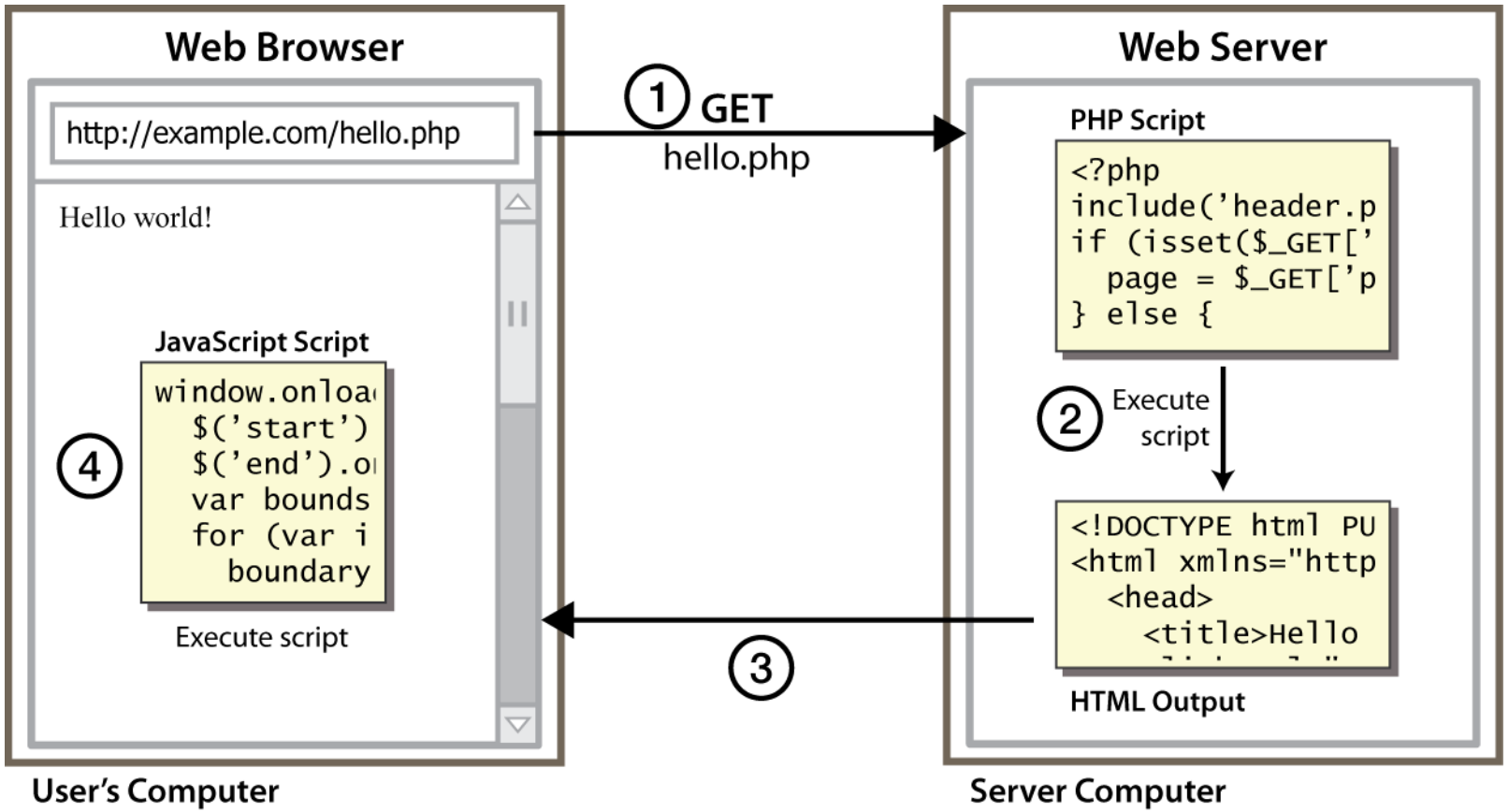
Lecture 5: JavaScript for Modern Web Apps

Making web pages interactive

- In this lecture we will learn about JavaScript, language that is used to make our web pages interactive. It is default scripting language to all modern browsers. As we learn, we find JavaScript syntax to be similar to languages we already know like C and Java, but it behaves drastically different in some cases, which makes JavaScript the most misunderstood programming language. ***Science of consciousness:*** As our consciousness grows, we become able to make discriminations between things that look similar on the surface level.

