Web Application Development

2019

TARTU ÜLIKOOL · UNIVERSITAS TARTUENSIS · 1632

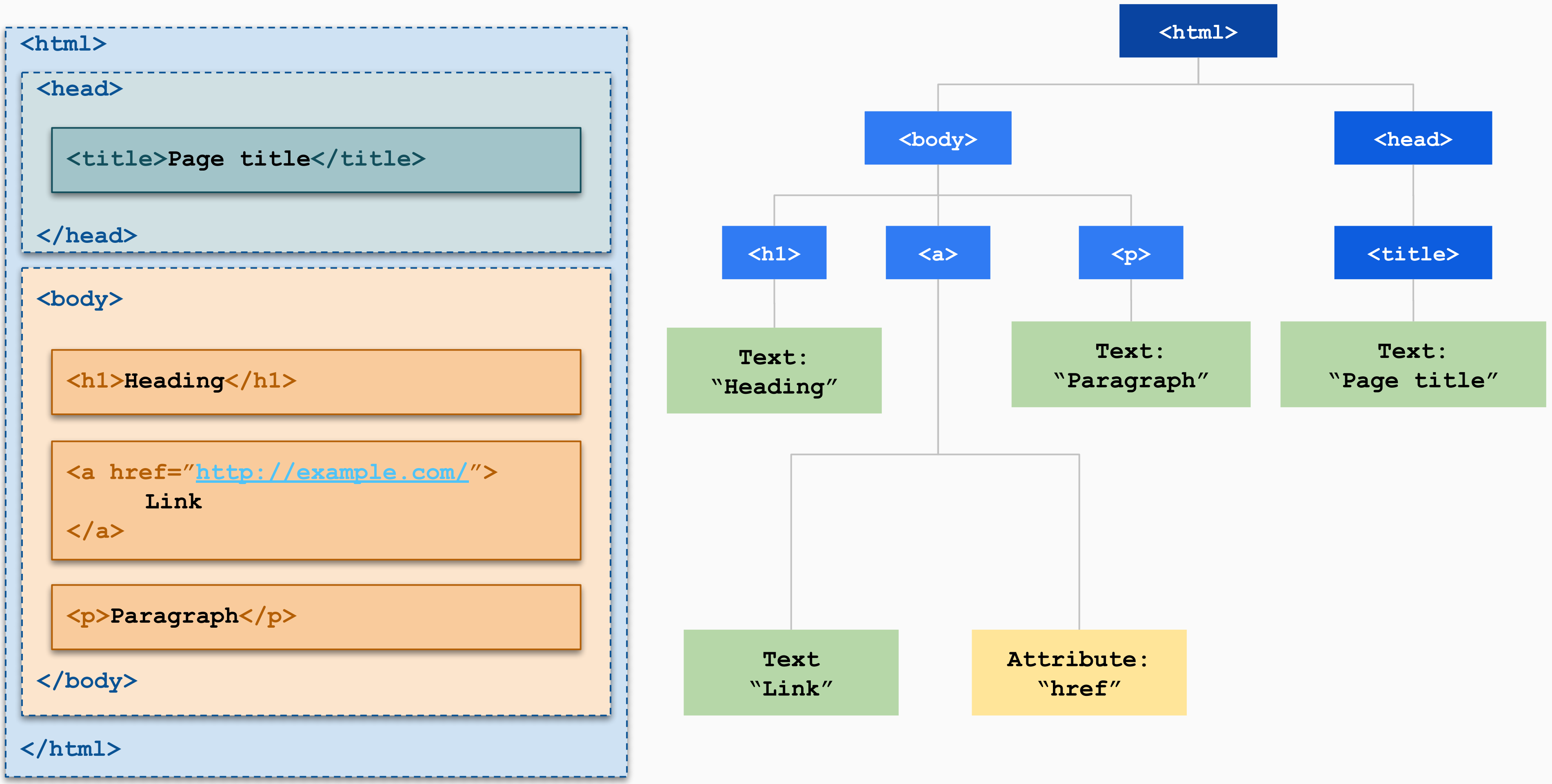
DOM

Document Object Model

DOM

"The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects.[1]"

https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction#What_is_the_DOM



```
<html>
<head>
<title>Page title</title>
</head>
<body>
<h1>Heading</h1>
<a href="http://example.com/">
     Link
</a>
<p>Paragraph</p>
</body>
</html>
```

Node Relationships

The nodes in the node tree have a hierarchical relationship to each other.

