10.1 Using JavaScript with HTML

Basics

Like a glossy brochure, a web page with just HTML and CSS is nice to look at but lacks interaction with the user. ***JavaScript*** is a programming language that enables a web page to have flexible behavior by specifying actions to be taken when events happen in the browser. The ***Document Object Model*** (or DOM) is a general way to represent and access all parts of HTML and XML documents. Together, JavaScript and the DOM provide browsers with the tools for making web pages interactive by accessing and changing any part of a web page.

The browser provides an object named ***document*** that is a data structure representing the entire web page DOM, including HTML and all other resources included by the web page, like CSS, image, and JavaScript files. Changes made to that data structure will be reflected in the browser presentation and/or behavior.

HTML5 pages add JavaScript code by using the `<script>` tag. JavaScript code between `<script></script>` tags is executed by the browser's JavaScript engine.

PARTICIPATION ACTIVITY 10.1.1: Writing JavaScript within the body of an HTML file.

The JavaScript code below uses the ***document.writeln()*** method, which outputs HTML into the document and alters the DOM.

1. Read the HTML and JavaScript below.
2. Render the web page to run the JavaScript code that displays a randomly generated response.
3. Add more responses to the `responses` array, and render the web page a few times until one of your new responses is displayed.

```
<!DOCTYPE html>
<html>
<title>Magic 8 Ball</title>
<meta charset="UTF-8">
<body>
   <h1>Magic 8 Ball</h1>
   <script>

   // Possible 8 Ball responses
   var responses = [ "Without a doubt", "Ask again later", "Don't count on it"

   // Display a randomly chosen response
   var randomNum = Math.floor(Math.random() * responses.length);
   document.writeln("<p>Magic 8 Ball says... <strong>" + responses[randomNum]

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


Render web page Reset code

Your web page

PARTICIPATION ACTIVITY 10.1.2: JavaScript Basics.

1) The DOM only refers to the HTML portions of the document.

   - ○ True
   - ○ False

2) The DOM is accessible via the global object named document.

- True
- False

3) `document.writeln("<div>test</div>")` adds a div element to the DOM.

- True
- False

Window object

JavaScript running in a web browser has access to the ***window*** object, which represents an open browser window. In a tabbed browser, each tab has a `window` object. The `document` object is a property of the `window` object and can be accessed as `window.document` or just `document`. Other properties of the `window` object include:

- ***window.location*** is a location object that contains information about the window's current URL. Ex: `window.location.hostname` is the URL's hostname.
- ***window.navigator*** is a navigator object that contains information about the browser. Ex: `window.navigator.userAgent` returns the browser's user agent string.
- ***window.innerHeight*** and ***window.innerWidth*** are the height and width in pixels of the window's content area. Ex: `window.innnerWidth` returns 600 if the browser's content area is 600 pixels wide.

The `window` object defines some useful methods:

- ***window.alert()*** displays an alert dialog box. Ex: `window.alert("Hello")` displays a dialog box with the message "Hello".
- ***window.confirm()*** displays a confirmation dialog box with OK and Cancel buttons. `confirm()` returns true if OK is pressed and false if Cancel is pressed. Ex: `window.confirm("Are you sure?")` displays a dialog box with the question.
- ***window.open()*** opens a new browser window. Ex: `window.open("http://www.twitter.com/")` opens a new browser that loads the Twitter web page.

PARTICIPATION ACTIVITY 10.1.3: Using the window object.

Use the `window.confirm()` method to ask if the user would like to see a popup window:

```
var okPressed = window.confirm("Would you like to see a popup window?");
```

Then render the web page, and click the OK button when prompted to see a small browser window created by `window.open()`. You may need to give your browser

permission to show the popup window since many browsers prevent popups from displaying by default.

```
<!DOCTYPE html>
<html>
<title>JavaScript Demo</title>
<meta charset="UTF-8">
<body>
   <h1>Popup Demo</h1>
   <script>

   var okPressed = false;
   if (okPressed) {
      var myWindow = window.open("", "", "width=250, height=100");
      myWindow.document.writeln("<h1>Hello, Popup!</h1>");
   }

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

```


Render web page Reset code

Your web page

PARTICIPATION ACTIVITY 10.1.4: Window object.

1) Window object properties or methods can be accessed without putting `window.` in front of the property or method.

   ○ True

- False

2) What window object property is useful for determining if the web page is loaded using HTTPS or HTTP?

- location
- navigator

3) What window object property likely produces the following output?

```
document.writeln(window._____);
```

```
Mozilla/5.0 (Windows NT 10.0;
WOW64) AppleWebKit/537.36
Chrome/53.0.2785.116
```

- navigator.userAgent
- navigator.language

4) What window method is ideal for displaying a pop-up advertisement?

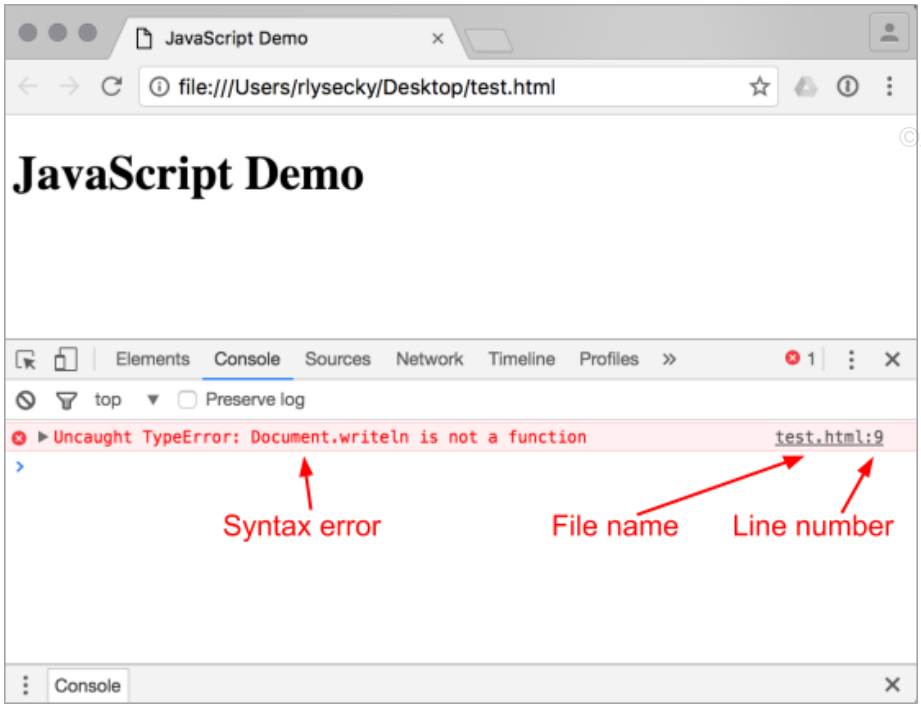
- alert
- open

Using the console

Modern browsers provide a ***console*** that allows the JavaScript code to produce informational and debugging output for the web developer, which does not affect the functionality or presentation of the web page. By default, the console is not open for viewing on a page. The console is viewable in Chrome by pressing Ctrl+Shift+J in Windows/Linux or Cmd+Opt+J on a Mac.

When a syntax error is present in JavaScript code or a run-time error occurs, the error is only made visible in the console. The figure below shows the syntax error created when the developer accidentally typed `Document.writeln()` with a capital "D". The console appears underneath the web page. Good practice is to leave the console open while writing and testing JavaScript code.

Figure 10.1.1: Chrome console showing a syntax error on line 8 of test.html.