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
  - React to events (ex: page load, user clicks, timer events, ajax events)
  - Perform DOM manipulations (style, value, structure)
  - Perform client side validations
  - Perform calculations (execute business logics) on browser
- A web standard (but not supported identically by all browsers)
- NOT related to Java other than by name and some syntactic similarities
  - Java was most popular language at time it was developed
  - international specification: ECMAScript

JavaScript vs. Java

- **Interpreted**, not compiled
- More relaxed syntax and rules
  - Fewer and "looser" data types
  - Variables don't need to be declared
  - Errors often silent (few exceptions)
- Key construct is the **function** rather than the class
  - "First-Class" functions are used in many situations
  - Functional programming language, if used properly
- OO language that does not have classes
- Contained within a web page and integrates with its HTML/CSS content
- Powerful and lightweight
- Many pitfalls **Alert!**
  - JavaScript: The Good Parts

First JavaScript Statement: `alert`

```
alert("message");
```

- a JS command that pops up a dialog box with a message
  - e.g. `alert("Hello, world!");`

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` from ES6
- Types are not specified, but JS does have types ("loosely typed")
  - can find out a variable's type by calling `typeof(variable_name)`;

Primitive types

Primitive type: is a type that's represented as a single value (not object)

- **`undefined`**
- **`null`**
- **`boolean`**: truthy or falsy
- **`number`**: float, int
- **`string`** (with single or double quotation)
- **`symbol`** (ES6) immutable type for objects (object wrapper)

Types are dynamic and JS Engine will automatically change between types or wrap a primitive type when needed (coercion)

```
var a = 1 + '2' // '12' concatenation in JS with +
```

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

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` is property (not a method as in Java)

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!"
let n1 = parseInt("42 is the answer");      // 42
let n2 = parseFloat("booyah");              // NaN
```

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

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

Template String (ES6)

- Template literals are strings literals allowing embedded expressions.
  - You can use multi-line strings and string interpolation features with them.

```
`string text`

`string text line 1
 string text line 2`

`string text ${expression} string text`
```

Boolean type

Any value can be used as a boolean

- "**falsey**" values: 0, 0.0, NaN, "", null, and undefined
- "**truthy**" values: anything else

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

- 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); }

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

Special values: `null` and `undefined`

```
let a = null;
const b = 9;
let c; // c is undefined
```

`undefined` : has been declared, but no value assigned

- Declare vars without giving them a value

`null` : var exists, and was explicitly 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

JS Syntax - (almost same as Java)

`for` loop , **`if/else`** statement, **`while`** loops

```
for (initialization; condition; update) {
        statements;
}
if (condition) {
        statements;
} else if (condition) {
        statements; }
else {
        statements;
}
while (condition) {
        statements;
}
do { statements; } while (condition);
```

Comments (same as Java)

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

Arrays

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

```
let 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
```

- 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

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

const b = a.map(function (elem, i, array) {
   return elem + 3;
})// [4,6,8,6,6]

const c = a.filter(function (elem, i, array) {
   return elem !== 3;
});//[1,5]
```

```
// find first element satisfying condition
const d = a.find(function (elem) {
   return elem > 1;
}); //3

// find  index of first element satisfying condition
const e = a.findIndex(function (elem) {
   return elem > 1;
}); //1

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

Using for of with Array (Iterable) – ES6

```
var things = ['a', 'b', 'c'];

for (const alphabet of things) {
        console.log(alphabet);
}

// a
// b
// c
```


```
var things = ['a', 'b', 'c'];

for (const alphabet of things.entries()) {
        console.log(alphabet);
}

// [0, 'a']
// [1, 'b']
// [2, 'c']
```

```
const alphabet = things.entries(); // ArrayIterator Object
const letter = alphabet.next();
// letter.done = false
// letter.value[0] = 0
// letter.value[1] = "a"
```

Splitting strings: `split` and `join`

```
var s = "the quick brown fox";
var a = s.split(" ");              // ["the", "quick", "brown", "fox"]
a.reverse();                       // ["fox", "brown", "quick", "the"]
s = a.join("!");                   // "fox!brown!quick!the"
```

- `split` breaks apart a string into an array using a delimiter
- Can also be used with **regular expressions** surrounded by /

```
var a = s.split(/[ \t]+/);
```

- `join` merges an array into a single string, placing a delimiter between them

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 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

- 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)

First-class functions

Functions can be assigned to variables

```
const myfunc = function(a, b) {
 return a * b;
};
```

Functions can be passed as parameters

```
function apply(a, b, myfunc) {
 const y = myfunc(a, b);
 return y;
}
const x = apply(2, 3, myfunc); // 6
```

Functions can be return values

```
function getAlert(str) {
 return function() { alert(str); }
}
const whatsUpAlert= getAlert("What's up!");
whatsUpAlert(); // "What's up!"
```

No Overloading!

