Lecture 5: JavaScript for Modern Web Apps

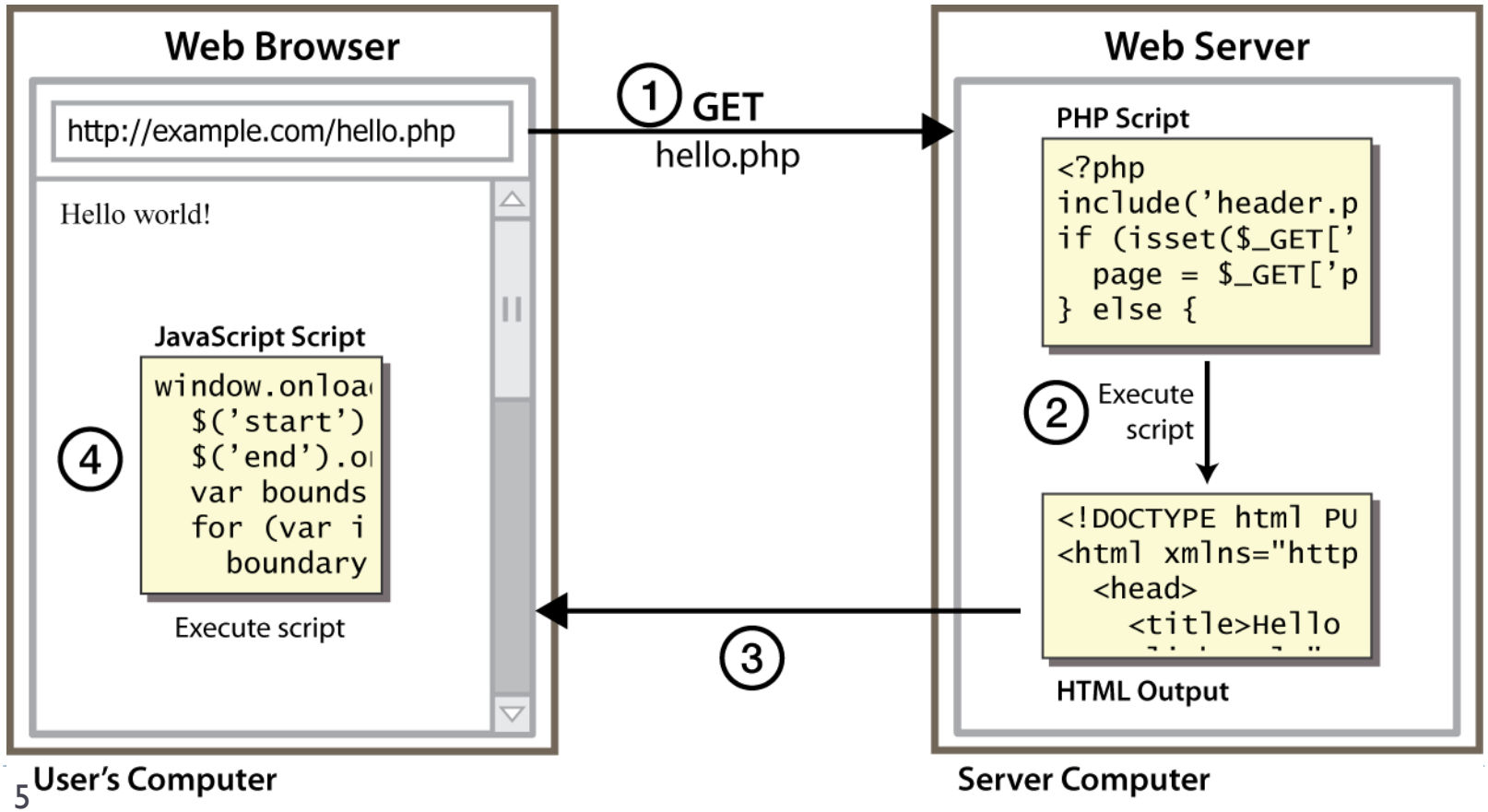
Maharishi University of Management -Fairfield, Iowa © 2016

Key JavaScript Concepts

- Key JavaScript Concepts
- JavaScript Syntax
- Program Logic
- Advanced JavaScript Syntax

Client-side Scripting

- client-side script: code runs in browser *after* page is sent back from server
  - often this code manipulates the page or responds to user actions



Why use client-side programming?

