JavaScript 1

**Kalob & Andrei 3**

Ajax and jquery

**3.1 Ajax**

**Ajax (Asynchronous JavaScript and XML)** is a *group* of *interrelated* *web development techniques* used on the client-side to create asynchronous web applications.

* With *Ajax*, web applications can *send* data to and *retrieve* data from, a *server* *asynchronously* (in the background) *without* *interfering* with the *display* and *behavior* of the existing *page*
* Data can be retrieved using the **XMLHttpRequest** object. Despite the name, the use of *XML* is not required (*JSON* is often used instead), and the requests do *not need to* be *asynchronous*.
* Ajax is not a single technology, but a group of technologies HTML and CSS can be used in combination to mark up and style information. The ***DOM*** is accessed with ***JavaScript*** to ***dynamically*** display, and to allow the ***user*** to ***interact*** with the ***information*** presented ***JavaScript*** and the ***XMLHttpRequest*** object provide a ***method*** for ***exchanging*** ***data*** ***asynchronously*** between ***browser*** and ***server*** to avoid full page reloads.

**3.3 How To Execute Ajax**

*AJAX* is a *developer's dream*, because you can:

* ***Read data*** from a web ***server*** - after the ***page*** has ***loaded***
* ***Update*** a ***web*** ***page*** without reloading the ***page***
* ***Send*** ***data*** to a ***web server*** - in the ***background***
* AJAX Example Explained

|  |
| --- |
| <!DOCTYPE html> <html> <body>  <div id="demo">   <h2>Let AJAX change this text</h2>   <button type="button" onclick="loadDoc()">Change Content</button> </div>  </body> </html> |

* The HTML page contains a ***<div>*** section and a ***<button>***.
* The ***<div>*** section is used to display information from a server.
* The ***<button>*** calls a function **loadDoc**() (if it is clicked). The function **loadDoc**() requests data from a web server and displays it:

function **loadDoc**() {

    var xhttp = new **XMLHttpRequest**();

    xhttp.**onreadystatechange** = function() {

      if (this.readyState == 4 && this.status == 200) {

       document.**getElementById**("demo").innerHTML = this.responseText;

      }

    };

    xhttp.**open**("GET", "ajax\_info.txt", true);

    xhttp.**send**();

  }

* What is AJAX: AJAX = Asynchronous JavaScript And XML.
* AJAX is *not a programming language*.
* AJAX just uses a combination of:

1. A ***browser*** built-in ***XMLHttpRequest*** ***object*** (to request data from a web server)
2. ***JavaScript*** and ***HTML DOM*** (to display or use the data)

* AJAX is a misleading name: AJAX applications might use ***XML*** to transport data, but it is equally ***common*** to transport data as ***plain text*** or ***JSON*** text.

AJAX allows *web pages* to be *updated ASYNCHRONOUSLY* by exchanging data with a *web server* behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

**How AJAX Works**



* 1. An *event occurs* in a web page (the page is loaded, a button is clicked)
  2. An *XMLHttpRequest* *object* is *created* by JavaScript
  3. The *XMLHttpRequest* object sends a *request* to a web server
  4. The *server* *processes* the *request*
  5. The *server* *sends* a *response* back to the web page
  6. The *response* is read by *JavaScript*
  7. *Proper action* (like page *update*) is performed by *JavaScript*
* To actually execute *AJAX* takes some work. Typically, we put all of this inside of a *function*, so we don't have to write this out over and over again.

var ajax;

if(window.XMLHttpRequest) {

    // this is for IE 7+, Chrome, Safari and Firefox.

    ajax = new **XMLHttpRequest**();

} else {

    // this is for IE 6 and IE 5.

    ajax = new **ActiveXObject**("Microsoft.XMLHTTP"); I

}

ajax.**onreadystatechange**=function(){

    if (ajax.readyState == 4 && ajax.status == 200) {

    // execute this section of code when the requested page has fully loaded

    // use ajax.responseText as the object that holds the page information

    document.**getElementByld**("myDiv").innerHTML = ajax.responseText;