```
<!DOCTYPE html>
<meta charset="UTF-8">
<html>
<title>JavaScript Demo</title>
<body>
   <h1>JavaScript Demo</h1>
   <script>

   Document.writeln("<p>Hello, JavaScript!</p>");

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

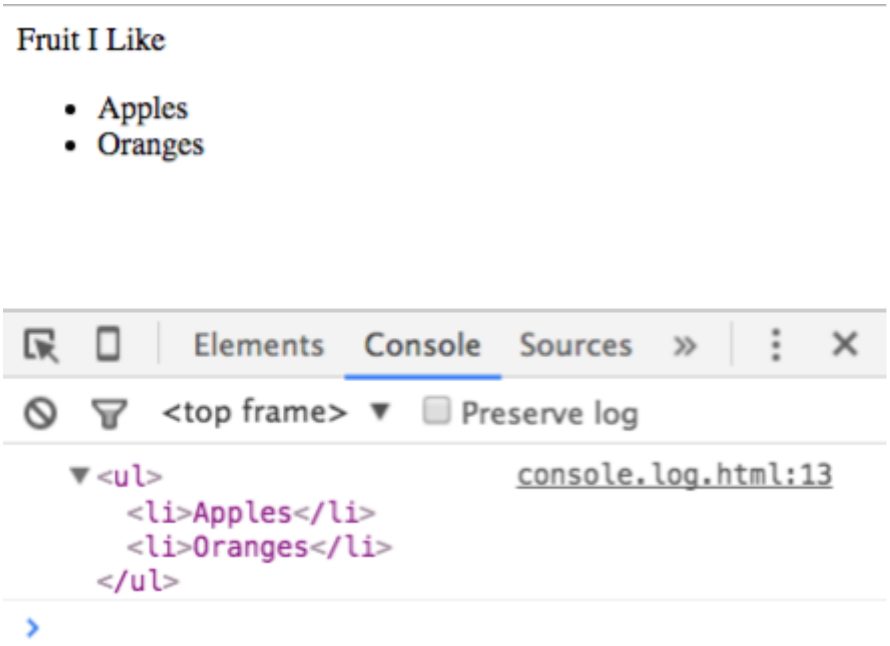
The browser provides a `console` object with a defined set of methods, or API, that the `console` object supports. An ***API*** (***Application Programming Interface***) is a specification of the methods and objects that defines how a programmer should interact with software components. The console API includes the following methods:

- ***console.log()*** displays informational data to the console.
- ***console.warn()*** displays warnings to the console. The browser usually has a special indicator to differentiate a warning from the standard log message. Ex: A yellow warning box.
- ***console.error()*** displays errors to the console. The browser usually has a special indicator to differentiate an error from a warning or the standard log message. Ex: A red error box.

- ***console.dir()*** displays a JavaScript object to the console. The browser usually supports a method for compactly representing the object. Ex: a hierarchical tree representation allowing a developer to expand and collapse the object contents.

Figure 10.1.2: console.log() output example.

When the web browser console is open, both the web page and the console are simultaneously visible.



console.log() can print both strings and concise representations of HTML elements.

```
<body>
   <p>
      Fruit I Like
   </p>
   <ul>
      <li>Apples</li>
      <li>Oranges</li>
   </ul>

   <script>
   console.log(document.getElementsByTagName("ul")[0]);
   </script>
</body>
```


PARTICIPATION ACTIVITY 10.1.5: Match terms with definitions.

Match the console method with the best use for that method.

dir | error | log | warn

	Helping determine why an algorithm isn't working as expected.
	Displaying a structured JavaScript object.
	Checking that assumptions in an algorithm are correct.
	Reporting unexpected problems.

Reset

Loading JavaScript from an external file

The <script> tag can be used to include JavaScript directly within the HTML file and is common practice when using small amounts of JavaScript. However, writing JavaScript directly within the document leads to a number of problems as a web page or website gets larger.

Good practice is to use the <script> tags to load JavaScript from external files, rather than writing the JavaScript directly within the HTML file. The <script> `src` attribute specifies a JavaScript file to load.

Example 10.1.1: Loading JavaScript from an external file.

```
<script src="bootstrap.js"></script>
```


A common error when loading an external JavaScript file is to forget the closing </script> tag, or trying to use a self-closing <script /> tag as in <script src="bootstrap.js" />. All modern browsers require a closing </script> tag.

PARTICIPATION ACTIVITY 10.1.6: Loading JavaScript from an HTML file.

Animation captions:

1. The web server sends the HTML file to the web browser.
2. Web browser reads the HTML file. Script tag with src attribute indicates the browser should load JavaScript from an external file.
3. Web browser requests JavaScript file from the web server.
4. Web browser reads the JavaScript file. JavaScript alert statement displays a dialog box and waits for the user to press enter.
5. After the user presses enter, web browser finishes reading the JavaScript file and continues reading the HTML file.
6. Web browser requests the image file, and the web server responds with the image file.
7. Web browser finishes reading HTML file.

PARTICIPATION ACTIVITY 10.1.7: Downloading JavaScript files.

1) A web browser will process the HTML following a script element that uses an external JavaScript file while the browser waits for the web server to return the JavaScript file.
   - True
   - False

2) One script element can be used to include both inline JavaScript and a reference to an external JavaScript file.
   - True
   - False

3) One script element can be used to reference multiple external JavaScript files.
   - True
   - False

Loading JavaScript with async and defer

Although the <script> tag can be included anywhere in the head or body, good practice is to include the <script> tag early in the document with the `async` and/or `defer` attributes set.

The `async` attribute allows the browser to process the web page concurrently with loading and processing the JavaScript.

Example 10.1.2: Loading JavaScript with the async attribute.

```
<script src="bootstrap.js" async></script>
```


PARTICIPATION ACTIVITY 10.1.8: Using the async attribute with the <script> tag.

Animation captions:

1. The web server sends the HTML file to the web browser.
2. Web browser reads the HTML file. Script tag's async attribute causes browser to continue reading HTML without waiting for JavaScript file to load.
3. Web server responds with the JavaScript file while the browser requests the image file.
4. Web browser begins reading the JavaScript file and pauses reading the HTML file. The web server concurrently responds to the image request. JavaScript alert statement displays a dialog box and waits for the user to press enter.
5. After the user presses enter, web browser finishes reading the JavaScript file and continues processing the HTML file by displaying the dog.jpg image that was received.
6. Web browser finishes reading HTML file.

The `defer` attribute allows the browser to load the web page concurrently with loading the JavaScript, but the JavaScript is not processed until the web page is completely loaded.

Example 10.1.3: Loading JavaScript with the defer attribute.

```
<script src="bootstrap.js" defer></script>
```


PARTICIPATION ACTIVITY 10.1.9: Using the defer attribute with the <script> tag.

Animation captions:

1. The web server sends the HTML file to the web browser.
2. Web browser reads the HTML file. Script tag's defer attribute causes browser to continue reading HTML without waiting for JavaScript file to load.
3. Web server responds with the JavaScript file while the browser requests the image file.
4. Web browser does not immediately process the JavaScript file due to the defer attribute. Instead, the browser continues to process the HTML.
5. After reading the HTML file, the web browser reads the JavaScript file. JavaScript alert statement displays a dialog box and waits for the user to press enter.
6. After the user presses enter, web browser finishes reading the JavaScript file.

PARTICIPATION ACTIVITY 10.1.10: Loading JavaScript.

1) The browser interprets the `defer` and `async` attributes for the script element the same.
   - True
   - False

2) When using a third-party JavaScript library, the `defer` attribute is usually better than the `async` attribute.
   - True
   - False

3) When writing custom JavaScript, the `defer` attribute is usually better than the `async` attribute.
   - True
   - False

4) Most web pages on the internet were written before the `defer` or `async` attributes were standardized.
   - True
   - False

Minification and obfuscation

To reduce the amount of JavaScript that must be downloaded from a web server,

developers often minify a website's JavaScript. ***Minification*** *or* ***minimization*** *is the process of removing unnecessary characters (like whitespace and comments)*

from JavaScript code so the code executes the same but with fewer characters. Minification software may also rename identifiers into shorter ones to reduce space. Ex: `var totalReturns = 10;` *may be converted into* `var a=10;`.

Minified JavaScript is typically stored in a file with a ".min.js" file extension. An example of minified code from the Bootstrap project is shown below.


```
// Excerpt from bootstrap.min.js
a.fn.button=b,a.fn.button.Constructor=c,a.fn.button.noConflict=function(){
return a.fn.button=d,this},a(document).on("click.bs.button.data-api",
'[data-toggle^="button"]',function(c){var d=a(c.target).closest(".btn");
b.call(d,"toggle"),a(c.target).is('input[type="radio"],
```

A JavaScript ***obfuscator*** *is software that converts JavaScript into an unreadable form that is very difficult to convert back into readable JavaScript. Developers obfuscate a website's JavaScript to prevent the code from being read or re-purposed by others. Obfuscated code may also be minified and appear in a ".min.js" file.*

CHALLENGE ACTIVITY 10.1.1: JavaScript with HTML.

