Lecture9: DOM, jQuery

Maharishi University of Management -Fairfield, Iowa © 2019

Problems with DOM

- "The *DOM*. It is what most people hate when they say they hate JavaScript" (Douglas Crockford)
- the DOM can be clunky to use
- the same code doesn't always work the same way in every browser
  - code that works great in Chrome, Firefox, Safari, ... may fail in IE and vice versa

jQuery framework

‣ the jQuery JavaScript library adds many useful features to JavaScript:

  ‣ many useful extensions to the DOM

  ‣ adds utility functions for built-in types String, Array, Object, Function

  ‣ improves event-driven programming

  ‣ many cross-browser compatibility fixes

  ‣ makes Ajax programming easier (seen later)

```
<script
  src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></
  /script>
```

jQuery Design Principles

▸ jQuery is so powerful in part because of these design principles

- ▸ an expressive method for defining a set of elements, a superset of CSS selectors

- ▸ useful and commonly needed methods for navigating the DOM tree

- ▸ heavily overloaded APIs

- ▸ functional programming techniques that apply operations to sets of elements at a time

- ▸ method chaining for succinct operations

window.onload

- We cannot use the DOM before the page has been constructed. jQuery gives us a more convenient way to do this.

- The DOM way

```javascript
window.onload = function() {
  // do stuff with the DOM
}
```

- The direct jQuery translation

```javascript
$(document).ready(function() {
  // do stuff with the DOM
});
```

- The jQuery way

```javascript
$(function() {
  // do stuff with the DOM
});
```

jQuery boilerplate

```
/* select the id=square, and give it
   event handler with an alert for
   click */


$('#square').click(function() {
  alert("Boink.");
});
```

Main Point Preview
jQuery selectors

When the argument to $() is a CSS selector the function will return a "jQuery object" that contains a group of selected DOM elements. CSS selectors are a simple, natural, and powerful tool used by jQuery to identify groups of DOM elements.

Science of Consciousness:

A mantra is a simple, natural, and powerful tool that we use in the TM Technique.

Aspects of the DOM and jQuery

- **Identification:**
  - how do I obtain a reference to the node that I want.
  - using css-like selectors to get target nodes
- **Traversal:**
  - Find nodes by tree traversal relations
  - using children, sibling, parent, etc links to get target nodes
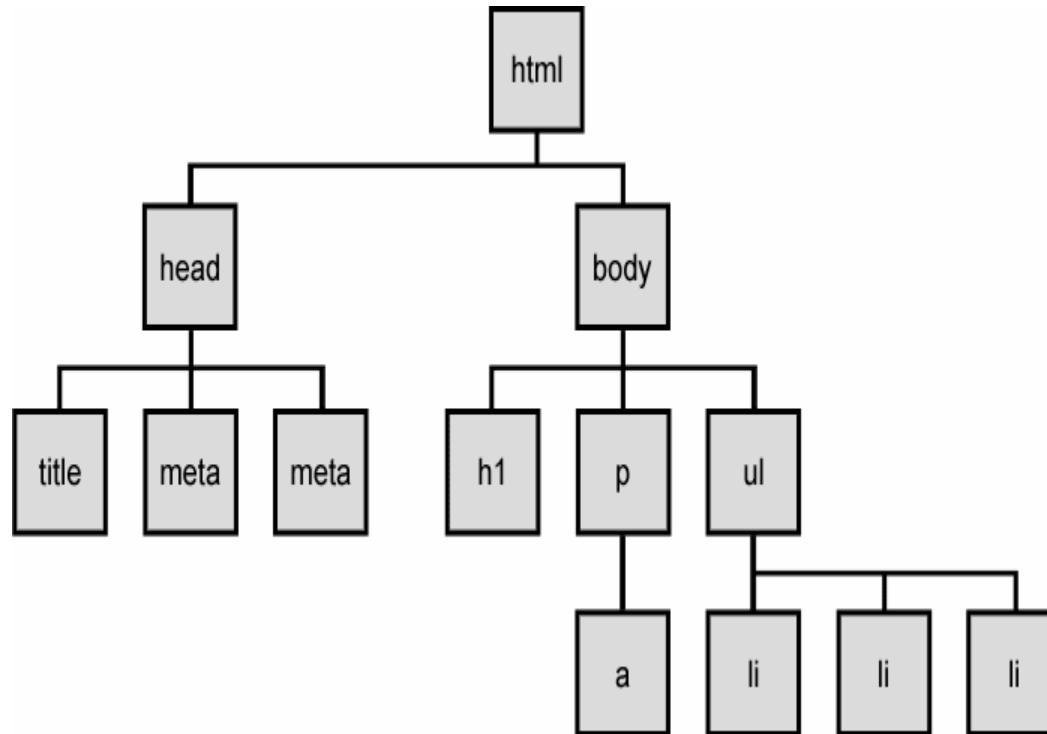- **Node Manipulation:**
  - how do I get or set aspects of a DOM node.
  - e.g., style, attributes, innerHTML
- **Tree Manipulation:**
  - how do I change the structure of the page.

