UNIT 2
CSS AND CLIENT SIDE PROGRAMMING

What Is CSS3 And Why Is It Used?

To help build highly interactive online pages, CSS3 is invariably used due to its importance in providing greater options in the design process. When marketing products and services, web design plays a vital part; a site should be created in a manner that will draw potential customers to explore and revisit a website more often. Many **web design firm**s are developing and enhancing websites through the use of CSS3 as this is a great form of web development. This article will help define CSS3 and will point out its advantages.

Definition

The acronym CSS stands for Cascading Style Sheets which is used to augment the functionality, versatility. and efficient performance of site content. It allows for the creation of content-rich websites that do not require much weight or codes; this translates into more interactive graphics and animation, superior user-interface, and significantly more organization and rapid download time.

It is used with HTML to create content structure, with CSS3 being used to format structured content. It is responsible for font properties, colors, text alignments, graphics, background images, tables and other components. This tool provides extra capabilities such as absolute, fixed and relative positioning of various elements. The increasing popularity of CSS3 when used by **web design firm**s stimulates major browsers such as Google Chrome, Firefox, Safari, and IE9 to adopt and embrace this programming language.

Advantages

Although CSS3 is not the only web development solution, it does allow provide greater advantages for several reasons.

- **Customization** – A web page can be customized and alterations created in the design by simply changing a modular file.
- **Bandwidth Requirements** – It decreases server bandwidth requirements, giving rapid download time when a site is accessed with desktop or hand-held devices, providing an improved user experience.
- **Consistency** – It delivers consistent and accurate positioning of navigational elements on the website.
- **Appealing** – It makes the site more appealing with adding videos and graphics easier.
- **Viewing** – It allows online videos to be viewed without the use of third-party plug-ins.
- **Visibility** – It delivers the opportunity to improve brand visibility by designing effective online pages.
- **Cost Effective** – It is cost-effective, time-saving, and supported by most browsers.

Since the introduction of CSS3, there is greater control of the presentation of content and various elements on a website; however it is not really responsible for overall design as it only specifies the structure and content presentation of certain web pages.

External, internal, and inline CSS styles

Cascading Style Sheets (CSS) are files with styling rules that govern how your website is presented on screen. CSS rules can be applied to your website's HTML files in various ways. You can use an **external stylesheet**, an **internal stylesheet**, or an **inline style**. Each method has advantages that suit particular uses.

An **external stylesheet** is a standalone .css file that is linked from a web page. The advantage of external stylesheets is that it can be created once and the rules applied to multiple web pages. Should you need to make widespread changes to your site design, you can make a single change in the stylesheet and it will be applied to all linked pages, saving time and effort.

An **internal stylesheet** holds CSS rules for the page in the **head** section of the HTML file. The rules only apply to that page, but you can configure CSS classes and IDs that can be used to style multiple elements in the page code. Again, a single change to the CSS rule will apply to all tagged elements on the page.

Inline styles relate to a specific HTML tag, using a **style** attribute with a CSS rule to style a specific page element. They're useful for quick, permanent changes, but are less flexible than external and internal stylesheets as each inline style you create must be separately edited should you decide to make a design change.

Using external CSS stylesheets

An HTML page styled by an external CSS stylesheet must reference the .css file in the document head. Once created, the CSS file must be uploaded to your server and linked in the HTML file with code such as:

```
<link href="style.css" rel="stylesheet" type="text/css">
```

You can name your stylesheet whatever you wish, but it should have a .css file extension.

Using internal CSS stylesheets

Rather than linking an external .css file, HTML files using an internal stylesheet include a set of rules in their **head** section. CSS rules are wrapped in <style> tags, like this:

```
<head>
<style type="text/css">

  h1 {
        color:#fff
        margin-left: 20px;
    }

  p {
```

```
    font-family: Arial, Helvetica, Sans Serif;
   }



</style>
</head>
```

Using inline styles

Inline styles are applied directly to an element in your HTML code. They use the **style** attribute, followed by regular CSS properties.

