LESSON 10
EVENTS & JQUERY

Actions Supported by All the Laws of Nature

Maharishi University of Management -Fairfield, Iowa © 2019

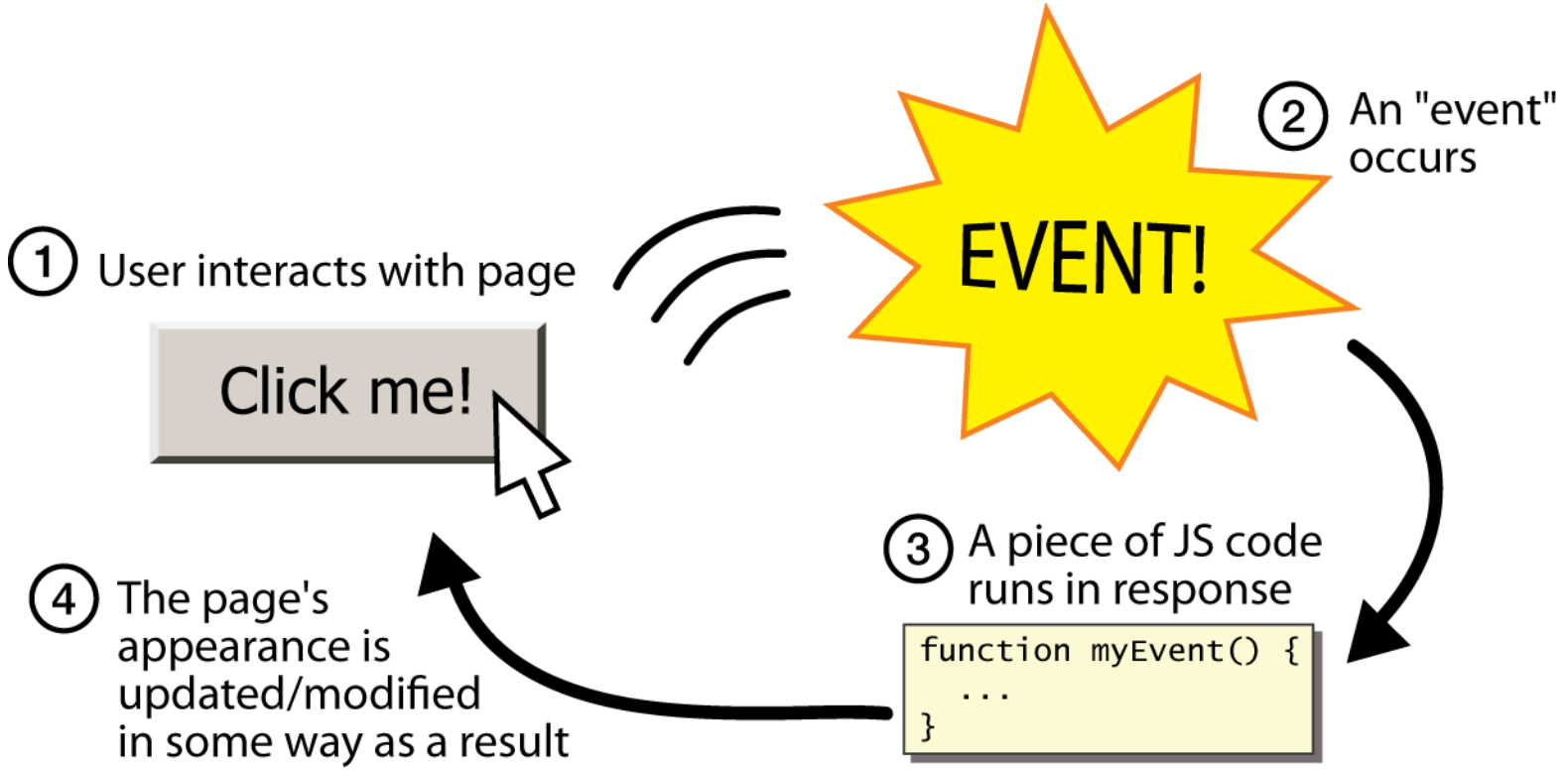
Main Point Preview

- JavaScript programs have no main. They respond to user actions called events.

