Contents

UNIT 6 – Event Handling, DOM, and Dynamic Documents with JavaScript

6.0 Structure of Event Handling, DOM, and Dynamic Documents with JavaScript

6.1 Learning Outcomes:

After the successful completion of this unit, the student will be able to:

- Explain the concepts of the Document Object Model(DOM) and DOM tree structure of an XHTML document.
- Explain different ways of accessing XHTML elements using JavaScript.[L2]
- Describe the event handling mechanism and change the style and content of XHTML elements dynamically using JavaScript.[L2]
- Explain client-side form validations using JavaScript.[L2]

6.2 Document Object Model (DOM)

The Document Object Model (DOM) is an application programming interface(API) for web documents that defines an interface between XHTML documents and application programs. It represents the structure of an XHTML document as a tree of objects, where

each object(node) corresponds to a part of the XHTML document, such as elements, attributes, and text. The objects have methods and properties that are associated with their respective node types. JavaScript can get access to XHTML as well as CSS of the web pages using DOM. The DOM provides a way for programs to traverse and manipulate the structure, css style, and content of web documents dynamically.

6.3 DOM Structure of XHTML

The DOM tree structure is the representation of the hierarchy of elements in an XHTML or XML document but in a tree-like structure composed of objects(nodes). Each element, attribute, and piece of text(content) in the XHTML document is represented as a node in the tree. The root of the tree branches out to include all the elements and content in the document. JavaScript interprets DOM easily, using it as a bridge to access and manipulate the elements using different methods and properties of the document object.

Example: An XHTML document (Figure 6.1) is represented in a DOM tree-like structure as shown in Figure 6.2.

XHTML document consists of <h2>, <p> tags, <ul> unordered list, and table is given in Figure 6.1.

```
 <html>
<head>
    <title>DOM Tree Structure of an XHTML document</title>
</head>
<body>

    <h2> Graphic Era University </h2>
      <p> Courses at GEU </p>
      <p>
            <ul>
                  <li>B.Tech</li>
                  <li>BBA</li>
                  <li>BCA</li>
            </ul>
      </p>
      <table>
            <tr>
                  <td>MBA</td>
                  <td>MCA</td>
            </tr>
      </table>
</body>
</html>
```

Figure 6.1: An XHTML document

The DOM-tree representation of the XHTML document is shown in Figure 6.2.

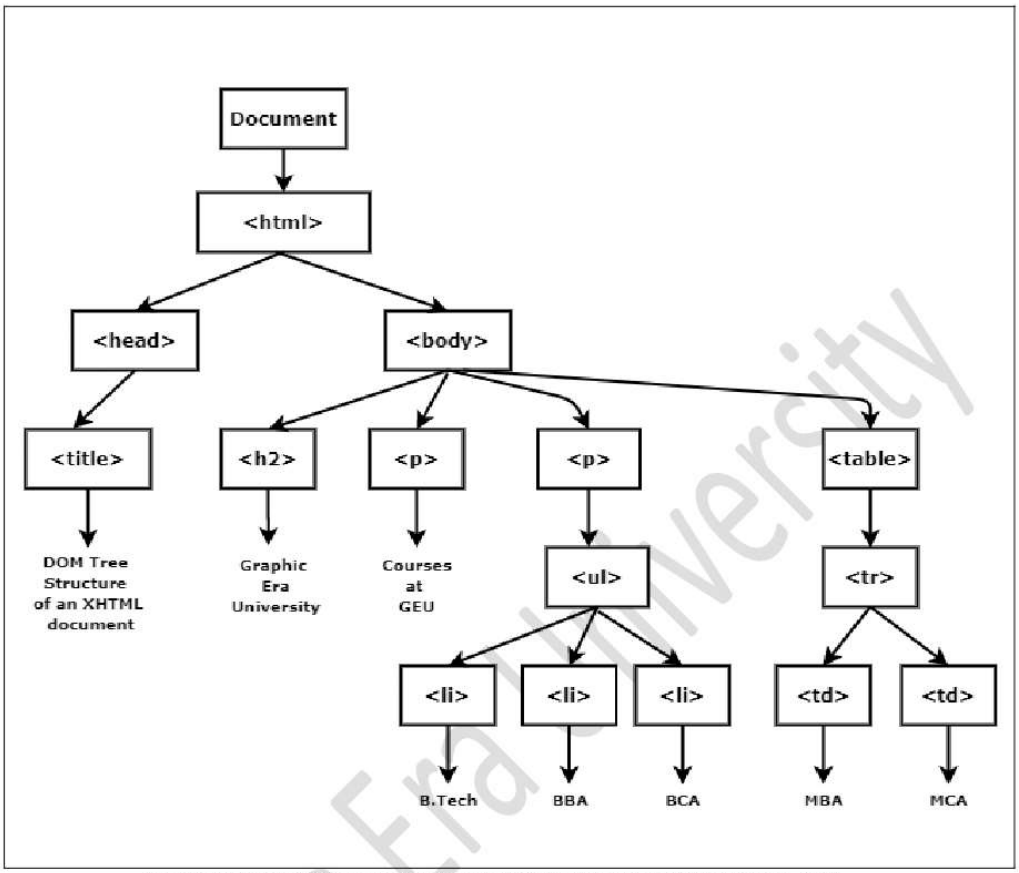

Figure 6.2: DOM-tree structure of an XHTML document (Figure 6.1)

The <html> element has two child elements: <head> and <body>.

- The <head> element has one child: <title>.
  - The <title> element has a text (leaf) node containing the title text(content) - "DOM tree Structure of an XHTML document"
- The <body> element has 4 child elements : <h2>, <p>, <p> and <table>
  - The <h2> and first <p> elements have text (leaf) nodes containing "Graphic Era University" and "Courses at GEU " respectively.
  - The second <p> element has one child element:
    The <ul> element has three children, each represented by <li> elements, and each <li> element has a text node containing the "B.Tech", "BBA", and "BCA".
  - The <table> element has one child element: <tr>
    The <tr> element has 2 child elements: <td> and <td> containing "MBA" and "MCA" as text nodes respectively.

This hierarchical structure allows developers to navigate through the document, select elements, and manipulate content using JavaScript. Understanding the DOM tree is crucial for effective web development, as it forms the basis for dynamic interactions and updates on web pages.

6.4 Accessing Elements in JavaScript

Accessing elements in JavaScript is a fundamental aspect of web development, and it involves selecting and interacting with XHTML elements in the DOM (Document Object Model). The structure of an XHTML document can be changed dynamically. There are various methods and properties associated with objects that facilitate element access in JavaScript. From the Document Object Model (DOM), elements in an XHTML document can be accessed in Javascript code using different ways.

Some of the ways of accessing elements are discussed here -

- get XHTML elements by ID - document.getElementById()
- get XHTML elements by name - document.getElementsByName()
- get XHTML elements by tag name - document.getElementsByTagName()
- get XHTML elements by class name - document.getElementsByClassName()

- Accessing elements using get elements by ID

  Generally, unique IDs are used for elements in an XHTML document so that they can easily be accessed/referenced in JavaScript Code. The getElementById() method is used to select an element based on its unique id attribute.

  Syntax:

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

  Here,

  The method takes one parameter - the ID of an XHTML element

  The method returns an object with a particular ID if it exists, else if the element with a specific ID does not exist it returns a null value.

  Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
      <title> Example of getElementByID method </title>
```

```
  </head>
  <body>
   <h2 id="t1"> Graphic Era University </h2>
   <p  id="p1"> First Paragraph  </p>

  <script type="text/javascript" >

     var x = document.getElementById("t1");
     var y = document.getElementById("p1");

     document.write("<br/><b> In JavaScript - Accessing elements by
                    ID </b><br/>");
     document.write("<br/> Content of heading 2 with ID <b> 't1'
                     -", x.innerHTML, "</b><br/>");
     document.write("<br/> Content of 1st paragraph with ID <b> 'p1'
                     -", y.innerHTML, "</b><br/>");
  </script>

</body>
</html>
```

Figure 6.3: Example – Accessing elements in JavaScript using getElementById()

Output:

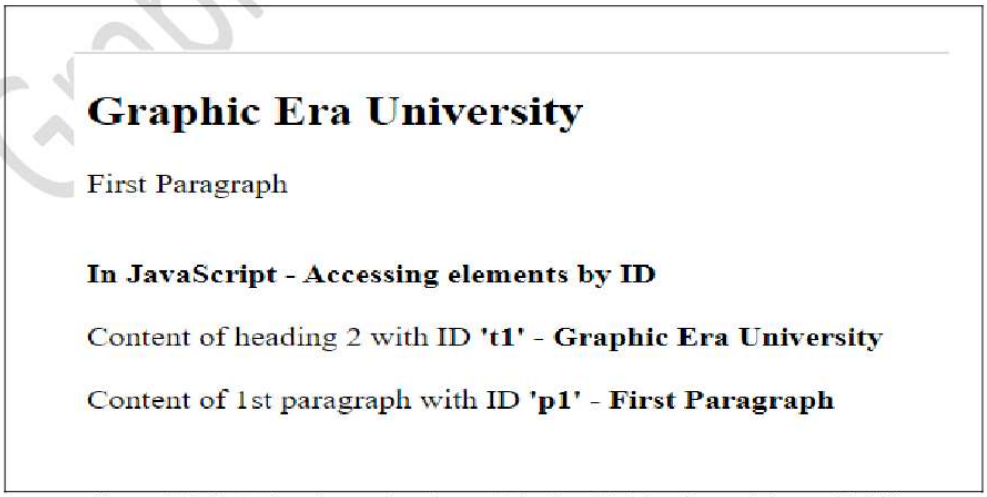

Figure 6.4: Output – Accessing elements in JavaScript using getElementById()

- Accessing elements using get elements by Name

  Elements in an XHTML document can easily be accessed/referenced in JavaScript Code by their names. The getElementsByName() method is used to select elements based on their name.

  Syntax:

  ```
  document.getElementsByName("elements_name");
  ```

  Here,

  The method takes one parameter - the name of XHTML elements that are to be accessed

  The method returns an array of objects that have elements with a specified name if it exists, else if the elements with a specific name do not exist it returns a null value.

  Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
   <head><title> Example of getElementsByName method </title>
   </head>
<body>
  <h2 name="t1"> Graphic Era University </h2>
  <p  name="p1"> First Paragraph  </p>
  <p  name="p1"> Second Paragraph  </p>
  <script type="text/javascript" >
    var x = document.getElementsByName("t1");
    var y = document.getElementsByName("p1");
    document.write("<br/><b> In JavaScript - Accessing elements by
                    Name </b><br/>");
    document.write("<br/> Content of heading-2  with name  <b> 't1'
                    - ", x[0].innerHTML, "</b><br/>");
    document.write("<br/> Content of 1st paragraph with name <b>
                   'p1' - ", y[0].innerHTML, "</b><br/>");
    document.write("<br/> Content of 2nd paragraph with name <b>
                   'p1'  - ", y[1].innerHTML, "</b><br/>");
  </script>
  </body>
</html>
```

Figure 6.5: Example – Accessing elements in JavaScript using getElementsByName()

Output

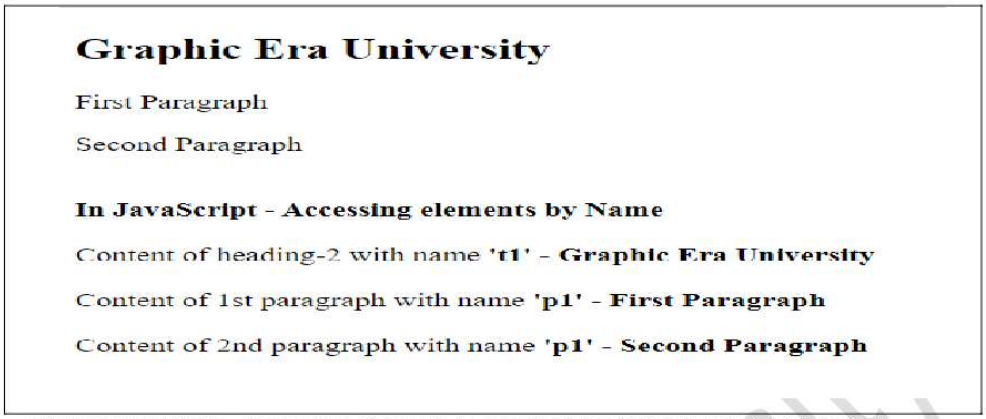

Figure 6.6: Output – Accessing elements in JavaScript using getElementsByName()

- Accessing elements using get elements by Tag Name

  Elements in an XHTML document can easily be accessed/referenced in JavaScript Code by their Tag names. The getElementsByTagName() method is used to select the elements based on their tag name.

  Syntax:

```
document.getElementsByTagName("elements_Tagname");
```

  Here,

  The method takes one parameter - the tag name of XHTML elements that are to be accessed

  The method returns an array of objects that has elements with a specified tag name if it exists, else if the elements with a specific tag name do not exist it returns a null value.

  Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
       <title> Example of getElementsByTagName method </title>
  </head>
<body>
  <h2 name="t1"> Graphic Era University </h2>
  <p  name="p1"> First paragraph </p>
  <p  name="p2"> Second paragraph </p>
```

```
<script type="text/javascript" >
  var x = document.getElementsByTagName("h2");
  var y = document.getElementsByTagName("p");

   document.write("<br/><b> In JavaScript - Accessing elements by
                Tag Name </b><br/>");
   document.write("<br/> Content of heading 2  with name <b>
                't1'  - ", x[0].innerHTML, "</b><br/>");
   document.write("<br/> Content of 1st paragraph with name <b>
                'p1'  - ", y[0].innerHTML, "</b><br/>");
   document.write("<br/> Content of 2nd paragraph with name  <b>
                'p2'  - ", y[1].innerHTML, "</b><br/>");

</script>
</body>
</html>
```

Figure 6.7: Example – Accessing elements in JavaScript using getElementsByTagName()

Output:

Graphic Era University

First paragraph

Second paragraph

In JavaScript - Accessing elements by Tag Name

Content of heading 2 with name **'t1' - Graphic Era University**

Content of 1st paragraph with name **'p1' - First paragraph**

Content of 2nd paragraph with name **'p2' - Second paragraph**

Figure 6.8: Output – Accessing elements in JavaScript using getElementsByTagName()

- Accessing elements using get elements by Class Name

  Elements in an XHTML document can easily be accessed/referenced in JavaScript Code by their class names. The getElementsByClassName() method is used to select the elements based on their class name.

Syntax:

```
document.getElementsByClassName("elements_classname");
```

Here,

The method takes one parameter - the class name of XHTML elements that are to be accessed

The method returns an array of objects that have elements with the same class name if it exists, else if the elements with a specific class name do not exist it returns a null value.

Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
       <title> Example of getElementsByClassName method </title>
  </head>
  <body>
    <h2 class="C1"> Graphic Era University </h2>
    <p  class="C1"> First Paragraph  </p>
    <p  class="C1"> Second Paragraph  </p>

    <script type="text/javascript" >
      var x = document.getElementsByClassName("C1");
      document.write("<br/><b> In JavaScript - Accessing elements
                    by Class Name </b><br/>");
      document.write("<br/> Content of heading 2 with class name
              <b> 'C1'  - ", x[0].innerHTML, "</b><br/>");
      document.write("<br/> Content of 1st paragraph with class
            name  <b> 'C1'  - ", x[1].innerHTML, "</b><br/>");
      document.write("<br/> Content of 2nd paragraph with class
            name  <b> 'C1'  - ", x[2].innerHTML, "</b><br/>");

  </script>
  </body>
  </html>
```

Figure 6.9: Example – Accessing elements in JavaScript using getElementsByClassName()

Output:

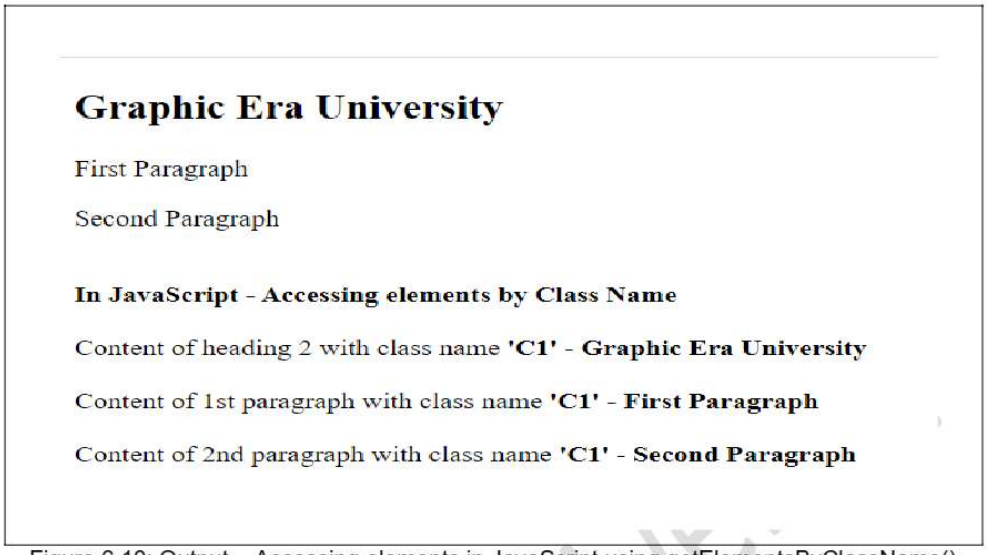

Figure 6.10: Output – Accessing elements in JavaScript using getElementsByClassName()

JavaScript also allows you to traverse and manipulate the DOM dynamically.