}

ajax.**open**("GET","mypage.html",true);

ajax.**send**();

* If we put this code into our website, it will automatically run this.
* Usually, you want the user to click a *button* or a *link* and then run the *AJAX*. For example: When you click a picture while on Facebook and that box appear on top of everything else with the picture, comments, likes and ads, that picture has been ajax'd onto that page. It was not pre-loaded, because the site holds tonnes of information it would take a very long time to load individual pages.
* So what we can do, is put most of this into a function and return the new XMLHttpRequest.

**3.4 AJAX requests**

* So what we can do: is put most of this into a function and return the new XMLHttpRequest.

// we can put most of this into a function and return the new XMLHttpRequest.

function **newAjax**(){

    var ajax;

    if(window.XMLHttpRequest) {

        // this is for IE 7+, Chrome, Safari and Firefox.

        ajax = new **XMLHttpRequest**();

    } else {

        // this is for IE 6 and IE 5.

        ajax = new **ActiveXObject**("Microsoft.XMLHTTP"); I

    }

    return ajax;

}

// We  use this with another function

function **LoadMe**() {

    var ajaxHandler = **newAjax**();

    ajaxHandler.**onreadystatechange** = function(){

        if (ajaxHandler.readyState==4 && ajaxHandler.status==200) {

            document.**getElementById**("loadMe").innerHTML = ajaxHandler.responseText;

        }

    }

    ajaxHandler.**open**("GET", "mypage.html", true);

    ajaxHandler.**send**()

}

<!DOCTYPE html>

<html lang="en">

    <head>

        <meta charset="utf-8">

        <title>Navigator</title>

        <link>

        <script src="JS\_file\_URL\_Link" type="text/javascript"></script>

        <script>

            // we can put most of this into a function and return the new XMLHttpRequest.

            function **newAjax**(){

                var ajax;

                if(window.XMLHttpRequest) {

                    // this is for IE 7+, Chrome, Safari and Firefox.

                    ajax = new **XMLHttpRequest**();

                } else {

                    // this is for IE 6 and IE 5.

                    ajax = new **ActiveXObject**("Microsoft.XMLHTTP"); I

                }

                return ajax;

            }

            // We  use this with another function

            function **LoadMe**() {

                var ajaxHandler = **newAjax**();

                ajaxHandler.**onreadystatechange** = function(){

                    if (ajaxHandler.readyState==4 && ajaxHandler.status==200) {

                        document.**getElementById**("loadMe").innerHTML = ajaxHandler.responseText;

                    }

                }

                ajaxHandler.**open**("GET", "mypage.html", true);

                ajaxHandler.**send**()

            }

        </script>

    </head>

    <body>

        <div id="loadMe">&nbsp;</div>

        <button onclick="**LoadMe**()">Ajax My page</button>

    </body>

</html>

* Send a Request To a Server: The ***XMLHttpRequest*** object is used to exchange data with a server. To send a request to a server, we use the ***open()*** and ***send()*** methods of the ***XMLHttpRequest*** object:

xhttp.open("GET", "ajax\_info.txt", true);

xhttp.send();

* **open**(method, url, async) Specifies the type of request
* method: the type of request: GET or POST
* url: the server (file) location
* async: true (asynchronous) or false (synchronous)
* **send()** Sends the request to the server (used for GET)
* **send(string)** Sends the request to the server (used for POST)
* GET or POST?
* GET is simpler and faster than POST, and can be used in most cases.
* However, always use POST requests when:
* A cached file is not an option (update a file or database on the server).
* Sending a large amount of data to the server (POST has no size limitations).
* Sending *user input* (which can contain *unknown characters*), *POST* is more *robust and secure* than GET.

**3.5 Ajax responses**

Previously, we learned how to create a request using ***open()*** and how to send that request using ***send()***.