The DOM tree

▸ **The elements of a page are nested into a tree-like structure of objects**

  ▸ half of the challenge is singling out elements that you want

DOM selectors

name	description
getElementById	returns the first element with the specified id.
getElementsByTagName	returns array of all elements with the given tag, such as "div"
getElementsByName	returns array of all elements with the given name attribute (mostly useful for accessing form controls)
querySelector *	returns the first element that would be matched by the given CSS selector string
querySelectorAll *	returns an array of all elements that would be matched by the given CSS selector string

jQuery / DOM comparison

DOM method	jQuery equivalent
getElementById("id")	$("#id")
getElementsByTagName("tag")	$("tag")
getElementsByName("somename")	$("[name='somename']")
querySelector("selector")	$("selector")
querySelectorAll("selector")	$("selector")

jQuery node identification

▸ The $ (aka jQuery) function selects elements from the DOM using most any CSS selector.

```javascript
// single argument selectors
const elem = $("#myid");
const elems = $("input")

// conjunction of selectors—requires both
const elems = $("#input.special");

// disjunction of selector--any can match
const elems = $("#myid, p");

// hierarchy selectors
const elems = $("#myid div p"); //space for descendent selection
const elems = $("#myid > div p");  // > for child selection

// context selectors
const $elem = $("#myid");
const specials = $("li.special", $elem); //or elem.find("li.special");

// combination
const elems = $("#myid > h1.special:not(.classy)");
```

# 5 core single argument selectors

1. ID
   ```
   $("#age")
   ```

2. type
   ```
   $("input")
   ```

3. attribute
   ```
   $("[required] ")
   $("[type=password]")
   ```

4. Class
   ```
   $(".special")
   ```

5. filter
   ```
   $("tr:first")
   ```

DOM contextual identification

- use querySelectorAll() and querySelector() on any DOM object.
- searches from that part of the DOM tree downward.
- Programmatic equivalent of a CSS context selector

```
var list = document.getElementsByTagName("ul")[0];

var specials = list.querySelectorAll('li.special');
```
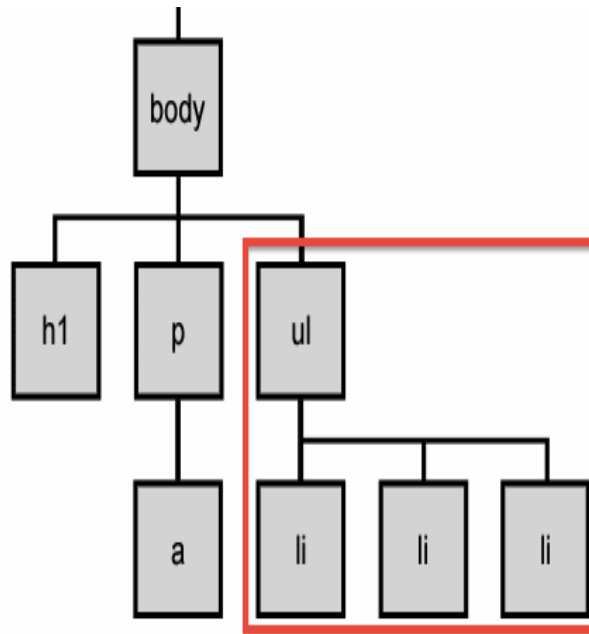
jQuery contextual identification