Start

Use the writeln method of the document object to display the inner height in a <p> tag in the webpage. Hint: The innerHeight property of the window object contains the inner height.

```
1 <h1>Demo</h1>
2 <script>
3
4    <!-- Your solution goes here -->
5
6 </script>
```

1 2

Check Next

Exploring further:

- Window object from MDN
- Console object from MDN
- async vs defer attributes from Growing with the Web
- JavaScript minifiers: javascript-minifier.com and jscompress.com
- JavaScript obfuscators: javascriptobfuscator.com and JS-obfus

10.2 Document Object Model (DOM)

Document Object Model (DOM) structure

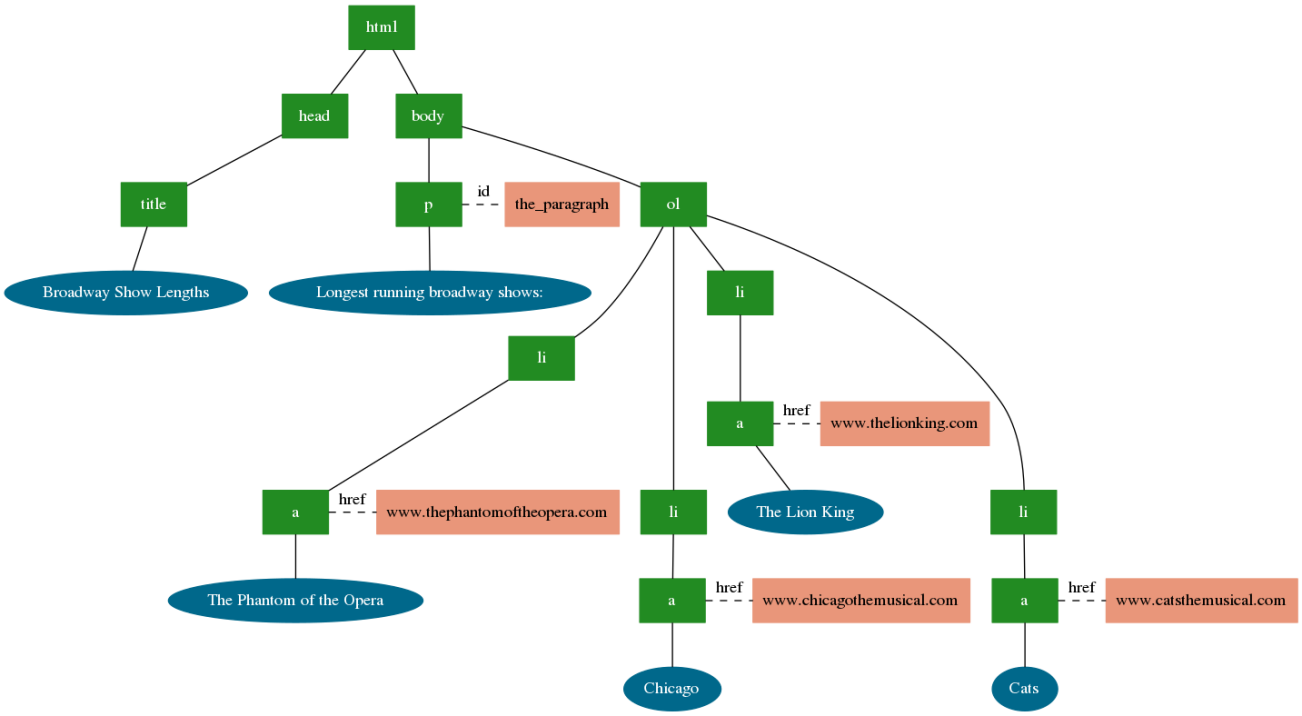
The Document Object Model (DOM) is a data structure corresponding to the HTML file. A DOM tree is a visualization of the DOM data structure. ***document.documentElement*** is the root of the DOM tree (the "top" node). Each tag in an HTML file is associated with an object in the DOM tree. A ***node*** is an individual object in the DOM tree.

DOM nodes have properties for accessing a node's parent and children nodes. The ***childNodes*** property is an array-like collection of objects for each of the node's children. Each node has a property ***parentNode*** that refers to the parent element containing the node. HTML attributes are represented in the DOM as similarly named properties on the corresponding node. Ex: The HTML title attribute becomes the node's title property in the corresponding node.

The DOM tree visualization below corresponds to the given HTML. Green nodes represent HTML elements. Pink nodes represent HTML attributes. Blue nodes represent text content of HTML elements. Note the <meta> tag is not included in the DOM tree since the meta tag is used to describe the contents of the document and is not part of the document contents.

Figure 10.2.1: Visualizing the DOM tree.

```
<!DOCTYPE html>
<html>
   <meta charset="UTF-8">
   <title>Broadway Show Lengths</title>
   <body>
      <p id="the_paragraph">Longest running broadway shows:</p>
      <ol>
         <li><a href="http://www.thephantomoftheopera.com/">The Phantom of the Opera</a>
</li>
         <li><a href="http://www.chicagothemusical.com/">Chicago</a></li>
         <li><a href="http://www.thelionking.com/">The Lion King</a></li>
         <li><a href="http://www.catsthemusical.com/">Cats</a></li>
      </ol>
   </body>
</html>
```

PARTICIPATION ACTIVITY 10.2.1: Creating the DOM from an HTML file.

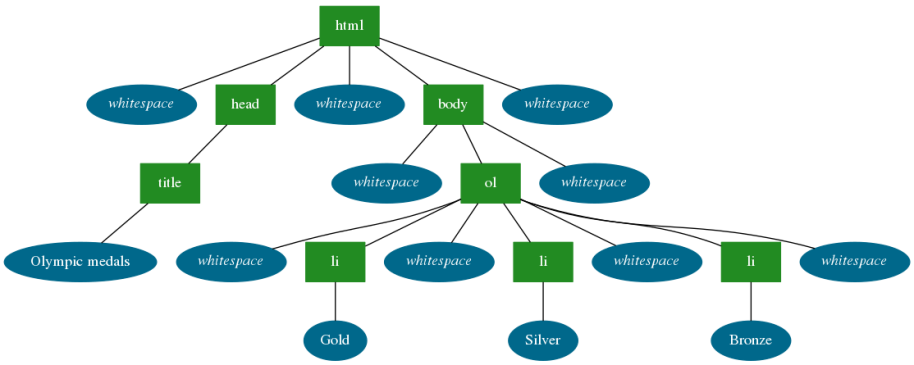
Animation captions:

1. Web browser reads the HTML file and creates the DOM root element.
2. The body element is a child of the root node within the DOM. The p element is contained within the body element, so the p node is a child of the body node in the DOM.
3. An attribute node is created in the DOM for the paragraph element's class attribute.
4. A text node is created for the paragraph element's text content.
5. Browser continues reading the HTML and creating DOM nodes as appropriate.

An idealized representation of the DOM tree excludes all text nodes that only contain whitespace. Normally, the DOM should be conceptualized with that idealized representation. However, a web developer occasionally needs to know the complete DOM tree, which includes whitespace as shown in the example below.

Figure 10.2.2: Complete DOM tree visualization with whitespace text nodes.


```
<!DOCTYPE html>
<html>
   <meta charset="UTF-8">
   <title>Olympic medals</title>
   <body>
      <ol>
         <li>Gold</li>
         <li>Silver</li>
         <li>Bronze</li>
      </ol>
   </body>
</html>
```



The DOM provides the ***children*** property for a node, which is similar to the childNodes property except that the children property only contains other DOM nodes and does not contain textual content. A common error is to use the childNodes property instead of the children property when iterating through the items of a list. The children property only contains the list items, while the childNodes property also contains the whitespace textual nodes between the list items.

PARTICIPATION ACTIVITY 10.2.2: DOM structure.

node **children** **childNodes** **documentElement** **parentNode**

	A DOM object created from an HTML element.

	The DOM object representing the top HTML element, which contains the rest of the document.
	A node's array-like object that contains the HTML elements and textual content directly contained within that node.
	A node's array-like collection that only contains the HTML elements directly contained within that node.
	A reference to a node's parent node.