* Well, we can send requests, but nothing will change unless we know how to harvest the information we receive after the request was sent.
* To receive the information, we use the ***responseText*** object. We simply attach this object to our ***ajaxHandler*** and we're able to access the information, whether it's XML, HTML, etc..
* Using the same function from earlier, the newA j ax () function...

function **newAjax**(){

    var ajax;

    if(window.XMLHttpRequest) {

        // this is for IE 7+, Chrome, Safari and Firefox.

        ajax = new **XMLHttpRequest**();

    } else {

        // this is for IE 6 and IE 5.

        ajax = new **ActiveXObject**("Microsoft.XMLHTTP"); I

    }

    return ajax;

}

* We need to use the ***ajaxHandler*** variable and identify' how far along the loading is.
* We don’t ever want to load partial amounts of information, that’s completely useless. Why? Because if your Ajax’d a page that starts with a <b> tag, but get’s cut off half way through, then all of the information will be in bold.
* So., we use the ***onreadystatechange*** object, assign it’s value a function, and check it this way.

var ajaxHandler = **newAjax**();

ajaxHandler.**onreadystatechange** = function() {

    // this is where we check our readyState and our status.

    if(ajaxHandler.readyState==4 && ajaxHandler.status==200) {

        // this means the file is done loading

**alert**(ajaxHandler.responseText)}

    }

ajaxHandler.**open**("POST", "mypage.html", true);

ajaxHandler.**send**();

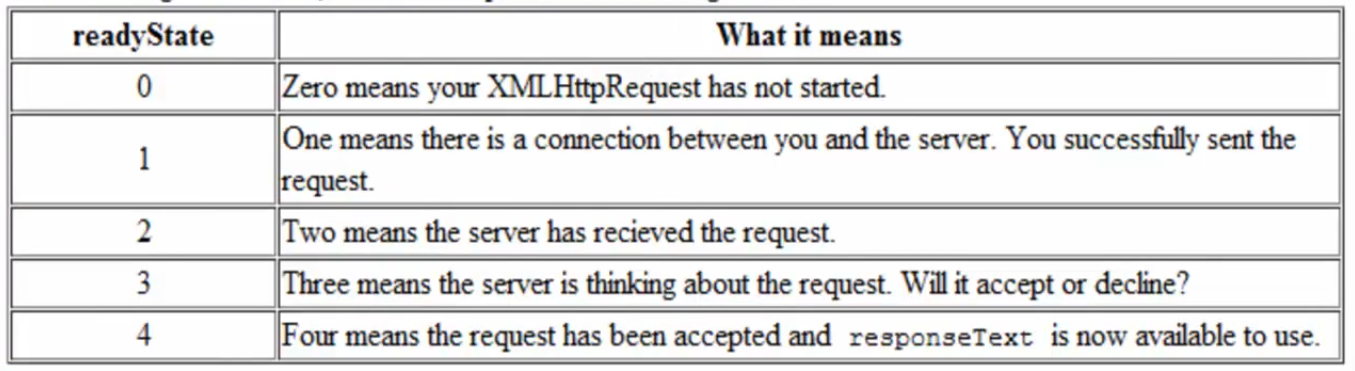
* We made a new request object (the ***XMLHttpRequest***) by using the **newAjax**() function.
* We told JS that when the ajax ***readyState*** changes, execute the code inside the first set of curly brackets.
* If the ***readyState=4*** AND the ***status=200***: alert us with the text from ''mypage.html".
* With ***.responseText*** we can use it anyway we like. We can put it into the ***innerHTML*** of an element, store it for later, or alert it. In every which way, the ***.responseText*** is the value from the file you loaded, and is yours to manipulate! )

Now let's see what ***readyState*** and ***status*** are. Let's look at this next.