- jQuery gives two ways to do contextual element identification
  - Second argument is root of subtree ("context")

```
const elem = $("#myid");

// alternate syntaxes, same jQuery implementation
const specials = $("li.special", elem);
const specials = elem.find("li.special");
```

jQuery selector references

▸ jQuery has a powerful set of selectors from CSS plus several of its own. (**bold** = jQuery specific)

ID selector (#)	**Descendent selector ( )**	**:button**	**:not()**
Class selector (.)	Child selector (>)	:text	:has()
Attribute selector [name='value']	:first-child :only-child	:input	:lt()
Element selector (tag)	:nth-child()	:checked	:gt()
Multiple selector (,)	:even	:file	:eq()

Key jQuery Concepts and Terms

- ## the jQuery function
  - refers to the global jQuery function that is normally aliased as $ for brevity

- ## a jQuery object
  - the object returned by the jQuery function that often represents a group of elements

- ## selected elements
  - the DOM elements that you have selected for, most likely by some CSS selector passed to the jQuery function and possibly later filtered further

A jQuery object

- The **$** function always (even for ID selectors) returns an array-like object called a jQuery object.
- This returned jQuery object wraps the originally selected DOM objects.
- You can access the actual DOM object by accessing the elements of the jQuery object.

```
// false
document.getElementById("id") === $("#myid");

// true
document.getElementById("id") === $("#myid")[0];
```

Using $ as a wrapper

▸ **$ adds extra functionality to DOM elements**

▸ **passing an existing DOM object to $ will give it the jQuery upgrade**

```
// convert regular DOM objects to a jQuery
   object
var elem = document.getElementById("myelem");
elem = $(elem);


var elems =
   document.querySelectorAll(".special");
elems = $(elems);
```

Main Point
jQuery selectors

When the argument to $() is a CSS selector the function will return a "jQuery object" that contains a group of selected DOM elements. CSS selectors are a simple, natural, and powerful tool used by jQuery to identify groups of DOM elements.

Science of Consciousness:

A mantra is a simple, natural, and powerful tool that we use in the TM Technique.

Main Point Preview
jQuery Traversal

In addition to selection by CSS-like selectors, jQuery has tree traversal relations that find children, sibling, parent, ancestor nodes

Science of Consciousness:

jQuery traversal operations facilitate relationships with all elements of the DOM. The experience of self-referral awareness facilitates one's relationships with family, friends, countrymen, and the world.

Aspects of the DOM and jQuery

- **Identification:**
  - how do I obtain a reference to the node that I want.
  - using css-like selectors to get target nodes
- **Traversal:**
  - Find nodes by tree traversal relations
  - using children, sibling, parent, etc links to get target nodes
- **Node Manipulation:**
  - how do I get or set aspects of a DOM node.
  - e.g., style, attributes, innerHTML
- **Tree Manipulation:**
  - how do I change the structure of the page.

Types of DOM nodes

```html
<p>
 This is a paragraph of text with a
 <a href="/path/page.html">link in it</a>.
</p>
```


- ▸ **element nodes** (HTML tag)

  - ▸ can have children and/or attributes

- ▸ **text nodes** (text in a block element)

- ▸ **attribute nodes** (attribute/value pair)
  - ▸ text/attributes are children in an element node
  - ▸ cannot have children or attributes
  - ▸ not usually shown when drawing the DOM tree
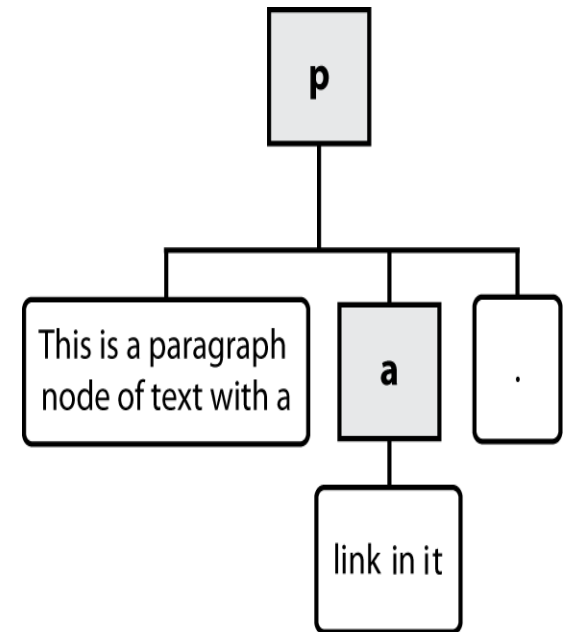
Traversing the DOM tree

