above, this is the **person object** that "owns" the fullName function.

In other words, this.firstName means the firstName property of **this object**.

## Accessing Object Methods

You access an object method with the following syntax:

*objectName.methodName()*

### **Example**

name = person.fullName();

If you access a method **without** the () parentheses, it will return the **function definition**:

### **Example**

name = person.fullName;

## Do Not Declare Strings, Numbers, and Booleans as Objects!

When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

x = new String();        // Declares x as a String object  
y = new Number();        // Declares y as a Number object  
z = new Boolean();       // Declares z as a Boolean object

Avoid String, Number, and Boolean objects. They complicate your code and slow down execution speed.

# JavaScript Objects

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

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

JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

## Creating Objects in JavaScript

There are 3 ways to create objects.

1. By object literal
2. By creating instance of Object directly (using new keyword)
3. By using an object constructor (using new keyword)

## 1) JavaScript Object by object literal

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

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

As you can see, property and value is separated by : (colon).

Let’s see the simple example of creating object in JavaScript.

**<script>**

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

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

**</script>**

#### **Output of the above example**

102 Shyam Kumar 40000

## 2) By creating instance of Object

The syntax of creating object directly is given below:

var objectname=new Object();

Here, **new keyword** is used to create object.

Let’s see the example of creating object directly.

**<script>**

var emp=new Object();

emp.id=101;

emp.name="Ravi Malik";

emp.salary=50000;

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

**</script>**

#### **Output of the above example**

101 Ravi 50000

## 3) By using an Object constructor

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

The **this keyword** refers to the current object.

The example of creating object by object constructor is given below.

**<script>**

function emp(id,name,salary)

{

this.id=id;

this.name=name;

this.salary=salary;

}

Emp();

e=new emp(103,"Vimal Jaiswal",30000);

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

**</script>**

#### **Output of the above example**

103 Vimal Jaiswal 30000

## Defining method in JavaScript Object

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.

The example of defining method in object is given below.

**<script>**

function emp(id,name,salary){

this.id=id;

this.name=name;

this.salary=salary;

this.changeSalary=changeSalary;

function changeSalary(otherSalary){

this.salary=otherSalary;

}

}

e=new emp(103,"Sonoo Jaiswal",30000);

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

e.changeSalary(45000);

document.write("**<br>**"+e.id+" "+e.name+" "+e.salary);

**</script>**

#### **Output of the above example**

103 Sonoo Jaiswal 30000  
103 Sonoo Jaiswal 45000

**JavaScript Date Object**

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

You can use different Date constructors to create date object. It provides methods to get and set day, month, year, hour, minute and seconds.

Constructor

You can use 4 variant of Date constructor to create date object.

1. Date()
2. Date(milliseconds)
3. Date(dateString)
4. Date(year, month, day, hours, minutes, seconds, milliseconds)

JavaScript Date Methods

Let's see the list of JavaScript date methods with their description.

