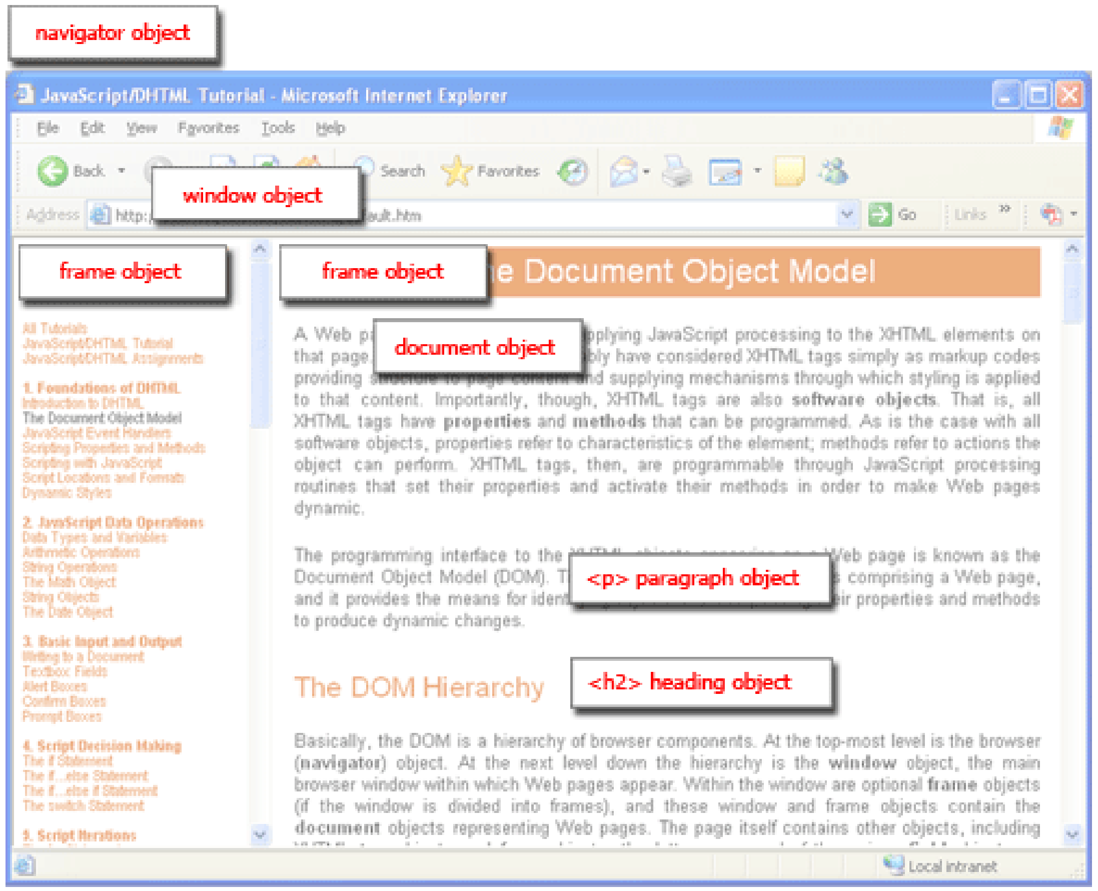
22AIE305: CLOUD COMPUTING

Document Object Model (DOM)

- HTML page is structured data
- DOM provides representation of this hierarchy
- Examples
  - Properties: document.alinkColor, document.URL, document.forms[ ], document.links[ ], document.anchors[ ], …
  - Methods: document.write(document.referrer)
    - These change the content of the page!
- Also Browser Object Model (BOM)
  - Window, Document, Frames[], History, Location, Navigator (type and version of browser)

Browser and Document Structure



All Tutorials
JavaScript/DHTML Tutorial
JavaScript/DHTML Assignments

1. Foundations of DHTML
Introduction to DHTML
The Document Object Model
JavaScript Event Handlers
Scripting Properties and Methods
Scripting with JavaScript
Script Locations and Formats
Dynamic Styles