▸ every node's DOM object has the following properties:

name(s)	description
firstChild, lastChild	start/end of this node's list of children
childNodes	array of all this node's children
nextSibling, previousSibling	neighboring nodes with the same parent
parentNode	the element that contains this node

▸ complete list of DOM node properties and methods

jQuery Traversing - Ancestors

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

- **parent() - returns the direct parent element of the selected element.**

- **parents() - returns all ancestor elements of the selected element.**
  - You can also use an optional parameter to filter the search for ancestors.

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

```
$("span").parent();
$("span").parents();
$("span").parents("ul"); // returns all ancestors of
  all <span> elements that are <ul> elements
$("span").parentsUntil("div");
```

jQuery Traversing - Descendants

A descendant is a child, grandchild, great-grandchild, and so on.

- **children() - returns all direct children of the selected element**
  - You can also use an optional parameter to filter the search for children
- **find() - returns descendant elements of the selected element**

```
$("div").children();


// returns all <p> elements with the class name "first", that are
  direct children of <div>
$("div").children("p.first");


// returns all <span> elements that are descendants of <div>:
$("div").find("span");


// returns all descendants of <div>
$("div").find("*");
```

jQuery Traversing - Siblings

- siblings() - returns all sibling elements of the selected element
  - can also use an optional parameter to filter the search for siblings
- next() - returns the next sibling element
- nextAll() - returns all following sibling elements
- nextUntil() - returns all sibling elements between two given arguments
- prev()
- prevAll()
- prevUntil()

jQuery Traversing - Filtering

```javascript
// selects the first <p> element inside the first <div> element
$("div p").first();

// selects the last <p> element inside the last <div> element
$("div p").last();

// returns an element with a specific index number of the
   selected elements
$("p").eq(1);
```

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

```javascript
// returns all <p> elements with class name "intro"
$("p").filter(".intro");

// returns all elements that do not match the criteria - opposite
   to filter
$("p").not(".intro");
```

➢ **All traversing methods**

Optimize selector performance

▸ DOM operations can be slow, especially with rich GUI applications
  ▸ More noticeable with mobile web apps

▸ Use ID selector if possible
▸ Cache selectors
  ▸ const $elm = $("#someID");
  ▸ $elm.addClass('special');

▸ Favor context selectors over CSS descendent
  ▸ var $parent = $("#parent1");
  ▸ var $elm = $parent.find('.special');
▸ Instead of $elm = $("#parent1 .special");

▸ Cache or chain selectors to avoid repeating searches

▸ Note $parent and $elm are just js identifiers. Some people use $ as first letter to indicate a jQuery Object

Main Point
jQuery Traversal

In addition to selection by CSS-like selectors, jQuery has tree traversal relations that find descendent, sibling, ancestor nodes

Science of Consciousness:

jQuery traversal operations facilitate relationships with all elements of the DOM. The experience of self-referral awareness facilitates one's relationships with family, friends, and the world.

Main Point Preview
Node Manipulation

The jQuery object returned by the selection mode of $() is a collection of DOM elements wrapped by jQuery functionality. This object can read style properties as well as set them by using the css method of jQuery.

Science of Consciousness:

Node manipulation involves locating nodes, observing or reading values, and changing values. Our TM practice develops our ability to locate quiet states of awareness. Advanced techniques and the TM-Sidhi Program develop abilities to experience and manipulate different characteristics of pure consciousness

Aspects of the DOM and jQuery

- **Identification:**
  - how do I obtain a reference to the node that I want.
  - using css-like selectors to get target nodes
- **Traversal:**
  - Find nodes by tree traversal relations
  - using children, sibling, parent, etc links to get target nodes
- **Node Manipulation:**
  - how do I get or set aspects of a DOM node.
  - e.g., style, attributes, innerHTML
- **Tree Manipulation:**
  - how do I change the structure of the page.