The terms parent, child, and sibling are used to describe the relationships.

- In a node tree, the **top** node is called the **root** (or root node)
- Every node has exactly **one** parent, except the root (which has no parent)
- A node can have **any** number of children
- Siblings (brothers or sisters) are nodes with the **same** parent

Navigating Between Nodes

You can use the following node properties to navigate between nodes with JavaScript:

- `parentNode`
- `childNodes[`*`nodenumber`*`]`
- `firstChild`
- `lastChild`
- `nextSibling`
- `previousSibling`

Javascript can:

- **Change** all the HTML elements in the page
- **Change** all the HTML attributes in the page
- **Change** all the CSS styles in the page
- **Remove** existing HTML elements and attributes
- **Add** new HTML elements and attributes
- **React** to all existing HTML events in the page
- **Create** new HTML events in the page

DOM Data Types

Data type (Interface)	Description
`Document`	When a member returns an object of type **`document`**, this object is the root **`document`** object itself.
`Node`	Every object located within a document is a node of some kind. In an HTML document, an object can be an element node but also a text node or attribute node.
`Element`	**`element`** objects implement the DOM **`Element`** interface and also the more basic **`Node`** interface. In an HTML document, elements are further enhanced by the HTML DOM API's **`HTMLElement`** interface as well as other interfaces describing capabilities of specific kinds of elements (for instance, **`HTMLTableElement`** for **`<table>`** elements).
`Attribute`	When an **`attribute`** is returned by a member (e.g., by the **`createAttribute()`** method), it is an object reference that exposes a special (albeit small) interface for attributes. Attributes are nodes in the DOM just like elements are, though you may rarely use them as such.
`NodeList`	A **`nodeList`** is an array of elements, like the kind that is returned by the method **`document.getElementsByTagName()`**. Items in a **`nodeList`** are accessed by index.
`NamedNodeMap`	A **`namedNodeMap`** is like an array, but the items are accessed by name or index.

Methods and Properties

```
let element = document.getElementById("demo");
element.innerHTML = "Hello World!";
```

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

In this case: `.innerHTML`

HTML DOM methods are **actions** you can perform (on HTML Elements).

In this case: `.getElementById()`

Finding Elements

There are several ways:

- Finding HTML elements by **id**
- Finding HTML elements by **tag name**
- Finding HTML elements by **class name**
- Finding HTML elements by **CSS selectors**

Finding HTML Element by **Id**

The easiest way to find an HTML element in the DOM, is by using the element id.

```html
<p id="intro">This is a paragraph.</p>
```

```javascript
var myElement = document.getElementById("intro");
```

Finding HTML Elements by **Tag Name**

It is possible to find an HTML element in the DOM, is by using the element tag name.

```html
<p id="intro">This is a paragraph.</p>
<p>This is another paragraph.</p>
```

```javascript
var myElement = document.getElementsByTagName("p");
```

Finding HTML Elements by **Class Name**

If you want to find all HTML elements with the same class name, use **`getElementsByClassName()`**

```html
<div class="text">This is a div.</div>
<p class="text">This is a paragraph.</p>
```

```javascript
var myElement = document.getElementsByClassName("text");
```

Finding HTML Elements by **CSS Selectors**

If you want to find all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the `querySelectorAll()` method.

```html
<div class="text">This is a div.</div>
<p class="text">This is a paragraph.</p>
```

```javascript
var myElement = document.querySelectorAll("p.text");
```