```
function log(){
	console.log("No Arguments");
}

function log(x){
	console.log("1 Argument: " + x);
}

function log(x, y){
	console.log("2 Arguments: " + x + ", " + y);
}

log(); // 2 Arguments: undefined, undefined
log(5); // 2 Arguments: 5, undefined
log(5, 10); // 2 Arguments: 5, 10
```

Flexible function Arguments

- JavaScript function calls are flexible and can be called with any number of parameters.
  - If more than expected, will simply be neglected.
  - If less than expected, rest will be undefined.

```
function a(x) {
        console.log(x);
}

a(); // undefined
a(5); // 5
a(5, 10); // 5
```

Flexible function Arguments

- But, flexibility might cost you in some case

```
function display(products){
    let count = products.length; //  Cannot read property 'length' of undefined
    ...
}
display();
```

- One solution is to check of presence of arguments as the first thing in the function but it does not scale well.

```
function display(products){
    let prods = typeof products !== 'undefined'? products: [];
}
```

Default parameters (ES6)

```
function log(x=10, y=5){
        console.log( x + ", " + y);
}

log(); // 10, 5
log(5); // 5, 5
log(5, 10); // 5, 10
```

- Scalable solution for the example on previous slide

```
function display(products=[]){
    let size = products.length;
}

display();
```

Rest operator (ES6)

- A **Rest** syntax allows us to represent variable number of arguments as an Array.
  - Its like `varargs` in Java and has same syntax.
  - Rest parameters should be the last parameter in a function.

```
function sum(x,y, ...more){
        var total = x + y;
        if(more.length > 0){
                for (var i=0; i<more.length; i++) {
                        total += more[i];
                }
        }
        console.log(total);
}

sum(4,4); // 8
sum(4,4,4); // 12
```

Spread operator (ES6)

- The spread syntax allows an expression to be expanded in places where multiple arguments (for function calls) or multiple elements (for array literals) or multiple variables (for destructuring assignments) are expected.

```
function listProducts(...products){
    for(let p of products){
        console.log(p);
    }
}

let products = ["apple", "ball"];
listProducts(...products);
```

Rest

Spread

- Syntax is same as for rest parameters, only use cases are different.

JavaScript Object Literal

A collection of name/value pairs

```
var address = {
        street: 'Main Street',
        "zip code": 52556,
        apartment: {
                floor: 3,
                number: 301
        }
    }
```

- Notice the **object literal** above, some names are with quotations!
- To access an object property: `address.street` or `address['street']`
- There is no difference in JS between single or double quotation!

Using `for in` with Objects/Arrays

```
var things = {'a': 97, 'b': 98, 'c': 99 };
for (const key in things) {
        console.log(key + ', ' + things[key]);
}
// a, 97
// b, 98
// c, 99


var things = ['a', 'b', 'c'];
for (const key in things) {
        console.log(key + ', ' + things[key]);
}
// 0, a
// 1, b
// 2, c
```

Summary ‘for’ loops

- ‘for’ is the basic for loop in JavaScript for looping
  - Almost exactly like Java for loop
  - If not sure what need, use this
- ‘for in’ is useful for iterating through the properties of objects, and can also be used to go through the indices of an array
- ‘for of’ is a new convenience (ES6) method for looping through values of ‘iterable’ collections (e.g., Array, Map, Set, String)
- ‘forEach’ is a another convenience method that executes a provided function once for each Array element.
  - forEach returns undefined rather than a new array
  - Intended use is for side effects, e.g., writing to output, etc.
- Best practice to use convenience methods when possible
  - Avoids bugs associated with indices at end points
  - map, filter, find, reduce best practice when appropriate

Destructuring Arrays and Objects (ES6)

- The destructuring assignment syntax is a JavaScript expression that makes it possible to extract data form arrays and objects into distinct variables.

