JQUERY

INTRODUCTION

- jQuery is an Open-Source JavaScript framework that simplifies cross-browser client side scripting.
  - Animations
  - DOM manipulation
  - AJAX
  - Extensibility through plugins
- jQuery was created by John Resig and released 01/06
- Most current release is 1.4.2 (2/19/10)

BENEFITS OF JQUERY

- Improves developer efficiency
- Excellent documentation // pats self on back
- Unobtrusive from the ground up
- Reduces browser inconsistencies
- At its core, a simple concept

JQUERY

jQuery is a JavaScript Library.

jQuery greatly simplifies JavaScript programming.

jQuery is easy to learn.

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 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.

ADDING JQUERY TO YOUR WEB PAGES

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

Download the jQuery library from jQuery.com

Include jQuery from a CDN, like Google

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.

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):

JQUERY CDN

You can include it from a CDN (Content Delivery Network).

Both Google and Microsoft host jQuery.

To use jQuery from Google or Microsoft, use one of the following:

Google CDN:

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquer
y/3.1.1/jquery.min.js"></script>
</head>
```

Microsoft CDN:

```
<head>
<script src="https://ajax.aspnetcdn.com/ajax/jQuery/jqu
ery-3.1.1.min.js"></script>
</head>
```

JQUERY SYNTAX

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)

THE DOCUMENT READY EVENT

```
$(document).ready(function(){
    // jQuery methods go here...
});
```

FIND SOMETHING

"Select" elements in the document

$

$()

$('div')

$('#id')

DO SOMETHING

1. Let elements "listen" for something to happen …
   - the document is ready
   - user does something
   - another "listener" acts
   - a certain amount of time elapses

DO SOMETHING

2.... and then do something

a. Manipulate elements
b. Animate elements
c. Communicate with the server

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

It is good practice to wait for the document to be fully loaded and ready before working with it. This also allows you to have your JavaScript code before the body of your document, in the head section.

Here are some examples of actions that can fail if methods are run before the document is fully loaded:

Trying to hide an element that is not created yet

The jQuery team has also created an even shorter method for the document ready event:

```
$(function(){
    // jQuery methods go here...
});
```

JQUERY SELECTORS

jQuery selectors allow you to select and manipulate HTML element(s).

jQuery 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: $().

THE ELEMENT SELECTOR

The jQuery element selector selects elements based on the element name.

You can select all <p> elements on a page like this:

```
$("p")
```

Example

When a user clicks on a button, all <p> elements will be hidden:

Example

```
$(document).ready(function(){
    $("button").click(function(){
        $("p").hide();
    });
});
```

THE #ID SELECTOR

The jQuery #id selector uses the id attribute of an HTML tag to find the specific element.

An id should be unique within a page, so you should use the #id selector when you want to find a single, unique element.

To find an element with a specific id, write a hash character, followed by the id of the HTML element:

```
$("#test")
```

Example

When a user clicks on a button, the element with id="test" will be hidden:

Example

```
$(document).ready(function(){
    $("button").click(function(){
        $("#test").hide();
    });
});
```

THE .CLASS SELECTOR

The jQuery class selector finds elements with a specific class.

To find elements with a specific class, write a period character, followed by the name of the class:

```
$(".test")
```

Example

When a user clicks on a button, the elements with class="test" will be hidden:

Example

```
$(document).ready(function(){
    $("button").click(function(){
        $(".test").hide();
    });
});
```

CSS SELECTORS

element {}

#id {}

.class {}

selector1, **selector2** {}

ancestor **descendant** {}

parent > **child** {}

:nth-child() {}

CSS SELECTORS

(jQuery Equivalents)

$('element')

$('#id')

$('.class')

$('**selector1**, **selector2**')

$('ancestor **descendant**')

$('parent > **child**')

$(':nth-child(n)')

CSS SELECTORS

(jQuery Equivalents)

$('element')

$('#id')

$('.class')

$('**selector1**, **selector2**')

$('ancestor **descendant**')

$('parent > **child**')

$(':nth-child(n)')

- *and others …*
- $('prev + **selector**')
- $('prevAll ~ **selector**')$(':nth-child(an+b')$(':not(selector)')
- $(':checked')
- $(':disabled')

CSS ATTRIBUTE SELECTORS

$('input[name=firstname\\[\\]]')	
$('[**title**]')	has the attribute
$('[attr="val"]')	attr equals val
$('[attr**!**="val"]')	attr does not equal val
$('[attr**~**="val"]')	attr has val as one of space-sep. vals
$('[attr**^**="val"]')	attr begins with val
$('[attr**$**="val"]')	attr ends with val
$('[attr*****="val"]')	attr has val anywhere within

CUSTOM FORM SELECTORS

$('div.myclass :checkbox')	
$(':input') <select>, <button>	<input>, <textarea>,
$(':text')	<input type="text">
$(':radio')	<input type="radio">
$(':button') <button>	<input type="button">,
$(':selected')	<option selected="selected">
etc.	

CUSTOM MISC. SELECTORS

$(':animated')

$(':**has**(descendant)')

$(':eq(n)')

$(':lt(n)')

$(':gt(n)')

$(':even')

$(':odd')

- $(':visible')
  $(':hidden')$(':header')$
  (':contains(string)')

FUNCTIONS IN A SEPARATE FILE

If your website contains a lot of pages, and you want your jQuery functions to be easy to maintain, you can put your jQuery functions in a separate .js file.

When we demonstrate jQuery in this tutorial, the functions are added directly into the <head> section.

JQUERY EVENT

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

An event represents the precise moment when something happens.

Examples:

moving a mouse over an element

selecting a radio button

clicking on an element

The term **"fires/fired"** is often used with events. Example: "The keypress event is fired, the moment you press a key".

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

JQUERY SYNTAX FOR EVENT METHODS

In jQuery, most DOM events have an equivalent jQuery method.

To assign a click event to all paragraphs on a page, you can do this:

```
$("p").click();
```

The next step is to define what should happen when the event fires. You must pass a function to the event:

```
$("p").click(function(){
  // action goes here!!
});
```

CLICK()

The click() method attaches an event handler function to an HTML element.

The function is executed when the user clicks on the HTML element.

The following example says: When a click event fires on a <p> element; hide the current <p> element:

Example

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

DBLCLICK()

The dblclick() method attaches an event handler function to an HTML element.

The function is executed when the user double-clicks on the HTML element:

Example

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

MOUSEENTER()

The mouseenter() method attaches an event handler function to an HTML element.

The function is executed when the mouse pointer enters the HTML element:

Example