The contents of the elements can be modified by using properties like innerHTML, textContent, and innerText.

Example: Changing the text content of a paragraph

```
var paraobj = document.getElementById("mypara1");
paraobj.textContent = "New text content";
```

Elements can be added, deleted, or modified using methods like createElement(), appendChild(), removeChild(), and setAttribute().

The properties that allow to access parent and child elements are parentNode, childNodes, firstChild, and lastChild.

The properties that allow us to navigate to the next or previous sibling nodes in the DOM tree are nextSibling, and previousSibling.

These are some of the basic methods and properties for accessing elements in JavaScript. The choice of method depends on the specific requirements of the task, and the flexibility of these methods allows developers to interact with different parts of the DOM effectively. Modern JavaScript libraries and frameworks often provide additional abstractions to simplify DOM manipulation further.

6.5 Event and Event Handling in JavaScript

In web development, an event is an occurrence of an action or incident that happens in the browser, such as a click of a button, hovering the mouse, resizing the window, pressing a key, loading or unloading of page, and many more. Events can be triggered by the user or by the browser itself.

Users interact with the XHTML document elements like buttons, radio buttons, checkboxes, text boxes, images, etc. by either using a mouse or keyboard. The actions that take place due to the user's interactions with the keyboard are called keyevents and those with the mouse are called mouseevents. Some events that occur due to browser windows are called window/document events. Events are JavaScript, their names are case-sensitive. For example, a click is an event, but Click is not.

Key events examples - keydown, keypress, keyup, etc.

Mouse events examples - click, dblclick, mouseover, mouseout, etc

Window/Document events examples – resize, scroll, load, unload, etc.

When an event occurs a specific task is performed in response to an event, this is called event handling. It is required to define and execute the code that should run in response to a specific event. All possible events that can occur have to be handled by an application. It should not happen that a user clicks a button and there is no response. Managing these events is called Event Handling.

The commonly used events, related tag attributes, tags, and descriptions are given in Table 6.1.

Table 6.1: Commonly used events in JavaScript

Event	Tag Attribute	Tag	Description
click	onclick	<a> Most elements	The user clicks the mouse on an element
dblclick	ondblclick	Most elements	The user double-clicks the mouse on an element
mouseover	onmouseover	Most elements	The cursor of the mouse is over the elements
mouseout	onmouseout	Most elements	The cursor of the mouse leaves the elements
mousedown	onmousedown	Most elements	The mouse is pressed over the element

mouseup	onmouseup	Most elements	The mouse is released over the element
keydown	onkeydown	<body>, form elements	The user presses the key
keyup	onkeyup	<body>, form elements	The user releases the key
focus	onfocus	<a> <input> <textarea> <select>	The user focuses on the element
blur	onblur	Form elements	The focus is away from the form elements
submit	onsubmit	Form elements	The user submits the form
reset	onreset	Form elements	The user resets the form
change	onchange	<input> <textarea> <select>	The user changes the values in the form element
load	onload	<body>	When the browser finishes loading the web page
unload	onunload	<body>	When the browser user leaves the current web page, the browser unloads it

For managing event handling in JavaScript, we need to understand the following concepts

- Event Handlers
- Event Registration
- Event Object

Event Handlers

The function/method that is invoked/called to perform a specific task on the occurrence of an event is called an event handler. These event handlers have to be registered with the event, only then they can be invoked/called.

Event Registration

The process of connecting/registering an event handler with an event is called Event Registration. The event handler is invoked when the event occurs and only when it has been registered.

There are 2 ways of event registration in the DOM 0 event model in JavaScript.

- Assign event handler to tag attributes
- Assign event handler address to object property

- **Event Registration - Assign event handler to tag attributes**

  The event handler is assigned directly to the tag attribute where the tag is declared.

  Example 1: Inbuilt method alert() is triggered onclick of the button which displays the message. The alert() method is registered with the tag attribute `onclick`.

  /* Event registered with event handler defined at tag attribute onclick of the button element */

```
<input type="button" id="btn1" value="First Button"
   onclick="alert('The first button is clicked: Event handler defined
           at tag attribute');" />
```

  Example 2: User-defined method (event handler) is invoked onclick of the button. The **`"displaymsg()"`** method is registered with the tag attribute `onclick`. The event handler **`"displaymsg()"`** in turn invokes the inbuilt **`alert()`** method to display the message.

  /* Event registered with event handler at tag attribute onclick of the button element */

```
<input type="button" id="btn2" value="Second Button"
      onclick="displaymsg();" />
```

  The event handler **`"displaymsg()"`** can be defined in the **`<script>`** tag as follows –

```
<script type="text/javascript" >
   function displaymsg(){
   alert('The Second Button is clicked : Event Registration done
          using tag attribute');
 }
 </script>
```

- Event Registration - Assign event handler address to object property

  The address of the event handler is assigned to the object property.

  Example: User-defined method (event handler) is invoked onclick of the button. The **"showmsg()"** method is registered with the object property onclick of the button element. The event handler **"showmsg()"** in turn invokes the inbuilt **alert()** method to display the message.

```
/* Event registered with event handler using object property - onclick of the button element */
<input type="button" id="btn3" value="Third Button" />
```

  The event handler **"showmsg()"** can be defined in the **<script>** tag as follows –

```
<script type="text/javascript" >
  function showmsg(){
     alert('The Third Button  is clicked : Event Registration done
             using Object Property');
 }
</script>
```

  The address of the event handler **"showmsg()"** is registered with the object property in the **<script>** tag as follows –

```
<script type="text/javascript" >
    /* Event Registered using object property */
    var docobj = document.getElementById("btn3").onclick = showmsg ;
</script>
```

An example of assigning an event handler to tag attributes and assigning an event handler address to an object property is given in Figure 6.11.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
  <head>
    <title> Example of event registration using tag attribute </title>
    <script type="text/javascript" >
```

```
      function displaymsg(){
         alert('The Second Button is clicked : Event Registration done
                using tag attribute');
      }
      function showmsg(){
          alert('The Third Button  is clicked : Event Registration done
                 using Object Property');
      }
    </script>
  </head>
 <body>
  <h2 name="t1">Graphic Era University </h2>
  <h3 name="t2">Examples of event registration with event handler  </h3>
  <h3> using tag attribute and object property </h3>
  <form>
    <!-- Event registered using event handler defined at tag attribute -->
    <input type="button" id="btn1" value="First Button"
    onclick="alert('The first button is clicked: Event handler defined
                   at tag attribute');" />

   <!-- Event registered - event handler at tag attribute -->
   <input type="button" id="btn2" value="Second Button"
    onclick="displaymsg();" />

   <!-- Event registered - using object property  -->
   <input type="button" id="btn3" value="Third Button" />
  </form>