|  |  |  |
| --- | --- | --- |
| **Methods** | **Description** | **Example** |
| [getDate()](https://www.javatpoint.com/javascript-date-getdate-method) | It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of local time. | <script type = "text/javascript">  var dt = new Date("December 25, 1995 23:15:00");  document.write("getDate() : " + dt.getDate() );  </script>  getDate() : 25 |
| [getDay()](https://www.javatpoint.com/javascript-date-getday-method) | It returns the integer value between 0 and 6 that represents the day of the week on the basis of local time. | <script type = "text/javascript">  var dt = new Date("December 25, 1995 23:15:00");  document.write("getDay() : " + dt.getDay() );  </script>  getDay() : 1 |
| [getFullYears()](https://www.javatpoint.com/javascript-date-getutcfullyear-method) | It returns the integer value that represents the year on the basis of local time. | <script type = "text/javascript">  var dt = new Date("December 25, 1995 23:15:00");  document.write("getFullYear() : " + dt.getFullYear() );  </script>  getFullYear() : 1995 |
| [getHours()](https://www.javatpoint.com/javascript-date-gethours-method) | It returns the integer value between 0 and 23 that represents the hours on the basis of local time. | <script type = "text/javascript">  var dt = new Date("December 25, 1995 23:15:00");  document.write("getHours() : " + dt.getHours() );  </script>  getHours() : 23 |
| [getMilliseconds()](https://www.javatpoint.com/javascript-date-getmilliseconds-method) | It returns the integer value between 0 and 999 that represents the milliseconds on the basis of local time. | <script type = "text/javascript">  var dt = new Date( );  document.write("getMilliseconds() : " + dt.getMilliseconds() );  </script>  getMilliseconds() : 632 |
| [getMinutes()](https://www.javatpoint.com/javascript-date-getminutes-method) | It returns the integer value between 0 and 59 that represents the minutes on the basis of local time. | <script type = "text/javascript">  var dt = new Date( "December 25, 1995 23:15:00" );  document.write("getMinutes() : " + dt.getMinutes() );  </script>  getMinutes() : 15 |
| [getMonth()](https://www.javatpoint.com/javascript-date-getmonth-method) | It returns the integer value between 0 and 11 that represents the month on the basis of local time. | <script type = "text/javascript">  var dt = new Date( "December 25, 1995 23:15:00" );  document.write("getMonth() : " + dt.getMonth() );  </script>  getMonth() : 11 |
| [getSeconds()](https://www.javatpoint.com/javascript-date-getseconds-method) | It returns the integer value between 0 and 60 that represents the seconds on the basis of local time. | <script type = "text/javascript">  var dt = new Date( "December 25, 1995 23:15:20" );  document.write("getSeconds() : " + dt.getSeconds() );  </script>  getSeconds() : 20 |
| setDate() | It sets the day value for the specified date on the basis of local time. | <script type = "text/javascript">  var dt = new Date( "Aug 28, 2008 23:30:00" );  dt.setDate( 24 );  document.write( dt );  Sun Aug 24 2008 23:30:00 GMT+0530 (India Standard Time) |
| setDay() | It sets the particular day of the week on the basis of local time. |  |
| setFullYears() | It sets the year value for the specified date on the basis of local time. | <script type = "text/javascript">  var dt = new Date( "Aug 28, 2008 23:30:00" );  dt.setFullYear( 2000 );  document.write( dt );  </script>  Mon Aug 28 2000 23:30:00 GMT+0530 (India Standard Time) |
| [setHours()](https://www.javatpoint.com/javascript-date-sethours-method) | It sets the hour value for the specified date on the basis of local time. | <script type = "text/javascript">  var dt = new Date( "Aug 28, 2008 23:30:00" );  dt.setHours( 02 );  document.write( dt );  </script>  Thu Aug 28 2008 02:30:00 GMT+0530 (India Standard Time) |
| [setMilliseconds()](https://www.javatpoint.com/javascript-date-setmilliseconds-method) | It sets the millisecond value for the specified date on the basis of local time. | <script type = "text/javascript">  var dt = new Date( "Aug 28, 2008 23:30:00" );  dt.setMilliseconds( 1010 );  document.write( dt );  </script>  Thu Aug 28 2008 23:30:01 GMT+0530 (India Standard Time) |
| [setMinutes()](https://www.javatpoint.com/javascript-date-setminutes-method) | It sets the minute value for the specified date on the basis of local time. | <script type = "text/javascript">  var dt = new Date( "Aug 28, 2008 23:30:00" );  dt.setMinutes( 45 );  document.write( dt );  </script> Thu Aug 28 2008 23:45:00 GMT+0530 (India Standard Time) |
| setMonth() | It sets the month value for the specified date on the basis of local time. | <script type = "text/javascript">  var dt = new Date( "Aug 28, 2008 23:30:00" );  dt.setMonth( 2 );  document.write( dt );  </script>  Fri Mar 28 2008 23:30:00 GMT+0530 (India Standard Time) |
| [setSeconds()](https://www.javatpoint.com/javascript-date-setseconds-method) | It sets the second value for the specified date on the basis of local time. | <script type = "text/javascript">  var dt = new Date( "Aug 28, 2008 23:30:00" );  dt.setSeconds( 80 );  document.write( dt );  </script>  Thu Aug 28 2008 23:31:20 GMT+0530 (India Standard Time) |
| [toString()](https://www.javatpoint.com/javascript-date-tostring-method) | It returns the date in the form of string. | <script type = "text/javascript">  var dateobject = new Date(1993, 6, 28, 14, 39, 7);  stringobj = dateobject.toString();  document.write( "String Object : " + stringobj );  </script>  String Object : Wed Jul 28 1993 14:39:07 GMT+0530 (India Standard Time) |
| Date() | Returns today's date and time | <script type = "text/javascript">  var dt = Date();  document.write("Date and Time : " + dt );  </script>  Date and Time : Fri Aug 05 2022 21:16:18 GMT+0530 (India Standard Time) |
| [setTime()](https://www.tutorialspoint.com/javascript/date_settime.htm) | Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC. | <script type = "text/javascript">  var dt = new Date( "Aug 28, 2008 23:30:00" );  dt.setTime( 5000000 );  document.write( dt );  </script>  Thu Jan 01 1970 06:53:20 GMT+0530 (India Standard Time) |

## JavaScript Date Output

By default, JavaScript will use the browser's time zone and display a date as a full text string:

**Mon Aug 01 2022 19:08:29 GMT+0530 (India Standard Time)**

## Creating Date Objects

Date objects are created with the new Date() constructor.

There are **4 ways** to create a new date object:

new Date()  
new Date(year, month, day, hours, minutes, seconds, milliseconds)  
new Date(milliseconds)  
new Date(date string)

## new Date()

new Date() creates a new date object with the **current date and time**:

### **Example**

const d = new Date();

Date objects are static. The computer time is ticking, but date objects are not.

## new Date(year, month, ...)

new Date(year, month, ...) creates a new date object with a **specified date and time**.

