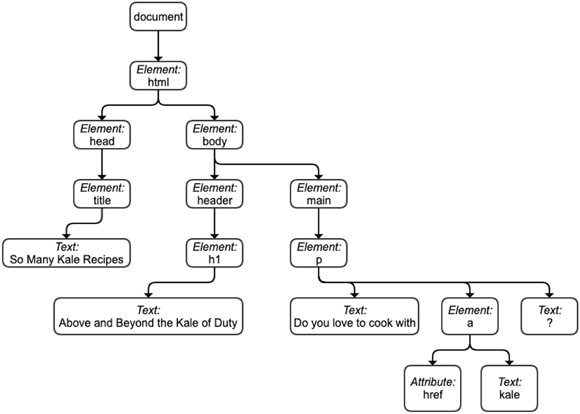
JavaScript Class – July 17, 2024

Document Object Model

1. document object
   1. a build-in JavaScript construct referring to whatever HTML file the script resides in
      1. document.write() – tells the browser to insert whatever is within the parentheses to the web page
      2. document.lastModified (method or property?) – returns the date and time the file was last changed and saved
      3. document.location – holds the address of the document currently displayed in the browser window
2. history object
   1. history.back() – tells browser to go back to the previously visited page
3. confirm() - ???
4. Introducing the Document Object Model
   1. Code for a simple web page:  
        
      <html lang=”en”>  
       <head>  
       <title>So Many Kale Recipes</title>  
       </head>  
       <body>  
       <header>  
       <h1>Above and Beyond the Kale of Duty</h1>  
       </header>  
       <main>  
       <p>  
       Do you love to cook with <a href=”kale.html”>kale</a>?  
       </p>  
       </main>  
       </body>  
      </html>
      1. html element is topmost element
         1. every other element contained within it
      2. next level down is head and body elements
         1. head contains title element
            1. title element contains text So Many Kale Recipes
         2. body element contains header and main element
   2. Here is the hierarchy:
      1. The arrows represent parent-to-child relationships
      2. Elements on the same level are siblings
      3. Every box represents an object
      4. Every object is one of three types:
         1. element
         2. text
         3. attribute
      5. Every object, regardless of its type, is called a node
      6. The page as a whole is represented by the document object
   3. This hierarchical object representation is known as the Document Object Model, or DOM
   4. The DOM enables your JavaScript code to access the complete structure of an HTML document
5. Specifying an element by id
   1. You can reference the element directly in your script as follows:
      1. Assign an id to your element  
           
         <div id=”kale-quotations”>
      2. Refer to the element in your code by using the document object’s getElementById() method
         1. Format - document.getElementById(id)
         2. To return a reference to the above div tag:  
              
            document.getElementById(“kale-quotations”)
   2. When coding the document object, don’t put <script> tag in the web page’s <head> section, because that will cause the browser to run the code before it has had a chance to create the document object
      1. Means your code will fail
      2. Instead place at the bottom of the web page, just before the </body> tag
6. Specifying elements by tag name
   1. Can work with collections of elements also, such as all the elements that use the same tag name (all the <a> tags, all the <div> tags, etc.)
   2. Mechanism for returning a collection of elements that have the same name tag is the getElementsByTagName() method  
        
      document.getElementsByTagName(tag)
      1. tag is a string representing the HTML name used by the tags you want to work with
   3. Method returns an array-like collection containing all the elements in the document that use the specified tag
      1. To return a collection that includes all the <div> elements in the current page:  
           
         const divs = document.getElementsByTagName(“div”);
7. Specifying elements by class name
   1. To return all the elements that share a specific class name:  
        
      document.getElementsByClassName(class)  
      1. class is a string representing the class name used by the elements you want to work with
   2. method returns an array-like collection that contains all the elements in the document that use the specified class name
   3. the collection order is the same as the order in which the elements appear in the document
   4. Example:  
        
      const keywords = document.getElementsByClassName(“keyword”);
8. Specifying elements by selector
   1. The same selectors and combinators used in CSS are also available in JavaScript to reference page elements by using the document object’s querySelector() and querySelectorAll() methods  
        
      document.querySelector(selector)  
      document.querySelectorAll(selector)
   2. querySelectorAll() returns a collection of all the elements that match the selector, whereas querySelector() returns only the first element that matches the selector
   3. The following statement returns the collection of all <section> elements that are direct children of an <article> element:  
        
      const articles = document.querySelectorAll(“article > section”);
9. Collections order
   1. Consider the following HTML code:  
        