   <script type="text/javascript" >
     /* Event Registered using object property */
      var docobj = document.getElementById("btn3").onclick = showmsg ;
   </script>
</body>
</html>
```

Figure 6.11: Example-Different ways of Registration of events with event handlers in JavaScript

Output:

The output is shown with the click of the buttons that are internally registered using different ways.

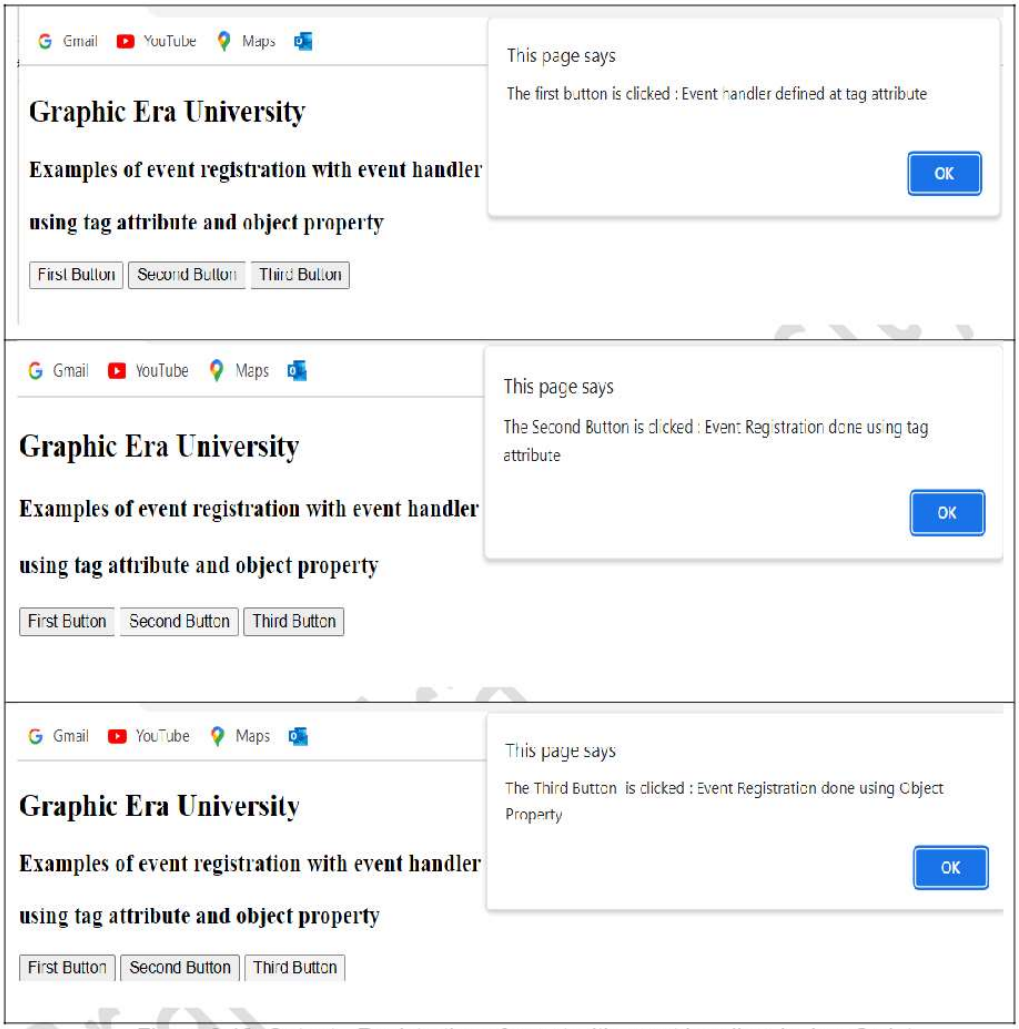

Figure 6.12: Output - Registration of event with event handlers in JavaScript

- Event Object

When an event occurs, the browser creates an event object that contains information about the event. This object is automatically passed to the event handler function.

Example:

```
function handleClick(event) {
  alert('Button clicked! Event details: ' + event.type);
}
```

In this example, the event parameter represents the event object, and its type property can be accessed to get the type of the event (e.g., 'click').

6.5.1 Handling events from body elements

Handling events from body elements in JavaScript involves capturing events that occur within the body of an XHTML document and registration of events with event handlers. Various events occur at the <body> elements.

Examples are load and unload events that can be handled by using the tag attributes "onload" and "onunload" of the opening <body> element.

Other events that can be handled at the <body> element are - click, dblclick, hovering of the mouse, keypress, keyup, and many more.

Example 1: An example illustrating how to handle load and unload events from the <body> element:

"load.js"

```
function load_greet()
{
   alert("Load Event triggered!!!");
}
```

Figure 6.13: Snippet Code of event handler used for handling load event in JavaScript

XHTML document

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE>
<html>
  <head>
    <title>Handling load and unload events in XHTML</title>
    <script type="text/javascript" src="load.js">
    </script>
  </head>
  <body onload="load_greet();"   onunload="alert('unloading');">
        <h3> GEU: Example of load and unload event handling </h3>
   </body>
</html>
```

Figure 6.14: Snippet Code of XHTML document for handling load and unload events in JavaScript

Here, the event handler "load_greet()" is stored in an external JavaScript file. The load event is triggered when the browser loads the XHTML document and the unload event is triggered when the XHTML document is closed or the user visits another web page. The

event handler "load_greet()" is registered by using the onload tag attribute for the load event and the onunload tag attribute is registered with the built-in method "alert()" for displaying the message for unload event.

Following is the output of the XHTML document on opening it in the browser.

Output:

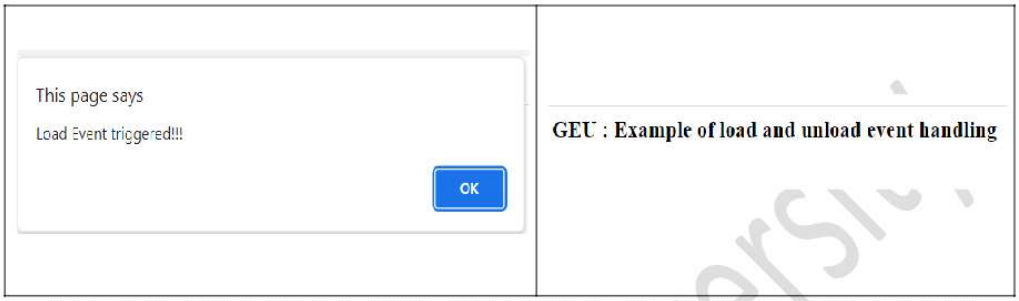

Figure 6.15: Output of XHTML document for handling load and unload events in JavaScript

Example 2: An example of handling click event on body elements - heading and paragraph elements.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html>
<head>
     <title>Example of event handling body elements </title>
</head>
<body>
 <h2> GEU: Example of event handling body elements </h2>
 <h1 style="color:blue" onclick="alert('You have clicked heading 1')" >
        Heading1 Heading1 Heading1 Heading1
 </h1>
 <h2 style="color:red" onclick="alert('You have clicked heading 2')" >
         Heading2 Heading2 Heading2 Heading2
 </h2>

 <p  style="color:blue; font-size:18pt"
             onclick="alert('You have clicked para1')" >
       paragraph1 paragraph1 paragraph1 paragraph1
 </p>
```

```
 <p  style="color:red;  font-size:18pt"
              onclick="alert('You have clicked para2')" >
       paragraph2 paragraph2 paragraph2 paragraph2
</p>