```
$("#p1").mouseenter(function(){
    alert("You entered p1!");
});
```

Try it Yourself »

MOUSELEAVE()

The mouseleave() method attaches an event handler function to an HTML element.

The function is executed when the mouse pointer leaves the HTML element:

Example

```
$("#p1").mouseleave(function(){
    alert("Bye! You now leave p1!");
});
```

MOUSEDOWN()

The mousedown() method attaches an event handler function to an HTML element.

The function is executed, when the left, middle or right mouse button is pressed down, while the mouse is over the HTML element:

Example

```
$("#p1").mousedown(function(){
    alert("Mouse down over p1!");
});
```

MOUSEUP()

The mouseup() method attaches an event handler function to an HTML element.

The function is executed, when the left, middle or right mouse button is released, while the mouse is over the HTML element:

Example

```
$("#p1").mouseup(function(){
    alert("Mouse up over p1!");
});
```

HOVER()

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

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:

Example

```
$("#p1").hover(function(){
    alert("You entered p1!");
},
function(){
    alert("Bye! You now leave p1!");
});
```

FOCUS()

The focus() method attaches an event handler function to an HTML form field.

The function is executed when the form field gets focus:

Example

```
$("input").focus(function(){
    $(this).css("background-color", "#cccccc");
});
```

BLUR()

The blur() method attaches an event handler function to an HTML form field.

The function is executed when the form field loses focus:

Example

```
$("input").blur(function(){
    $(this).css("background-color", "#ffffff");
});
```

THE ON() METHOD

The on() method attaches one or more event handlers for the selected elements.

Attach a click event to a <p> element:

Example

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

JQUERY EFFECTS - HIDE AND SHOW

jQuery hide() and show()

Example

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

$("#show").click(function(){
    $("p").show();
});
```

SYNTAX

$(*selector*).hide(*speed,callback*);

$(*selector*).show(*speed,callback*);

The optional speed parameter specifies the speed of the hiding/showing, and can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the hide() or show() method completes

JQUERY TOGGLE()

With jQuery, you can toggle between the hide() and show() methods with the toggle() method.

Example

```
$("button").click(function(){
    $("p").toggle();
});
```

SYNTAX

$(*selector*).toggle(*speed,callback*);

The optional speed parameter can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after toggle() completes.

JQUERY EFFECTS - FADING

With jQuery you can fade elements in and out of visibility.

jQuery has the following fade methods:

fadeIn()

fadeOut()

fadeToggle()

fadeTo()

JQUERY FADEIN() METHOD

The jQuery fadeIn() method is used to fade in a hidden element.

Syntax:

$(*selector*).fadeIn(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

EXAMPLE

```
$("button").click(function(){
    $("#div1").fadeIn();
    $("#div2").fadeIn("slow");
    $("#div3").fadeIn(3000);
});
```

JQUERY FADEOUT() METHOD

The jQuery fadeOut() method is used to fade out a visible element.

Syntax:

$(*selector*).fadeOut(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

EXAMPLE

```
$("button").click(function(){
    $("#div1").fadeOut();
    $("#div2").fadeOut("slow");
    $("#div3").fadeOut(3000);
});
```

JQUERY FADETOGGLE() METHOD

The jQuery fadeToggle() method toggles between the fadeIn() and fadeOut() methods.

If the elements are faded out, fadeToggle() will fade them in.

If the elements are faded in, fadeToggle() will fade them out.

Syntax:

$(*selector*).fadeToggle(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

EXAMPLE

```
$("button").click(function(){
    $("#div1").fadeToggle();
    $("#div2").fadeToggle("slow");
    $("#div3").fadeToggle(3000);
});
```

JQUERY FADETO() METHOD

The jQuery fadeTo() method allows fading to a given opacity (value between 0 and 1).

Syntax:

$(*selector*).fadeTo(*speed,opacity,callback*);

The required speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The required opacity parameter in the fadeTo() method specifies fading to a given opacity (value between 0 and 1).

The optional callback parameter is a function to be executed after the function completes.

EXAMPLE

```
$("button").click(function(){
    $("#div1").fadeTo("slow", 0.15);
    $("#div2").fadeTo("slow", 0.4);
    $("#div3").fadeTo("slow", 0.7);
});
```

JQUERY EFFECTS - SLIDING

With jQuery you can create a sliding effect on elements.

jQuery has the following slide methods:

slideDown()

slideUp()

slideToggle()

JQUERY SLIDEDOWN() METHOD

The jQuery slideDown() method is used to slide down an element.

Syntax:

$(*selector*).slideDown(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

EXAMPLE

```
$("#flip").click(function(){
    $("#panel").slideDown();
});
```

JQUERY SLIDEUP() METHOD

The jQuery slideUp() method is used to slide up an element.

Syntax:

$(*selector*).slideUp(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

EXAMPLE

```
$("#flip").click(function(){
    $("#panel").slideUp();
});
```

JQUERY SLIDETOGGLE() METHOD

The jQuery slideToggle() method toggles between the slideDown() and slideUp() methods.

If the elements have been slid down, slideToggle() will slide them up.

If the elements have been slid up, slideToggle() will slide them down.

$(*selector*).slideToggle(*speed,callback*);

The optional speed parameter can take the following values: "slow", "fast", milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

EXAMPLE

```
$("#flip").click(function(){
    $("#panel").slideToggle();
});
```

JQUERY EFFECTS - ANIMATION

jQuery Animations – The animate() Method

The jQuery animate() method is used to create custom animations.

Syntax:

$(*selector*).animate({*params*},*speed*,*callback*);

The required params parameter defines the CSS properties to be animated.

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the animation completes.

EXAMPLE

```
$("button").click(function(){
    $("div").animate({left: '250px'});
});
```

By default, all HTML elements have a static position, and cannot be moved.
To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!

JQUERY ANIMATE() - MANIPULATE MULTIPLE PROPERTIES

Example

```
$("button").click(function(){
    $("div").animate({
        left: '250px',
        opacity: '0.5',
        height: '150px',
        width: '150px'
    });
});
```

JQUERY ANIMATE() - USING RELATIVE VALUES

It is also possible to define relative values (the value is then relative to the element's current value). This is done by putting += or -= in front of the value:

Example

```
$("button").click(function(){
    $("div").animate({
        left: '250px',
        height: '+=150px',
        width: '+=150px'
    });
});
```

JQUERY ANIMATE() - USING PRE-DEFINED VALUES

You can even specify a property's animation value as "show", "hide", or "toggle":

Example

```
$("button").click(function(){
    $("div").animate({
        height: 'toggle'
    });
});
```

JQUERY ANIMATE() - USES QUEUE FUNCTIONALITY

By default, jQuery comes with queue functionality for animations.