For example:

```
<h1 style="color:red;margin-left:20px;">Today's Update</h1>
```

Rule Cascading

Cascade and inheritance

Conflicting rules

CSS stands for **Cascading Style Sheets**, and that first word *cascading* is incredibly important to understand — the way that the cascade behaves is key to understanding CSS.
At some point, we will find that the CSS have created two rules which could potentially apply to the same element. The **cascade**, and the closely-related concept of **specificity**, are mechanisms that control which rule applies when there is such a conflict. Which rule is styling your element may not be the one you expect, so you need to understand how these mechanisms work.
Also significant here is the concept of **inheritance**, which means that some CSS properties by default inherit values set on the current element's parent element, and some don't. This can also cause some behavior that you might not expect.

The cascade

Stylesheets **cascade** — at a very simple level this means that the order of CSS rules matter; when two rules apply that have equal specificity the one that comes last in the CSS is the one that will be used.

EXAMPLE
In the below example, we have two rules that could apply to the `h1`. The `h1` ends up being colored blue — these rules have an identical selector and therefore carry the same specificity, so the last one in the source order wins.

```
h1 {
    color: red;
```

```
}

h1 {

   color: blue;

}

   <h1>This is my heading.</h1>
```

OUTPUT

This is my heading.

Specificity

Specificity is how the browser decides which rule applies if multiple rules have different selectors, but could still apply to the same element. It is basically a measure of how specific a selector's selection will be:

- An element selector is less specific — it will select all elements of that type that appear on a page — so will get a lower score.
- A class selector is more specific — it will select only the elements on a page that have a specific `class` attribute value — so will get a higher score.

Example time! Below we again have two rules that could apply to the `h1`. The below `h1` ends up being colored red — the class selector gives its rule a higher specificity, and so it will be applied even though the rule with the element selector appears further down in the source order.

EXAMPLE

```
main-heading {

   color: red;

}

      h1 {

   color: blue;

}
```

. <h1 class="main-heading">This is my heading.</h1>

OUTPUT

This is my heading.

Inheritance

Inheritance also needs to be understood in this context — some CSS property values set on parent elements are inherited by their child elements, and some aren't.

For example, if you set a color and font-family on an element, every element inside it will also be styled with that color and font, unless you've applied different color and font values directly to them.

Some properties do not inherit — for example if you set a width of 50% on an element, all of its descendants do not get a width of 50% of their parent's width. If this was the case, CSS would be very frustrating to use!

```
body {
   color: blue;
}

span {
   color: black;
}
```

```
<p>As the body has been set to have a color of blue this is inherited through the
descendants.</p>

<p>We can change the color by targetting the element with a selector,
 such as this

<span>span</span>.</p>
```

OUTPUT

As the body has been set to have a color of blue this is inherited through the descendants.

We can change the color by targetting the element with a selector, such as this **span.**

CSS3 Shadow Effects

With CSS3 you can create two types of shadows: text-shadow (adds shadow to text) and box-shadow (adds shadow to other elements).

CSS3 Text Shadow

The text-shadow property can take up to four values:

- the horizontal shadow
- the vertical shadow
- the blur effect
- the color

Examples:

- Normal text shadow

```
h1 {
text-shadow: 2px 2px 5px crimson;
}
```

- Glowing text effect

```
h1 {
text-shadow: 0 0 4px #00FF9C;
}
```

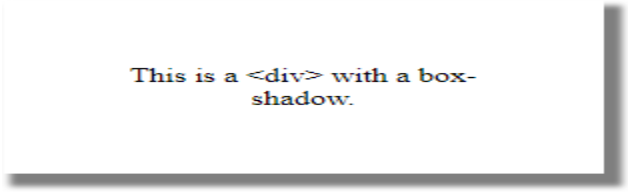
CSS3 Box Shadow

The box-shadow property can take up to six values:

- (optional) the inset keyword (changes the shadow to one inside the frame)
- the horizontal shadow
- the vertical shadow
- the blur effect
- the spreading
- the color

Examples:

```
.first-div {
        box-shadow: 1px 1px 5px 3px grey;
}
```



CSS Animations

CSS allows animation of HTML elements without using JavaScript or Flash!