      <div id=”div1”>  
       This, of course, is div 1.  
      </div>  
      <div id=”div2”>  
       Yeah, well <em>this</em> is div 2!  
      </div>  
      <div id=”div3”>  
       Ignore those dudes. Welcome to div 3!  
      </div>
   2. Consider the following statement:  
        
      divs = document.getElementsByTagName(“div”);
   3. In the resulting collection ,the first item (divs[0]) will be the <div> element with the id div1
   4. The second item (divs[1]) will be the <div> element with the id div2
   5. The third item (divs[2]) will be the <div> element with the id div3
   6. You can refer to elements in the collection using either their index number in the array or by directly using their id values
      1. These statements are equivalent:  
           
         const firstDiv = divs[0];  
         const firstDiv = divs.div1;
   7. To learn how many items are in the collection, use the length property  
        
      const totalDivs = divs.length;
   8. To perform one or more operations on each item in the collection, you can use a for…of loop to run through the collection one at a time
      1. Syntax:  
           
         for (const item of collection) {  
          statements (the code you want to use to manipulate, view, etc the item  
         }
      2. Example:  
           
         divs = document.getElementsByTagName(“div”);  
         for (const d of divs) {  
          console.log(d.id);  
         }  
         // div1  
         // div2  
         // div3
      3. for…of loop is an ES6 edition. For older browsers, just use a regular for loop
10. Touring the DOM with code
    1. Here is the code we’re working with:   
         
       <html lang=”en”>  
        <head>  
        <title>So Many Kale Recipes</title>  
        </head>  
        <body>  
        <header id=”page-banner”>  
        <h1>Above and Beyond the Kale of Duty</h1>  
        </header>  
        <main id=”page-content”>  
        <p>  
        Do you love to cook with <a href=”kale.html”>kale</a>?  
        </p>  
        </main>  
        </body>  
       </html>

* 1. Working with the children, parent or siblings of some element in the page is known as traversing the DOM, because you’re using these techniques to move up, down and along the DOM hierarchy
  2. Getting the children of a parent element
     1. Every parent element offers several properties that enable you to work with all or just some of its child nodes
        1. All the child nodes
        2. The first child node
        3. The last child node
     2. Getting all the child nodes
        1. To return a collection of all the child elements of a parent, use the children property  
             
           parentElement.children
        2. The following statement stores all the child element nodes of the body element in a variable:  
             
           const bodyChildElements = document.body.children;
        3. Result is an HTMLCollection object, an array-like collection of element nodes
        4. If we console.log(bodyChildElements) we get the following:  
             
           HTMLCollection { 0: header, 1: main, length: 2 }
           1. 0 and 1 are the index numbers of each child
        5. Can use bodyChildElements[0] to refer to the first element in the collection, which is the <header> element
     3. Getting the first child node
        1. If using the parent element’s childNodes or children property to return the parent’s child nodes, you can refer to the first time in the collection by tacking [0] on the collection’s variable name  
             
           bodyChildren[0]  
           bodyChildElements[0]
        2. More direct route: parent.firstChild
        3. Example: working with the first child node of the <main> element:  
             
           const content = document.getElementById(“page-content”);  
           const firstContentChildNode = content.firstChild;  
           1. Resulting node is a text node (the white space between the <main> and <p> tags)
        4. To get the first child element node, use parent.firstElementChild  
             
           const content = document.getElementById(“page-content”);  
           const firstContentChildElement = content.firstElementChild;  
           1. This returns the <p> element
     4. Getting the last child node
        1. parent.lastChild
        2. to work with the last child node of the <p> element:  
             
           const para = document.querySelector(“main > p”);  
           const lastParaChildNode = para.lastChild;
           1. result is a text node representing the question mark and white space to the <p> tag
        3. To get the last child element node, use parent.lastElementChild
        4. To get the last child element node of the <p> element:  
             
           const para = document.querySelector(“main > p”);  
           const lastParaChildElement = para.lastElementChild;  
           1. Returns the <a> element
     5. Getting the parent of a child element
        1. child.parentNode
        2. to work with the parent element of the <h1> element:  
             
           const childElement = document.querySelector(“h1”);  
           const parentElement = childElement.parentNode;
     6. Getting the siblings of an element
        1. A parent’s child nodes appear in the DOM in the same order in which they appear in the HTML code, which means the siblings also appear in the order they appear in the HTML
        2. For a given child element, there are two sibling possibilities:
           1. Previous sibling – the sibling that appears in the DOM immediately before the child element you’re working with