This means that if you write multiple animate() calls after each other, jQuery creates an "internal" queue with these method calls. Then it runs the animate calls ONE by ONE.

EXAMPLE 1

```
$("button").click(function(){
    var div = $("div");
    div.animate({height: '300px', opacity: '0.4'}, "slow");
    div.animate({width: '300px', opacity: '0.8'}, "slow");
    div.animate({height: '100px', opacity: '0.4'}, "slow");
    div.animate({width: '100px', opacity: '0.8'}, "slow");
});
```

JQUERY STOP ANIMATIONS

The jQuery stop() method is used to stop an animation or effect before it is finished.

The stop() method works for all jQuery effect functions, including sliding, fading and custom animations.

Syntax:

$(*selector*).stop(*stopAll,goToEnd*);

The optional stopAll parameter specifies whether also the animation queue should be cleared or not. Default is false, which means that only the active animation will be stopped, allowing any queued animations to be performed afterwards.

The optional goToEnd parameter specifies whether or not to complete the current animation immediately. Default is false.

So, by default, the stop() method kills the current animation being performed on the selected element.

EXAMPLE

```
$("#stop").click(function(){
    $("#panel").stop();
});
```

A callback function is executed after the current effect is 100% finished.

JavaScript statements are executed line by line. However, with effects, the next line of code can be run even though the effect is not finished. This can create errors.

To prevent this, you can create a callback function.

A callback function is executed after the current effect is finished.

Typical syntax: **$(*selector*).hide(*speed,callback*);**

EXAMPLE WITH CALLBACK

```
$("button").click(function(){
    $("p").hide("slow", function(){
        alert("The paragraph is now hidden");
    });
});
```

EXAMPLE WITHOUT CALLBACK

```
$("button").click(function(){
    $("p").hide(1000);
    alert("The paragraph is now hidden");
});
```

JQUERY - CHAINING

With jQuery, you can chain together actions/methods.

Chaining allows us to run multiple jQuery methods (on the same element) within a single statement.

JQUERY METHOD CHAINING

Until now we have been writing jQuery statements one at a time (one after the other).

However, there is a technique called chaining, that allows us to run multiple jQuery commands, one after the other, on the same element(s).

To chain an action, you simply append the action to the previous action.

EXAMPLE

The following example chains together the css(), slideUp(), and slideDown() methods. The "p1" element first changes to red, then it slides up, and then it slides down:

```
$("#p1").css("color", "red").slideUp(2000).slideDown(2000
);
```

When chaining, the line of code could become quite long. However, jQuery is not very strict on the syntax; you can format it like you want, including line breaks and indentations.

This also works just fine:

Example

```
$("#p1").css("color", "red")
  .slideUp(2000)
  .slideDown(2000);
```

GET CONTENT AND ATTRIBUTES

JQuery contains powerful methods for changing and manipulating HTML elements and attributes.

DOM Manipulation:

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.

TEXT(), HTML(), AND VAL()

Three 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

GET ATTRIBUTES - ATTR()

The JQuery attr() method is used to get attribute values.

Example:

```
$("button").click(function(){
    alert($("#w3s").attr("href"));
});
```

SET CONTENT AND ATTRIBUTES

Set Content – text(), html(), and val():

- We will use the same three methods from the previous page to **set content**.

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

A CALLBACK FUNCTION FOR TEXT(), HTML(), AND VAL()

All of the three jQuery methods above: text(), html(), and val(), also come with a callback function.

The callback function has two parameters: the index of the current element in the list of elements selected and the original (old) value.

It return the string you wish to use as the new value from the function.

SET ATTRIBUTES - ATTR()

- The jQuery attr() method is also used to set/change attribute values.
- Demonstrates how to change (set) the value of the href attribute in a link.

Example:

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

{
    $("#w3s").attr("href", "https://www.w3schools.com/jqu
ery");
});
```

The attr() method also allows you to set multiple attributes at the same time.

A Callback Function for attr():

- The jQuery method attr(), also come with a callback function.
- The callback function has two parameters: the index of the current element in the list of elements selected and the original (old) attribute value.
- Return the string you wish to use as the new attribute value from the function.

Example:

```
$("button").click(function(){
    $("#w3s").attr("href", function(i, origValue){
        return origValue + "/jquery";
    });
});
```

JQuery - Add Elements

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

Add New HTML Content:

- We will look at four jQuery methods that are used to add new 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

JQUERY APPEND() METHOD

- The jQuery append() method inserts content AT THE END of the selected HTML elements.

Example:

```
$("p").append("Some appended text.");
```

JQuery prepend() Method:

- The JQuery prepend() method inserts content AT THE BEGINNING of the selected HTML elements.

Example:

```
$("p").prepend("Some prepended text.");
```

Add Several New Elements With append() and prepend():

The append() and prepend() methods can take an infinite number of new elements as parameters.

The new elements can be generated with text/HTML with JQuery, or with JavaScript code and DOM elements.

Example:

```
function appendText()
{
   var txt1 = "<p>Text.</p>";
   var txt2 = $("<p></p>").text("Text.");

    var txt3 = document.createElement("p");
    txt3.innerHTML = "Text.";
    $("body").append(txt1, txt2, txt3);
}
```

JQuery after() and before() Methods:

- The jQuery after() method inserts content AFTER the selected HTML elements.
- The jQuery before() method inserts content BEFORE the selected HTML elements.

Example:

```
$("img").after("Some text after");

$("img").before("Some text before");
```

Add Several New Elements With after() and before():

- The after() and before() methods can take an infinite number of new elements as parameters.
- The new elements can be generated with text/HTML with jQuery, or with JavaScript code and DOM elements.

Example:

```
function afterText()
 {
    var txt1 = "<b>I </b>";
    var txt2 = $("<i></i>").text("love ");

    var txt3 = document.createElement("b");

  txt3.innerHTML = "jQuery!";
    $("img").after(txt1, txt2, txt3);

<img>
}
```

JQuery - Remove Elements:

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

Remove Elements/Content:

To remove elements and content, there are mainly two jQuery methods.

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

empty() - Removes the child elements from the selected element

JQuery remove() Method

The JQuery remove() method removes the selected element(s) and its child elements.

Example:

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

JQUERY EMPTY() METHOD:

The JQuery empty() method removes the child elements of the selected element(s).

Example:

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

Filter the Elements to be Removed:

- The jQuery remove() method also accepts one parameter, which allows you to filter the elements to be removed.
- The parameter can be any of the jQuery selector syntaxes.

Example:

```
$("p").remove(".test");
```

JQUERY - GET AND SET CSS CLASSES

With JQuery, it is easy to manipulate the CSS of elements.

JQuery Manipulating CSS:

JQuery has several methods for CSS manipulation.

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

JQUERY ADDCLASS() METHOD

It shows how to add class attributes to different elements. Of course you can select multiple elements, when adding classes.

Example:

```
$("button").click(function(){
    $("h1, h2, p").addClass("blue");
    $("div").addClass("important");
});
```

We can also specify multiple classes within the addClass() method.

Example:

```
$("button").click(function(){
    $("#div1").addClass("important blue");
});
```

JQuery removeClass() Method:

It shows how to remove a specific class attribute from different elements.

Example:

```
$("button").click(function()
{
    $("h1, h2, p").removeClass("blue");
});
```

JQuery toggleClass() Method:

It show how to use the JQuery toggleClass() method. This method toggles between adding/removing classes from the selected elements.

Example:

```
$("button").click(function(){
    $("h1, h2, p").toggleClass("blue");
});
```

JQuery css() Method:

The css() method sets or returns one or more style properties for the selected elements.

Return a CSS Property:

To return the value of a specified CSS property, use the following syntax:

css("*propertyname*");

Example:

```
$("p").css("background-color");
```

Set a CSS Property:

To set a specified CSS property, use the following syntax:

Syntax:

css("*propertyname*","*value*");

Set Multiple CSS Properties:

To set multiple CSS properties, use the following syntax:

Syntax:

css({"*propertyname*":"*value*","*propertyname*":"*value*",...});

Example:

```
$("p").css({"background-color": "yellow", "font-
size": "200%"});
```

JQuery - Dimensions

JQuery Dimension Methods:

JQuery has several important methods for working with dimensions:

width()

height()

innerWidth()

innerHeight()

outerWidth()

outerHeight()

JQuery width() and height() Methods:

- The width() method sets or returns the width of an element (excludes padding, border and margin).
- The height() method sets or returns the height of an element (excludes padding, border and margin).

Example:

```
$("button").click(function()
{
    var txt = "";
    txt += "Width: " + $("#div1").width() + "</br>";
    txt += "Height: " + $("#div1").height();
    $("#div1").html(txt);
});
```

JQuery innerWidth() and innerHeight() Methods:

- The innerWidth() method returns the width of an element (includes padding).
- The innerHeight() method returns the height of an element (includes padding).

Example:

```
$("button").click(function()
{
    var txt = "";
    txt += "Inner width: " + $("#div1").innerWidth()
+ "</br>";
    txt += "Inner height: " + $("#div1").innerHeight();
    $("#div1").html(txt);
});
```

JQuery outerWidth() and outerHeight() Methods:

- The outerWidth() method returns the width of an element (includes padding and border).
- The outerHeight() method returns the height of an element (includes padding and border).

Example:

```
$("button").click(function()
{
    var txt = "";
    txt += "Outer width: " + $("#div1").outerWidth()
+ "</br>";
    txt += "Outer height: " + $("#div1").outerHeight();
    $("#div1").html(txt);
});
```

Note: The outerWidth(true) method returns the width of an element (includes padding, border, and margin).

The outerHeight(true) method returns the height of an element (includes padding, border, and margin).

JQuery More width() and height():

It returns the width and height of the document (the HTML document) and window (the browser viewport).

Example:

```
$("button").click(function()
{
    var txt = "";
    txt += "Document width/height: " +
$(document).width();
    txt += "x" + $(document).height() + "\n";
    txt += "Window width/height: " + $(window).width();
    txt += "x" + $(window).height();
    alert(txt);
});
```

JQUERY TRAVERSING

Traversing:

JQuery traversing, which means "move through", are used to "find" HTML elements based on their relation to other elements.

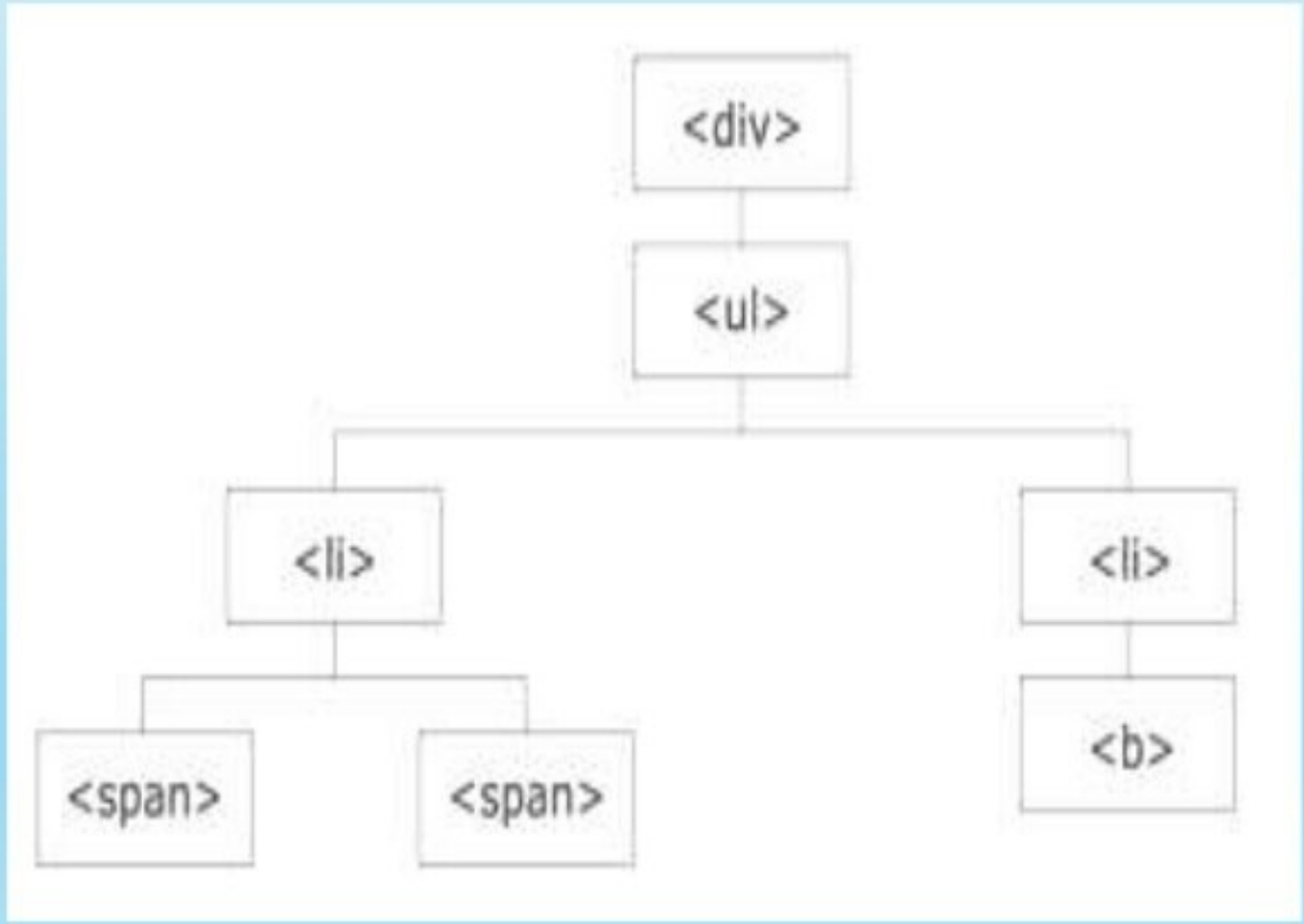
Start with one selection and move through that selection until you reach the elements you desire.

The image below illustrates a family tree.

With JQuery traversing, We can easily move up, down and sideways in the family tree, starting from the selected element.

This movement is called traversing – or moving through – the DOM.

- An ancestor is a parent, grandparent, great-grandparent, and so on.
- A descendant is a child, grandchild, great-grandchild, and so on.
- Siblings share the same parent.



Traversing the DOM:

- JQuery provides a variety of methods that allows us to traverse the DOM.
- The largest category of traversal methods are tree-traversal.
- The next chapters will show us how to travel up, down and sideways in the DOM tree.

JQUERY TRAVERSING - ANCESTORS:

An ancestor is a parent, grandparent, great-grandparent, and so on.

With jQuery you can traverse up the DOM tree to find ancestors of an element.

Traversing Up the DOM Tree

Three useful JQuery methods for traversing up the DOM tree

- parent()
- parents()
- parentsUntil()

JQuery parent() Method:

- The parent() method returns the direct parent element of the selected element.
- This method only traverse a single level up the DOM tree.

Example:

```
$(document).ready(function()
{
    $("span").parent();
});
```

- It returns the direct parent element of each elements

JQuery parents() Method:

- The parents() method returns all ancestor elements of the selected element, all the way up to the document's root element (<html>).

Example:

```
$(document).ready(function()
{
    $("span").parents();
});
```

- Example returns all ancestors of all elements.
- We can also use an optional parameter to filter the search for ancestors.

Example:

```
$(document).ready(function(){
    $("span").parents("ul");
});
```

- Example returns all ancestors of all elements that are <ul> elements

JQuery parentsUntil() Method:

- The parentsUntil() method returns all ancestor elements between two given arguments.

Example:

```
$(document).ready(function()
{
    $("span").parentsUntil("div");
});
```

- Example returns all ancestor elements between a and a element

JQuery Traversing - Descendants:

- A descendant is a child, grandchild, great-grandchild, and so on.
- With jQuery you can traverse down the DOM tree to find descendants of an element.

Traversing Down the DOM Tree:

Two useful jQuery methods for traversing down the DOM tree are:

- children()
- find()

children() Method:

- The children() method returns all direct children of the selected element.
- This method only traverse a single level down the DOM tree.

Example:

```
 $(document).ready(function()
{
    $("div").children();
});
```

JQuery find() Method:

- The find() method returns descendant elements of the selected element, all the way down to the last descendant.

Example

```
$(document).ready(function()
{
    $("div").find("span");
});
```

JQuery Traversing - Siblings:

- Siblings share the same parent.
- With jQuery you can traverse sideways in the DOM tree to find siblings of an element.

Traversing Sideways in The DOM Tree

- There are many useful jQuery methods for traversing sideways in the DOM tree:
  - siblings()
  - next()
  - nextAll()
  - nextUntil()
  - prev()
  - prevAll()
  - prevUntil()

JQuery siblings() Method:

- The siblings() method returns all sibling elements of the selected element.
- You can also use an optional parameter to filter the search for siblings.
- It returns all sibling elements.

Example:

```
$(document).ready(function()
{
    $("h2").siblings();
});
```

JQuery next() Method:

- The next() method returns the next sibling element of the selected element.
- It returns the next sibling of <h2>

Example:

```
$(document).ready(function(){
    $("h2").next();
});
```

JQuery nextAll() Method:

- The nextAll() method returns all next sibling elements of the selected element.

Example:

```
$(document).ready(function()
{
    $("h2").nextAll();
});
```

JQuery nextUntil() Method:

- The nextUntil() method returns all next sibling elements between two given arguments.

Example:

```
$(document).ready(function()
{
    $("h2").nextUntil("h6");
});
```

JQuery prev(), prevAll() & prevUntil() Methods:

- The prev(), prevAll() and prevUntil() methods work just like the methods above but with reverse functionality.
- They return previous sibling elements (traverse backwards along sibling elements in the DOM tree, instead of forward).

JQUERY TRAVERSING - FILTERING

Narrow Down The Search For Elements

The three most basic filtering methods are first(), last() and eq(), which allow you to select a specific element based on its position in a group of elements.

Other filtering methods, like filter() and not() allow you to select elements that match, or do not match, a certain criteria.

JQUERY FIRST() METHOD

The first() method returns the first element of the selected elements.

The following example selects the first <p> element inside the first element:

Example

```
$(document).ready(function(){
    $("div p").first();
});
```

JQUERY LAST() METHOD

The last() method returns the last element of the selected elements.

The following example selects the last <p> element inside the last element:

Example

```
$(document).ready(function(){
    $("div p").last();
});
```

JQUERY EQ() METHOD

The eq() method returns an element with a specific index number of the selected elements.

The index numbers start at 0, so the first element will have the index number 0 and not 1. The following example selects the second <p> element (index number 1):

Example

```
$(document).ready(function(){
    $("p").eq(1);
});
```

JQUERY FILTER() METHOD

The filter() method lets you specify a criteria. Elements that do not match the criteria are removed from the selection, and those that match will be returned.

The following example returns all <p> elements with class name "intro":

Example

```
$(document).ready(function(){
    $("p").filter(".intro");
});
```

FILTER

.filter(selector)

- `.filter('.some-class')`

.filter(function)

-
```
.filter(function() {
    return $(this).parents('li').length >= 2;
});
```

JQUERY NOT() METHOD

The not() method returns all elements that do not match the criteria.

Tip: The not() method is the opposite of filter().

The following example returns all <p> elements that do not have class name "intro":

Example

```
$(document).ready(function(){
    $("p").not(".intro");
});
```

FILTER

.not(selector)

```
.not('.some-class')
```

.not(function)