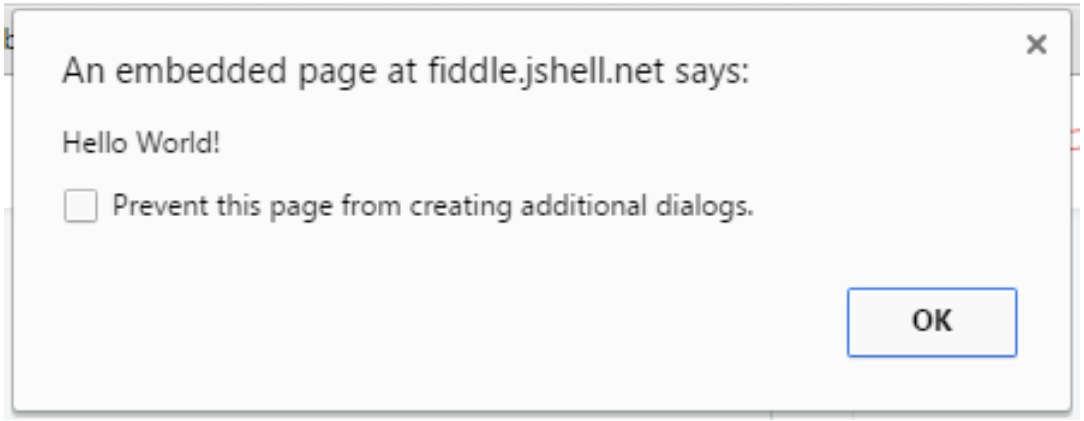
- client-side scripting (JavaScript) benefits:
  - usability: can modify a page without having to post back to the server (faster UI)
  - efficiency: can make small, quick changes to page without waiting for server
  - event-driven: can respond to user actions like clicks and key presses

What is JavaScript?

- a lightweight programming language ("scripting language")
- used to make web pages interactive
  - insert dynamic text into HTML (ex: user name)
  - react to events (ex: page load user click)
  - get information about a user's computer (ex: browser type)
  - perform calculations on user's computer (ex: form validation)
- a web standard (but not supported identically by all browsers)
- NOT related to Java other than by name and some syntactic similarities

A JavaScript statement: alert

```
alert("Hello World!");
```



- A JS command that pops up a dialog box with a message

Variables and types

```
var name = expression;

var age = 32;
var weight = 127.4;
var clientName = "Connie Client";
```

- variables are declared with the var keyword (case sensitive)
  - Replaced by let and const with JS6
- types are not specified, but JS does have types ("loosely typed")
  - Number, Boolean, String, Object (Array , Function) , Null, Undefined
  - can find out a variable's type by calling typeof

Block-scoped variable (ES6)

```
let callbacks = []
for (let i = 0; i <= 2; i++) {
  callbacks[i] = function() {
     return i * 2
  }
}
Alert(i);
callbacks[0]() === 0
callbacks[1]() === 2
callbacks[2]() === 4
```

Constants (ES6)

- Support for constants (also known as "immutable variables"), i.e., variables which cannot be re-assigned new content.
  - Like var , but cannot be reassigned
- Notice: this only makes the variable itself immutable, not its assigned content (for instance, in case the content is an object, this means the object itself can still be altered).

```
const PI = 3.1415926;
PI = 10;  //error
const  point = {x:1, y: 10};
point.y = 20;  //ok
```

Number Type

```
let enrollment = 99;
let medianGrade = 2.8;
let credits = 5 + 4 + (2 * 3);
```

- integers and real numbers are the same type (no int vs. double)
- same operators: + - * / % ++ -- = += -= *= /= %=
- similar precedence to Java
- many operators auto-convert types: "2" * 3 is 6
  - What is "2" + 3 ?

Logical Operators

- >, <, >=, <=, &&, ||, ! ==, !=, ===, !==
- most logical operators automatically convert types:
  - 5 < "7" is true
  - 42 == 42.0 is true
  - "5.0" == 5 is true
- === and !== are *strict equality* tests; checks both type and value
  - "5.0" === 5 is false
- Always use *strict* equality

Comments (same as Java)

```
// single-line comment
/* multi-line comment */
```

- identical to Java's comment syntax
- recall: 4 comment syntaxes
  - HTML: <!-- *comment* -->
  - CSS/JS/PHP: /* *comment* */
  - Java/JS/PHP: // *comment*
  - Python: # *comment*