Changing CSS

The HTML DOM allows JavaScript to change the style of HTML elements.

```html
<div id="intro">This is a div.</div>
<p class="text">This is a paragraph.</p>
```

```javascript
document.getElementById("intro").style.color = "blue";
```

Creating New HTML Elements (Nodes)

To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

```
<div id="div1">
    <p id="p1">This is a paragraph.</p>
    <p id="p2">This is another paragraph.</p>
</div>

<script>
    var paragraph = document.createElement("p");
    var node = document.createTextNode("New Text.");
    paragraph.appendChild(node);

    var parent = document.getElementById("div1");
    parent.appendChild(para);
</script>
```

BOM

Browser Object Model

The Window Object

The window object is supported by all browsers. It represents the browser's window.

All global JavaScript objects, functions, and variables automatically become members of the window object.

- `window.innerHeight` - the inner height of the browser window (in pixels)
- `window.innerWidth` - the inner width of the browser window (in pixels)

- `window.open()` - open a new window
- `window.close()` - close the current window
- `window.moveTo()` - move the current window
- `window.resizeTo()` - resize the current window

Window Screen

The window.screen object contains information about the user's screen.

The `window.screen` object can be written without the window prefix.

- `screen.width`
- `screen.height`
- `screen.availWidth`
- `screen.availHeight`
- `screen.colorDepth`
- `screen.pixelDepth`

Window Location

The window.location object can be used to get the current page address (URL) and to redirect the browser to a new page.

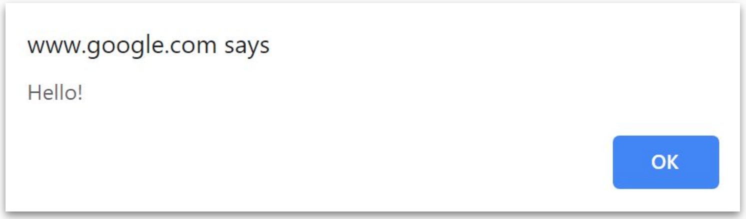
The `window.location` object can be written without the window prefix.

- `location.href` returns the href (URL) of the current page
- `location.hostname` returns the domain name of the web host
- `location.pathname` returns the path and filename of the current page
- `location.protocol` returns the web protocol used (http: or https:)
- `location.assign()` loads a new document

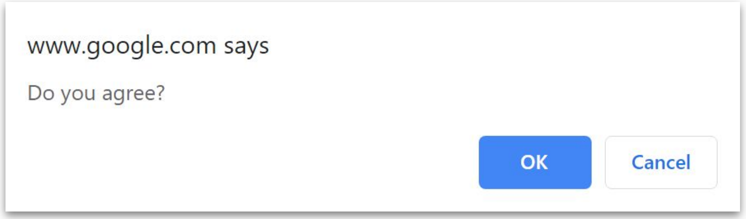
Popup Boxes

JavaScript has three kind of popup boxes:

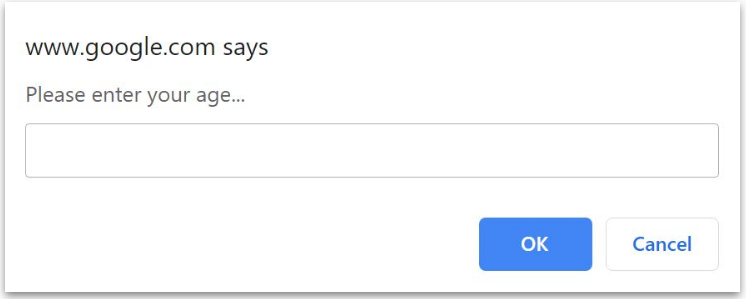
- Alert box
- Confirm box
- Prompt box


www.google.com says

Hello!


www.google.com says

Do you agree?