```
.not(function()  {
    return $(this).parents('li').length >= 2;

});
```

FILTER

.slice()

- `.slice(2)`
- `.slice(-2)`
- `.slice(3, 6)`
- `.slice(2, -1)`

.eq()

- `.eq(2)`
- `.eq(-2)`

TRAVERSAL METHODS

List of all traversal methods on the jQuery API site

http://api.jquery.com/category/traversing

CONTEXT

$('selector', 'context')

- Different from "or" selector – $('selector1, selector2')
- Same as $('context').find('selector')
- Not worth using; too confusing.

.add()

.andSelf()

.end()

CHECK

.hasClass(class)

.is(selector)

** returns a boolean

CHAINING

JavaScript has chaining built in.

- 'swap this text'.replace(/w/, 'n').replace(/this/,'that');
- '616-555-1212'.split('-').join('.');

jQuery takes advantage of this concept by having almost all methods return the jQuery object.

CHAINING

Chain traversal methods together

```
$('a').parent('li').siblings().find('a')
```

CHAINING

Attach multiple behaviors.

```
$('a').removeClass('old').addClass('new');
```

CHAINING

DOM Traversal methods are different from other jQuery chaining methods!

- New jQuery instance is created with each one.

```
$('a').addClass('foo').parent('li').removeClass('foo')
```

CHAINING

JavaScript ignores white space, so use it to your advantage.

```
var lis = $('.container li:first')
.addClass('first-li')
   .next()
   .addClass('second-li')
.end()
   .nextAll()
   .addClass('not-first-li')
.end(); // unnecessary; added for symmetry
```

LOOPING

Implicit Iteration

Explicit Iteration (Looping)

```
$('li').removeClass('myclass');          //implicit
```

```
$('li').each(function(index) {           //explicit
   $(this).append( ' #' + (index+1) );
});
```

THIS KEYWORD

Refers to the current object

jQuery sets **this** to matched elements in the jQuery object.

```
$('li').each(function() {
  console.log( this ); // DOM element
  console.log( $(this) );
});
```

TIPS

Store selectors used more than once in variables

Use length property to check existence

- ...but often no need for the check

```
var $listItems = $('li');
var numItems = $listItems.length

//no need for length check
$listItems.addClass('pretty');

if (numItems) {
  // do something with other elements
}
```

TIPS

Concatenate to pass in a variable

```
$('#menu li').each(function(index) {
    $(this).click(function() {
        $('#footer li:eq(' + index + ')')
        .addClass('active');
    });
});
```

TIPS

Avoid jQuery's custom selectors when possible

```
// bad
$(':checkbox')
// better
$('input:checkbox')
// best
$('input[type="checkbox"]')
```

TIPS

Avoid jQuery's custom selectors when possible

```
// uncool
$('div:first')

// cool
$('div').first();
```

THE BASICS

In JavaScript, you can work with the following things:

Strings: textual content. wrapped in quotation marks (single or double).

- 'hello, my name is Karl'
- "hello, my name is Karl"

Numbers: integer (2) or floating point (2.4) or octal (012) or hexadecimal (0xff) or exponent literal (1e+2)

Booleans: true or false

THE BASICS

In JavaScript, you can work with the following things:

Arrays: simple lists. *indexed* starting with 0

- ['Karl', 'Sara', 'Ben', 'Lucia']
- ['Karl', 2, 55]
- [['Karl', 'Sara'], ['Ben', 'Lucia']]

Objects: lists of key, value pairs

- {firstName: 'Karl', lastName: 'Swedberg'}
- {parents: ['Karl', 'Sara'], kids: ['Ben', 'Lucia']}

VARIABLES

Always declare your variables!

If you don't, they will be placed in the **global scope** (more about that later).

- **bad:** myName = 'Karl';
- **good:** **var** myName = 'Karl';
- **still good:** **var** myName = 'Karl';

```
// more stuff
myName = 'Joe';
```

CONDITIONALS AND OPERATORS

conditionals:

- if, else
- switch

operators:

- +, -, *, %, ++, --
- >, <, ==, !=, >=, <=, ===, !==
- !, &&, ||

LOOPS

Loops *iterate* through a list of some kind.

A common pattern in JavaScript is to build a list, or collection, and then do something with each item in that list.

LOOPS

CSS uses *implicit* iteration.

- div { color: red; } /* applies to ALL divs */

JavaScript relies on *explicit* iteration.
Must explicitly loop through each div

jQuery allows for both (because it does the looping for you)

LOOPS

The two most common loops...

- **for** loops — for general-purpose iteration. Used with arrays or array-like objects)
- **for-in** loops — used with arrays or objects (but don't use with arrays)

The other two are...

- **while** loops
- **do-while** loops

FOR LOOPS

three statements and a code block

1. initial value
2. condition
3. increment

```
for (initial value; condition; increment) {
  // code block
}
```

FOR LOOPS

```
for (var i = 0; i < 3; i++) {
  alert(i+1);
}
```

This is your variable, so it can be anything! (but developers often use “i”)

FOR LOOPS

```
var divs =
document.getElementsByTagName('div');

for (var i = 0; i < divs.length; i++) {
  // do something with each div
individually
    divs[i].style.color = 'red';
}
```

FOR LOOPS

```
var divs = document.getElementsByTagName('div');


// better to store length in variable first
var divCount = divs.length


for (var i = 0; i < divCount; i++) {
  // do something with each div individually
    divs[i].style.color = 'red';}
```

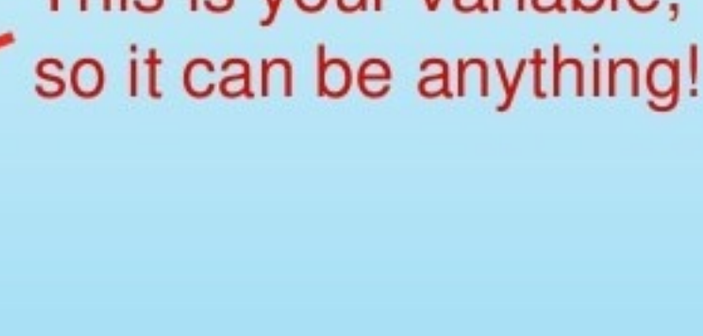
FOR LOOPS

```
var divs = document.getElementsByTagName('div');

// can store it directly in the initializer
for (var i=0, divCount=divs.length; i < divCount;
i++) {
  // do something with each div individually
  divs[i].style.color = 'red';
}
```

FOR-IN LOOPS

```
var family = {
  dad: 'Karl',
  mom: 'Sara',
  son: 'Benjamin',
  daughter: 'Lucia'
}
for (var person in family) {
  alert('The ' + person + ' is ' + family[person]);
}
```



WHILE AND DO-WHILE