7 numbers specify year, month, day, hour, minute, second, and millisecond (in that order):

### **Example**

const d = new Date(2018, 11, 24, 10, 33, 30, 0);

**Note:** JavaScript counts months from **0** to **11**:

**January = 0**.

**December = 11**.

Specifying a month higher than 11, will not result in an error but add the overflow to the next year:

Specifying:

const d = new Date(2018, 15, 24, 10, 33, 30);

Is the same as:

const d = new Date(2019, 3, 24, 10, 33, 30);

Specifying a day higher than max, will not result in an error but add the overflow to the next month:

Specifying:

const d = new Date(2018, 5, 35, 10, 33, 30);

Is the same as:

const d = new Date(2018, 6, 5, 10, 33, 30);

## Using 6, 4, 3, or 2 Numbers

6 numbers specify year, month, day, hour, minute, second:

### **Example**

const d = new Date(2018, 11, 24, 10, 33, 30);

5 numbers specify year, month, day, hour, and minute:

### **Example**

const d = new Date(2018, 11, 24, 10, 33);

4 numbers specify year, month, day, and hour:

### **Example**

const d = new Date(2018, 11, 24, 10);

3 numbers specify year, month, and day:

### **Example**

const d = new Date(2018, 11, 24);

2 numbers specify year and month:

### **Example**

const d = new Date(2018, 11);

You cannot omit month. If you supply only one parameter it will be treated as milliseconds.

### **Example**

const d = new Date(2018);

## new Date(dateString)

new Date(dateString) creates a new date object from a **date string**:

### **Example**

const d = new Date("October 13, 2014 11:13:00");

## JavaScript Stores Dates as Milliseconds

JavaScript stores dates as number of milliseconds since January 01, 1970, 00:00:00 UTC (Universal Time Coordinated).

Zero time is January 01, 1970 00:00:00 UTC.

Now the time is: **1659361109593** milliseconds past January 01, 1970

## new Date(milliseconds)

new Date(milliseconds) creates a new date object as**zero time plus milliseconds**:

### **Example**

const d = new Date(0);

01 January 1970 **plus** 100 000 000 000 milliseconds is approximately 03 March 1973:

### **Example**

const d = new Date(100000000000);

January 01 1970 **minus** 100 000 000 000 milliseconds is approximately October 31 1966:

### **Example**

const d = new Date(-100000000000);

### **Example**

const d = new Date(86400000);

One day (24 hours) is 86 400 000 milliseconds.

# JavaScript setTimeout() and clearTimeout() method

The **setTimeout()** method in JavaScript is used to execute a function after waiting for the specified time interval. This method returns a numeric value that represents the ID value of the timer.

Unlike the **setInterval()** method, the **setTimeout()** method executes the function only once. This method can be written with or without the **window** prefix.

We can use the **clearTimeout()** method to stop the timeout or to prevent the execution of the function specified in the **setTimeout()** method. The value returned by the **setTimeout()** method can be used as the argument of the **clearTimeout()** method to cancel the timer.

The commonly used syntax of the **setTimeout()** method is given below.

### **Syntax**

window.setTimeout(function, milliseconds);

### **Parameter values**

This method takes two parameter values **function** and **milliseconds** that are defined as follows.

**function:** It is the function containing the block of code that will be executed.

**milliseconds:** This parameter represents the time-interval after which the execution of the function takes place. The interval is in milliseconds. Its default value is 0. It defines how often the code will be executed. If it is not specified, the value **0** is used.

Let's understand the use of **setTimeout()** method by using some illustrations.

### **Example1**

This is a simple example of using the **setTimeout()** method. Here, an alert dialog box will display at an interval of two seconds. We are not using any method to prevent the execution of the function specified in **setTimeout()** method. So the **setTimeout()** method executes the specified function only once, after the given time interval.

**<html>**

**<head>**

**<title>** setTimeout() method **</title>**

**</head>**

**<body>**

**<h1>** Hello World :) :) **</h1>**

**<h3>** This is an example of using the setTimeout() method **</h3>**

**<p>** Here, an alert dialog box will display after two seconds. **</p>**

**<script>**

  var a;

a = setTimeout(fun, 2000);

function fun() {

alert(" Welcome to the javaTpoint.com ");

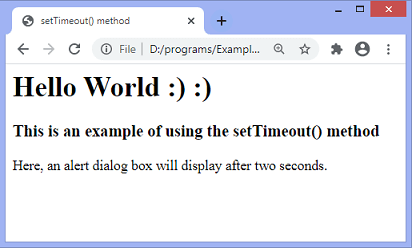
}

**</script>**

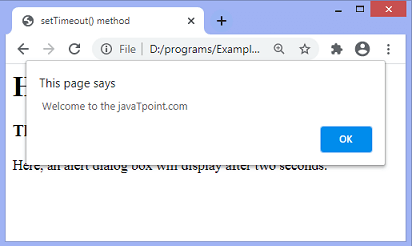
**</body>**

**</html>**

**Output**



After an interval of two seconds, the output will be -



### **Example2**

In the above examples, we have not used any method to prevent the execution of function specified in **setTimeout()**. Here, we are using the **clearTimeout()** method to stop the function's execution.

We have to click the given **stop** button before two seconds to see the effect.

**<html>**

**<head>**

**<title>** setTimeout() method **</title>**

**</head>**

**<body>**

**<h1>** Hello World :) :) **</h1>**

**<h3>** This is an example of using the setTimeout() method **</h3>**

**<p>** Click the following button before 2 seconds to see the effect. **</p>**

**<button** onclick = "stop()"**>** Stop **</button>**

**<script>**

var a = setTimeout(fun1, 2000);

function fun1()

{

var win1 = window.open();

win1.document.write(" **<h2>** Welcome to the javaTpoint.com **</h2>**");

setTimeout(function(){win1.close()}, 2000);

}

function stop() {

  clearTimeout(a);

}

**</script>**

**</body>**

**</html>**

**Output**



The output will remain same if the user clicks the **stop** button before two seconds. Otherwise, a new tab will open after two seconds and close after two seconds of opening.