String Type

```
const s = "Connie Client";
let fName = s.substring(0, s.indexOf(" ")); //
  "Connie"
let len = s.length; // 13
let s2 = 'Melvin Merchant'; // can use "" or ' '
```

- methods: charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase
  - charAt returns a one-letter String (there is no char type)
- length property (not a method as in Java)
- concatenation with + : 1 + 1 is 2, but "1" + 1 is "11"

More about String

- escape sequences behave as in Java: \' \" \& \n \t \\
- to convert between numbers and Strings:

```
let count = 10;
let s1 = "" + count; // "10"
let s2 = count + " bananas, ah ah ah!"; // "10
  bananas, ah ah ah!"
const n1 = parseInt("42 is the answer"); // 42
const n2 = parseFloat("booyah"); // NaN
```

- to access characters of a String, use [*index*] or charAt:

```
const firstLetter = s[0];
let firstLetter = s.charAt(0);
let lastLetter = s.charAt(s.length - 1);
```

Boolean Type

```
const iLikeWebApps = true;
const ieIsGood = "IE6" > 0; // false
if ("web dev is great") { /* true */ }
if (0) { /* false */ }
```

- any value can be used as a Boolean
  - "falsey" values: false, 0, 0.0, NaN, empty String(""), null, and undefined
  - "truthy" values: anything else, include objects
- !! Idiom – gives boolean value of any variable
  - const x=5;
  - console.log(!x);
  - console.log(x);
  - console.log(!!x);

Special values: null and undefined

```
let ned = null;
const benson = 9;
let caroline;
// at this point in the code,
// ned is null
// benson's 9
// caroline is undefined
```

- undefined : has been declared, but no value assigned
  - e.g., caroline variable above
  - vars that are "hoisted" to beginning of a function
- null : exists, and was specifically assigned an value of null
- reference error when try to evaluate a variable that has not been declared
  - reference error different from undefined
  - undefined means declared, but no value assigned

if/else statement (same as Java)

```
if (condition) {
  statements;
} else if (condition) {
  statements;
} else {
  statements;
}
```

- identical structure to Java's if/else statement
- JavaScript allows almost anything as a *condition*
  - *JS idioms*
  - //initialize a variable if not set yet
    - if (!a) { a = 10;}
  - //only use a variable if it has a value
    - if (b) { console.log(b); }

for loop (same as Java)

```
for (initialization; condition; update){
 statements;
}

let sum = 0;
for (let i = 0; i < 100; i++) {
 sum = sum + i;
}

const s1 = "hello";
let s2 = "";
for (let i = 0; i < s.length; i++) {
 s2 += s1[i] + s1[i];
} // s2 stores "hheellllloo"
```

while loops (same as Java)

```
while (condition) {
statements;
}

do {
statements;
} while (condition);
```

- break and continue keywords also behave as in Java

Arrays

```
let name = []; // empty array
let name = [value, value, ..., value]; // pre-filled
name[index] = value; // store element

const ducks = ["Huey", "Dewey", "Louie"];
let stooges = []; // stooges.length is 0
stooges[0] = "Larry"; // stooges.length is 1
stooges[1] = "Moe"; // stooges.length is 2
stooges[4] = "Curly"; // stooges.length is 5
stooges[4] = "Shemp"; // stooges.length is 5
```

- two ways to initialize an array – see example for another two ways
- length property (grows as needed when elements are added)

Array methods

```
let a = ["Stef", "Jason"]; // Stef, Jason
a.push("Brian"); // Stef, Jason, Brian
a.unshift("Kelly"); // Kelly, Stef, Jason, Brian
a.pop(); // Kelly, Stef, Jason
a.shift(); // Stef, Jason
a.sort(); // Jason, Stef
```

- array serves as many data structures: list, queue, stack, ...
- methods: concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift
  - push and pop add/remove from back
  - unshift and shift add/remove from front
  - shift and pop return the element that is removed

Array methods: map, filter, reduce

```
//functional programming:  map, filter, reduce can replace many loops

const a = [1,3,5,3,3];

//translate/map all elements in an array to another set of values
const b = a.map(function(elem, i, array) {
  return elem + 3;})
console.log(b); // [4,6,8,6,6]

//remove unwanted elements based on a condition
const c = a.filter(function(elem, i, array){
  return elem !== 3;});
console.log(c); //[1,5]

//find a cumulative or concatenated value based on elements across the array
const d = a.reduce(function(prevVal, elem, i, array){
  return prevVal + elem;});
console.log(d); //15
```

Function Declaration

```
function name() {
 statement ;
 statement ;
 ...
 statement ;
}

function square(number) {
 return number*number;
}
```

- declarations are "hoisted" (vs function expressions) – see Lecture07
  - They can be declared anywhere in a file, and used before the declaration.