```
var i = 1;
while (i < 4) {
  alert(i);
  i++;
}
var j = 1; // code block always executed at least once
do {
  alert(j);
  j++;
} while (j < 4)
```

THE BASICS: FUNCTIONS

In JavaScript, you can also work with **functions:**

Functions allow you to **define** a block of code, name that block, and then **call** it later as many times as you want.

- **function** myFunction( ) { /* code goes here */ } // defining
- **myFunction**( ) // calling the function *myFunction*

You can define functions with **parameters**

- function myFunction(**param1**, **param2**) { /* code goes here */ }

You can call functions with **arguments**:

- myFunction(**'one'**, **'two'**)

FUNCTIONS

```
// define a function
function doSomething() {
  alert('I am something');
}
// call the function
doSomething();
```

FUNCTIONS

```
// define a function
function sumThing(a, b) {
  return a + b;
}
// call the function
alert( sumThing(1, 2) );
```

FUNCTIONS

```
// define a function
function sumThing(a, b) {
  return a + b;
}
var mySum = sumThing(1, 2);
// call the function
alert( mySum );
```

THE `ARGUMENTS` OBJECT

Every function has an `arguments` object

- a collection of the arguments passed to the function when it is called
- an "array-like object" in that it is indexed and has a **length** property but can't attach array methods to it
- can be looped through
- allows for variable number of arguments

FUNCTIONS

```
// call the function
function logThing() {
  for (var i=0; i < arguments.length; i++) {
    console.log(arguments[i]);
  }
}
// call the function
logThing(1, 2, 'three');
/* prints to the console:  >> 1  >> 2  >> three
*/
```

EXERCISE

Convert the sumThing function to allow for variable number of arguments.

```
function sumThing(a, b) {
   return a + b;
}
```

Use a *for* loop to loop through the *arguments* object, adding to a "sum" variable with each iteration.

After the loop, return *sum*.

(SIMPLE) SOLUTION

```
// define a function
function sumThing() {
  var sum = 0,
      countArgs = arguments.length;
  for (var i = 0; i < countArgs; i++) {
  sum += arguments[i];
  }
  return sum;
}
// call the function
console.log( sumThing(1, 2, 4 ) );
```

RETURNING FUNCTIONS

Functions can return other functions

```
function multiple(n) {
  function f(x) {
    return x * n;
  }
  return f;
}
var triple = multiple(3);
var quadruple = multiple(4);

console.log( triple(5) ); // 15
console.log( quadruple(5) ); // 20
console.log( multiple(4)(5) ); // 20
```

NAMED VS. ANONYMOUS FUNCTIONS

Named:

- function **foo**() { } // function **declaration**
- var foo = function **foo**() { }; // function **expression**

Anonymous:

- var foo = function() { }; // function **expression**

ANONYMOUS FUNCTIONS

Prevalent in jQuery

Good for creating closures

Used as "callback" functions

Can be used as object properties (methods)

let's take a look ...

ANONYMOUS FUNCTIONS

Prevalent in jQuery

```
$(document).ready(function() {
});
```

ANONYMOUS FUNCTIONS

Good for creating closures

```
function() {
    // variables are defined within this scope
    // avoid name collisions
}
```

ANONYMOUS FUNCTIONS

Good for creating closures

Can be *defined* and then immediately *invoked:*
"immediately invoked function expression," (a.k.a. **IIFE**; pronounced "**iffy**")

```
(function() {
  // variables are defined within this scope
  // avoid name collisions
})();
```

ANONYMOUS FUNCTIONS

Good for creating closures

Used by plugins to keep jQuery safe.

```
(function($) { // "$" is the function's param

})(jQuery); // function is called with "jQuery"
```

ANONYMOUS FUNCTIONS

Used as "callback" functions

```
$('p').slideDown('slow', function() {
      // code in here is not executed
      // until after the slideDown is
finished
      // jQuery calls the code in here when
effect ends
});
```

OBJECTS

In JavaScript, everything is an object. Well, almost everything.

Objects are objects : **{ }**

Arrays are objects : **[ ]**

even Functions are objects : **function( ) { }**

jQuery is an object

Numbers, strings, and booleans (true/false) are primitive data types, but they have object wrappers.

GLOBAL OBJECT

In the browser environment, the global object is **window**
It collects all functions and variables that are global in scope.
Usually implied.

Comes with some useful properties and methods:

```
location
parseInt(); parseFloat()
isNaN()
encodeURI(); decodeURI()
setTimeout(); clearTimeout()
setInterval(); clearInterval()
```

DATE OBJECT

```
var now = new Date(); // current date and time
var then = new Date('08/12/2000 14:00');

console.log( then.getTime() ); // 966103200000

console.log( then.toString() );
  // Sat Aug 12 2000 14:00:00 GMT-0400 (EDT)

console.log( then.getMonth() ); // 7 !!!!
```

REGEXP OBJECT
REGULAR EXPRESSION

Object constructor

```
var re = new RegExp('hello');
```

Regular expression literal

```
var re = /hello/;
```

CREATING A REGEXP

Object constructor

```
var re = new RegExp('hello');
```

Regular expression literal

```
var re = /hello/;
```

USING A REGEXP

```
var text = 'The quick brown fox';

var re = new RegExp('quick');
console.log( re.test(text) ); // true


console.log( /brown/.test(text) ); // true
console.log( /red/.test(text) ); // false
```

REGEXP SYNTAX

Most characters (incl. all alphanumerics) represent themselves

Special characters can be escaped with a backslash (\)

CHARACTER CLASSES

/t.p/ matches 'tap' and 'tip' and 'top'

/t[ai]p/ matches 'tap' and 'tip', not 'top'

/t[a-k]p/ matches 'tap' and 'tip', not 'top'

/t[^m-z]p/ matches 'tap' and 'tip', not 'top'

REPETITION

/frog*/ matches 'fro', 'frog', 'frogg', ...

/frog+/ matches 'frog', 'frogg', ...

/frog?/ matches 'fro' or 'frog'

/frog{2,3}/ matches 'frogg' or 'froggg'

GROUPING

Grouping

- /(frog)*/ matches "frog" or "frogfrog"

Alternation

- /th(is|at)/ matches "this" and "that"

ANCHOR POINTS

^ matches the beginning of a string

$ matches the end of a string

\b matches word boundaries

EXERCISES

Write a regular expression that matches any word that starts with a vowel.

Write a regular expression that matches any HTML tag.

STRING REGEXP METHODS

str.search(re)

- str.match(re)
- str.replace(re, replacement)
- str.split(re)

STRING REPLACEMENT