# JavaScript setInterval() and **clearInterval()** method

The **setInterval()** method in JavaScript is used to repeat a specified function at every given time-interval. It evaluates an expression or calls a function at given intervals. This method continues the calling of function until the window is closed or the **clearInterval()** method is called. This method returns a numeric value or a non-zero number that identifies the created timer.

Unlike the **setTimeout()** method, the **setInterval()** method invokes the function multiple times. This method can be written with or without the **window** prefix.

The commonly used syntax of **setInterval()** method is given below:

### **Syntax**

window.setInterval(function, milliseconds);

### **Parameter values**

This method takes two parameter values **function** and **milliseconds** that are defined as follows.

**function:** It is the function containing the block of code that will be executed.

**milliseconds:** This parameter represents the length of the time interval between each execution. The interval is in milliseconds. It defines how often the code will be executed. If its value is less than 10, the value 10 is used.

## How to stop the execution?

We can use the **clearInterval()** method to stop the execution of the function specified in **setInterval()** method. The value returned by the **setInterval()** method can be used as the argument of **clearInterval()** method to cancel the timeout.

Let's understand the use of **setInterval()** method by using some illustrations.

### **Example1**

This is a simple example of using the **setInterval()** method. Here, an alert dialog box displays at an interval of 3 seconds. We are not using any method to stop the execution of the function specified in **setInterval()** method. So the method continues the execution of the function until the window is closed.

**<html>**

**<head>**

**<title>** setInterval() method **</title>**

**</head>**

**<body>**

**<h1>** Hello World :) :) **</h1>**

**<h3>** This is an example of using the setInterval() method **</h3>**

**<p>** Here, an alert dialog box displays on every three seconds. **</p>**

**<script>**

var a;

a = setInterval(fun, 3000);

function fun() {

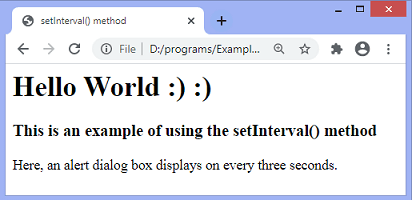
alert(" Welcome to the javaTpoint.com ");

}**</script>**

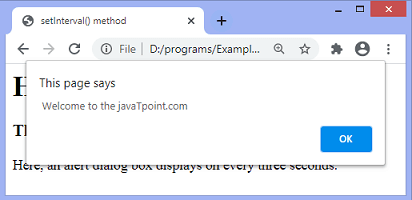
**</body>**

**</html>**

**Output**



After the time interval of three seconds, the output will be -



Now, there is another example of using the **setInterval()** method.

### **Example2**

Here, the background color will change on every 200 milliseconds. We are not using any method to stop the execution of the function specified in **setInterval()** method. So the method continues the execution of the function until the window is closed.

**<html>**

**<head>**

**<title>** setInterval() method **</title>**

**</head>**

**<body>**

**<h1>** Hello World :) :) **</h1>**

**<h3>** This is an example of using the setInterval() method **</h3>**

**<p>** Here, the background color changes on every 200 milliseconds. **</p>**

**<script>**

var var1 = setInterval(color, 200);

function color() {

var var2 = document.body;

var2var2.style.backgroundColor = var2.style.backgroundColor == "lightblue" ? "lightgreen" : "lightblue";

}

**</script>**

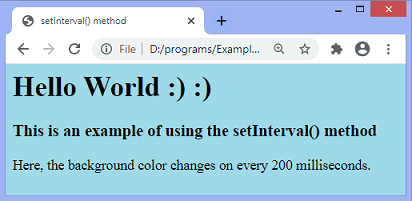
**</body>**

**</html>**

**Output**



The background will keep changing from **lightgreen** to **lightblue** on an interval of 200 milliseconds. After 200 milliseconds, the output will be -



### **Example3**

In the above example, we have not used any method to stop the toggling between the colors. Here, we are using the **clearInterval()** method to stop the toggling of colors in the previous example.

We have to click the specified **stop** button to see the effect.

**<html>**

**<head>**

**<title>** setInterval() method **</title>**

**</head>**

**<body>**

**<h1>** Hello World :) :) **</h1>**

**<h3>** This is an example of using the setInterval() method **</h3>**

**<p>** Here, the background color changes on every 200 milliseconds. **</p>**

**<button** onclick = "stop()"**>** Stop **</button>**

**<script>**

var var1 = setInterval(color, 200);

function color() {

var var2 = document.body;

var2var2.style.backgroundColor = var2.style.backgroundColor == "lightblue" ? "lightgreen" : "lightblue";

}

function stop() {

clearInterval(var1);

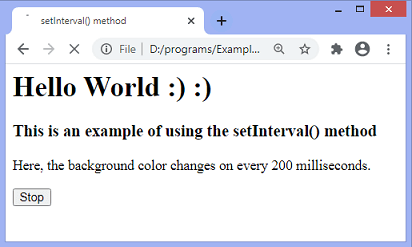
}

**</script>**

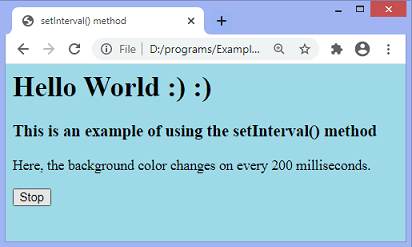
**</body>**

**</html>**

**Output**



The color of the background will start changing after 200 milliseconds. On clicking the specified **stop** button, the toggling between the colors will be stopped on the corresponding background color. The output after clicking the button will be -



# Document Object Model (DOM model)

The **document object** represents the whole html document.