Looping over the DOM

```
$("li").each(function(idx, e) {
 // do stuff with e
});
```

‣ You cannot use a regular JavaScript for each loop on either because they are not technically arrays, just array-like objects.

Inside the jQuery each loop

```
$("li").each(function(idx, e) {
 // do stuff with e
});
```

▸ return false to exit the loop early
▸ e is a plain old DOM object
   ▸ We can upgrade it again using $ if we want

```
$("li").each(function(idx, e) {
 e = $(e);
 // do stuff with e
});
```

▸ the `this` keyword can be used instead of e (alternate syntax if don't need idx)
▸ when a function is called by an object, that object becomes the context or `'this'` parameter

```
$("li").each(function() {
 // do stuff with this
     this = $(this);
});
```

jQuery .each()

```javascript
//$.each(array/obj, fn) versus $(selector ).each(fn)

$.each([52, 97], function(index, value) {
 console.log(index + ": " + value);
});
//0: 52
//1: 97

var obj = {
 "course": "CS472",
 "name": "WAP"
};
$.each(obj, function(key, value) {
 console.log(key + ": " + value);
});
//course: CS472
//name: WAP
//would this work with JavaScript forEach?
```

Modifying DOM nodes

▸ DOM nodes have fields that correspond to the attributes in HTML tags. There are a few exceptions

HTML attributes	DOM fields
title	.title
id	.id
class	.className
style="prop: value"	.style.prop = value

Getting/setting CSS classes in DOM

```javascript
function highlightField() {
 // turn text yellow and make it bigger
 var elem = document.getElementById("id");

 if (!elem.className) {
      elem.className = "highlight";
 } else if (elem.className.indexOf("invalid") < 0) {
      elem.className += " highlight";
 }

}
```

▸ JS DOM's className property corresponds to HTML `class` attribute

▸ somewhat clunky when dealing with multiple space-separated classes as one big string

▸ className value is a string, not an array like we would want

Getting/setting CSS [classes in jQuery](#)

```
function highlightField() {
 // turn text yellow and make it bigger
 if (!$("#myid").hasClass("invalid")) {
 $("#myid").addClass("highlight");
 }
}
```

- addClass, removeClass, hasClass, toggleClass manipulate CSS classes

- similar to existing className DOM property, but don't have to manually split by spaces

Adjusting styles with the DOM

```html
<button id="clickme">Color Me</button>
```

```javascript
window.onload = function() {
  document.getElementById("clickme").onclick =
  changeColor;
};
function changeColor() {
 var clickMe = document.getElementById("clickme");
 clickMe.style.color = "red";
}
```

Property	Description
style	lets you set any CSS style property for an element

▸ contains same properties as in CSS, but with camelCasedNames
  ▸ examples: backgroundColor, borderLeftWidth, fontFamily

Adjusting styles in jQuery

```
function biggerFont() {
  // turn text yellow and make it bigger
  $("#clickme").css("color", "yellow");
  var size = parseInt($("#clickme").css("font-size"));
  var newsize = size + 16 + "px";
  $("#clickme").css("font-size", newsize );
}
```

▸ css function of the jQuery object works even if styles not previously set

▸ Accepts familiar font-size syntax in addition to fontSize

▸ css(*property*) gets the property value

▸ css(*property*, *value*) sets the property value

Common bug: incorrect usage of existing styles

```
// bad!
$("#main").css("top", $("#main").css("top") + 100 +
"px");
```

▸ the above example computes e.g. "200px" + 100 + "px" , which would evaluate to "200px100px"

▸ a corrected version:

```
// correct
$("#main").css("top",
  parseInt($("#main").css("top")) + 100 + "px");
```

Recall: Unobtrusive styling

```
function okayClick() {
  this.style.color = "red";
  this.className = "highlighted";
}
.highlighted { color: red; }
```