What are CSS Animations?

An animation lets an element gradually change from one style to another.

You can change as many CSS properties you want, as many times you want.

To use CSS animation, you must first specify some keyframes for the animation.

Keyframes hold what styles the element will have at certain times.

The @keyframes Rule

When you specify CSS styles inside the @keyframes rule, the animation will gradually change from the current style to the new style at certain times.

To get an animation to work, you must bind the animation to an element.

The following example binds the "example" animation to the element. The animation will last for 4

seconds, and it will gradually change the background-color of the element from "red" to "yellow":

Example

```
/* The animation code */
@keyframes example {
```

```
  from {background-color: red;}
  to {background-color: yellow;}
}

/* The element to apply the animation to */
div {
  width: 100px;
  height: 100px;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
}
```

Note: The animation-duration property defines how long time an animation should take to complete. If the animation-duration property is not specified, no animation will occur, because the default value is 0s (0 seconds).

In the example above we have specified when the style will change by using the keywords "from" and "to" (which represents 0% (start) and 100% (complete)).

It is also possible to use percent. By using percent, you can add as many style changes as you like.

The following example will change the background-color of the element when the animation is 25% complete, 50% complete, and again when the animation is 100% complete:

Example

```
/* The animation code */
@keyframes example {
  0%  {background-color: red;}
  25%  {background-color: yellow;}
  50%  {background-color: blue;}
  100% {background-color: green;}
}

/* The element to apply the animation to */
div {
  width: 100px;
  height: 100px;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
}
```

CSS Animation Properties

The following table lists the @keyframes rule and all the CSS animation properties:

Property	Description
@keyframes	Specifies the animation code
animation	A shorthand property for setting all the animation properties

animation-delay	Specifies a delay for the start of an animation
animation-direction	Specifies whether an animation should be played forwards, backwards or in alternate cycles
animation-duration	Specifies how long time an animation should take to complete one cycle
animation-fill-mode	Specifies a style for the element when the animation is not playing (before it starts, after it ends, or both)
animation-iteration-count	Specifies the number of times an animation should be played
animation-name	Specifies the name of the @keyframes animation
animation-play-state	Specifies whether the animation is running or paused
animation-timing-function	Specifies the speed curve of the animation

CSS Transitions

CSS Transitions is a module of CSS that lets you create gradual transitions between the values of specific CSS properties. The behavior of these transitions can be controlled by specifying their timing function, duration, and other attributes.

Properties

- transition
- transition-delay
- transition-duration
- transition-property
- transition-timing-function

The **transition** CSS property is a shorthand property for transition-property, transition-duration, transition-timing-function, and transition-delay.