When html document is loaded in the browser, it becomes a document object. It is the **root element** that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

As mentioned earlier, it is the object of window. So

window.document

Is same as

document

According to W3C - *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*

# DOM (Document Object Model)

The Document Object Model (DOM) is a ***programming interface*** for **HTML(HyperText Markup Language)** and **XML**(Extensible markup language) documents. It defines the **logical structure** of documents and the way a document is accessed and manipulated.

**Note**: It is called a Logical structure because DOM doesn’t specify any relationship between objects.

DOM is a way to represent the webpage in a structured hierarchical way so that it will become easier for programmers and users to glide through the document. With DOM, we can easily access and manipulate tags, IDs, classes, Attributes, or Elements of HTML using commands or methods provided by the Document object. Using DOM, the JavaScript gets access to HTML as well as CSS of the web page and can also add behavior to the HTML elements. so basically **Document Object Model is an API that represents and interacts with HTML or XML documents.**

**Why DOM is required?**

HTML is used to **structure**the web pages and Javascript is used to add **behavior**to our web pages. When an HTML file is loaded into the browser, the javascript can not understand the HTML document directly. So, a corresponding document is created(DOM). **DOM is basically the representation of the same HTML document but in a different format with the use of objects**. Javascript interprets DOM easily i.e javascript can not understand the tags(<h1>H</h1>) in HTML document but can understand object h1 in DOM. Now, Javascript can access each of the objects (h1, p, etc) by using different functions.

**Structure of DOM**: DOM can be thought of as a Tree or Forest(more than one tree). The term **structure model**is sometimes used to describe the tree-like representation of a document.  Each branch of the tree ends in a node, and each node contains objects  Event listeners can be added to nodes and triggered on an occurrence of a given event. One important property of DOM structure models is ***structural isomorphism***: if any two DOM implementations are used to create a representation of the same document, they will create the same structure model, with precisely the same objects and relationships.

**Why called an Object Model?**  
Documents are modeled using objects, and the model includes not only the structure of a document but also the behavior of a document and the objects of which it is composed like tag elements with attributes in HTML.

**Properties of DOM**: Let’s see the properties of the document object that can be accessed and modified by the document object.



*Representation of the DOM*

* [**Window Object**](https://www.geeksforgeeks.org/properties-of-window-object/#:~:text=It%20represents%20an%20array%20that,frames%20of%20a%20given%20window.&text=It%20returns%20a%20reference%20to%20a%20DOMPoint%20object%2C%20which%20represents,point%20in%20a%20coordinate%20system.&text=It%20provides%20information%20of%20the%20URLs%20visited%20in%20the%20current%20window.&text=It%20represents%20the%20number%20of%20frames%20in%20the%20current%20window.)**:** Window Object is object of the browser which is always at top of the hierarchy.  It is like an API that is used to set and access all the properties and methods of the browser. It is automatically created by the browser.
* **Document object:** When an HTML document is loaded into a window, it becomes a document object. The ‘document’ object has various properties that refer to other objects which allow access to and modification of the content of the web page. If there is a need to access any element in an HTML page, we always start with accessing the ‘document’ object. Document object is property of window object.
* **Form Object:** It is represented by ***form*** tags.
* [**Link Object**](https://www.geeksforgeeks.org/html-dom-link-object/)**:** It is represented by ***link***tags.
* [**Anchor Object**](https://www.geeksforgeeks.org/html-dom-anchor-object/)**:** It is represented by ***a href*** tags.
* **Form Control Elements:**: Form can have many control elements such as text fields, buttons, radio buttons, checkboxes, etc.

## Methods of document object

We can access and change the contents of document by its methods.

The important methods of document object are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| write("string") | writes the given string on the doucment. |
| writeln("string") | writes the given string on the doucment with newline character at the end. |
| getElementById() | returns the element having the given id value. |
| getElementsByName() | returns all the elements having the given name value. |
| getElementsByTagName() | returns all the elements having the given tag name. |
| getElementsByClassName() | returns all the elements having the given class name. |

### **Accessing field value by document object**

In this example, we are going to get the value of input text by user. Here, we are using **document.form1.name.value** to get the value of name field.

Here, **document** is the root element that represents the html document.

**form1** is the name of the form.

**name** is the attribute name of the input text.

**value** is the property, that returns the value of the input text.

Let's see the simple example of document object that prints name with welcome message.