Reset

Traversing the DOM

The document object provides five primary methods of accessing specific nodes within the DOM:

1. ***document.getElementById***: returns the DOM node whose id attribute is the same as the method's parameter.
   Ex: document.getElementById("early_languages") returns the p node in the HTML below.
2. ***document.getElementsByTagName***: returns an array containing all the DOM nodes whose type is the same as the method's parameter.
   Ex: document.getElementsByTagName("li") returns a list of the four li nodes from in the HTML below.
3. ***document.getElementsByClassName***: returns an array containing all the DOM nodes whose class attribute matches the method's parameter.
   Ex: document.getElementsByClassName("traditional") returns an array containing the ol node with the class attribute matching the word traditional.
4. ***document.querySelectorAll***: returns an array of containing all the DOM nodes that match the CSS selector passed as the method's parameter.
   Ex: document.querySelectorAll("li a") returns an array containing the two `a` nodes in the HTML below .
5. ***document.querySelector***: returns the first element found in the DOM that matches the CSS selector passed as the method's parameter. document.querySelector expects the same types of parameters as document.querySelectorAll, but only returns the first element found

while navigating the DOM tree in a depth-first traversal.
Ex: document.querySelector("li") returns the li node about Fortran.

Figure 10.2.3: Example HTML for traversing the DOM.

```
<!DOCTYPE html>
<html>
   <meta charset="UTF-8">
   <title>Early Programming Languages</title>
   <body>
      <p id="early_languages">Early Programming Languages Still Used:</p>
      <ol class="traditional">
         <li><a href="https://en.wikipedia.org/wiki/Fortran">Fortran - 1954</a>
</li>
         <li><a
href="https://en.wikipedia.org/wiki/Lisp_(programming_language)">Lisp - 1958</a>
</li>
         <li>COBOL - 1959</li>
         <li>BASIC - 1964</li>
      </ol>
   </body>
</html>
```


Note

A DOM traversal method name indicates whether the method returns one node, or an array of nodes. If the method name starts with getElements or ends in All, then the method will return an array, even if the array contains one node or is empty. getElementById and querySelector methods either return a single node, or return null if no node matched the method parameters.

PARTICIPATION ACTIVITY 10.2.3: DOM traversal.

Refer to the HTML below.

```
<!DOCTYPE html>
<html>
   <meta charset="UTF-8">
   <title>Web development languages</title>
   <body>
      <p>Languages used in web development.</p>
      <ul id="list">
         <li id="item-1">HTML for content</li>
         <li id="item-2">CSS for presentation</li>
         <li id="item-3">JavaScript for functionality</li>
      </ul>
   </body>
</html>
```

1) Which HTML elements are returned by: `document.getElementById("list")`

- The paragraph tag (<p>)
- The unordered list tag (<ul>)
- One of the list item tags (<li>)
- All of the list item tags ([<li>, <li>,])

2) Which HTML elements are returned by:

```
document.getElementsByTagName("li")
```

- The paragraph tag (<p>)
- The unordered list tag (<ul>)
- One of the list item tags (<li>)
- All of the list item tags ([<li>, <li>,])

3) Which HTML element is referenced by:

```
document.getElementsByTagName("ul")
[0]
```

- The paragraph tag (<p>)
- The unordered list tag (<ul>)
- One of the list item tags (<li>)
- All of the list item tags ([<li>, <li>,])

4) Which HTML elements are referenced by:

```
document.getElementsByTagName("ul")
[0].children
```

- The paragraph tag (<p>)
- The unordered list tag (<ul>)
- One of the list item tags (<li>)
- All of the list item tags ([<li>, <li>,])

5) Which HTML element is referenced by:

```
document.documentElement.children[1]
```

-

The body tag (<body>)

- The meta tag (<meta>)
- The title tag (<title>)
- The paragraph tag (<p>)

6) Which HTML element is referenced by:

```
document.documentElement.children[1].children[1]
```


- The paragraph tag (<p>)
- The unordered list tag (<ul>)
- One of the list item tags (<li>)
- All of the list item tags ([<li>, <li>,])

Additionally, the DOM provides accessor properties for each node that allow direct navigation to sibling or parent nodes:

1. ***nextSibling*** is a node property that refers to the node with the same parent following the current node in the document. Ex: In the figure below, the ol node is the nextSibling for the p node.
2. ***prevSibling*** is a node property that refers to the node with the same parent preceding the current node in the document. Ex: In the figure below, the p node is the prevSibling for the ol node.
3. ***parentNode*** is a node property that refers to the current node's parent node. Ex: In the figure below, the ol node is the parentNode for all of the li nodes.

Figure 10.2.4: Example HTML for sibling methods.

```
<!DOCTYPE html>
<html>
   <meta charset="UTF-8">
   <title>Geologic eons of earth</title>
   <body>
      <p>The four geologic eons on earth:</p>
      <ol>
         <li><a href="https://en.wikipedia.org/wiki/Hadean">Hadean</a></li>
         <li><a href="https://en.wikipedia.org/wiki/Archean">Archean</a></li>
         <li><a href="https://en.wikipedia.org/wiki/Proterozoic">Proterozoic</a></li>
         <li><a href="https://en.wikipedia.org/wiki/Phanerozoic">Phanerozoic</a></li>
      </ol>
   </body>
</html>
```


PARTICIPATION ACTIVITY 10.2.4: Using nextSibling, prevSibling, and parentNode.

Refer to the HTML below.

```
<!DOCTYPE html>
<html>
   <meta charset="UTF-8">
   <title>Web development languages</title>
   <body>
      <p>Languages used in web development.</p>
      <ul id="list">
         <li id="item-1">HTML for content</li>
         <li id="item-2">CSS for presentation</li>
         <li id="item-3">JavaScript for functionality</li>
      </ul>
   </body>
</html>
```


1) Which HTML element is referenced by:

```
document.getElementById("item-2").prevSibling
```

- The list item node with the id item-1.
- The list item node with the id item-2.
- The list item node with the id item-3.
- The ul node with the id list.

2) Which HTML element is referenced by:

```
document.getElementById("item-2").parentNode
```

- The list item node with the id item-1.
- The list item node with the id item-2.
- The list item node with the id item-3.
- The ul node with the id list.

3) Which HTML element is referenced by:

```
document.getElementById("item-1").nextSibling
```

- The list item node with the id item-1.
- The list item node with the id

item-2.

- The list item node with the id item-3.
- The ul node with the id list.

DOM attributes

Every attribute for an HTML element has an identically named property in the element's DOM node. Ex: `<a href="http://www.nasa.gov/" id="nasa_link">NASA</a>` has a corresponding DOM node with properties named href and id. Each attribute property name acts as both a getter and a setter.

Using the property name to read the value allows a program to examine the attribute's value. Writing to a property allows a program to modify the attribute, which is reflected in the rendered web page. Ex:
`document.getElementById("nasa_link").href = "http://www.spacex.com/"` changes the hyperlink for the element with id of nasa_link to refer to SpaceX.

By modifying attribute properties, JavaScript programs can perform actions including:

- Change which image is displayed by modifying an <img> node's src attribute.
- Determine which image is currently being displayed by reading the <img> node's src attribute.
- Change the style of part of the page by modifying a node's class attribute.
- Change the functionality of the page by modifying an element's event attributeEx: Change the onsubmit attribute for a form node.

Each DOM node provides a method named ***removeAttribute*** which takes a name of an attribute as a parameter. The removeAttribute method removes the corresponding attribute from the node. Ex: If `aNode` is an anchor node in the DOM, `aNode.removeAttribute("href")` removes the link from the anchor so that clicking on the HTML element no longer performs an action.

PARTICIPATION ACTIVITY 10.2.5: Reading and modifying DOM node attribute values.

Refer to the HTML below.


```
<!DOCTYPE html>
<html>
   <meta charset="UTF-8">
   <title>Orion mission</title>
   <body>
      <h1>Orion moon mission update.</h1>
      <img src="inFlight.png" id="status" alt="Image of spaceship en route to the
moon">
   </body>
</html>
```

1) Fill in the missing attribute to make the image load success.png.

```
var pic =
document.getElementById("status");
pic._______ = "success.png";
```

Answer field

Check Show answer

2) Fill in the node and attribute required to read the current image name.

```
var pic =
document.getElementById("status");
if ( __________ != "success.png")
{
    alert("Wrong picture!");
}
```

Answer field

Check Show answer

Modifying DOM nodes

The ***nodeValue*** property sets or gets the value of text nodes. As the DOM tree represents textual content separately from HTML elements, the textual content of an HTML element is the first child node of the HTML element's node. So, to access the textual content of an HTML element within the DOM, .firstChild.nodeValue is used to access the value of the HTML's element's first child.