```
// Destructuring Array
const details = ['Jim', 123, 'mum.edu'];
const [name, id, website] = details;

// Destructuring Object
const settings = { width: 300, color: 'black' }
const { width, color} = settings;

// Swap variables
var color1 = 'Green';
var color2 = 'red';
[color1, color2] = [color2, color1];

// Object Destructuring with variable renaming & default values
const { w: width, h: height} = { w: 800, h: 300}
```

Value types vs Reference types

```
// value (primitives)
let a = 1;
let b;

b = a;
a = 2;

console.log(a); // 2
console.log(b); // 1
```

```
// reference (all objects)
const a = { fname: 'George' };
let b;

b = a;
a.fname = 'Mike';
console.log(a.fname); // Mike
console.log(b.fname); // Mike

b.fname = 'Jim';
console.log(a.fname); // Jim
console.log(b.fname); // Jim

a = { fname: 'George' };
console.log(a.fname); // George
console.log(b.fname); // Jim
```

= operator always sets a new memory space

Semi-colon ;

Semi-colons in JS are optional, JS automatically add them to our code

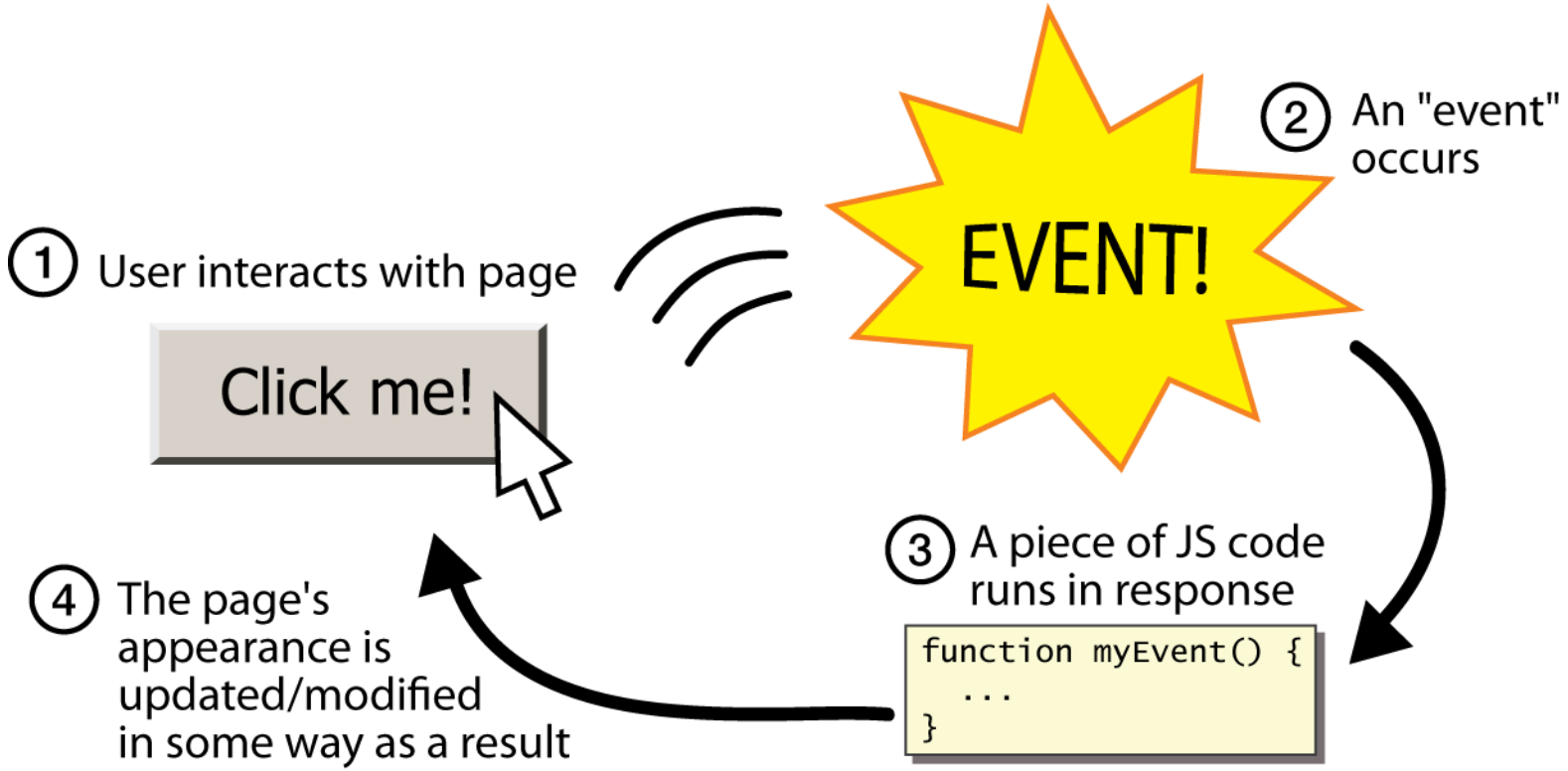
```
function a(){
        return {
                name: 'George'
        }
} // return an object

function a(){
        return
        {
                name: 'George'
        }
}  // return void, why?
```

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:*** 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



Event handlers (Obtrusive)

- A JavaScript can be executed when an event occurs, like when user clicks on an HTML element.
- To execute code when user clicks on an element, add JavaScript code to an HTML event attribute:

```
<button onclick="sayHi()">Say Hi!</button>
```

```
<script>
    function sayHi(){
        alert('Hi!');
    }
</script>
```

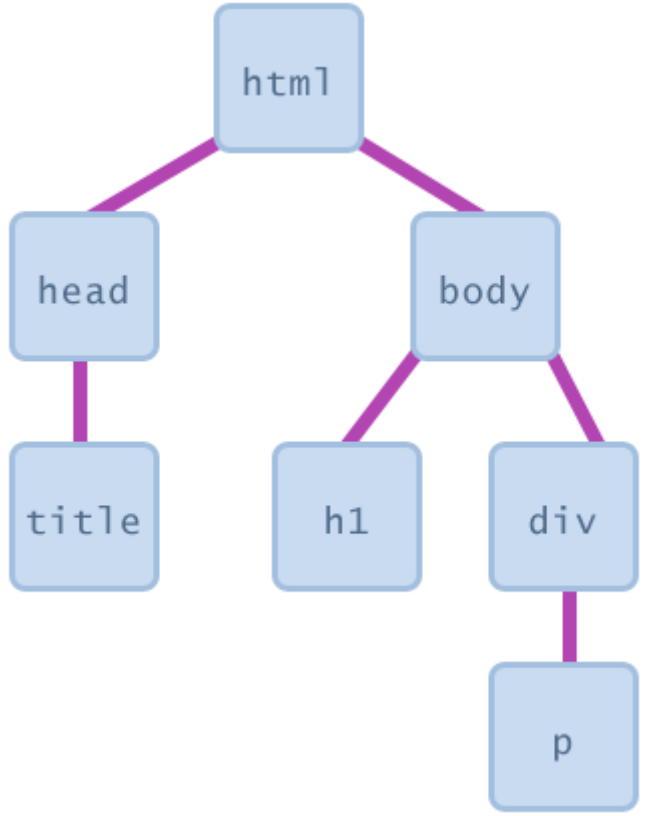
Main Point

- 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.

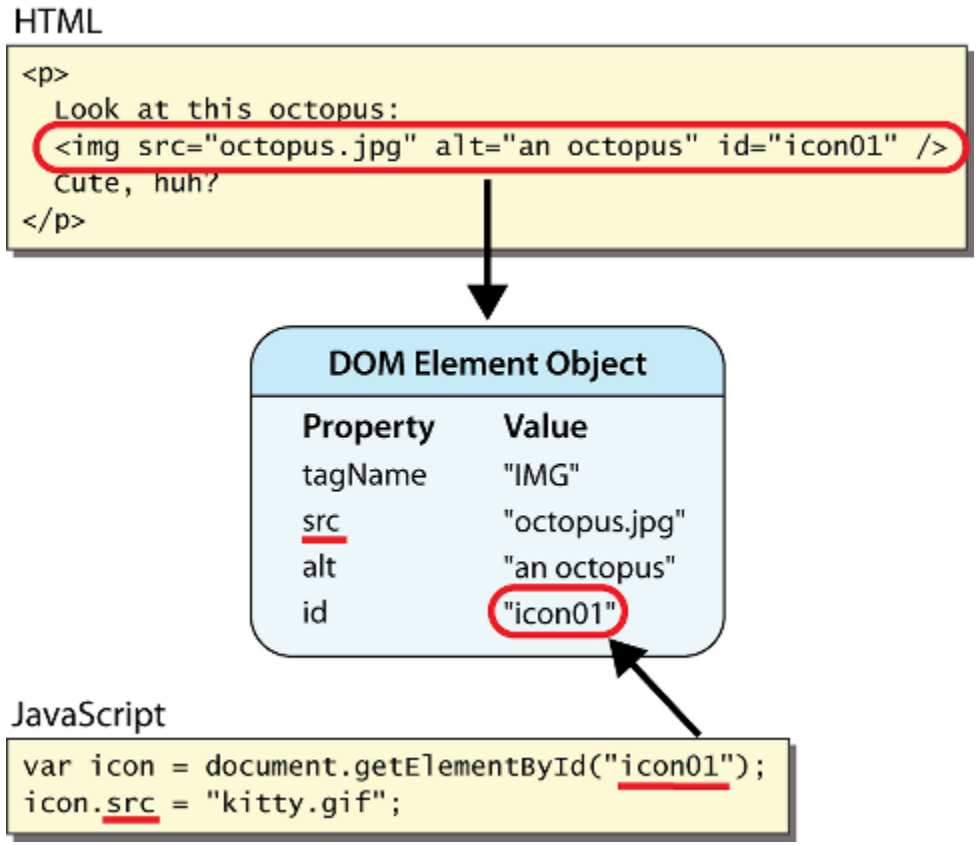
Document Object Model (DOM)

- All HTML elements are represented in browsers as objects
- All objects are nested together in one tree (DOM tree)
- Elements can have parents, siblings and children
- Most JS code manipulates elements (objects) on the DOM
  - it can examine elements' state (see whether a box is checked)
  - it can change state (insert some new text into a div)
  - it can change styles (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.propertyName`**



Accessing DOM Elements