Function Expressions

- Can be Anonymous function
  - Widely used in JS with event handlers

```
const square = function(number) { return number *
  number };
let x = square(4) // x gets the value 16
```

Can also have a name to be used inside the function to refer to itself

```
//NFE (Named Function Expression)
const factorial = function fac(n)
     { return n < 2 ? 1 : n * fac(n - 1) };

console.log(factorial(3));
```

- Basically, a function expression is same syntax as a declaration, just used where an expression is expected ('lhs' vs 'rhs')

Which is better: Function Declaration or Function Expression?

- Function declarations have two advantages over function expressions:
  - They are hoisted, so you can call them before they appear in the source code. (some consider this poor style)
  - They have a name. - the name of a function is useful for debugging; especially anonymous function. (ES6 infers name)
- Conclusion
  - Neither of the above are significant
  - Can use functions declarations if desired, but can always use function expressions to accomplish same thing (except hoisting)
- `function f1(){} f1.name // 'f1'`
- `const f2 = function(){}; f2.name //'' or 'f2' in ES6`
- `const f3 = function f3(){}; f3.name //'f3'`
  - `Named Function Expression (NFE)`

Anonymous functions

- JavaScript allows you to declare anonymous functions
- Can be stored as a variable, attached as an event handler, etc.
- Keeping unnecessary names out of namespace for performance and safety

```
window.onload = function(){
  alert("Hello World!");
}
```

Semicolon ;

- Semicolons are (technically) 'optional'
  - JS implicitly adds them to our code if it makes the parser happy
  - in certain cases that can cause problems
    - return, var, break, throw, …
  - Best practice to explicitly include them
  - Include brackets at the end of line versus new line
    - K&R (Kernighan and Ritchie) versus OTBS (one true brace style) styles

```
function a() {
  return {
           a: 1 ;
    }
}
function b()
{  //OTBS – ok, but not good practice according to some (Crockford, …)
 return  //semicolon gets inserted here
    {
    a: 1 ;
    }
}
console.log(a()); //object
console.lo(b());  //undefined
```

Web Storage

- With web storage, web applications can store data locally within the user's browser.
- Before HTML5, application data had to be stored in cookies, included in every server request. Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance.
- Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server.
- Web storage is per origin (per domain and protocol). All pages, from one origin, can store and access the same data.

The *localStorage* Object

- The *localStorage* object stores the data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.

Example

```
// Store
localStorage.setItem("lastname", "Smith");
// Retrieve
document.getElementById("result").innerHTML
  = localStorage.getItem("lastname");
```

The *localStorage* Object

- The previous example above could also be written like this:

```
// Store
localStorage.lastname = "Smith";
// Retrieve
document.getElementById("result").innerHTML
= localStorage.lastname;
```

- The syntax for removing the "lastname" localStorage item is as follows:

```
localStorage.removeItem("lastname");
```

The *sessionStorage* Object

- The *sessionStorage* object is equal to the *localStorage* object, except that it stores the data for only one session.
- The data is deleted when the user closes the specific browser tab.
- The following example counts the number of times a user has clicked a button, in the current session:

```
if (sessionStorage.clickcount) {
    sessionStorage.clickcount =
        Number(sessionStorage.clickcount) + 1;
} else {
    sessionStorage.clickcount = 1;
}
document.getElementById("result").innerHTML =
  "You have clicked the button " +
   sessionStorage.clickcount +
   " time(s) in this session.";
```

The *sessionStorage* Object

- The following example counts the number of times a user has clicked a button, in the current session:

```
if (sessionStorage.clickcount) {
    sessionStorage.clickcount =
        Number(sessionStorage.clickcount) + 1;
} else {
    sessionStorage.clickcount = 1;
}
document.getElementById("result").innerHTML =
  "You have clicked the button " +
   sessionStorage.clickcount +
   " time(s) in this session.";
```