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

function printvalue(){

var name=document.form1.name.value;

alert("Welcome: "+name);

}

**</script>**

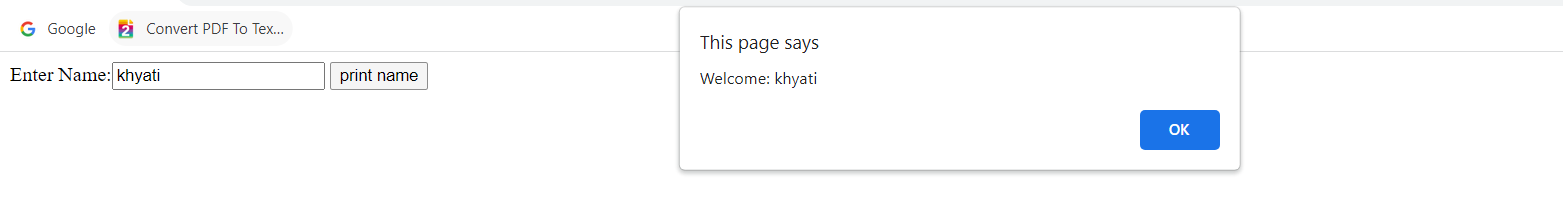
**<form** name="form1"**>**

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

**<input** type="button" onclick="printvalue()" value="print name"**/>**

**</form>**

#### **Output of the above example**

Top of Form

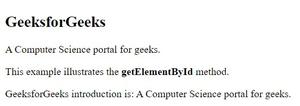
Bottom of Form

**Example:**In this example, We use HTML element id to find the DOM HTML element.

* HTML

|  |
| --- |
| <!DOCTYPE html>  <**html**>    <**body**>      <**h2**>GeeksforGeeks</**h2**>        <!-- Finding the HTML Elements by their Id in DOM -->      <**p** id="intro">A Computer Science portal for geeks.</**p**>    <**p**>This example illustrates the <**b**>getElementById</**b**> method.</**p**>      <**p** id="demo"></**p**>      <**script**>      const element = document.getElementById("intro");      document.getElementById("demo").innerHTML = "GeeksforGeeks introduction is: " + element.innerHTML;      </**script**>  </**body**>  </**html**> |

**Output:**



*Getting the HTML element by getElementById() Method*

# Javascript - document.getElementById() method

The **document.getElementById()** method returns the element of specified id.

In the previous page, we have used **document.form1.name.value** to get the value of the input value. Instead of this, we can use document.getElementById() method to get value of the input text. But we need to define id for the input field.

Let's see the simple example of document.getElementById() method that prints cube of the given number.

Skip Ad

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

function getcube(){

var number=document.getElementById("number").value;

alert(number\*number\*number);

}

**</script>**

**<form>**

Enter No:**<input** type="text" id="number" name="number"**/><br/>**

**<input** type="button" value="cube" onclick="getcube()"**/>**

**</form>**

#### **Output of the above example**

Top of Form

Bottom of Form

# Javascript - document.getElementsByName() method

The **document.getElementsByName()** method returns all the element of specified name.

The syntax of the getElementsByName() method is given below:

document.getElementsByName("name")

Here, name is required.

### **Example of document.getElementsByName() method**

In this example, we going to count total number of genders. Here, we are using getElementsByName() method to get all the genders.

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

function totalelements()

{

var allgenders=document.getElementsByName("gender");

alert("Total Genders:"+allgenders.length);

}

**</script>**

**<form>**

Male:**<input** type="radio" name="gender" value="male"**>**

Female:**<input** type="radio" name="gender" value="female"**>**

**<input** type="button" onclick="totalelements()" value="Total Genders"**>**

**</form>**

#### **Output of the above example**



# Javascript-document.getElementsByTagName() method

The **document.getElementsByTagName()** method returns all the element of specified tag name.

The syntax of the getElementsByTagName() method is given below:

document.getElementsByTagName("name")

Here, name is required.

### **Example of document.getElementsByTagName() method**

In this example, we going to count total number of paragraphs used in the document. To do this, we have called the document.getElementsByTagName("p") method that returns the total paragraphs.

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

function countpara(){

var totalpara=document.getElementsByTagName("p");

alert("total p tags are: "+totalpara.length);

}

**</script>**

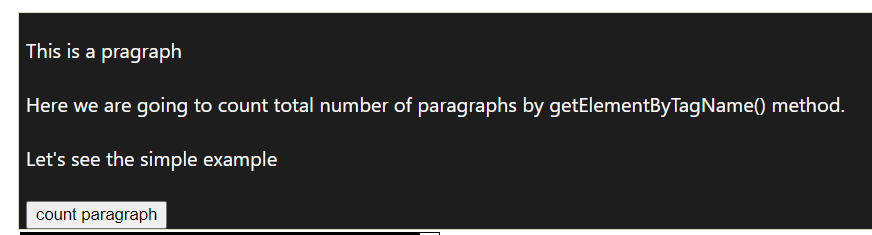
**<p>**This is a pragraph**</p>**

**<p>**Here we are going to count total number of paragraphs by getElementByTagName() method.**</p>**

**<p>**Let's see the simple example**</p>**

**<button** onclick="countpara()"**>**count paragraph**</button>**

#### **Output of the above example**



### **Another example of document.getElementsByTagName() method**

In this example, we going to count total number of h2 and h3 tags used in the document.