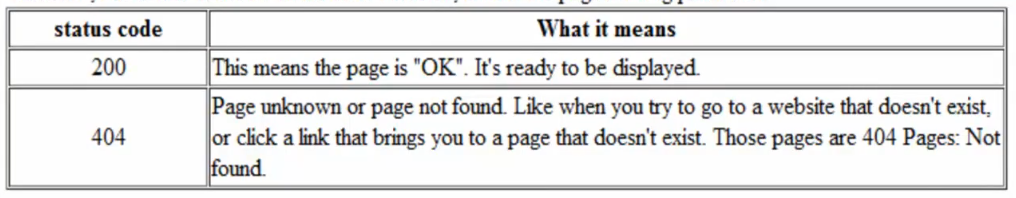
**3.6 readyState and status**

**onreadystatechange** = function() {}

* ***onreadystatechange*** is a function you define right there, or a function you defined earlier that will *execute every single time* the ***readyState*** *changes* in an Ajax request.
* ***readyState*** is status of the ***XMLHttpRequest*** you've been making with the **newAjax**() function. There are 5 states. 0 being the first one, because computers start counting from 0.



* In most cases, we ***don't need to check*** when the **readyState<4**. Usually, we *only care* if the request was *accepted* and when we can *use* the *information* that's been given to us through ***responseText*** .
* So if *readyState* is the *status*, what is *status*?.
* Status: **status()** are following codes. 200 = ok, 404 = not found



* Following example clear the idea:

// ready sates and status

ajaxHandler.**onreadystatechange** = function() {

    // this function will run 5 times because there are 5 readyState1s.

    if(ajaxHandler.readyState==4 && ajaxHandler.status==200) {

        // this means we can use responseText because readyState==4

        // this means the file was loaded successfully, because status=200

**alert**(ajaxHandler.responseText);

    document.**getElementByld**("myid").innerHTML = ajaxHandler.responseText;

    }

}

**3.7 jQuery**

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

* There are many *Javascript libraries* out there, such as *Moo Tools*, *Scriptaculous*, *Dojo*, *Prototype* and many more.
* Because of the simplicity and flexibility jQuery gives us, we can *animate* *elements*, *add* *remove classes*, *attributes*, *id’s*, etc, and even use **ajax**.
* *jQuery* takes a lot of common tasks that require *many lines of JavaScript* code to accomplish, and *wraps* them into *methods* that you can call with a *single line of code*.
* *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

Tip: In addition, jQuery has plugins for almost any task out there.

**3.8 Installing jQuery**

There are several ways to start using jQuery on your web site. You can:

1. File download: Download the jQuery library from jQuery.com. The jQuery library is a single JavaScript file, and you reference it with the HTML <script> tag (notice that the <script> tag should be inside the <head> section):

        <head>

            <script src="jquery-3.5.1.min.js"></script>

        </head>

1. CDN: Include jQuery from a CDN, like Google. If you don't want to download and host jQuery yourself, you can include it from a CDN (Content Delivery Network). Google is an example of someone who host jQuery:

    <head>

        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

    </head>

* Order matters: In case of using Bootstrap, we need to include Jqery at the bottom of the document, just before bootstrap.js inclusion.

<!DOCTYPE html>

<html lang="en">

    <head>

        <meta charset="utf-8">

        <title>Navigator</title>

        <link>

        <script src="JS\_file\_URL\_Link" type="text/javascript"></script>

        <script src="jquery-3.5.1.min.js"></script>

    </head>

    <body>

        <div id="loadMe">&nbsp;</div>

        <button onclick="**LoadMe**()">Ajax My page</button>

        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

    </body>

</html>

**3.9 jQuery Syntax**

Before we learn about ***selectors*** and ***events***, ***animation*** or even ***ajax***, we need to know how the syntax works. So let's create a demo script that tells us that jQuery has been installed on our web page.