Ex:
`document.getElementById("saleprice").firstChild.nodeValue = "$25.99"`:

1. Gets the DOM node for the element with id "saleprice",
2. uses .firstChild to access the textual content node for the element, and then
3. uses nodeValue to update the content.

The ***innerHTML*** property sets or gets a DOM node's content, including all of the node's children, as a string instead of as a tree. Ex: The innerHTML property of an ordered list element can be assigned a string containing multiple list items elements using
`"<li>first item</li><li>second item</li><li>third item</li>"`

The innerHTML property is not officially part of the W3C standardized API, but every browser has implemented innerHTML for over a decade. Many web developers prefer to modify the DOM using innerHTML because multiple changes to the DOM can be made with one line of code.

Additionally, the innerHTML property depends on the internal parser for the web browser, which is better optimized than manually written DOM API methods.For those reasons, good practice is to use innerHTML whenever possible.

PARTICIPATION ACTIVITY 10.2.6: .firstChild.nodeValue vs. .innerHTML.

Animation captions:

1. JavaScript uses the firstChild.nodeValue to modify the paragraph's contents.
2. JavaScript uses .innerHTML to modify the entire contents of the ordered list. The argument to .innerHTML specifies three list itemts to replace the existing contents.

Each HTML element has a ***style*** object that represents the active CSS styling for the element. The style object has separate properties for each CSS rule applied to the element. The name of the style properties are the same as the CSS properties with some minor exceptions. Ex: The CSS property `background-color` becomes the JavaScript property `backgroundColor`. Setting the style property for a specific HTML element in the JavaScript has the same effect as if the CSS had a selector that matched the same HTML element, which implies that the property may cascade to lower elements in the DOM tree. Ex: `document.getElementsByTagName("body")[0].style.color = "red"` changes the text color of all HTML elements to red unless an element's text color was explicitly overridden by another style rule.

PARTICIPATION ACTIVITY 10.2.7: Using .removeAttribute(), .innerHTML, and .style with DOM nodes.

Add JavaScript in the changePage function so that clicking on the Use Current Astronomy button does the following:

1. Uses removeAttribute() to remove the hidden attribute from the paragraph with the id p2, causing the paragraph to become visible.
2. Uses the .innerHTML property of the span with the id lastPlanet to change the name of the farthest planet to "Neptune". The quotation marks around Neptune are necessary.
3. Uses .style.textDecoration to set style attribute of the span text to "underline". The quotation marks around "underline" are necessary.

Use document.getElementById() to access the DOM nodes.

HTML | JavaScript

```
<body>
   <h1>The farthest planet</h1>

   <p id="p1">Pluto was discovered in 1930 and designated as a planet.</p>
   <p id="p2" hidden>In 2006, Pluto was reclassified as a dwarf planet.</p>

   <p><span id="lastPlanet">Pluto</span> is the farthest planet from the Sun.<

   <input type="button" value="Use Current Astronomy" onClick="changePage()">
</body>
```


Render web page Reset code

Your web page

Modifying the DOM structure

Each DOM node object provides methods for changing node locations within the DOM:

- The ***appendChild*** method appends a DOM node to the child nodes of the method's caller. The code below moves the first ordered list's first list item to the last list item of the same ordered list.

```
ol = document.getElementsByTagName("ol")[0];
li = ol.getElementsByTagName("li")[0];
ol.appendChild(li);
```

- The ***insertBefore*** method inserts a DOM node as a child node before an existing child node of the method's caller The code below moves the first ordered list's fourth list item to the

first list item of the same ordered list.

```
ol = document.getElementsByTagName("ol")[0];
items = ol.getElementsByTagName("li");
ol.insertBefore(items[0], items[3]);
```

- The ***removeChild*** method removes a node from the method's caller's children. The most common usage pattern is to get a DOM node, n, and call removeChild on n's parent passing n as a parameter. Ex: `n.parentNode.removeChild(n)`

PARTICIPATION ACTIVITY 10.2.8: Adding and removing DOM nodes.

removeChild | appendChild | insertBefore

	a method on a DOM node which moves one DOM node to be a previous sibling to another DOM node
	a method on a DOM node which deletes a DOM node from the DOM tree
	a method on a DOM node which turns another DOM node into the first DOM node's last child

Reset

The document object provides methods for creating new nodes:

- The ***createElement*** method creates a DOM node from a string parameter for an HTML element. Ex: `document.createElement("p")` creates a new paragraph DOM node. The createElement method does not add the created DOM node to the DOM tree, so the programmer must use appendChild or insertBefore to add the new node to the existing DOM tree. A common error is to forget to add a newly created node to the DOM tree.
- The ***createTextNode*** method creates a DOM node containing the text specified by a string argument. Ex: `document.createTextNode("new paragraph contents")` creates the text "new paragraph contents", which can be added to the paragraph node created above. A text node must be added to the DOM using appendChild or insertBefore.

Existing nodes in the DOM can be duplicated using cloneNode.

- The ***cloneNode*** method creates a DOM node or tree identical to the tree rooted at the method's caller. The method's boolean argument indicates whether the method should clone the node's children. Ex: `x.cloneNode(true)` creates an identical tree as that rooted at x, and `x.cloneNode(false)` creates a single node identical to x. The created tree or node must be added to the DOM using appendChild or insertBefore. A common error is to forget to modify any id attributes in the cloned tree. The cloneNode method does not ensure that new nodes have unique id attributes.

PARTICIPATION ACTIVITY 10.2.9: Creating new DOM nodes.

1) Which method for a DOM node creates a copy of the node and the node's children?
   - createElement
   - createTextNode
   - cloneNode(true)
   - cloneNode(false)

2) Which method for the document object creates a new DOM node for a specific HTML element?
   - createElement
   - createTextNode
   - cloneNode(true)
   - cloneNode(false)

3) Which method for the document object creates a new DOM node to hold content?
   - createElement
   - createTextNode
   - cloneNode(true)
   - cloneNode(false)

4) Which method of a DOM node creates a copy of another DOM node, but not the node's children?
   - createElement
   - createTextNode

- cloneNode(true)
- cloneNode(false)

CHALLENGE ACTIVITY 10.2.1: Using the Document Object Model.

Start

Assign listNodes with all elements with a class name of 'programming-language'.

HTML | JavaScript

```
<>Top 10 TIOBE index for June 2017:</> <!-- https://www.tiobe.com/tiobe-index/
<ol class="languages-list">
   <li class="programming-language">Java</li>
   <li class="programming-language">C</li>
   <li class="programming-language">C++</li>
   <li class="programming-language">Python</li>
   <li class="programming-language">C#</li>
   <li class="programming-language">Visual Basic .NET</li>
   <li class="programming-language">JavaScript</li>
   <li class="programming-language">PHP</li>
   <li class="programming-language">Perl</li>
   <li class="programming-language">Assembly Language</li>
</ol>
```

1 2 3 4 5 6

Check Next

Exploring further:

- Document Object Model (DOM) from MDN

10.3 Event-driven programming

An ***event*** is an action, usually caused by a user, that the web browser responds to. Ex: A mouse movement, a key press, or a network response from a web server. Typically, the occurrence and timing of an event are unpredictable, since the user or web server can perform an action at any time.

Event-driven programming is a programming style where code runs only in response to various events. Code that runs in response to an event is called an ***event handler*** or ***event listener***.

The web browser supports event-driven programming to simplify handling the many events a web page must process. When an event happens, the browser calls the event's specified handlers. The web browser internally implements the code for detecting events and executing event handlers, thus helping web developers focus on writing the event handlers.

PARTICIPATION ACTIVITY 10.3.1: Focus and blur event handling.