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

function counth2(){

var totalh2=document.getElementsByTagName("h2");

alert("total h2 tags are: "+totalh2.length);

}

function counth3(){

var totalh3=document.getElementsByTagName("h3");

alert("total h3 tags are: "+totalh3.length);

}

**</script>**

**<h2>**This is h2 tag**</h2>**

**<h2>**This is h2 tag**</h2>**

**<h3>**This is h3 tag**</h3>**

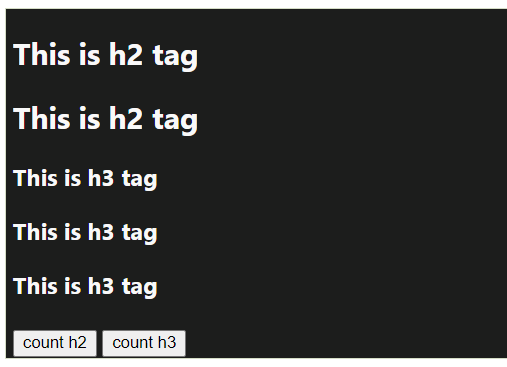
**<h3>**This is h3 tag**</h3>**

**<h3>**This is h3 tag**</h3>**

**<button** onclick="counth2()"**>**count h2**</button>**

**<button** onclick="counth3()"**>**count h3**</button>**

#### **Output of the above example**



#### **Note: Output of the given examples may differ on this page because it will count the total number of para , total number of h2 and total number of h3 tags used in this document.**

# Javascript - innerHTML

The **innerHTML** property can be used to write the dynamic html on the html document.

It is used mostly in the web pages to generate the dynamic html such as registration form, comment form, links etc.

### **Example of innerHTML property**

In this example, we are going to create the html form when user clicks on the button.

In this example, we are dynamically writing the html form inside the div name having the id mylocation. We are identifing this position by calling the document.getElementById() method.

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

function showcommentform() {

var data="Name:**<input** type='text' name='name'**><br>**Comment:**<br><textarea** rows='5' cols='80'**></textarea>**

**<br><input** type='submit' value='Post Comment'**>**";

document.getElementById('mylocation').innerHTML=data;

}

**</script>**

**<form** name="myForm"**>**

**<input** type="button" value="comment" onclick="showcommentform()"**>**

**<div** id="mylocation"**></div>**

**</form>**

# Javascript - innerText

The **innerText** property can be used to write the dynamic text on the html document. Here, text will not be interpreted as html text but a normal text.

It is used mostly in the web pages to generate the dynamic content such as writing the validation message, password strength etc.

## Javascript innerText Example

In this example, we are going to display the password strength when releases the key after press.

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

function validate() {

var msg;

if(document.myForm.userPass.value.length**>**5){

msg="good";

}

else{

msg="poor";

}

document.getElementById('mylocation').innerText=msg;

 }

**</script>**

**<form** name="myForm"**>**

**<input** type="password" value="" name="userPass" onkeyup="validate()"**>**

Strength:**<span** id="mylocation"**>**no strength**</span>**

**</form>**

#### **Output of the above example**



# HTML DOM Document write()

## Definition and Usage

The write() method writes directly to an open (HTML) document stream.

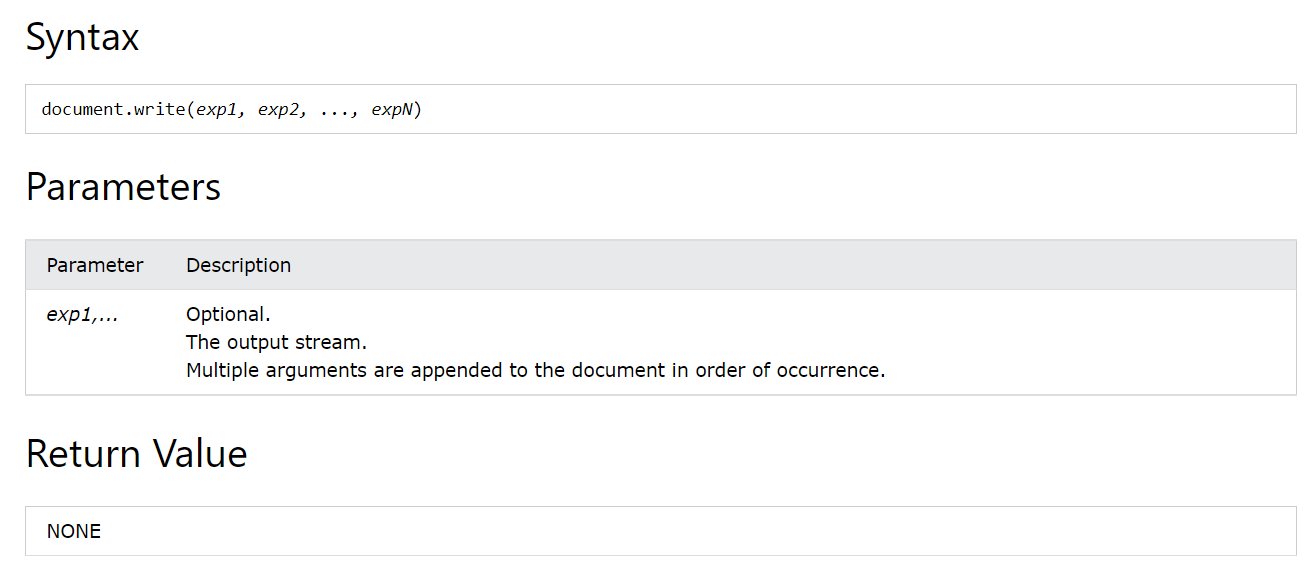
## Warning

The write() method deletes all existing HTML when used on a loaded document.