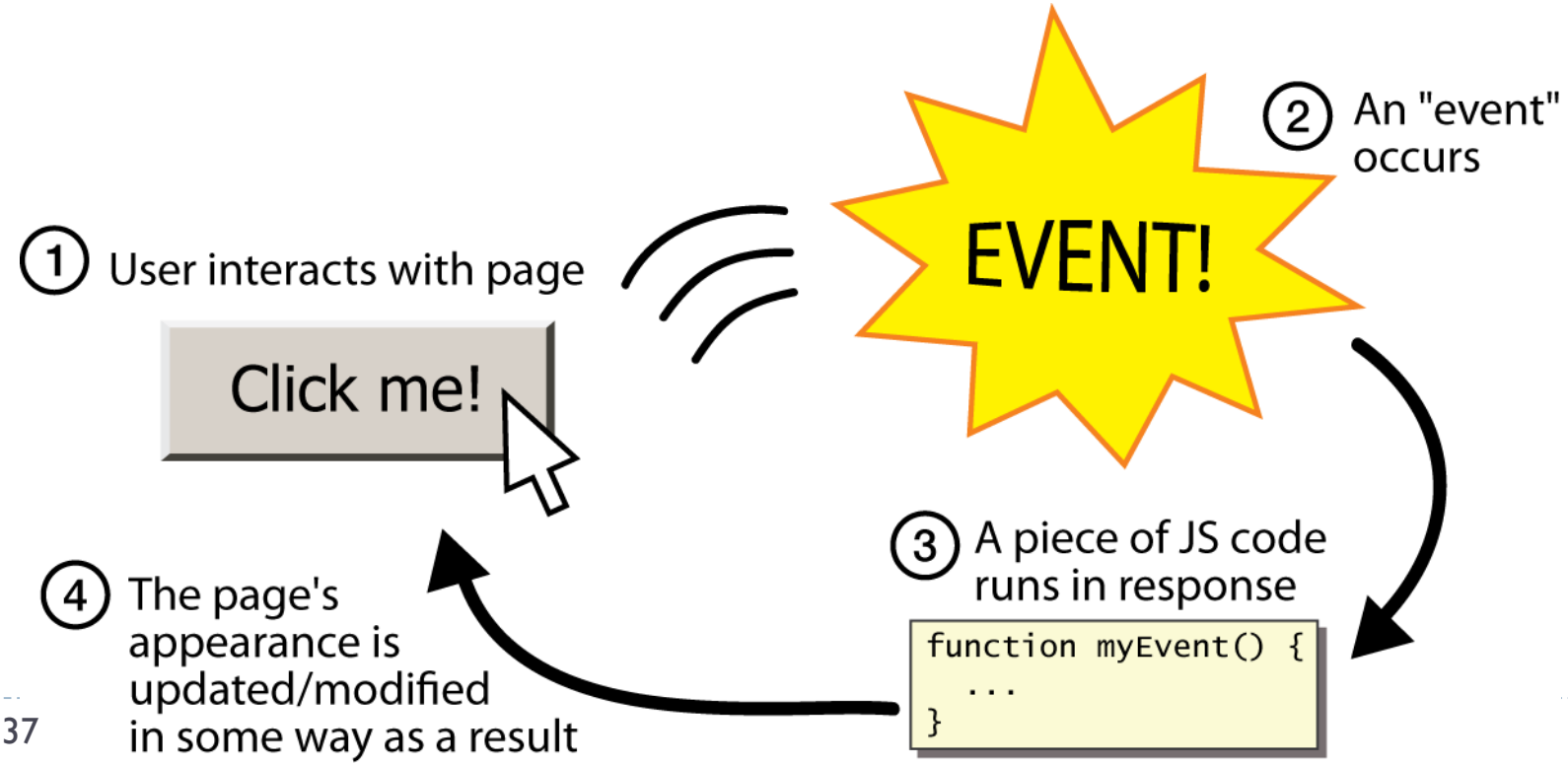
Main Point

JavaScript is a loosely typed language. It has types, but does no compile time type checking. Programmers must be cautious of automatic type conversions, including conversions to Boolean types. It has a flexible and powerful array type as well as distinct types of null and undefined.

Science of Consciousness: To be an effective JavaScript programmer one needs to understand the principles and details of the language. If our awareness is established in the source of all the laws of nature then our actions will spontaneously be in accord with the laws of nature for a particular environment.

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

Event handlers

```
<element attributes onclick="function();">...

<button onclick="myFunction();">Click me!</button>
```

- JavaScript functions can be set as **event handlers**
  - when you interact with the element, the function will execute
- onclick is just one of many event HTML attributes we'll use

- but popping up an alert window is disruptive and annoying
  - A better user experience would be to have the message appear on the page...