- well-written JavaScript code should contain as little CSS as possible
- use JS to set CSS classes/IDs on elements
- define the styles of those classes/IDs in your CSS file
- Conclusion: unobtrusive styling means avoid using the .css method in jQuery
  - Set classes in JavaScript code to handle any style changes

jQuery method behavior

▸ Getters typically operate only on the first of the jQuery object's selected elements.

```html
<ul>
 <li style="font-size: 10px">10px font size</li>
 <li style="font-size: 20px">20px font size</li>
 <li style="font-size: 30px">30px font size</li>
</ul>
$("li").css("font-size"); // returns '10px'
```

▸ Setters typically operate on all of the selected DOM elements.

```html
$("li").css("font-size", "15px"); // sets all selected
  elements to '15px'
<ul>
 <li style="font-size: 15px">10px font size</li>
 <li style="font-size: 15px">20px font size</li>
 <li style="font-size: 15px">30px font size</li>
</ul>
```

jQuery method parameters

▸ **Many jQuery object methods are overloaded**

▸ getter syntax:
```
$("#myid").css(propertyName);
```

▸ setter syntax:
```
$("#myid").css(propertyName, value);
```

▸ multi-setter syntax:
```
$("#myid").css({
  'propertyName1': value1,
  'propertyName2': value2,
  ...
});
```

▸ modifier syntax:
```
$("#myid").css(propertyName,
    function(idx, oldValue) {
        return newValue;
});
```

▸ function allows for computation based on the old value
▸ What do you think the multi-modifier syntax is?

common jQuery mistake

```
// bad jQuery
$("div").css("top",
  parseInt($("div").css("top")) + 100 +
  "px");
```

▸ **Does not work if multiple selected objects. Why?**

▸ **a corrected version:**

```
$("div").css("top", function(idx, old) {
 return parseInt(old) + 100 + "px";
}); // good jQuery
```

jQuery method returns

▸ When there is no other return to make, jQuery methods return the same jQuery object back to you

method	return type
$("#myid");	jQuery object
$("#myid").children();	jQuery object
$("#myid").css("margin-left");	String
$("#myid").css("margin-left", "10px");	**jQuery object**
$("#myid").addClass("special");	**jQuery object**

jQuery chaining

```
$("img").css("color", "red");
$("img").attr("id", "themainarea");
$("img").addClass("special");
```

▸ The implictly returned jQuery object allows for chaining of method calls.

```
$("img") // good jQuery style
 .css("color", "red")
 .addClass("special")
 .attr("src", "foo.png");
// we could chain further right here
```

More node manipulation with jQuery

jQuery method	functionality
.hide()	set CSS display: none
.show()	set CSS display to original value, e.g., block, inline
.empty()	remove everything inside the element, innerHTML = ""
.html()	get/set the innerHTML without escaping html tags
.text()	get/set the innerHTML, HTML escapes the text first
.val()	get/set the value of a form input, select, textarea, ...
.height()	get/set the height in pixels, returns a Number
.width()	get/set the width in pixels, return a Number

Main Point
Node Manipulation

The jQuery object returned by the selection mode of $() is a collection of DOM elements wrapped by jQuery functionality. This object can read and write node properties such as id, name, value, class, style, etc.

Science of Consciousness:

Node manipulation involves finding nodes and then reading and writing values. TM practice develops our ability to locate quiet states of awareness. Advanced techniques and the TM-Sidhi Program develop abilities to experience and manipulate different characteristics of pure consciousness

Main Point Preview
Tree Manipulation

An efficient and clear style is to use object literals to create complex DOM elements in jQuery. Lists of elements should be created as a single list element before insertion for efficiency.

Science of Consciousness: Actions arising from quiet levels of awareness are naturally efficient.

Aspects of the DOM and jQuery

▸ **Identification:**

  ▸ how do I obtain a reference to the node that I want.

  ▸ using css-like selectors to get target nodes

▸ **Traversal:**

  ▸ how do I move around the DOM tree.

  ▸ using children, sibling, parent, etc links to get target nodes

▸ **Node Manipulation:**

  ▸ how do I get or set aspects of a DOM node.