The write() method cannot be used in XHTML or XML.

## Note

The write() method is most often used to write to output streams opened by the the open() method.



### **Examples**

Write some text directly to the HTML output:

document.write("Hello World!");

Write some HTML elements directly to the HTML output:

document.write("<h2>Hello World!</h2><p>Have a nice day!</p>");

## More Examples

Write a date object directly to the HTML ouput:

document.write(Date());

Open an output stream, add some HTML, then close the output stream:

document.open();  
document.write("<h1>Hello World</h1>");  
document.close();

Open a new window and write some HTML into it:

const myWindow = window.open();  
myWindow.document.write("<h1>New Window</h1>");  
myWindow.document.write("<p>Hello World!</p>");

# HTML DOM Document writeln()

## Definition and Usage

The writeln() method writes directly to an open (HTML) document stream.

The writeln() method is identical to the write() method, with the addition of writing a newline character (U+000A) after each statement.

## Warning

The writeln() method deletes all existing HTML when used on a loaded document.

The writeln() method cannot be used in XHTML or XML.

### **Example**

document.writeln("Hello World!");  
document.writeln("Have a nice day!");

## Note

It makes no sense to use **writeln()** in HTML.

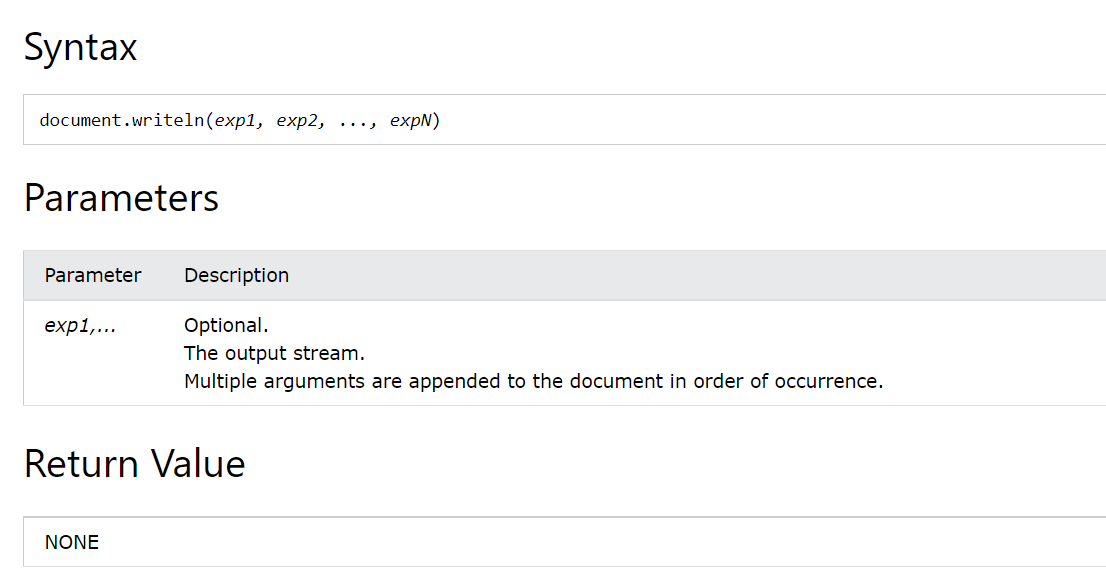
It is only useful when writing to text documents (type=".txt").

Newline characters are ignored in HTML.

If you want new lines in HTML, you must use paragraphs or <br>:

### **Examples**

document.write("Hello World!");  
document.write("<br>");  
document.write("Have a nice day!");



## The Difference Between write() and writeln()

The writeln( ) method is only useful when writing to text documents (type=".txt").

### **Example**

<!DOCTYPE html>

<html>

<body>

<h1>The Document Object</h1>

<h2>The write() and writeln() Methods</h2>

<p>write() does NOT add a new line (CR) after each statement.</p>

<p>writeln() DOES add a new line (CR) after each statement.</p>

<pre>

<script>

document.write("Hello World!");

document.write("Have a nice day!");

document.write("<br>");

document.writeln("Hello World!");

document.writeln("Have a nice day!");

</script>

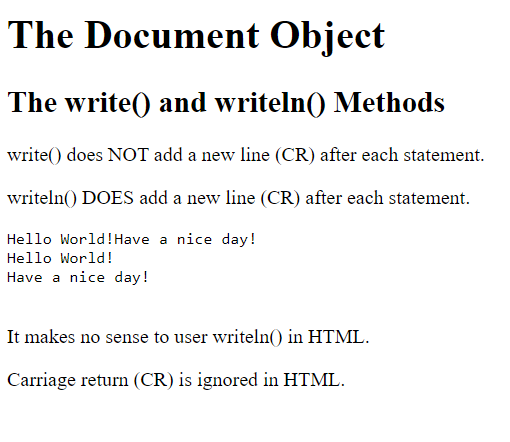
</pre>

<p>It makes no sense to user writeln() in HTML.</p>

<p>Carriage return (CR) is ignored in HTML.</p>

</body>

</html>



Top of Form