Animation captions:

1. User clicks in the Name input box. Browser calls the input element's focus event handler, which changes the element's style. Browser then gives focus to input box.
2. User key presses are sent to Name input box.
3. User clicks the Answer input box. Browser calls the Name element's blur event handler, then calls the Answer element's focus handler, and then gives focus to the Answer input box.
4. User key presses are sent to Answer input box.
5. When the user clicks elsewhere, the browser calls the Answer blur event handler.

PARTICIPATION ACTIVITY 10.3.2: Event-driven programming.

1) The actions a web browser notices are called handlers.
   - ○ True
   - ○ False

2) A web developer must implement the code to detect events and call the appropriate handlers.
   - ○ True
   - ○ False

3) A mouse click causes an event.
   - ○ True

False

Each event is given a name that represents the corresponding action. Ex: The event name for a mouse movement is `mousemove`, and the event name for a key press is `keypress`.

PARTICIPATION ACTIVITY 10.3.3: Mouse and keyboard events.

mousemove **keydown** **keyup** **mouseout** **keypress** **click** **mouseover**

	Caused by a mouse click.
	Caused by mouse entering the area defined by an HTML element.
	Caused by mouse exiting the area defined by an HTML element.
	Caused by mouse moving within the area defined by an HTML element.
	Caused by the user pushing down a key on keyboard.
	Caused by the user releasing a key on the keyboard.
	Caused by the user pressing and releasing a key on the keyboard.

Reset

The following are events for which web developers commonly write handlers.

- A ***change*** event is caused by an element value being modified. Ex: Selecting an item in a radio button group causes a change event.
- An ***input*** event is caused when the value of an input or textarea element is changed.
- A ***load*** event is caused when the browser completes loading an HTML element, usually the body.
- A ***DOMContentLoaded*** event is caused when the HTML file has been loaded and parsed, although other related resources such as CSS, JavaScript, and image files may not yet be loaded.
- A ***focus*** event is caused when an element becomes the current receiver of keyboard input. Ex: Clicking in an input field causes a focus event.
- A ***blur*** event is caused when an element loses focus and the element will no longer receive future keyboard input.
- A ***submit*** event is caused when the user submits a form to the web server.

PARTICIPATION ACTIVITY 10.3.4: Other common browser events.

1) A submit event occurs when any button is clicked.
   - True
   - False

2) A blur event occurs when the mouse is moved over another input element.
   - True
   - False

Setting up event handlers

Handlers are written in three ways:

1. Embedding the handler as part of the HTML. Ex: `<button onclick="clickHandler()">Click Me</button>` sets the click event handler for thebutton element by using the `onclick` attribute. The attribute name used to register the handler adds the prefix "on" to the event name. Ex: The attribute for a mousemove event is `onmousemove`. Embedding a handler in HTML mixes content and functionality and thus should be avoided whenever possible.
2. Setting the DOM node event handler property directly using JavaScript. Ex: `document.getElementById("myButton").onclick = clickHandler` sets the click event handler for the element with an id of "myButton" by overwriting the `onclick` JavaScript property. Using DOM node properties is better than embedding handlers within

the HTML but has the disadvantage that setting the property only allows one handler for that element to be registered.

3. Using the JavaScript ***addEventListener()*** method to register an event handler for a DOM object. Ex:
`document.getElementById("myButton").addEventListener("click", clickHa`
registers a click event handler for the element with the id "myButton". Good practice is to use the `addEventListener()` method whenever possible, rather than using element attributes or overwriting JavaScript properties. The `addEventListener()` method allows for separation of content and functionality and allows multiple handlers to be registered with an element for the same event.

Every handler has an optional ***event object*** parameter that provides details of the event. Ex: For a keypress event, the event object indicates which key was pressed, or for a click event, which element was clicked.

PARTICIPATION ACTIVITY 10.3.5: Registering event handlers with addEventListener().

Animation captions:

1. The window's addEventListener() method registers the handler loadedHandler() for the DOMContentLoaded event.
2. After the rest of the HTML is loaded and parsed, the DOMContentLoaded event occurs, and loadedHandler() is called.
3. The text box's addEventListener() method registers the handler keypressHandler() for the keypress event.
4. When the user types the first letter, a keypress event occurs, which results in keypressHandler() being called.
5. The keypressHandler's event object's charCode attribute is the Unicode value of the pressed key (80 for P).
6. Each keypress causes keypressHandler() to execute. When the user presses Enter, event.charCode is 13, and the if statement is true.
7. The event object's target attribute is the text box object that caused the keypress event. An alert dialog displays "Hello, Pam!"

In the animation above, the `keypressHandler()` used `event.target` to access the text box object where the keypress event occured. Inside an event handler, the `this` keyword refers to the element to which the handler is attached. So `event.target` and `this` both refer to the text box object in the event handler.

PARTICIPATION ACTIVITY 10.3.6: Registering event handler using addEventListener.

Create and register a JavaScript event handler, `myClickHandler`, to handle click events for each element with the `hide` class attribute. When the click event occurs for an element, myClickHandler should reveal the hidden text by changing the `style.color` of the `event.target` to `"black"`.

HTML | JavaScript | CSS


```
<!DOCTYPE html>
<html>
<title>Event Demo</title>
<body>
   <p>
      Challenge your knowledge about event-driven programming by guessing the
      each sentence below.  See if you have guessed correctly by revealing the
   </p>

   <ul>
      <li>
         Event handlers are also known as
         <span class="highlight"><span class="hide">callback functions</span><
         because the handlers are "called back" when the appropriate event hap
      </li>
      <li>
         Event-driven programming allows web pages to react to
         <span class="highlight"><span class="hide">user actions</span></span>
         <span class="highlight"><span class="hide">web server actions</span><
```

Render web page Reset code

Your web page **Expected web page**

PARTICIPATION ACTIVITY 10.3.7: Registering event handlers.

Refer to the HTML below.

```
<body>
  <h1>Calculator</h1>
  <input type="text" id="num1" size="5"><br>
  <input type="text" id="num2" size="5"><br>
  <input type="button" value="Add" id="addBtn">
</body>
```

1) What event registers `loadedHandler()` to be executed after the HTML has been loaded and parsed?


```
window.addEventListener("_____",
loadedHandler);
```

- click
- DOMContentLoaded
- ready

2) What is missing to register the `addNumbers()` function as a click event handler?

```
function loadedHandler() {
   var addBtn =
document.getElementById("addBtn");

addBtn.addEventListener("click",
_______);
}
```

- addNumbers()
- addNumbers(1, 2)
- addNumbers

3) What code registers an anonymous function as a click event handler for the add button?


```
function loadedHandler() {
   var addBtn =
document.getElementById("addBtn");

addBtn.addEventListener("click",
_______);
}
```

- function addNumbers() { ... }
- function() { ... }
- function { ... }

4) The `highlightField()` function is an event handler for the mouseover and mouseout events. What parameter is `highlightField()` missing?

```
function highlightField(______) {
   if
(event.target.style.background ==
"yellow") {

event.target.style.background =
"white";
   }
   else {

event.target.style.background =
"yellow";
   }
}
```

- event
- field
- color

5) What parameter is `highlightField()` missing to change the `textBox` background color to yellow?

```
textBox.addEventListener("mouseover",
highlightField);
function highlightField() {
   ____.style.background = "yellow";
}
```

- event.target
- event
- this

Capturing, at target, and bubbling phases

When an event occurs, the browser follows a simple DOM traversal process to determine which handlers are relevant and need to be called. This traversal process follows three phases: capturing, at target, and bubbling.

1. In the ***event capturing*** phase, the browser traverses the DOM tree from the root to the event target node, at each node calling any event-specific handlers that were explicitly registered for activation during the capturing phase.

2. In the ***at target*** phase, the browser calls all event-specific handlers registered on the target node.
3. In the ***event bubbling*** phase, the browser traverses the DOM tree from the event target node back to the root node, at each node calling all event-specific handlers registered for the bubbling phase on the current node.

The optional third parameter for the `addEventListener` method indicates whether the handler is registered for the capturing phase or bubbling phase. If the third parameter is `false` or not specified, or if the event handler is registered using any other mechanism, the browser registers the handler for the event bubbling phase. If the parameter is `true`, the browser registers the handler for the capturing phase.

Some events do not bubble, such as blur, focus, and change. When a non-bubbling event occurs, the browser will follow the event capturing phase, the at target phase, and then stop.