www.google.com says

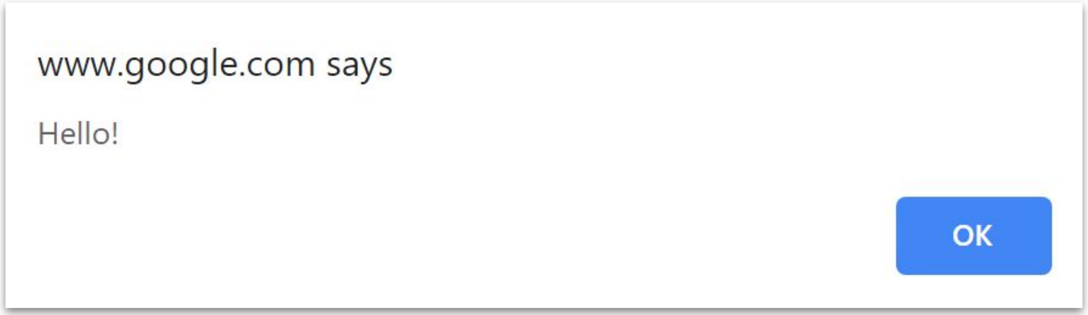
Please enter your age...

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

```
window.alert("Hello!");
```


www.google.com says

Hello!

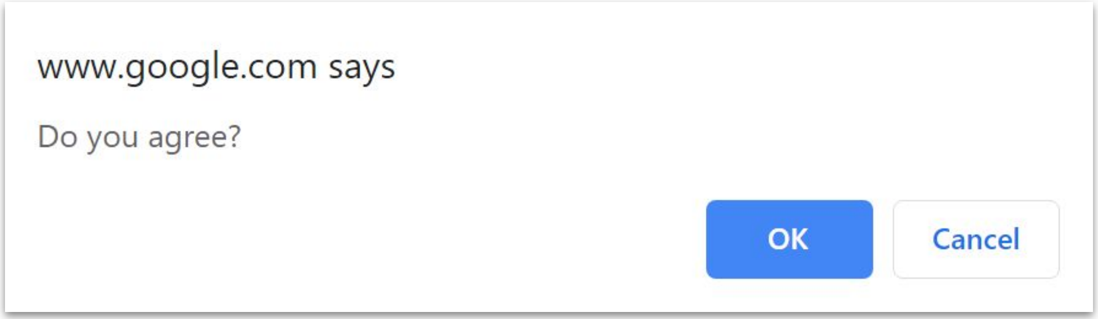
Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns **true**. If the user clicks "Cancel", the box returns **false**.

```
if (confirm("Press a button!")) {
  txt = "You pressed OK!";
}
else {
  txt = "You pressed Cancel!";
}
```


www.google.com says

Do you agree?

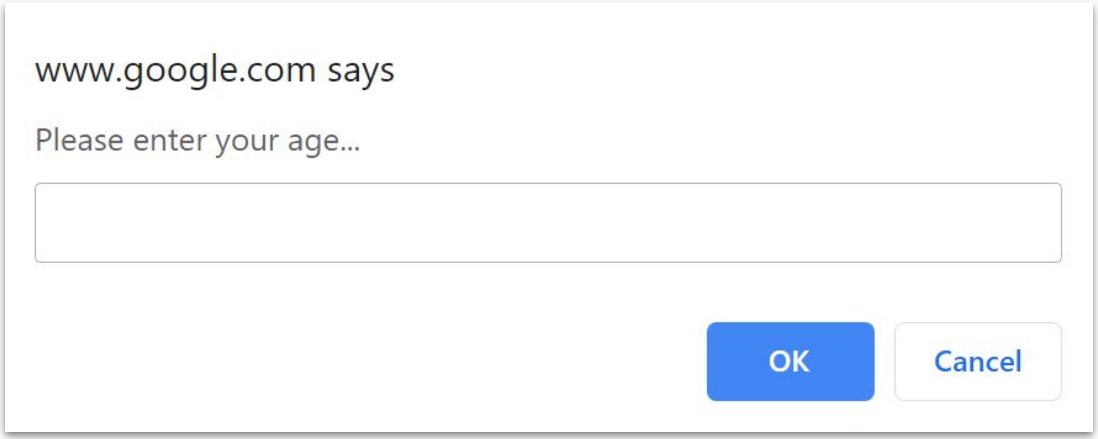
Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the **input value**. If the user clicks "Cancel" the box returns **null**.

```
var age = prompt("Please enter your age...");
if (age == null || age == "") {
  txt = "User cancelled the prompt.";
}
else {
  txt = age;
}
```


www.google.com says

Please enter your age...