```
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 access text in most *form controls* by using `value` property
- Browser automatically updates the screen when any DOM object is changed

Value vs. innerHTML

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

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

- Can access text inside non-form elements by via. `innerHTML` property

Adjusting styles with the DOM

- To change a DOM element style we use **`style`** property which allows you to set any CSS style property
- It contains same properties as in CSS, but with **camelCasedNames**
  - examples: **`backgroundColor`**, **`borderLeftWidth`**, **`fontFamily`**

```
element.style.fontSize="14pt";
```

Common DOM styling errors

Many students forget to write .style when setting styles

```
var clickMe = document.getElementById("clickme");
clickMe.color = "red";
clickMe.style.color = "red";
```

style properties are capitalized as camelCasedNames

```
clickMe.style.font-size = "14pt";
clickMe.style.fontSize = "14pt";
```

style properties must be set as strings, often with units at the end

```
clickMe.style.width = 200;
clickMe.style.width = "200px";
clickMe.style.padding = "0.5em";
```

The below example computes "200px" + 100 + "px" which would evaluate to "200px100px"

```
var top = document.getElementById("main").style.top;
top = top + 100 + "px";
top = parseInt(top) + 100 + "px";
```

DOM manipulation (snippets)

```
const textbox = document.getElementById("MyText");
textbox.value = "Hello!";
textbox.disabled = true;

const checkbox = document.getElementById("MyCheckBox");
if (checkbox.checked) { … }


const span = document.getElementById("output2");
span.innerHTML = "Hello!";
span.className = "highlight";
span.style.fontFamily = "Tahoma";  // Style properties in camelCasedNames


const img = document.getElementById("logo");
img.src = "cat.png";
```

Other DOM selection APIs

- `getElementsByName("name")`
  - Get all the elements with the specified name
- getElementsByTagName("tag")
  - Get all the elements in the document with the specified tag name
- querySelector("selector")
  - Get the first element in the document that matches the specified CSS selector(s) in the document.
- querySelectorAll("selector")
  - Returns all the elements in the document that matches a specified CSS selector(s)

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.

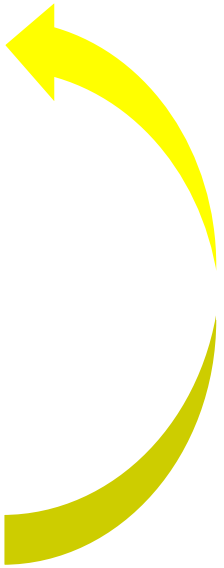
- **Science of Consciousness:** The purpose of thought is to produce successful actions and achievements in the world, and more powerful thoughts will produce more successful actions.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

JavaScript for Modern Web Apps: Spontaneous Right Action

1. JavaScript code loads along with an HTML page and is executed on the browser.

2. JavaScript reacts to browser events and manipulates the web page using the HTML DOM API.



---

3. **Transcendental consciousness** is the source of thought and the home of all the laws of nature.

4. **Impulses within the transcendental field:** Actions arising from this level will spontaneously be in accord with all the laws of nature.

5. **Wholeness moving within itself:** In unity consciousness, all of one's perceptions and actions are grounded in the experience of pure consciousness.