```
var str =

  'The quick brown fox jumps over the lazy dog.';

console.log(str.replace(/[aeiou]/, '*'));

// Th* quick brown fox jumps over the lazy dog.
```

REGEXP FLAGS

Placed after closing / character

Global (g): find as many as possible

Case insensitive (i)

Multiline (m): ^ and $ work with newlines

STRING REPLACEMENT

```
var str =

  'The quick brown fox jumps over the lazy dog.';

console.log(str.replace(/[aeiou]/g, '*'));
// Th* q**ck br*wn f*x j*mps *v*r th* l*zy d*g.

console.log(str.replace(/the/gi, 'a'));
// a quick brown fox jumps over a lazy dog.
```

BACKREFERENCES

```
var str = 'The quick brown fox jumps over the lazy dog.';

console.log(str.replace(/r(.)/g, '$1x'));

// The quick boxwn fox jumps ove xthe lazy dog.
```

REPLACEMENT FUNCTIONS

```
var str = 'Kill 5+9 birds with 2+5 stones.';

function add(match, first, second) {
  return parseInt(first, 10) + parseInt(second, 10);
}
str = str.replace(/([0-9]+)\+([0-9]+)/g, add);
console.log(str);
// Kill 14 birds with 7 stones.
```

EXERCISES

Write a function that uppercases all the vowels in a string.

Write a function that strips the angle brackets from around any HTML tags in a string.

GREEDINESS

Repeat operators usually match as much of the string as possible; they are *greedy*

JavaScript supports *reluctant* repetition as well

- Append ? to the repeat operator

AND MUCH MORE

JavaScript supports most Perl regular expression extensions

- POSIX character classes
- Unicode character escapes
- Look-ahead assertions

MATH OBJECT

Not a constructor, a singleton

Gathers useful methods and properties

```
Math.PI
Math.abs(), Math.sin(), Math.pow(),
Math.random(),
Math.max(), Math.min()
Math.round(), Math.floor(), Math.ceil()
```

CSS TIP

Object literal notation looks a lot like CSS style rule notation!

CSS:

```
h3 {
font-size: 1.2em;
line-height: 1;
}
```

JS:

```
var h3 = {
fontSize: '1.2em',
'line-height': 1
};
```

OBJECT LITERALS

person is the **object**

firstName and **lastName** are **properties**

hello is a **method** (a property that is a function)

```
var person = {
   firstName: 'Karl',
   lastName: 'Swedberg',
   hello: function() {
      return 'Hello, my name is ' +
            this.firstName + ' ' + this.lastName;
   }
};
```

OBJECT LITERALS

interests is a **property** *and* an **object**

```
var person = {
  firstName: 'Karl',
  lastName: 'Swedberg',
  hello: function() {
    return 'Hello, my name is ' +
        this.firstName + ' ' + this.lastName;
  },
  interests: {
    athletic: ['racquetball', 'karate', 'running'],
    musical:  ['rock', 'folk', 'jazz', 'classical']
  }
};
```

OBJECT LITERALS

```
var person = {
  firstName: 'Karl',
  lastName: 'Swedberg',
  hello: function() {
    return 'Hello, my name is ' +
        this.firstName + ' ' + this.lastName;
  } // ← notice, no comma here!
};
```

OBJECT LITERALS
"DOT" NOTATION

```
var person = {
  firstName: 'Karl',
  lastName: 'Swedberg',
  hello: function() {
    return 'Hello, my name is ' +
        this.firstName + ' ' + this.lastName;
  }
};

// "dot" notation
person.firstName; // 'Karl'
person.lastName; // 'Swedberg'
person.hello() // 'Hello, my name is Karl Swedberg'
```

OBJECT LITERALS
ARRAY NOTATION

```
var person = {
  firstName: 'Karl',
  lastName: 'Swedberg',
  hello: function() {
    return 'Hello, my name is ' +
        this.firstName + ' ' + this.lastName;
  }
};

// array notation
person['firstName']; // 'Karl'
person['lastName']; // 'Swedberg'
person['hello']() // 'Hello, my name is Karl Swedberg'
```

OBJECT LITERALS

```
var person = {
  firstName: 'Karl',
  lastName: 'Swedberg',
  hello: function() {
    return 'Hello, my name is ' +
          this.firstName + ' ' + this.lastName;
  },
  interests: {
    athletic: ['racquetball', 'karate', 'running'],
    musical:  ['rock', 'folk', 'jazz', 'classical']
  }
};
// person['interests']['musical'][1] == ??
// == person.interests.musical[1]
```

OBJECT LITERALS

```
var person = {
  firstName: 'Karl',
  lastName: 'Swedberg',
  hello: function() {
    return 'Hello, my name is ' +
        this.firstName + ' ' + this.lastName;
  }
};
person.firstName = 'Karl';
var prop = 'firstName';
person[ prop ]; // 'Karl'
prop = 'lastName';
person[ prop ]; // 'Swedberg'
```

OBJECT LITERALS

```
var blah;
var person = {
  firstName: 'Karl',
  lastName: 'Swedberg',
  hello: function() {
    return 'Hello, my name is ' +
          this.firstName + ' ' + this.lastName;
  }
};
for (var el in person) {
  blah = typeof person[el] == 'function' ?
          person[el]() :
          person[el];
  console.log( blah );
}
```

OBJECT LITERALS

Great as function arguments

single argument allows flexibility when calling the function

```
doSomething({
  speed: 'fast',
  height: 500,
  width: 200,
  somethingElse: 'yes'
});

doSomething({width: 300});
```

REFERENCING SCRIPTS IN THE HTML

browser slides

SELECTORS & TRAVERSAL

AT THE HEART OF JQUERY...

Find something

Do something

TIPS

Avoid jQuery's custom selectors when possible

```
// slower
$('li:eq(3)')
$('li:lt(3)')

// faster
$('li').eq(3)
$('li').slice(0, 3)
```

JSON
JAVASCRIPT OBJECT NOTATION

a *data interchange* format. In other words, a format for passing data back and forth

"discovered" and popularized by Douglas Crockford

a *subset* of JavaScript Object Literal Notation

- a tree-like structure of object(s) and/or array(s)
- no functions
- all strings, including object keys, take double quotes

JSON


```
{
  "firstName": "Karl",
  "lastName": "Swedberg",
  "age": 24,
  "interests": {
    "athletic": [
      "racquetball",
      "karate"
    ]
  }
}
```

JSON


```
{"firstName":"Karl",
"lastName":"Swedberg", "age":24,
"interests":
{"athletic":["racquetball",
"karate"]
}
}
```

Thank You