  ▸ e.g., style, attributes, innerHTML

▸ **Tree Manipulation:**

  ▸ how do I change the structure of the page.

DOM innerHTML hacking

▶ Why not just code this way?

```
document.getElementById("myid").innerHTML +=
 "<p>A paragraph!</p>";
```

▶ Imagine that the new node is more complex

```
document.getElementById("myid").innerHTML +=
 "<p style='color: red; " +
 "margin-left: 50px;' " +
 "onclick='myOnClick();'>" +
 "A paragraph!</p>";
```

   ▶ ugly
   ▶ bad style on many levels
   ▶ HTML (and CSS and JS!) code embedded within JS

▶ error-prone: must carefully distinguish " and '

Create nodes in jQuery

▸ **jQuery creates a DOM node if the argument is an HTML element**

```
var newElement = $("<div>");
```

▸ **creating element does not add it to the page**
▸ **can be added as child of an existing element**
  ▸ *append* for last child, *prepend* for first child

```
$("#myid").append(newElement);
```

▸ **jQuery programmers typically - 1 line**

```
$("#myid").append($("<div>"));
```

▸ `Can remove any matched elements`
  `$("li:contains('kangeroo')").remove();`

Creating complex nodes in jQuery

▸ The terrible way, this is no better than innerHTML hacking

```
$("<p id='myid' class='special'>My paragraph is
  awesome!</p>")
```

▸ The bad way, decent jQuery, but we can do better

```
$("<p>")
  .attr("id", "myid")
  .addClass("special")
  .text("My paragraph is awesome!");
```

▸ The good way

```
$("<p>", { "id": "myid", "class": "special", "text":
  "My paragraph is awesome!" });
```

Poor jQuery example

```
$("#ex2 span.special").each(function(i, elem) {
 var img = $("<img>")
 .attr("src", "../images/laughing_man.jpg")
 .attr("alt", "laughing man")
 .css("vertical-align", "middle")
 .css("border", "2px solid black")
 .click(function() {
     alert("clicked");
 });
 $(elem).prepend(img);
});
```

Good jQuery example

```
$("#ex3 span.special").prepend($("<img>", {
 "src": "../images/laughing_man.jpg",
 "alt": "laughing man",
 "css": {
      "vertical-align": "middle",
      "border": "2px solid black"
 },
 "click": function() {
      alert("clicked");
 }
}));
```

Insert via .append or .prepend

➤ **if creating list of elements, best practice is to create entire list before inserting**

- ➤ entire DOM redraws whenever modified

➤ **.append**

- ➤ make first child of matched elements

➤ **.prepend**

- ➤ make last child of matched elements

Insert via .before and .after

➢ **.after, .before**

  ➢ insert new (or move selected) content after or before
    matched elements

Main Point
Tree Manipulation

An efficient and clear style is to use object literals to create complex DOM elements in jQuery. Lists of elements should be created as a single list element before insertion for efficiency.

Science of Consciousness: Actions arising from quiet levels of awareness are naturally efficient.

jQuery $ function signatures

- Responding to the page ready event
  `$(function);`

- Identifying elements
  `$("selector", [context]);`

- Upgrading DOM elements
  `$(elements);`

- Creating new elements
  `$("<html>", [properties]);`

The TM Technique and Pure Consciousness

▸ 1. jQuery is a powerful and widely used JavaScript API for working with the DOM that provides cross-browser compatibility.

▸ 2. The $() function is heavily overloaded. It will be a different function depending on the arguments it has, including running a callback function on page load, selecting DOM elements, wrapping DOM elements, and creating new DOM elements.

▸ **3. Transcendental consciousness**. The TM Technique is a convenient and cross-platform API for experiencing transcendental consciousness.

▸ **4. Impulses within the transcendental field:** Thoughts and actions at quiet levels of awareness are simple, natural, and effective because they are in accord with all the laws of nature that reside at these deep levels.

▸ **5. Wholeness moving within itself:**   Advanced TM Techniques and the TM-Sidhi Programs are powerful APIs for bringing the calm dynamism and bliss of pure consciousness into the experiences of daily activity.