PARTICIPATION ACTIVITY 10.3.8: Capturing and bubbling.

Animation captions:

1. User moves the mouse cursor over item two in the list. A mouseover event occurs with the second li node as the target node.
2. Event capturing phase traverses the DOM tree from the root to event target node. No capturing handlers are registered for the mouseover event.
3. At target phase looks for mouseover event handlers registered on the target node, but no handlers are registered for the mouseover event.
4. Event bubbling traverses DOM tree from the event node back to the root node. The ol node's bubbling event handler is called.
5. A mouseout event occurs with the second li node as the target node.
6. Event capturing phase traverses the DOM tree from the root to event target node. The ol node's mouseout event handler is called.
7. At target phase looks for relevant mouseover event handlers registered on the target node, but no handlers are registered for the mouseout event.
8. Event bubbling phase looks for any relevant mouseout event handlers by moving up to the DOM tree, but no elements have mouseout event handlers registered.

PARTICIPATION ACTIVITY 10.3.9: Capturing and bubbling.

Given the HTML and JavaScript below, match the order of alerts to the action performed by the user.

```
<div id="div1">
   <div id="div2">
      <div id="div3">
      </div>
   </div>
</div>
```

```
var div1 = document.getElementById("div1");
var div2 = document.getElementById("div2");
var div3 = document.getElementById("div3");

div1.addEventListener("click", function(){ alert("Capture 1"); }, true);
div2.addEventListener("click", function(){ alert("Capture 2"); }, true);
div3.addEventListener("click", function(){ alert("Capture 3"); }, true);

div1.addEventListener("click", function(){ alert("Bubble 1"); });
div3.addEventListener("click", function(){ alert("Bubble 3"); });
```


Capture 1, Bubble 1

Capture 1, Capture 2, Capture 3, Bubble 3, Bubble 1

Capture 1, Capture 2, Bubble 1

	User clicks on div with `div1` id attribute.
	User clicks on div with `div2` id attribute.
	User clicks on div with `div3` id attribute.

Reset

Preventing default behavior

The event capturing and bubbling process can be stopped by calling the ***stopPropagation()*** method on the event object provided to the handler. Once the stopPropagation is called, the browser stops the traversal process but still calls relevant registered handlers on the current node.

A web developer may want to prevent the browser from using a built-in handler for an event. Ex: Whenever a user clicks a form's submit button, the web browser sends the form data to the web server. The event object's ***preventDefault()*** method stops the web browser from performing the built-in handler. The built-in handlers that are often prevented are clicking elements, submitting forms, and moving the mouse into or out of an element.

PARTICIPATION

PARTICIPATION ACTIVITY | 10.3.10: Bubbling and capturing.

1) The web browser performs the event capturing process before the bubbling process.
   - True
   - False

2) A web developer cannot prevent the web browser from performing built-in handlers.
   - True
   - False

3) If a web developer creates a "default" handler for a DOM node high in the tree and a more specific handler for a DOM node lower in the tree, the web browser will run both handlers for an event.
   - True
   - False

4) Bubbling is the preferred process for the web browser to find appropriate handlers for an event.
   - True
   - False

Timers

Some events are related to time instead of user actions. Ex: A website may wish to refresh stock data at regular intervals to show updated inventory information. Web browsers provide four methods for handling time-based events:

1. The ***setTimeout()*** method takes two arguments: an event handler, and a time delay in milliseconds (1/1000th of a second) until the timeout occurs. The browser calls the handler after the timeout has occurred. The `setTimeout()` method returns a unique integer identifier that refers to the timeout that was created. Ex: `var uid = setTimeout(function() { alert("Hello!"); }, 2500);` causes a timeout event to occur after about 2.5 seconds.

2. The ***clearTimeout()*** method takes one argument: a unique identifier for a timeout that has been created. If the timeout has not yet occurred, `clearTimeout()` turns off the associated timer. Otherwise, `clearTimeout()` doesn't do anything. Ex: `clearTimeout(uid);` stops the timer and causes the timeout event associated with the `uid` unique identifier to not occur.
3. The ***setInterval()*** method takes two arguments: a recurring timeout in milliseconds (t), and a handler. The browser calls the handler every t milliseconds until the recurring timeout has been canceled. A recurring timeout is also called an interval. The `setInterval()` method returns a unique integer identifier that refers to the recurring timeout that was created.
4. The ***clearInterval()*** method takes one argument: a unique identifier for an interval that has been created. After a call to `clearInterval()`, no further calls to the associated handler will be made.

PARTICIPATION ACTIVITY 10.3.11: Timeouts.

1) `setInterval()` and `setTimeout()` both return identifiers for their associated handlers.
   - ○ True
   - ○ False

2) Since `clearInterval()` and `clearTimeout()` take an identifier as a parameter, the methods can be used interchangeably.
   - ○ True
   - ○ False

3) The following two JavaScript fragments are functionally equivalent.

```
setTimeout(function() {
alert("Hello!"); }, 2500);
```

```
function hello() {
   alert("Hello!");
}
setTimeout(hello, 2500);
```

   - ○ True
   - ○ False

PARTICIPATION

PARTICIPATION ACTIVITY | 10.3.12: Interval event handler.

Modify the JavaScript to create a countdown timer.

1. Add code to `startbutton`'s click event handler to start a recurring timeout that will call the `countdown()` function every second.
2. Store the unique identifier returned by `setInterval()` so the recurring timeout events can be canceled.
3. Add code to `countdown()` function to clear the countdown recurring timeout.
4. Add code to `stopbutton`'s click event handler to clear the countdown recurring timeout.

HTML | JavaScript

```
<!DOCTYPE html>
<html>
<body>
   <p>Enter the countdown starting number, then click the start button.</p>
   <input type="number" id="number" min="0" value="5">
   <input type="button" id="startbutton" value="start">
   <input type="button" id="stopbutton" value="stop" disabled>
</body>
</html>

```

Render web page | Reset code

©zyBooks 06/02/19 18:13 473675
Irving Jimenez
StrayerCIS273Spring2019

Your web page	Expected web page

CHALLENGE ACTIVITY 10.3.1: Event-driven programming.

Start

Register the updateCount event handler to handle input changes for the textarea tag.
Note: The function counts the number of characters in the textarea.

HTML | JavaScript

```
1 <label for="userName">User name:</label>
2 <textarea id="userName" cols="40" rows="3"></textarea><br>
3 <p id="stringLength">0</p>
4
```


1 2 3 4

Check Next

Exploring further:

- Event reference from MDN
- EventTarget.addEventListener() from MDN
- Event flow tutorial from Java2s
- JavaScript timers from MDN

10.4 Form validation

Since data integrity is essential to most applications, many web forms require specific formats for users to enter data. Ex: A credit card must contain 16 digits, a date cannot have a fifteenth month, and only 50 valid names of states exist for the United States of America. While a web server must check that the submitted data is valid, a better user experience occurs when the web client also performs the same checks before posting. Any invalid data on the client can be immediately noticed and flagged as needing modifications without waiting for the server to respond. Validation can either be done as the user enters data in the form by adding a JavaScript function as the change handler for the appropriate field, or immediately prior to submitting the entire form by adding a function as the form's submit.

PARTICIPATION ACTIVITY 10.4.1: Validating form input.

Animation captions:

1. The web page uses JavaScript to validate the web form.
2. User enters invalid form data and does not check the checkbox.
3. User clicks the submit button. The browsers executes code to validate the form input, and highlights invalid fields in red.
4. User must correct the form data before submitting the form data to the web server. After the user clicks the submit button again, the browser updates the page to reflect that all data is valid.

Each textual input element in an HTML document has a ***value*** attribute that is associated with the user-entered text. The `value` attribute can be used to validate user-entered text by checking desired properties, such as:

- Checking for a specific length using the `length` property on the `value` attribute
- Checking if entered text is a specific value using `===`
- Checking is the text contains a specific value using the string `indexOf()` method on the `value` attribute
- Checking that text matches a desired pattern using a regular expression and the string `match()` method

Drop-down menus also have a `value` attribute that is associated with the user-selected menu option .

Checkboxes and radio buttons have a ***checked*** attribute that is a boolean value indicating whether the user has chosen a particular checkbox or radio button. The checked attribute can be used to ensure an input element is either checked or unchecked before form submission. Ex: Agreeing to a website's terms of service.