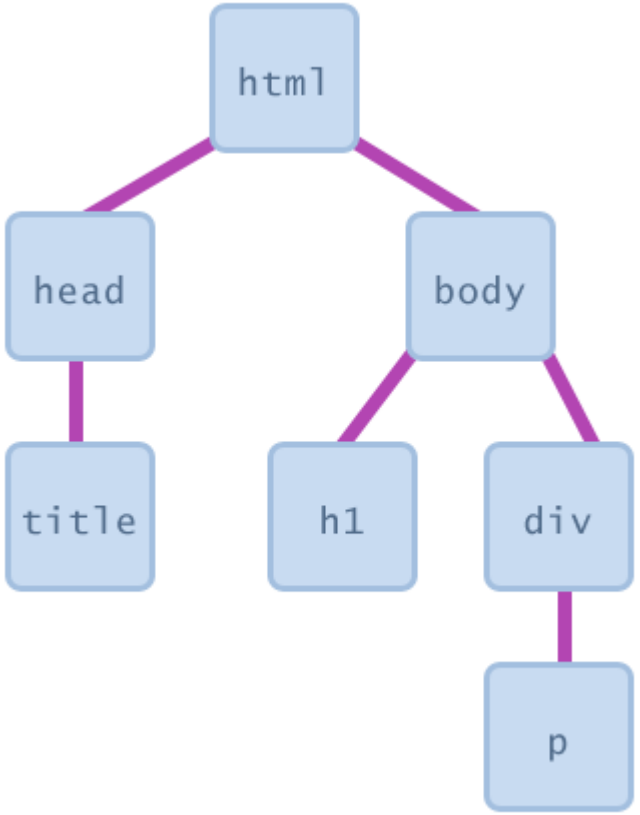
Main Point

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.

Document Object Model (DOM)

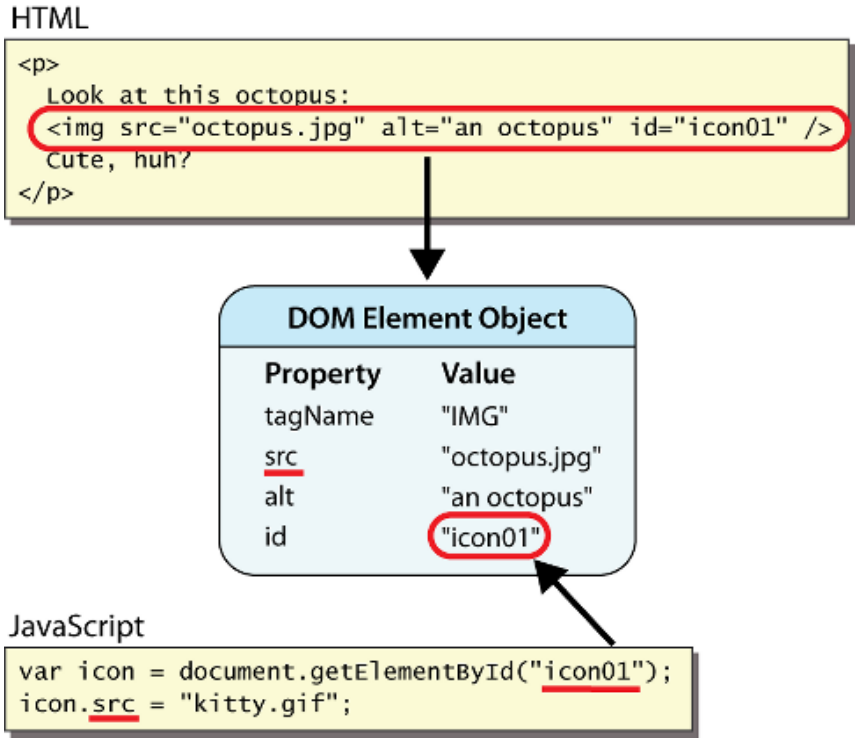
- most JS code manipulates elements on an HTML page
- we can examine elements' state
  - e.g. see whether a box is checked
- we can change state
  - e.g. insert some new text into a div
- we can change styles
  - e.g. make a paragraph red



DOM element objects

- every element on the page has a corresponding DOM object

- access/modify the attributes of the DOM object with *objectName.attributeName*



Accessing elements: document.getElementById

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

<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;
}
```

can change the text inside most elements by setting the innerHTML property
See examples: See example: lecture05_examples/click2.html
lecture05_examples/click1.html

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.