        <script>

**$**(window).**ready**(function(){

**alert**('Loaded');

            });

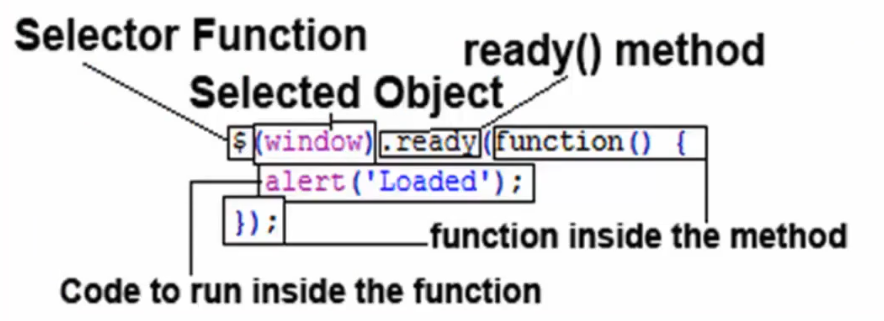
        </script>

**$**(window).**ready**(function() {

                // our code in here

            });

* This is a very common syntax for jQuery. A selector using the **$** *function*, an *event* such as **ready()** , our *code* inside the **function()** , and the *closing syntax* together (back to back) which looks like **});** .
* **The Document Ready Event:**



* **$** is the function to *select something*, it's similar to:

            function **$**(id){

                return document.**getElementById**(id);

            }

* The ***window*** is the *selected item*. Meaning jQuery is selecting the entire window.
* The ***ready()*** method *executes the specified code* when the window is ready to start using jQuery'.
* Inside the ***ready()*** method we have ***function(){*** which allows us to put a *large block of code inside* the ***ready()*** method.
* The ***alert(’ Loaded ’);*** is the code we can put inside the ***ready()*** method and it will run as soon as jQuery is done loading.
* The ***});*** , on the last line, is the closing curly' bracket from our ***function(){*** and the closing bracket from the ***ready(*** method ended with a semi-colon.
* With jQuery you select (query) HTML elements and perform "actions" on them. The jQuery syntax is tailor-made for selecting HTML elements and performing some action on the element(s).
* 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". \*/

* Are you familiar with CSS selectors?: jQuery uses CSS syntax to select elements.

**3.10 jQuery selectors**

The syntax is similar to css selectors.

* The element Selector:

**$**("p").**hide**()       /\* - hides all <p> elements. \*/

* The #id Selector:

**$**("#test").**hide**()   /\* - hides the element with id="test". \*/

* The .class Selector:

**$**(".test").**hide**()   /\* - hides all elements with class="test". \*/

* The attribute Selector:

**$**("[href]")                 /\* Selects all elements with an href attribute   \*/

**$**("a[target='\_blank']")     /\* Selects all <a> elements with a target attribute value equal to "\_blank"  \*/

**$**("a[target!='\_blank']")    /\* Selects all <a> elements with a target attribute value NOT equal to "\_blank" \*/

**$("[name = 'test\_Name']")**

**Other jqery selectors:**

// JQERY selectors

**$**("\*")          /\* 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>***  \*/

**$**("[href]")             /\* Selects ***all elements*** with an ***href attribute***   \*/

**$**("a[target='\_blank']") /\* Selects ***all <a>*** elements with a ***target attribute*** value equal to ***"\_blank"***  \*/

**$**("a[target!='\_blank']")    /\* Selects all ***<a> elements*** with a ***target attribute*** value ***NOT equal to*** "\_blank"  \*/

**$**(":button")    /\* Selects all ***<button>*** elements and ***<input>*** elements of ***type="button"***   \*/

**$**("tr:even")    /\* Selects all ***even <tr>*** elements    \*/

**$**("tr:odd")     /\* Selects all ***odd <tr>*** elements \*/

* If you find these selectors do not work when you try them, you might need to put it inside the ***ready()*** method.
* The reason may be, the browser is *trying to load your selectors* before jQuery was *done loading* and is resulting in a problem.
* So to make sure our jQuery only runs AFTER jQuery' is finished loading fully, we put our code inside the ***ready()*** method, like so:

**$**(window).**ready**(function(){

                /\* We told the browser to not load this until jQuery is finished loading. \*/

                var getBoldClass = **$**(".bold");

            });

**3.11 jQuery Events (Listeners)**

When we are writing code *inside* an *event* in jQuery, we can use **$(this)** to reference the selector:

**$("div");** **$(this)**

inside an event

* **focus()** and **blur()**:
* The function is executed when the form field gets focus.
* The function is executed when the form field loses focus.

                /\*  The function is executed when the form field gets focus:    \*/

              $("input").focus(function(){

                $(this).css("background-color", "yellow");

              });

                /\*  The function is executed when the form field loses focus:  \*/

              $("input").blur(function(){

                $(this).css("background-color", "green");

              });

* **hover():** The hover() method takes two functions and is a combination of the ***mouseenter()*** and ***mouseleave()*** methods.

            //  ---------  hover()  -----------

                $("#p1").hover(

                    function(){

                        alert("You entered p1!");

                        },

                    function(){

                        alert("Bye! You now leave p1!");

                    });

                $("#p2").hover(

                    function(){

                        $(this).css("border", "1px solid green");

                        },

                    function(){

                        $(this).css("border", "0");

                    });

* The first function is *executed* when the *mouse enters* the HTML element, and the *second* function is executed when the *mouse leaves* the HTML element:

(Part l) 6:22

Events (Listeners) (Part 2)

When A Key Is Pressed

Basic jQuery effects

Executing code when a function is complete callbacks

Selecting and changing element inner HTML

Changing CSS

Animate your CSS and HTML

Managing HTML element attributes

Simple Ajax requests using jQuery

Introduction To Javascript

What Is Javascript

Your First Javascript

Variables

Control Flow

Javascript On Our Webpage

Functions

Data Structures Arrays

Data Structures Objects

Exercise Build Facebook

Javascript Terminology

Loops

Exercise Build Facebook

Javascript Keywords

Document Object Model

DOM Selectors

DOM Events

Exercise Background Generator

jQuery

DEVELOPER FUNDAMENTALS V

Scope

Advanced Control Flow

ES5 and ES6

Advanced Functions

Advanced Arrays

Advanced 0bjects

New ES7

New ES8

Debugging

New How Javascript Works

Modules

1. JavaScript Essentials
2. What is JavaScript
3. JavaScript Internal vs. External
4. JavaScript comments
5. document.write()
6. Display info from the browser ***alert*** & ***confirm***
7. Prompting the user for information
8. Programming fundamentals Variables
9. Add two sentences together concatenation
10. Basic math in JavaScript
11. Redirecting users and opening new windows
12. Creating empty hyperlinks
13. String Manipulation
14. Comparing variables and values
15. Programming fundamentals If...Else Statements
16. Programming fundamentals Else...If Statements
17. Programming fundamentals Switch Statements
18. Programming fundamentals Functions
19. JavaScript Events
20. Selecting HTML elements using getElementById()
21. Escaping content
22. Programming fundamentals Arrays
23. Programming fundamentals For Loops
24. Programming fundamentals While Loops
25. Programming fundamentals Breaking Out Of Loops
26. Programming fundamentals Skipping A Loop Cycle
27. Getting parts of a value split & subst
28. Programming fundamentals Try..Catch And Throw
29. Getting the users date and time
30. Some more complex math (no homework)
31. Programming fundamentals Regular Expressions
32. Get the users browser (navigator)
33. Add timing setlnterval & setTimeout
34. Programming fundamentals Objects
35. What Is Ajax
36. HowTo Execute Ajax
37. Making an Ajax requests
38. Understanding Ajax responses
39. When Ajax is ready readyState and status
40. Putting It Together