</body>
</html>
```

Figure 6.16: Snippet Code of XHTML document for handling click events of body elements

Here, the built-in method "alert()" used as an event handler for displaying messages is registered with the onclick tag attribute of <h1>, <h2>, and <p> tags. The click event is triggered when the user clicks on the "heading1", "heading2", "paragraph1" or "paragraph2" and the corresponding message is displayed.

Following is the output of the XHTML document on opening it in the browser.

Output:

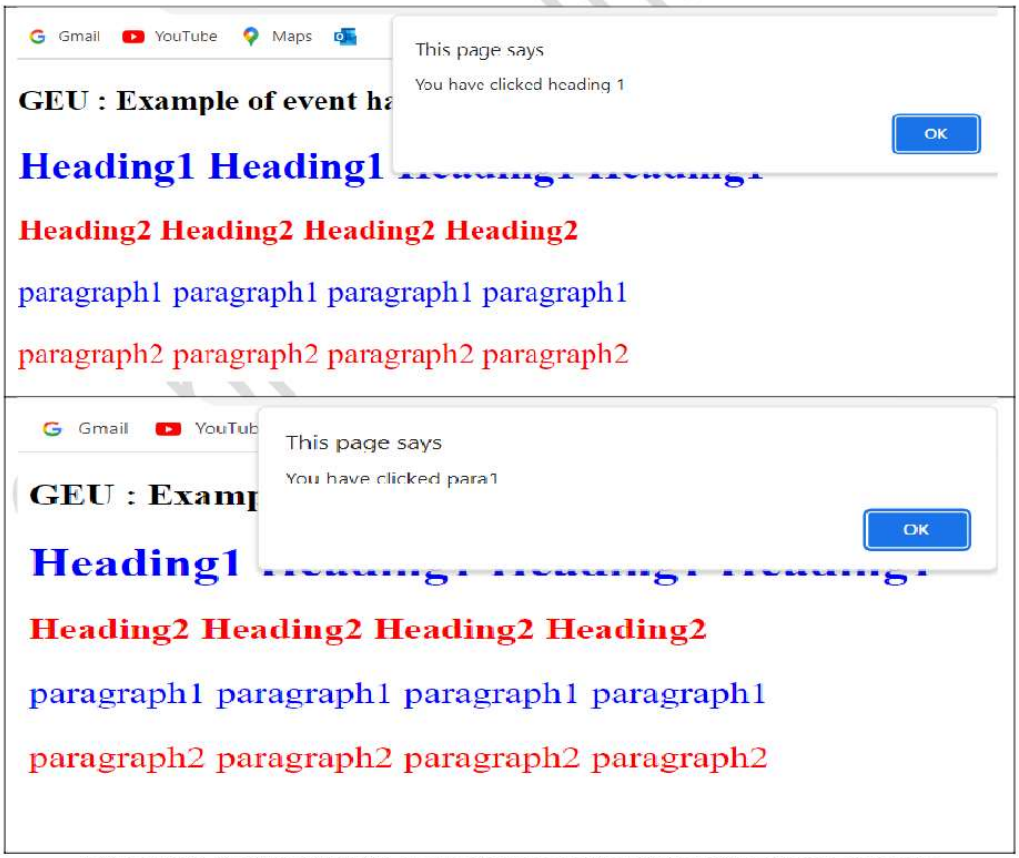

Figure 6.17: Output of XHTML document for handling click events of body elements

6.5.2 Handling events from button elements

Buttons in XHTML documents provide and simple and effective way of collecting information from the user. Handling events from button elements in JavaScript involves capturing events that occur on click of button elements like radio buttons, checkboxes, input buttons, button elements, etc, and registration of events with event handlers. Events that occur due to buttons are click, submit, and reset events.

Example: An example of handling events using radio buttons.

"evthandlerradio.js"

```
function evthandlerradio()
{ var d1 = document.getElementsByName('games');
  for(i = 0; i < d1.length;i++)
   {   if(d1[i].checked)
       {    x=d1[i].value;
            break;
       }
   }
   switch(x) {
       case "1":
            alert("Cricket game is selected");
            break;
       case "2":
            alert("Volleyball game is selected");
            break;
       case "3":
            alert("Football game is selected");
            break;
       case "4":
            alert("Badminton game is selected");
            break;
       default:
            alert("Error in Javascript function");
            break;
    }
}
```

Figure 6.18: Snippet Code of event handler used for handling radio button events in JavaScript

XHTML document

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html>
<head>
   <title>Example of event handling - Radio Buttons </title>
   <script type="text/javascript" src="evthandlerradio.js">
   </script>
</head>
<body>
      <h3>GEU - Example of event handling - Radio buttons </h3>
      <h2> Select any one game </h2>
      <form id="myform" action="">
      <p>
      <label><input type="radio" name="games" id="r1" value="1" />
             Cricket</label>
      <br/>
      <label><input type="radio" name="games" id="r2" value="2" />
             Volleyball</label>
      <br/>
      <label><input type="radio" name="games" id="r3"  value="3" />
              Football</label>
      <br/>
      <label><input type="radio" name="games" id="r4"  value="4" />
             Badminton</label>
      </p>
</form>
<!-- Script for registering the event handlers -->
<script type="text/javascript" >
      document.getElementById("r1").onclick = evthandlerradio;
      document.getElementById("r2").onclick = evthandlerradio;
      document.getElementById("r3").onclick = evthandlerradio;
      document.getElementById("r4").onclick = evthandlerradio;
</script>
</body>
</html>
```

Figure 6.19: Snippet Code of XHTML document for handling radio button events

Here, the method `"evthandlerradio()"` is used as an event handler for displaying messages and is registered with the onclick object property of each radio button. Here the event is registered with the address of the event handler. The click event is triggered when the user clicks on any one of the radio buttons and the corresponding message is displayed. Only one radio button can be selected at a time.

In the event handler `"evthandlerradio()"` all radio buttons based on the same name `"games"` are collected in an array of objects by using the method `document.getElementsByName('games').` By iterating through the array of objects the checked radio buttons value is obtained and the appropriate message is displayed using the switch statement.

Following is the output of the XHTML document on opening it in the browser.

Output:

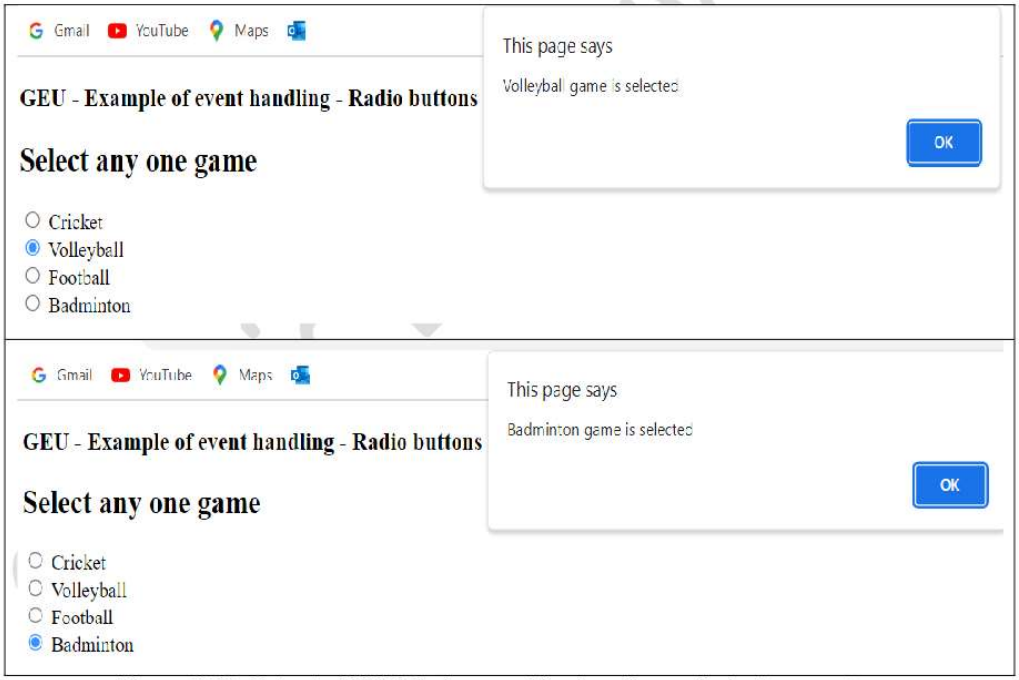

Figure 6.20: Output of XHTML document for handling radio button events

6.5.3 Handling events from textboxes

Handling events from text boxes in JavaScript involves capturing and responding to events that occur on input elements of type text. Common events for text boxes include events related to user input, such as the 'input', 'change', 'focus', and 'blur' events.

Example: An example demonstrating how to handle the click event of the button and using the onfocus tag attribute and **"this.blur()"** inbuilt method for making a text box read-only. Here, the area of a triangle is calculated by taking input from the user in text boxes.