Timing Events

The window object allows execution of code at specified time intervals.

These time intervals are called timing events.

- `setTimeout(function, milliseconds)`
  Executes a function, after waiting a specified number of milliseconds.

- `setInterval(function, milliseconds)`
  Same as setTimeout(), but repeats the execution of the function continuously.

The setTimeout() Method

The first parameter is a function to be executed.

The second parameter indicates the number of milliseconds before execution.

```
<button onclick="setTimeout(myFunction, 3000)">
    Try it
</button>

<script>
function myFunction() {
  alert('Hello');
}
</script>
```

Stop the Execution

The clearTimeout() method stops the execution of the function specified in setTimeout().

The clearTimeout() method uses the variable returned from setTimeout()

```
<button onclick="myVar = setTimeout(myFunction,3000)">
    Try it
</button>

<button onclick="clearTimeout(myVar)">
    Stop it
</button>
```

The setInterval() Method

The setInterval() method repeats a given function at every given time-interval.

The first parameter is the function to be executed.

The second parameter indicates the length of the time-interval between each execution.

```
<p id="demo"></p>
<script>
var myVar = setInterval(myTimer, 1000);
function myTimer() {
    var d = new Date();
    var demo = document.getElementById("demo");
    demo.innerHTML = d.toLocaleTimeString();
}
</script>
```

Stop the Execution

The clearInterval() method stops the executions of the function specified in the setInterval() method.

The clearInterval() method uses the variable returned from setInterval()

```
<p id="demo"></p>
<button onclick="clearInterval(myVar)">
  Stop time
</button>

<script>
var myVar = setInterval(myTimer, 1000);
function myTimer() {
    var d = new Date();
    var demo = document.getElementById("demo");
    demo.innerHTML = d.toLocaleTimeString();
}
</script>
```

Cookies

Cookies

Cookies are data, stored in small text files, on your computer.

When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Cookies were invented to solve the problem "how to remember information about the user":

When a user visits a web page, his/her name can be stored in a cookie. Next time the user visits the page, the cookie "remembers" his/her name.

Cookies are saved in name-value pairs:

```
username = John Doe
```

Cookies

JavaScript can create, read, and delete cookies with the document.cookie property.

```javascript
document.cookie = "username=John Doe";
document.cookie = "username=John Doe; expires=Thu, 18 Dec 2020 12:00:00 UTC; path=/;";

var x = document.cookie;
//username=John Doe

//Deleting cookie
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";
```

jQuery

What is jQuery?

jQuery is a lightweight, "write less, do more", JavaScript library.

The purpose of jQuery is to make it much easier to use JavaScript on your website.

jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

The jQuery library contains the following features:

- HTML/DOM manipulation
- CSS manipulation
- HTML event methods
- Effects and animations
- AJAX
- Utilities

Adding jQuery to Your Web Pages

Downloading jQuery:

There are two versions of jQuery available for downloading:

- **Production version** - this is for your live website because it has been minified and compressed
- **Development version** - this is for testing and development (uncompressed and readable code)

Both versions can be downloaded from jQuery.com.

```html
<head>
    <script src="jquery.js"></script>
</head>
```

jQuery CDN:

If you don't want to download and host jQuery yourself, you can include it from a CDN (Content Delivery Network).

To use jQuery from Google use one of the following:

```html
<head>
    <script
    src="https://ajax.googleapis.com/
    ajax/libs/jquery/3.4.1/
    jquery.min.js"></script>
</head>
```

jQuery syntax

Basic syntax is: $(*selector*).*action*()

- A $ sign to define/access jQuery
- A (*selector*) to "query (or find)" HTML elements
- A jQuery *action*() to be performed on the element(s)

Examples:

`$(this).hide()` - hides the current element.