CSS transition Property

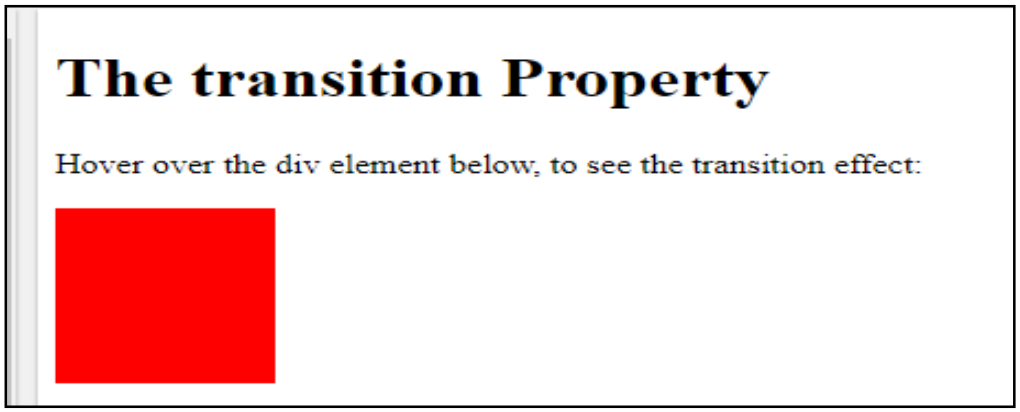
Example

Hover over a element to gradually change the width from 100px to 300px:

```
div {
  width: 100px;
  transition: width 2s;
}
```

```
div:hover {
  width: 300px;
}
```

OUTPUT



Definition and Usage

The transition property is a shorthand property for:

- transition-property
- transition-duration
- transition-timing-function

Property Values

Value	Description
transition-property	Specifies the name of the CSS property the transition effect is for
transition-duration	Specifies how many seconds or milliseconds the transition effect takes to complete
transition-timing-function	Specifies the speed curve of the transition effect
transition-delay	Defines when the transition effect will start
initial	Sets this property to its default value. Read about *initial*
inherit	Inherits this property from its parent element. Read about *inherit*

Example

When an <input type="text"> gets focus, gradually change the width from

100px to 250px:

```
input[type=text] {
```

```
  width: 100px;
  transition: width .35s ease-in-out;
}

input[type=text]:focus {
```

```
  width: 250px;
}
```

OUTPUT

The width Property

Set the width of the input field to 100 pixels. However, when the input field gets focus, make it 250 pixels wide:

Search:

CSS background-color

The `background-color` property specifies the background color of an element.

Example

The background color of a page is set like this:

```
body {
  background-color: lightblue;
}
```

CSS background-image

The `background-image` property specifies an image to use as the background of an element.

By default, the image is repeated so it covers the entire element.

Example

The background image for a page can be set like this:

```
body {
  background-image: url("paper.gif");
}
```

CSS background - Shorthand property

To shorten the code, it is also possible to specify all the background properties in one single property. This is called a shorthand property.

The shorthand property for background is `background`.

Example

Use the shorthand property to set all the background properties in one declaration:

```
body {
  background: #ffffff url("img_tree.png") no-repeat right top;
}
```

CSS Border - Shorthand Property

As you can see from the examples above, there are many properties to consider when dealing with borders.

To shorten the code, it is also possible to specify all the individual border properties in one property.

The `border` property is a shorthand property for the following individual border properties:

- `border-width`
- `border-style` (required)
- `border-color`

Example

```
p {
  border: 5px solid red;
}
```

Result:

Some text

3. **Give** any four methods of Date objects.

JavaScript Get Date Methods

These methods can be used for getting information from a date object:

Method	Description
getFullYear()	Get the **year** as a four digit number (yyyy)
getMonth()	Get the **month** as a number (0-11)
getDate()	Get the **day** as a number (1-31)
getMinutes()	Get the **minute** (0-59)
getMilliseconds()	Get the **millisecond** (0-999)
getDay()	Get the weekday as a number (0-6)

JavaScript Set Date Methods

seconds, milliseconds) for a Date Object.

Set Date Methods

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

4. **Write** the JavaScript methods to retrieve the data and time based on the computer locale.

```
<script>

// Use of Date.now() function
var d = Date(Date.now());

// Converting the number of millisecond in date string
a = d.toString()

// Printing the current date
document.write("The current date is: " + a)

</script>
```

OUTPUT

The current date is: Thu Jan 09 2020 20:35:39 GMT+0530 (India Standard Time)

5. Can you **list** the different methods defined in document and window object of JavaScript.

Window Object

The window object represents an open window in a browser.