XHTML document

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html>
<html>
<head>
    <title>Example Calculating Area of Triangle </title>
    <script type="text/javascript" src="calarea.js"></script>
</head>
<body>
   <p> </p>
   <form id ="f1" >
    <h3> GEU- Event handling textbox events </h3>
    <h4> Calculating Area of triangle </h4>
    <label>Base : <input type="text"  maxlength="10" id="txtbase"/>
    </label>
    <p>
      <label>height:<input type="text"  maxlength="10" id="txthgt"/>
       </label>
    </p>
    <p>
       <input type="button" id = "b1" value="calculate"/>
    </p>
    <p>
       <label>Area:
           <!-- onfocus event -->
           <input type="text"  maxlength="10" id="tarea"
                              onfocus = "this.blur()"/>
        </label>
    </p>
    </form>
      <script language="javascript">
          /* Registration of event handler using object property */
```

```
        document.getElementById("b1").onclick=fnarea;
    </script>
</body>
</html>
```

Figure 6.21: Snippet Code of XHTML document for handling textbox events

"calarea.js"

```
function fnarea()
{     var x = document.getElementById("txtbase");
      var y = document.getElementById("txthgt").value;
      var z  = 0.5 * x.value * y ;
      document.getElementById("tarea").value = z;
}
```

Figure 6.22: Snippet Code of event handler – calculating the area of a triangle

Here, the method "fnarea()" is used as an event handler for calculating the area of a triangle based on the input given by the user in the text boxes. The address of the event handler is registered with the onclick object property of the button element as –

```
document.getElementById("b1").onclick=fnarea;
```

The click event occurs when the user clicks the button "calculate" to calculate the area of a triangle.

The method/event handler "fnarea()" updates the textbox with the id "tarea" with the newly calculated area as -

```
document.getElementById("tarea").value = z;
```

If the user clicks on the textbox with the id "tarea" the onfocus event gets triggered and the text within it gets blurred due to the "this.blur()" method being called. This textbox is read-only and the displayed area cannot be edited by the user.

Following is the output of the XHTML document on opening it in the browser.

Output:

Figure 6.23: Output of - calculating the area of a triangle and onfocus event handling in textboxes

6.6 Form Validations (Example)

Form validation is an essential part of web development to ensure that user input is accurate and meets the specified criteria. Validation of form inputs, before sending form data to the server reduces the processing load on the server and also network traffic. Validations on the client side prevent unwanted/inaccurate/bad data from being sent to the server for processing and provide quick responses to the users thereby reducing response time.

When the users fill in form data, the event handlers should detect errors/invalid data if any, and display appropriate messages informing users using alert so that users can rectify the inputs and only submit valid data to the server for further processing. The users should also be informed where exactly the error exists and the correct format of the user inputs.

The focus() and select() methods can be called on the document object created by using document.getElementById(). The focus() method gets focus on the form element where the incorrect data is entered. The select() method after getting focus highlights the text in the form element.

For example

```
alert("Invalid Input");
document.getElementById("t1").focus();      // "t1" is the ID of textbox
document.getElementById("t1").select();     // "t1" is the ID of textbox
```

Example: An example of form validation – Validating subject code

"verify.js"

```
function verifycode()
{   var x = document.getElementById("t1").value;
    // matches "23OMC30X" where X is a digit between 1-9
    let pattern = /^23OMC30[1-9]$/;
      if ( x.match(pattern) == null) {
            alert("Subject Code is not correct - Its Invalid ");
            document.getElementById("t1").focus();
            document.getElementById("t1").select();
            return false;
      }
      else {
            alert("Subject code is correct - Its Valid");
            return true;
} }
```

Figure 6.24: Snippet Code of event handler – validating subject code

Valid subject codes are 23OMC301, 23OMC302, 23OMC303, …, and 23OMC309. All other subject codes entered by the user are considered invalid. The pattern used for validation is `/^23OMC30[1-9]$/.` The match() method checks for the pattern, if the valid pattern is not found it returns null and displays the error message otherwise it displays the `"Subject code is correct – It's Valid".`

XHTML document:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html>
<html>
<head>
 <title>Example of Form Validation in XHTML using JavaScript </title>
 <script type="text/javascript" src="verify.js"></script>
</head>
<body>
  <h4> GEU: Example of Form Validation in XHTML using JavaScript  </h4>
  <form method="GET">
      <h4> Validating Subject Code Example: 23OMC30
      <span style="color:red;font-weight:bold;font-size:14pt">
       X </span>  where X value is between 1 to 9  </h4>

      <label> Subject Code
        <input  type="text"  id= "t1"
              name="txtsubcode" size="35" maxlength="30"
              placeholder="Enter Subject Code"  required />
      </label><br/><br/>
       <input   type="Submit" id="t2" value="Verify" />
  </form>

      <script language="javascript">
           /* Registration of event handler using object property */
            document.getElementById("t2").onclick = verifycode;
      </script>
</body>
</html>
```

Figure 6.25: Snippet Code of XHTML document for form validation

Following is the output of the XHTML document on opening it in the browser.

Output:

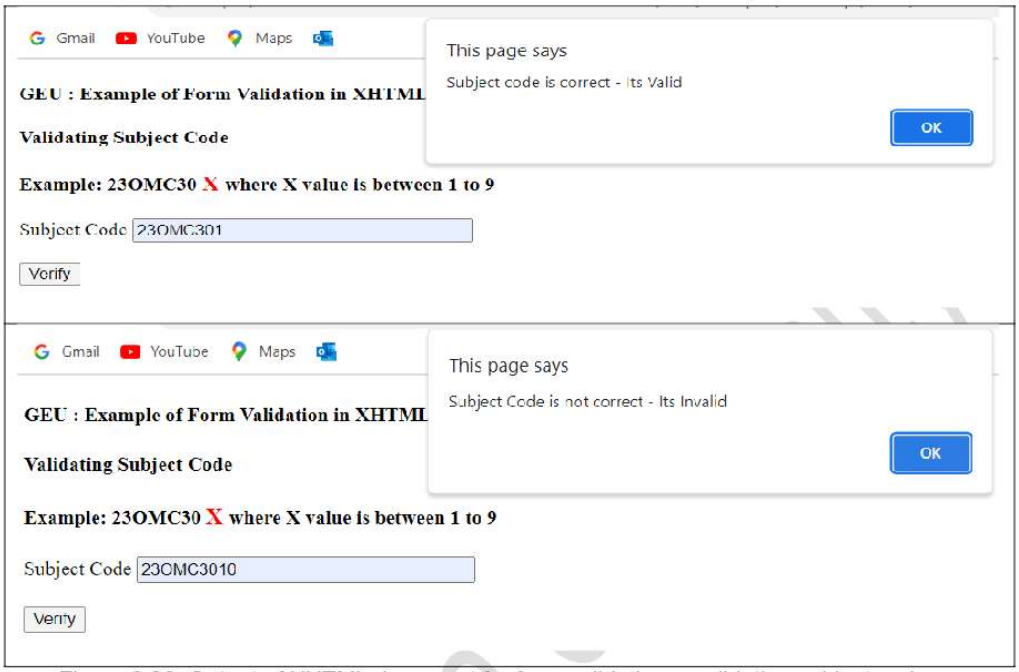

Figure 6.26: Output of XHTML document for form validation - validating subject code

6.7 Changing Colors and Fonts

The background and foreground colors, font-size of the text displayed on the web page can be changed dynamically. Changing colors and fonts on a web page can be done through CSS (Cascading Style Sheets) properties and JavaScript.

Example: An example that demonstrates how to dynamically change the background and foreground color of an XHTML element using CSS properties and JavaScript:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html >
<html>
<head>
   <title>Example of dynamically changing background and foreground
          Color
   </title>
```

```
    <script type="text/javascript" >
     function changecolor()
     {
        x = document.getElementById("backcolor");
        y = document.getElementById("forecolor");
        /* Dynamically changing background color */
        document.body.style.backgroundColor = x.value;
        /* Dynamically changing foreground color */
        document.body.style.color = y.value;
     }

 </script>
</head>
<body>
   <p> GEU - Example of dynamically changing foreground and background
       color </p>
   <form id ="f1" >

      <label> Background : <input type="text"  maxlength="10"
                           id="backcolor"   /></label>
      <label> Foreground : <input type="text"  maxlength="10"
                           id="forecolor"  /></label></p>
      <p><input type="button" id = "b1" value="Change it"
                          onclick = changecolor() /></p>

   </form>
</body>
</html>
```

Figure 6.27: Snippet Code of XHTML document for changing background and foreground colors

Here, the inputs are taken from the user and dynamically the colors of the background and foreground are changed.

The background color can be changed dynamically as follows –

```
/* "backcolor" is the ID of the textbox where the name of the color is entered by the user */
var x = document.getElementById("backcolor");
```

```
/* Dynamically changing background color */
document.body.style.backgroundColor = x.value;
```

The foreground color can be changed dynamically as follows –