If the child element is the first sibling, it will have no previous sibling

* + - * 1. Next sibling – the sibling that appears in the DOM immediately after the child element you’re working with

If the child element is the last sibling, it will have no next sibling

* + 1. Getting the previous sibling
       1. element.previousElementSibling
       2. Storing the previous sibling of the <main> element in a variable:  
            
          const currentElement = document.querySelector(“main”);

const prevSib = currElement.previousElementSibling;

* + 1. Getting the next sibling
       1. element.nextElementSibling
       2. Storing the next sibling of the <header> element in a variable:  
            
          const currElement = document.querySelector(“header”);  
          const nextSib = currElement.nextElementSibling;

1. Adding, Modifying, and Removing Elements
   1. When you add an element, you always specify the parent element to which it will be added, and then you decide whether you want the new element added to the end or to the beginning of the parent’s collection of children
   2. To add an element to the page, three steps:
      1. Create an object for the type of element you want to add
      2. Add the new object from step 1 as a child element of an existing element
      3. Insert some text and tags into the new object from step 1
   3. To create the element: document.createElement(elementName)
      1. Example:  
           
         const newArticle = createElement(“article”);
   4. To add the element to an existing parent element, you have four choices:
      1. Append the new element to the end of the parent’s collection of child elements
         1. parent.append(child)
            1. parent – the parent element to which the new element will be appended
            2. child – a reference to the child element you’re appending
            3. you can append multiple elements at the same time by separating each element with a comma
            4. the child parameter can also be a text string
      2. Prepend the new element to the beginning of the parent’s collection of child elements
         1. parent.prepend(child)
            1. Multiple elements, text string
      3. Insert the new element just after an existing child element of the parent
         1. child.after(sibling)
            1. child – a reference to the child element after which the new element will be inserted
            2. sibling – a reference to the new element you’re inserting
            3. multiple elements, text string
      4. Insert the new element just before an existing child element of the parent
         1. child.before(sibling)
            1. multiple elements, text string
      5. To create a new <article> element and append it to the <main> element:  
           
         const newArticle = document.createElement(“article”);  
         document.querySelector(“main”).append(newArticle);
      6. To create a new <nav> element and prepend it to the <main> element:  
           
         const newNav = document.createElement(“nav”);  
         document.querySelector(“main”).prepend(newNav);
   5. Adding text and tags to the new element
      1. With the element created and appended to a parent, the final step is to add some text and tags using the innerHTML property
         1. element.innerHTML = text
            1. element – a reference to the new element within which you want to add the text and tags
            2. text – a string containing the text and HTML tags you want to insert
      2. whatever value you assign to the innerHTML property completely overwrites an elements existing text and tags, so use caution
      3. To add a heading to the previously created newNav element:  
           
         newNav.innerHTML = “<h2>Navigation</h2>”;
   6. Inserting text or HTML into an element
      1. if you want to keep an elements existing tags and text and insert new tags and text, there are two methods:
         1. to insert just text into an element: element.insertAdjacentText(location, text)
            1. element – a reference to the element into which the new text will be inserted
            2. location – a string specifying where you want the text inserted
            3. text – a string containing the text you want to insert
         2. to insert tags and text into an element: element.insertAdjacentHTML(location, data)
            1. element – a reference to the element into which the new tags and text will be inserted
            2. location – a string specifying where you want the tags and text inserted
            3. data – a string containing the tags and text you want to insert
         3. You can use one of the following strings for the location argument
            1. beforebegin: inserts the data outside of and just before the element
            2. afterbegin: inserts the data inside the element, before the element’s first child
            3. beforeend: inserts the data inside the element, after the element’s last child
            4. afterend: inserts the data outside of and just after the element
      2. If we have the following element