Window Object Methods

Method	Description
alert()	Displays an alert box with a message and an OK button
atob()	Decodes a base-64 encoded string
blur()	Removes focus from the current window
btoa()	Encodes a string in base-64
clearInterval()	Clears a timer set with setInterval()
clearTimeout()	Clears a timer set with setTimeout()
close()	Closes the current window
confirm()	Displays a dialog box with a message and an OK and a Cancel button
focus()	Sets focus to the current window
getComputedStyle()	Gets the current computed CSS styles applied to an element
getSelection()	Returns a Selection object representing the range of text selected by the user
matchMedia()	Returns a MediaQueryList object representing the specified CSS media query string
moveBy()	Moves a window relative to its current position
moveTo()	Moves a window to the specified position
open()	Opens a new browser window
print()	Prints the content of the current window
prompt()	Displays a dialog box that prompts the visitor for input

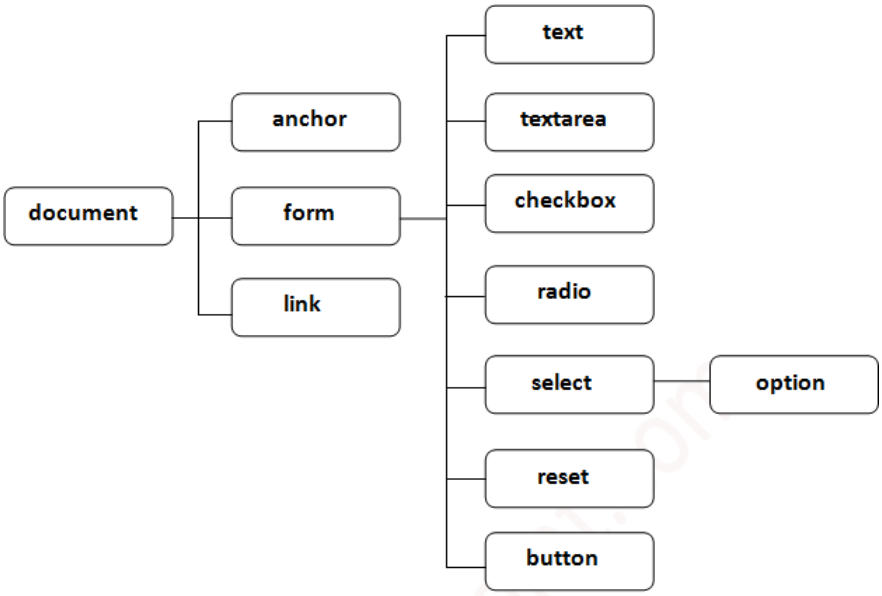
requestAnimationFrame()	Requests the browser to call a function to update an animation before the next repaint
resizeBy()	Resizes the window by the specified pixels
resizeTo()	Resizes the window to the specified width and height
scroll()	Deprecated. This method has been replaced by the scrollTo() method.
scrollBy()	Scrolls the document by the specified number of pixels
scrollTo()	Scrolls the document to the specified coordinates
setInterval()	Calls a function or evaluates an expression at specified intervals (in milliseconds)
setTimeout()	Calls a function or evaluates an expression after a specified number of milliseconds
stop()	Stops the window from loading

Document Object Model

1. Document Object
2. Properties of document object
3. Methods of document object
4. Example of document object

- 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

- 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."*



Properties of document object

Let's see the properties of document object that can be accessed and modified by the document object.

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.