```
/* "forecolor" is the ID of the textbox where the name of the color is entered by the user */
y = document.getElementById("forecolor");
/* Dynamically changing foreground color */
document.body.style.color = y.value;
```

Following is the output of the XHTML document on opening it in the browser.

Output:

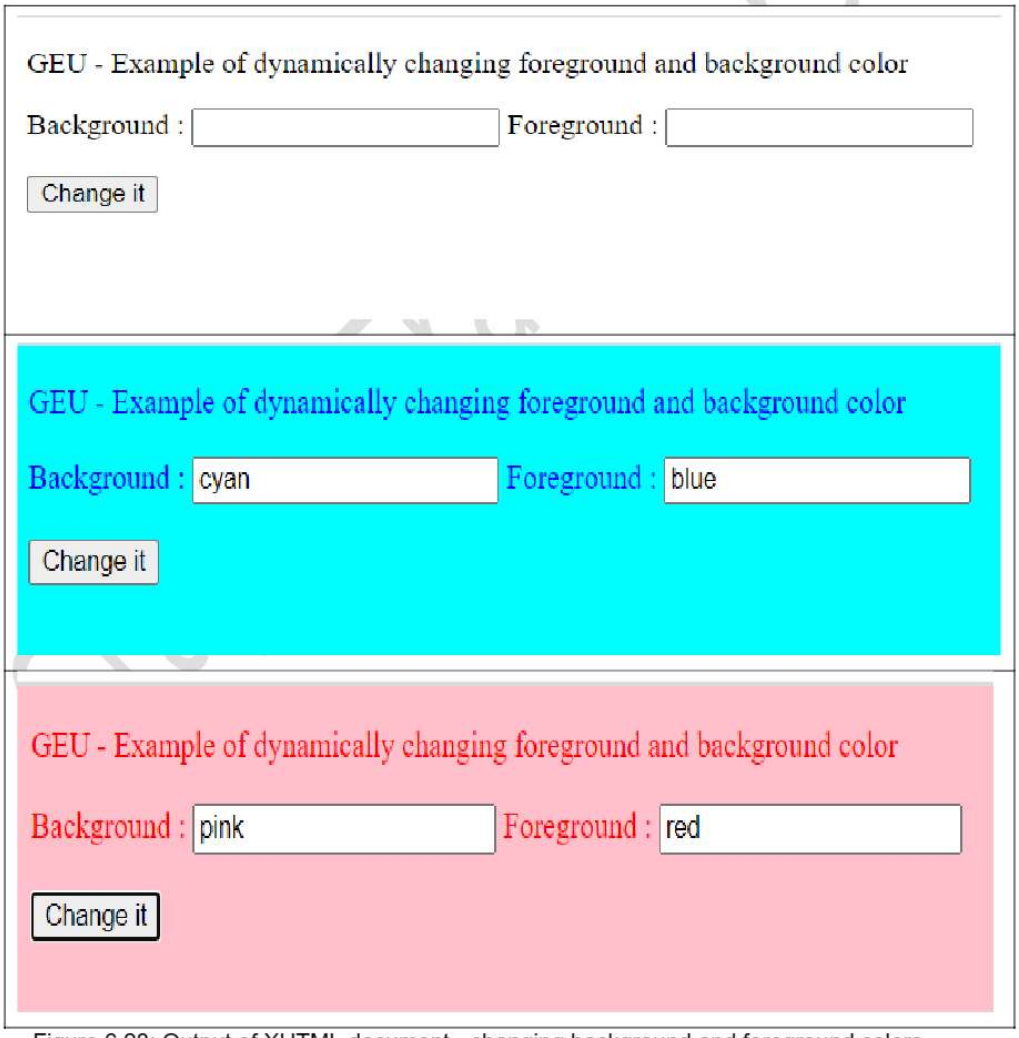

Figure 6.28: Output of XHTML document - changing background and foreground colors

Changing text color and fonts

Example: An example that demonstrates how to dynamically change the text color and the font-size an XHTML element using CSS properties and JavaScript.

XHTML document

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html>
<html>
<head>
  <title>Example of dynamically changing font-size and color </title>
</head>
<body>
   <h3> GEU - Example of dynamically changing font size and colors </h3>
     <p  onmouseover= "this.style.color ='red' ;
       this.style.backgroundColor = 'cyan';
       this.style.font = 'italic 28pt Arial' ;"
       onmouseout= "this.style.color = 'blue';
       this.style.backgroundColor = 'yellow';
       this.style.font = 'bold 20pt Times' ;"  >
       Graphic Era University, Dehradun
   </p>
</body>
</html>
```

Figure 6.29: Snippet Code of XHTML document for dynamically changing background and foreground colors and font size

Here, "onmouseover" and "onmouseout" events the font size and colors of paragraphs are changed dynamically

The font size, background, and foreground color of the paragraphs are changed dynamically on mouseover by using the tag attribute "onmouseover" in the opening tag of the paragraph – the text color is set to `'red', the` background color is set to `'cyan'` and font-size is set to `'italic 28pt Arial'` as follows -

```
onmouseover = "this.style.color ='red' ;
this.style.backgroundColor = 'cyan';
this.style.font = 'italic 28pt Arial' ;"
```

The font size, background, and foreground color of the paragraphs are changed dynamically on mouseout by using the tag attribute "onmouseout" in the opening tag of the paragraph – the text color is set to `'blue'` background color is set to `'yellow'` and font-size is set to `'bold 20pt Times'` as follows -

```
onmouseout= "this.style.color = 'blue';
this.style.backgroundColor = 'yellow';
this.style.font = 'bold 20pt Times' ;"
```

Following is the output of the XHTML document on opening it in the browser.

Output:

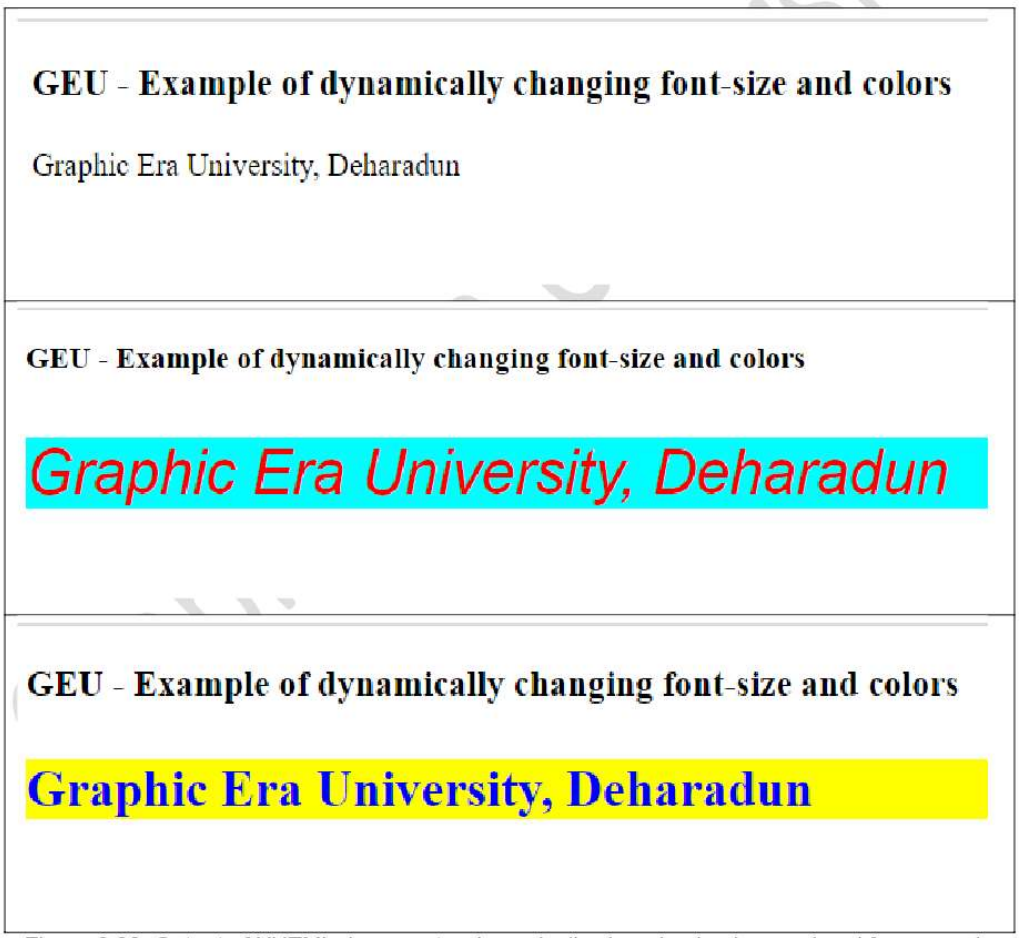

Figure 6.30: Output of XHTML document - dynamically changing background and foreground colors and font size

6.8 Self-Assessment Questions

Q1. What is DOM? Explain with a neat diagram, the DOM tree structure and the corresponding XHTML document.[8 marks, L2]