1. What isjQuery
2. Installing jQuery
3. jQuery Syntax
4. Selecting is super easy in jQuery
5. Events (Listeners) (Part l)
6. Events (Listeners) (Part 2)
7. When A Key Is Pressed
8. Basic jQuery effects
9. Executing code when a function is complete callbacks
10. Selecting and changing element inner HTML
11. Changing CSS
12. Animate your CSS and HTML
13. Managing HTML element attributes
14. Simple Ajax requests using jQuery
15. Introduction To Javascript
16. What Is Javascript
17. Your First Javascript
18. Variables
19. Control Flow
20. Javascript On Our Webpage
21. Functions
22. Data Structures Arrays
23. Data Structures Objects
24. Exercise Build Facebook
25. Javascript Terminology
26. Loops
27. Exercise Build Facebook
28. Javascript Keywords
29. Document Object Model
30. DOM Selectors
31. DOM Events
32. Exercise Background Generator
33. jQuery
34. DEVELOPER FUNDAMENTALS V
35. Scope
36. Advanced Control Flow
37. ES5 and ES6
38. Advanced Functions
39. Advanced Arrays
40. Advanced 0bjects
41. New ES7
42. New ES8
43. Debugging
44. New How Javascript Works
45. Modules
46. JavaScript Essentials
47. What is JavaScript
48. JavaScript Internal vs. External
49. JavaScript comments
50. document.write()
51. Display info from the browser ***alert*** & ***confirm***
52. Prompting the user for information
53. Programming fundamentals Variables
54. Add two sentences together concatenation
55. Basic math in JavaScript
56. Redirecting users and opening new windows
57. Creating empty hyperlinks
58. String Manipulation
59. Comparing variables and values
60. Programming fundamentals If...Else Statements
61. Programming fundamentals Else...If Statements
62. Programming fundamentals Switch Statements
63. Programming fundamentals Functions
64. JavaScript Events
65. Selecting HTML elements using getElementById()
66. Escaping content
67. Programming fundamentals Arrays
68. Programming fundamentals For Loops
69. Programming fundamentals While Loops
70. Programming fundamentals Breaking Out Of Loops
71. Programming fundamentals Skipping A Loop Cycle
72. Getting parts of a value split & subst
73. Programming fundamentals Try..Catch And Throw
74. Getting the users date and time
75. Some more complex math (no homework)
76. Programming fundamentals Regular Expressions
77. Get the users browser (navigator)
78. Add timing setlnterval & setTimeout
79. Programming fundamentals Objects
80. What Is Ajax
81. HowTo Execute Ajax
82. Making an Ajax requests
83. Understanding Ajax responses
84. When Ajax is ready readyState and status
85. Putting It Together
86. What isjQuery
87. Installing jQuery
88. jQuery Syntax
89. Selecting is super easy in jQuery
90. Events (Listeners) (Part l)
91. Events (Listeners) (Part 2)
92. When A Key Is Pressed
93. Basic jQuery effects
94. Executing code when a function is complete callbacks
95. Selecting and changing element inner HTML
96. Changing CSS
97. Animate your CSS and HTML
98. Managing HTML element attributes
99. Simple Ajax requests using jQuery
100. Introduction To Javascript
101. What Is Javascript
102. Your First Javascript
103. Variables
104. Control Flow
105. Javascript On Our Webpage
106. Functions
107. Data Structures Arrays
108. Data Structures Objects
109. Exercise Build Facebook
110. Javascript Terminology
111. Loops
112. Exercise Build Facebook
113. Javascript Keywords
114. Document Object Model
115. DOM Selectors
116. DOM Events
117. Exercise Background Generator
118. jQuery
119. DEVELOPER FUNDAMENTALS V
120. Scope
121. Advanced Control Flow
122. ES5 and ES6
123. Advanced Functions
124. Advanced Arrays
125. Advanced 0bjects
126. New ES7
127. New ES8
128. Debugging
129. New How Javascript Works
130. Modules