- **Science of Consciousness**: JavaScript was designed as a language that could effectively respond to browser and DOM events. We respond most effectively to events in our environment if our awareness is settled and alert.

Event-driven programming

- JS programs have no main; they respond to user actions called **events**
- **event-driven programming**: writing programs driven by user events



Button: <button>

```
<button>Click me!</button>
```

- button's text appears inside tag; can also contain images
- To make a responsive button or other UI control:
  - choose the control (e.g. button) and event (e.g. mouse click) of interest
  - write a JavaScript function to run when the event occurs
  - attach the function to the event on the control

Unobtrusive JavaScript

```
// where element is a DOM element object
element.onevent = function;

<button id="ok">OK</button>
var okButton = document.getElementById("ok");
okButton.onclick = okayClick;
```

- it is legal to attach event handlers to elements' DOM objects in your JavaScript code
  - notice that you do **not** put parentheses after the function's name
- this is better style than attaching them in the HTML
- Where should we put the above code?

Linking to a JavaScript file: script

- JS code can be placed directly in the HTML file's body or head (like CSS)
  - but this is bad style (should separate content, presentation, and behavior)
- script tag should be placed in HTML page's head
- script code should be stored in a separate .js file (like CSS)

```
<script src="filename" type="text/javascript"></script>
<script src="example.js" type="text/javascript"></script>
```

When does my code run?

```
<html>
 <head>
  <script src="myfile.js" type="text/javascript"></script>
 </head>
 <body> ... </body> </html>

// global code
var x = 3;
function f(n) { return n + 1; }
function g(n) { return n - 1; }
x = f(x);
```

- your file's JS code runs the moment the browser loads the script tag
  - any variables are declared immediately
  - any functions are declared but not called, unless your global code explicitly calls them
- at this point in time, the browser has not yet read your page's body
  - none of the DOM objects for tags on the page have been created yet

A failed attempt at being unobtrusive

```
<html>
 <head>
 <script src="myfile.js" type="text/javascript"></script> </head>
 <body> ... </body> </html>
<div><button id="ok">OK</button></div>

// global code
document.getElementById("ok").onclick = okayClick; // error: null
```

- problem: **`myfile.js`** code runs the moment the script is loaded
- script in head is processed before page's body has loaded
  - no elements are available yet or can be accessed yet via the DOM
- we need a way to attach the handler after the page has loaded...

The window.onload event

```
// this will run once the page has finished loading
function functionName() {
 element.event = functionName;
 element.event = functionName;
 ...
}
window.onload = functionName; // global code
```

- we want to attach our event handlers right after the page is done loading
  - there is a global event called window.onload event that occurs at that moment
- in window.onload handler we attach all the other handlers to run when events occur

Common unobtrusive JS errors

- many students mistakenly write () when attaching the handler

```
window.onload = pageLoad();
window.onload = pageLoad;
okButton.onclick = okayClick();
okButton.onclick = okayClick;
```

  - IMPORTANT FUNDAMENTAL CONCEPT !!!
    - Function reference versus evaluation

- event names are all lowercase, not capitalized like most variables

```
window.onLoad = pageLoad;
window.onload = pageLoad;
```