Q2. Explain the different ways of accessing XHTML elements in JavaScript with a suitable example for each. [8 marks, L2]

Q3. Describe the use of the following methods for accessing XHTML elements in JavaScript with suitable examples for each. [6 marks, L2]

- document.getElementById()
- document.getElementsByName()
- document.getElementsByTagName()
- document.getElementsByClassName()

Q4. Explain with a suitable example how XHTML elements can be accessed by using ID and name attributes. [8 marks, L2]

Q5. Explain with a suitable example how XHTML elements can be accessed by using Tag names and Class names. [8 marks, L2]

Q6. What is an event? List some of the events and the tag attributes used for the event handling in JavaScript. [5 marks, L2]

Q7. What is an event and event handler? Explain the event-handling mechanism in JavaScript with a suitable example for each [6 marks each, L2]

- handling events from body elements
- handling events from button elements
- handling events from textboxes

Q8. Explain with a suitable example how form data validation can be done on the client side using JavaScript. [8 marks, L2]

Q9. What are the benefits of form data validation on the client side? [4 marks, L2]

Q10. Explain with a suitable example the use of "focus()" and "select()" methods in event handling for form data validation. [6 marks, L2]

Q11. Explain with a suitable example how background and foreground colors of XHTML elements can be changed dynamically using CSS properties and JavaScript. [6 marks, L2]

Q12. Explain with a suitable example how the font size of XHTML elements can be changed dynamically using CSS properties and JavaScript. [6 marks, L2]

6.9 Self-Assessment Activities

A1. Explain how events and handled in JavaScript using radio buttons with a suitable example.

A2. Create a login form in an XHTML document. Validate the username and password on submitting the form data. Display appropriate messages- if the data entered by the user is invalid display an error message and if the data is valid display a success message.

6.10 Multiple-Choice Questions

1. In Web development DOM stands for _____. [1 mark, L1]
   a) Document Object Model
   b) Data Object Model
   c) Dynamic Object Model
   d) Document Order Model

2. The term "traversing the DOM" means _____.[1 mark, L1]
   a) Navigating through the browser's history
   b) Moving up and down the DOM hierarchy to access elements
   c) Changing the document's structure dynamically
   d) Scrolling through the content of a web page

3. The primary function of the DOM tree in web development is ____. [1 mark, L1]
   a) Handling server-side logic
   b) Managing database queries
   c) To represent the structure of XHTML documents
   d) Controlling the browser's navigation

4. The _____method is commonly used to access an XHTML element by its ID in JavaScript.[1 mark, L1]
   a) document.accessElementById
   b) document.getElementById
   c) document.selectElementById
   d) document.findElementById

5. An event in the context of web development is _____. [1 mark, L1]
   a) A style applied to an XHTML element

b) A predefined JavaScript function

c) A user action or occurrence detected by the browser

d) A type of XHTML tag

6. Event handling in JavaScript is _____. [1 mark, L1]

a) The process of capturing and responding to events in the browser

b) Attaching styles to HTML elements

c) Creating new events in the DOM

d) Changing the structure of the DOM

7. ____is used to access and modify the content of an XHTML element in JavaScript. [1 mark, L1]

a) changeContent

b) textContent

c) modifyText

d) innerHTML

8. In JavaScript, the getElementsByTagName method returns _____. [1 mark, L1]

a) All elements with a specific class name

b) All elements with a specific tag name

c) The first element with a specific ID

d) The last element in the document

9. Event handler is nothing but _____.[1 mark, L1]

a) a function that handles or responds to an event.

b) an interface that handles or responds to an event.

c) an event

d) None of the above

10. The background color and font color can be changed in the JavaScript when the form is successfully submitted _____.[1 mark, L1]

a) By using the changeStyle function.

b) By setting the bodyColor and fontColor properties.

c) By directly modifying the style property of the document.body.

d) None of the above

6.11 Key Answers to Multiple-Choice Questions

1. In Web development DOM stands for Document Object Model.[a]
2. The term "traversing the DOM" means moving up and down the DOM hierarchy to access elements.[b]
3. The primary function of the DOM tree in web development is to represent the structure of XHTML documents.[c]
4. The document.getElementById method is commonly used to access an XHTML element by its ID in JavaScript.[b]
5. An event in the context of web development is a user action or occurrence detected by the browser.[c]
6. Event handling in JavaScript is the process of capturing and responding to events in the browser.[a]
7. innerHTML is used to access and modify the content of an XHTML element in JavaScript.[d]
8. In JavaScript, the getElementsByTagName method returns all elements with a specific tag name.[b]
9. Event handler is nothing but a function that handles or responds to an event.[a]
10. The background color and font color can be changed in JavaScript when the form is successfully submitted by directly modifying the style property of the document.body.[c]

6.12 Summary

The Document Object Model (DOM) is an application programming interface(API) for web documents that defines an interface between XHTML documents and application programs. It represents the structure of an XHTML document as a tree of objects, where each object(node) corresponds to a part of the XHTML document, such as elements, attributes, and text. The objects have methods and properties that are associated with their respective node types. JavaScript can get access to XHTML as well as CSS of the web pages using DOM. The DOM provides a way for programs to traverse and manipulate the structure, css style, and content of web documents dynamically. JavaScript interprets DOM easily, using it as a bridge to access and manipulate the elements using different methods and properties of the document object.

Accessing elements in JavaScript is a fundamental aspect of web development, and it involves selecting and interacting with XHTML elements in the DOM (Document Object Model). The structure of an XHTML document can be changed dynamically. There are various methods and properties associated with objects that facilitate element access in

JavaScript. From the Document Object Model (DOM), elements in an XHTML document can be accessed in Javascript code using different ways.

Some of the ways of accessing elements are

- get XHTML elements by ID - document.getElementById(),
- get XHTML elements by name - document.getElementsByName(),
- get XHTML elements by tag name - document.getElementsByTagName(),
- get XHTML elements by class name - document.getElementsByClassName()

In web development, an event is an occurrence of an action or incident that happens in the browser, such as a click of a button, hovering the mouse, resizing the window, pressing a key, loading or unloading of page, and many more. Events can be triggered by the user or by the browser itself.

Users interact with the XHTML document elements like buttons, radio buttons, checkboxes, text boxes, images, etc. by either using a mouse or keyboard. The actions that take place due to the user's interactions with the keyboard are called keyevents and those with the mouse are called mouseevents. Some events that occur due to browser windows are called window/document events. Events are JavaScript, their names are case-sensitive. For example, a click is an event, but a Click is not.

Key events examples - keydown, keypress, keyup, etc.

Mouse events examples - click, dblclick, mouseover, mouseout, etc

Window/Document events examples – resize, scroll, load, unload, etc.

When an event occurs a specific task is performed in response to an event, this is called event handling. It is required to define and execute the code that should run in response to a specific event. All possible events that can occur have to be handled by an application. It should not happen that a user clicks a button and there is no response. Managing these events is called Event Handling.

There are 2 ways of event registration in the DOM 0 event model in JavaScript.

- Assign event handler to tag attributes
- Assign event handler address to object property

Events can be handled from body elements, button elements, text boxes, etc.

Form validation is an essential part of web development to ensure that user input is accurate and meets the specified criteria. Validation of form inputs, before sending form data to the server reduces the processing load on the server and also network traffic. Validations on the client side prevent unwanted/inaccurate/bad data from being sent to the server for processing and provide quick responses to the users thereby reducing response time.

The background and foreground colors, font size of the text displayed on the web page can be changed dynamically. Changing colors and fonts on a web page can be done through CSS (Cascading Style Sheets) properties and JavaScript.

6.13 Keywords

- Document Object Model (DOM)
- Tree-Structure
- Event
- Event handler
- Event Registration
- Tag attributes
- innerHTML
- Form validation

6.14 Recommended resources for further reading

a. Essential Reading

1. Sebesta, R. W. (2010). Programming the World Wide Web (6th ed.), Pearson education.
2. Subramanian, V. (2019). Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node (2nd Ed.), Apress.

b. Recommended Reading

1. DT Editorial Services. (2016). HTML 5: Covers CSS3, JavaScript, XML, XHTML, AJAX, PHP & jQuery: Black Book, Dreamtech Press.
2. Koroliova, E. W. I., (2018). MERN Quick Start Guide: Build Web applications with MongoDB, Express.js, React and Node, Packt.

--*--