6.	**Name** which parser is best in parsing in large size documents. Why?
7.	**Summarize** benefits of using JavaScript code in an HTML document. **Advantages of JavaScript** The merits of using JavaScript are − • **Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server. • **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something. • **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard. • **Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.
8.	**Predict** the need for client and server side scripting. **Client-side scripting** (embedded scripts) is code that exists inside the client's HTML page. This code will be processed on the client machine and the HTML page will NOT perform a PostBack to the web-server. Traditionally, client-side scripting is used for page navigation, data validation and formatting. The language used in this scripting is JavaScript. JavaScript is compatible and is able to run on any internet browser. The two main benefits of client-side scripting are: 1. The user's actions will result in an immediate response because they don't require a trip to the server. 2. Fewer resources are used and needed on the web-server. **Server-side scripting** is a technique used in web development which involves employing **scripts** on a web **server** which produce a response customized for each user's (client's) request to the website. The alternative is for the web **server** itself to deliver a static web page. The **client-side script** executes the code to the **client side** which is visible to the users while a **server-side script** is executed **in the server** end which users cannot see.
9.	**Interpret** how exceptions are handled in Java script.

The try...catch...finally Statement

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the **try...catch...finally** construct as well as the **throw** operator to handle exceptions.

You can **catch** programmer-generated and **runtime** exceptions, but you cannot **catch** JavaScript syntax errors.

Here is the **try...catch...finally** block syntax –

```
<script type = "text/javascript">
  <!--
    try {
      // Code to run
      [break;]
    }

    catch ( e ) {
      // Code to run if an exception occurs
      [break;]
    }

    [ finally {
      // Code that is always executed regardless of
      // an exception occurring
    }]
  //-->
</script>
```

The **try** block must be followed by either exactly one **catch** block or one **finally** block

(or one of both). When an exception occurs in the **try** block, the exception is placed in **e** and the

catch block is executed. The optional **finally** block executes unconditionally after try/catch.

Examples

Here is an example where we are trying to call a non-existing function which in turn is raising an exception. Let us see how it behaves without **try...catch**−

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function myFunc() {
          var a = 100;
          alert("Value of variable a is : " + a );
        }
```

```
//-->
```

```
    </script>
  </head>

  <body>
    <p>Click the following to see the result:</p>

    <form>
      <input type = "button" value = "Click Me" onclick = "myFunc();" />
    </form>
  </body>
</html>
```

10.

Define JavaScript statement with an example.

JavaScript Statements

Example

```
var x, y, z;     // Statement 1
x = 5;           // Statement 2
y = 6;           // Statement 3
z = x + y;       // Statement 4
```

JavaScript statements are composed of:
Values, Operators, Expressions, Keywords, and Comments.
This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Statements</h2>

<p>In HTML, JavaScript statements are executed by the browser.</p>

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML = "Hello Dolly.";

</script>

</body>

</html>
```

OUTPUT

	JavaScript Statements In HTML, JavaScript statements are executed by the browser. Hello Dolly.

11.	**Point out** any two techniques of event programming.
	What is an Event ?
	JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.
	When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.
	Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.
	Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.
	Please go through this small tutorial for a better understanding HTML Event Reference. Here we will see a few examples to understand a relation between Event and JavaScript −
	onclick Event Type
	This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.
	Example
	Try the following example.
	`<html>`
	`<head>`
	`<script type = "text/javascript">`
	`<!--`
	`function sayHello() {`
	`alert("Hello World")`
	`}`
	`//-->`
	`</script>`
	`</head>`

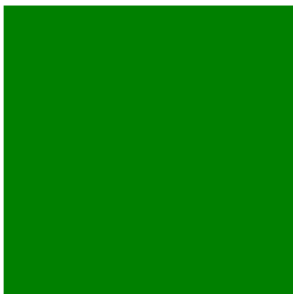
```
  <body>
    <p>Click the following button and see result</p>
    <form>
      <input type = "button" onclick = "sayHello()" value = "Say Hello" />
    </form>
  </body>
</html>
```

Example 2

```
<!doctype html>
<html>
<head>
        <script>
        function hov() {
                var e = document.getElementById('hover');
                e.style.display = 'none';
        }
        </script>
</head>
<body>
        <div id="hover" onmouseover="hov()"
        style="background-color:green;height:200px;width:200px;">
        </div>
</body>
</html>
```

OUTPUT

Before mouse is taken over green square-



Green square gets disappear after mouse is taken over it.