`$("p").hide()` - hides all <p> elements.

`$(".test").hide()` - hides all elements with class="test".

`$("#test").hide()` - hides the element with id="test".

Document ready event

This is to prevent any jQuery code from running before the document is finished loading (is ready).

```
//Document ready event

$(document).ready(function(){


  // jQuery methods go here...


});
```

```
//Shorthand

$(function(){


  // jQuery methods go here...


});
```

jQuery Selectors

Query selectors are used to "find" (or select) HTML elements based on their name, id, classes, types, attributes, values of attributes and much more. It's based on the existing CSS Selectors, and in addition, it has some own custom selectors.

All selectors in jQuery start with the dollar sign and parentheses: $().

$("*")	Selects all elements
$(this)	Selects the current HTML element
$("p.intro")	Selects all <p> elements with class="intro"
$("p:first")	Selects the first <p> element
$("ul li:first")	Selects the first <li> element of the first <ul>
$("ul li:first-child")	Selects the first <li> element of every <ul>

Events

All the different visitors' actions that a web page can respond to are called events.

An event represents the precise moment when something happens.

```
$("p").click(function(){
  $(this).hide();
});
```

Get / Set

One very important part of jQuery is the possibility to manipulate the DOM.

jQuery comes with a bunch of DOM related methods that make it easy to access and manipulate elements and attributes.

Simple, but useful, jQuery methods for DOM manipulation are:

- `text()` - Sets or returns the text content of selected elements
- `html()` - Sets or returns the content of selected elements (including HTML markup)
- `val()` - Sets or returns the value of form fields
- `attr()` method is used to get / set attribute values.

Add New HTML Content

With jQuery, it is easy to add new elements/content.

- `append()` - Inserts content at the end of the selected elements
- `prepend()` - Inserts content at the beginning of the selected elements
- `after()` - Inserts content after the selected elements
- `before()` - Inserts content before the selected elements

- `$("p").append("Some appended text.");`
- `$("p").prepend("Some prepended text.");`
- `$("img").after("Some text after");`
- `$("img").before("Some text before");`

Remove Elements

With jQuery, it is easy to remove existing HTML elements.

- `remove()` - Removes the selected element (and its child elements)
- `empty()` - Removes the child elements from the selected element

- `$("#div1").remove();`
- `$("#div1").empty();`

Get and Set CSS Classes

With jQuery, it is easy to manipulate the style of elements.

- `addClass()` - Adds one or more classes to the selected elements
- `removeClass()` - Removes one or more classes from the selected elements
- `toggleClass()` - Toggles between adding/removing classes from the selected elements
- `css()` - Sets or returns the style attribute

```
$("button").click(function(){

  $("h1, h2, p").addClass("blue");

  $("div").addClass("important");

});
```

Traversing the DOM

jQuery provides a variety of methods that allow us to traverse the DOM.

- `parent()` - returns the direct parent element of the selected element
- `parents()` - returns all ancestor elements of the selected element, all the way up to the document root element (`<html>`)
- `parentsUntil()` - returns all ancestor elements between two given arguments

- `children()` - returns all direct children of the selected element. This method only traverses a single level down the DOM tree
- `find()` - returns descendant elements of the selected element, all the way down to the last descendant

Traversing the DOM

jQuery provides a variety of methods that allow us to traverse the DOM.

- `siblings()` - returns all sibling elements of the selected element
- `next()` - returns the next sibling element of the selected element
- `nextAll()` - returns all next sibling elements of the selected element
- `nextUntil()` - returns all next sibling elements between two given arguments
- `prev()` - returns the previous sibling element of the selected element
- `prevAll()` - returns all previous sibling elements of the selected element
- `prevUntil()` - returns all previous sibling elements between two given arguments

References

https://developer.mozilla.org/en-US/

https://www.w3schools.com/js

https://jquery.com/

Questions?

Next: AJAX / JSON