PARTICIPATION ACTIVITY 10.4.2: Identify why the field is invalid.

1) Enter 5-digit ZIP code:

ZIP

- Some input characters are not digits.
- The input field is empty.
- The input is too long.

2) Enter 5-digit ZIP code:

103

- Some input characters are not digits.
- The input field is empty.
- The input length is incorrect.

3) Enter 5-digit ZIP code:

31M4N

- Some input characters are not digits.
- The input field is empty.
- The input length is incorrect.

Validating form data upon submission

Validating form data using JavaScript that executes when the user submits the form can be performed by:

1. Register a handler for the form's submit event that executes a validation function.
2. Within the validation function, inspect the form's input fields via the appropriate DOM elements and element attributes.
3. If the form is invalid, call the `preventDefault()` method on the event to cancel the form submission and prevent the form data from being sent to the server.

Figure 10.4.1: Ensuring a checkbox is selected before the form is submitted.

```
<!DOCTYPE html>
<html>
<meta charset="UTF-8">
<title>Terms of Service</title>
<script src="validate.js" defer></script>
<body>
   <form id="tosForm" action="https://example.com" target="_blank" method="POST">
      <label for="tos">I agree to the terms of service:</label>
      <input type="checkbox" id="tos"><br>
      <input type="submit">
   </form>
</body>
</html>
```

```
function checkForm(event) {
   var tosWidget = document.querySelector("#tos");

   // Cancel form submission if tos not checked
   if (!tosWidget.checked) {
      event.preventDefault();
   }
}

var formWidget = document.querySelector("#tosForm");
formWidget.addEventListener("submit", checkForm);
```

PARTICIPATION ACTIVITY 10.4.3: Practice validating form prior to submission.

Complete the JavaScript `validateForm()` function so that `validateForm()` sets the input `style.backgroundColor` to `LightGreen` for each field that passes the validation check and sets the input field's `style.backgroundColor` to `Orange` if the validation fails.

Validation rules:

- The screen name field must not be empty.
- The ZIP code field must be of length 5.
- The TOS field must contain "yes".

HTML | CSS | JavaScript

```
<!DOCTYPE html>
<html>
<meta charset="UTF-8">
<title>Terms of Service</title>
<body>
   <form id="myForm">
      <label for="screenName">Screen name:</label>
      <input type="text" id="screenName" name="screenName" placeholder="Screen
      <label for="zip">ZIP code:</label>
      <input type="text" id="zip" name="zip" placeholder="5-digit ZIP code">
      <label for="tos">Type yes if you agree to the terms of service:</label>
      <input type="text" name="agreement" id="tos">
      <input type="button" id="validate" value="Validate Form">
   </form>
</body>
</html>
```


Render web page Reset code

Your web page **Expected web page**

Validating each field as data is entered

Alternatively, form data can be validated as the user enters data in the form by:

1. For each field that should be validated:

   a. Register an input event handler for the field.
   b. Create a global variable to track whether the field is currently valid. In most cases, this global variable should be initialized to false since the form typically starts with the field as invalid.
   c. Modify the global variable as appropriate within the field's event handler.
2. Register a submit event handler for the form that verifies the global variables for each field are true.
3. If one or more of the global variables are false, call the `preventDefault()` method on the event to cancel the form submission and keep the form from being sent to the server.

Note

*The JavaScript function **isNaN(num)** returns true if the parameter, num, is not a number. Mozilla's isNaN provides a comprehensive discussion of how to determine whether a value represents a JavaScript number.*

Figure 10.4.2: Checking a ZIP code field as the user updates the field.

```
<!DOCTYPE html>
<html>
<meta charset="UTF-8">
<title>Terms of Service</title>
<script src="validate.js" defer></script>
<body>
   <form id="tosForm">
      <input type="text" id="zipcode">
      <input type="submit">
   </form>
</body>
</html>
```

```
var zipcodeValid = false;
var zipcodeWidget = document.querySelector("#zipcode");
zipcodeWidget.addEventListener("input", checkZipcode);

function checkZipcode() {
   var val = zipcodeWidget.value;
   zipcodeValid = val.length == 5 && !isNaN(parseInt(val, 10));
}

var formWidget = document.querySelector("#tosForm");
formWidget.addEventListener("submit", checkForm);

function checkForm(event) {
   if (!zipcodeValid) {
      event.preventDefault();
   }
}
```

PARTICIPATION ACTIVITY 10.4.4: Using JavaScript to validate input fields.

1) What does the validation function return for `checkGrade("-5")`?

```
function checkGrade(grade) {
   return grade.length > 0 &&
!isNaN(grade);
}
```


- true
- false
- null

2) What does the validation function return for `checkGrade("95.3")`?

```
function checkGrade(grade) {
   return !isNaN(grade) &&
      grade >= 0 && grade <= 10;
}
```

- true
- false

3) What does the validation function return for `checkTemperature("-40")`?

```
function checkTemperature(temp) {
   return temp.length > 0 &&
!isNaN(temp) &&
      temp >= 0 && temp <= 1000;
}
```

- true
- false

4) What does the validation function return for `checkTemperature("    ")`?


```
function checkTemperature(temp) {
   return temp.length > 0 &&
!isNaN(temp) &&
      temp >= 0 && temp <= 1000;
}
```

true

false

Using HTML5 form validation

HTML5 form elements enable the browser to do form validation automatically, which reduces the need for JavaScript validation.

Note

A browser that does not support a particular HTML5 input element will transform that element into a text input, which then requires JavaScript to validate the form data.

HTML5 provides customized input elements that can only contain valid values, such as date or color. Customized elements are automatically checked by the browser and/or filled in by a pop-up input picker in the browser, ensuring the submitted value matches a common specification. HTML5 also provides attributes that allow the browser to do validation without using JavaScript (Ex: required, max, pattern, etc.). HTML5 form validation attributes include:

- The ***required*** attribute indicates that the field must have a value (text or selection) prior to submitting the form.
- The ***max*** and ***min*** attributes indicate the maximum and minimum values respectively that can be entered in an input field with ranges, such as a date or number.
- The ***maxlength*** and ***minlength*** attributes indicate the maximum and minimum length of input allowed by an input field.
- The ***pattern*** attribute provides a regular expression that valid input must match.
- The ***title*** attribute can be used to provide a description of valid input when using the pattern attribute.

Figure 10.4.3: Using HTML5 form validation.


```
<form>
  <input type="range" name="age" min="5" max="120">
  <input type="checkbox" name="agree" required>
  <input type="password" name="password" minlength="10" maxlength="16">
  <input type="text" name="credit" pattern="^\d{16}$" title="exactly 16 digits">
  <input type="submit">
</form>
```

HTML5 forms provide pseudo-classes to help CSS styling of forms. The following are four examples of HTML5 pseudo-classes:

1. The ***:valid*** pseudo-class is active on an element when the element meets all the stated requirements in field attributes.
2. The ***:invalid*** pseudo-class is active on an element when one or more of the attributes in the field are not fully met.
3. The ***:required*** pseudo-class is active on an element if the element has the `required` attribute set.
4. The ***:optional*** pseudo-class is active on an element if the element does not have the required attribute set.

PARTICIPATION ACTIVITY 10.4.5: Form validation questions.

1) If all the fields in a form have been validated before submitting the form data to a server, does the server need to repeat the field validation?
   - Yes
   - No

2) Is validating input fields as the user fills in each field better than validating the entire form after all the form data has been entered?
   - Yes
   - No

3) If validation shows that a form input value is invalid, should the input value be reset to the initial value?
   - Yes
   - No

4) If most browsers support HTML5, can web developers do all form data validation using HTML5 input element attributes and not use JavaScript validation?
   - Yes
   - No

CHALLENGE ACTIVITY 10.4.1: Form validation.

Start

Use the checkSubmittedForm function to validate the form when the user clicks "submit".

HTML | JavaScript

```
1 <form id="ageForm" action="https://learn.zybooks.com" method="POST">
2    <label for="userAge">Age:</label>
3    <input type="number" id="userAge"><br>
4    <input type="submit" id="submitForm">
5 </form>
```

1 2 3 4

Check Next

Exploring further:

- Data form validation from MDN