Anonymous function example

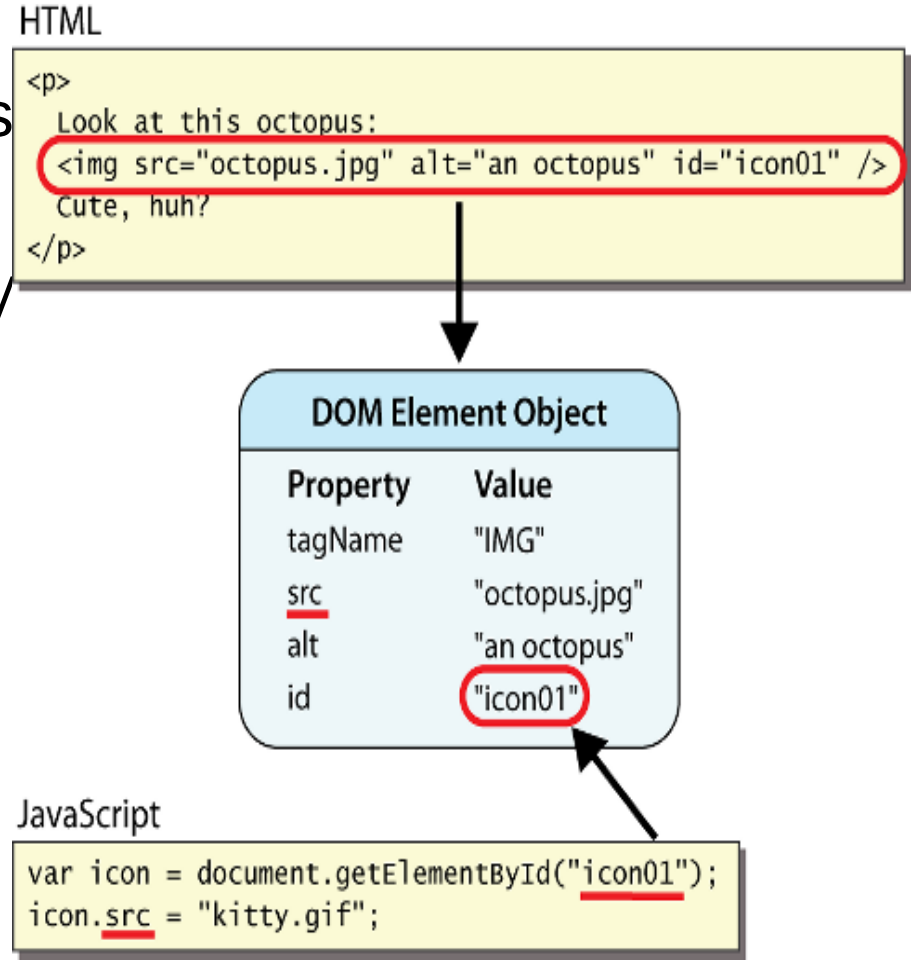
```
window.onload = function() {
 var okButton = document.getElementById("ok");
 okButton.onclick = okayClick;
};
function okayClick() {
 alert("booyah");
}
```

or the following is also legal (though harder to read):

```
window.onload = function() {
 var okButton = document.getElementById("ok");
 okButton.onclick = function() {
  alert("booyah");
 };
};
```

DOM element objects

- every element on the page has a corres
- access/modify the attributes of the DOM with *objectName.attributeName*



Accessing elements: document.getElementById

```
const name = document.getElementById("id");

<button onclick="changeText();">Click me!</button>
<input id="output" type="text" value="replace me" />

function changeText() {
 const textbox = document.getElementById("output");
 textbox.value = "Hello, world!";
}
```

- document.getElementById returns the DOM object for an element with a given id
- can change the text in most *form controls* by setting the value property
- Browser automatically updates the screen when any DOM object is changed

More advanced example

```
<button onclick="swapText();">Click me!</button>
<span id="output2">Hello</span>
<input id="textbox2" type="text" value="Goodbye" />

function swapText() {
 var span = document.getElementById("output2");
 var textBox = document.getElementById("textbox2");
 var temp = span.innerHTML;
 span.innerHTML = textBox.value;
 textBox.value = temp;
}
```

In class Exercise: Swap image

can change the text inside most elements by setting the innerHTML property

Main Point

- The purpose of most JavaScript code is to manipulate the HTML DOM, which is a set of JavaScript objects that represent each element on an HTML page.

- **Science of Consciousness**: The purpose of thought is to produce successful actions and achievements in the world, and more powerful thoughts will produce more successful actions.

Main Point Preview

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

Science of Consciousness: 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.

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

jQuery $ function signatures

- Responding to the page ready event

```
$(function);
```

- Identifying elements

```
$("selector", [context]);
```

- Upgrading DOM elements

```
$(elements);
```

- Creating new elements

```
$("<html>", [properties]);
```

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

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

- The direct jQuery translation

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

- The jQuery way

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

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.

```
// 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)");
```

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);
```

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

Main Point

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

Science of Consciousness: 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.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

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.