2. JavaScript Data Operations
Data Types and Variables
Arithmetic Operations
String Operations
The Math Object
String Objects
The Date Object

3. Basic Input and Output
Writing to a Document
Textbox Fields
Alert Boxes
Confirm Boxes
Prompt Boxes

4. Script Decision Making
The If Statement
The If...else Statement
The If...else If Statement
The switch Statement

5. Script Iterations

e Document Object Model

A Web p pplying JavaScript processing to the XHTML elements on that page. bly have considered XHTML tags simply as markup codes providing nd supplying mechanisms through which styling is applied to that content. Importantly, though, XHTML tags are also **software objects**. That is, all XHTML tags have **properties** and **methods** that can be programmed. As is the case with all software objects, properties refer to characteristics of the element; methods refer to actions the object can perform. XHTML tags, then, are programmable through JavaScript processing routines that set their properties and activate their methods in order to make Web pages dynamic.

The programming interface to the Veb page is known as the Document Object Model (DOM). T s comprising a Web page, and it provides the means for ident ir properties and methods to produce dynamic changes.

The DOM Hierarchy

Basically, the DOM is a hierarchy of browser components. At the top-most level is the browser (**navigator**) object. At the next level down the hierarchy is the **window** object, the main browser window within which Web pages appear. Within the window are optional **frame** objects (if the window is divided into frames), and these window and frame objects contain the **document** objects representing Web pages. The page itself contains other objects, including

W3C standard differs from models supported in existing browsers

Reading Properties with JavaScript

Sample HTML

```
<ul id="t1">
<li> Item 1 </li>
</ul>
```

Sample script

```
1. document.getElementById('t1').nodeName
2. document.getElementById('t1').nodeValue
3. document.getElementById('t1').firstChild.nodeName
4. document.getElementById('t1').firstChild.firstChild.nodeName
5. document.getElementById('t1').firstChild.firstChild.nodeValue
```

- Example 1 returns "ul"
- Example 2 returns "null"
- Example 3 returns "li"
- Example 4 returns "text"
  - A text node below the "li" which holds the actual text data as its value
- Example 5 returns " Item 1 "

Objects

- An object is a collection of named properties
- Think of it as an associative array or hash table
  - Set of name:value pairs
    - objBob = {name: “Bob", grade: 'A', level: 3};
  - Play a role similar to lists in Lisp / Scheme
- New members can be added at any time
    - objBob.fullname = 'Robert';
- Can have methods
- Can refer to this

Object features

- Dynamic lookup
  - Method depends on run-time value of object
- Encapsulation
  - Object contains private data, public operations
- Subtyping
  - Object of one type can be used in place of another
- Inheritance
  - Use implementation of one kind of object to implement another kind of object

Concurrency

- JavaScript itself is single-threaded
  - How can we tell if a language provides concurrency?
- AJAX provides a form of concurrency
  - Create XMLHttpRequest object, set callback function
  - Call request method, which continues asynchronously
  - Reply from remote site executes callback function
    - Event waits in event queue…
  - Closures important for proper execution of callbacks
- Another form of concurrency
  - Use SetTimeout to do cooperative multi-tasking

JavaScript eval

- Evaluate string as code (seen this before?)
  - The eval function evaluates a string of JavaScript code, in scope of the calling code
    - var code = "var a = 1";
    - eval(code); // a is now '1�much
    - var obj = new Object();
    - obj.eval(code); // obj.a is now 1
  - Common use: efficiently deserialize a complicated data structure received over network via XMLHttpRequest
- What does it cost to have eval in the language?
  - Can you do this in C? What would it take to implement?

Three ways to create an object

- You can use an object literal:
  - `var course = { number: "CIT597", teacher="Dr. Dave" }`
- You can use new to create a “blank” object, and add fields to it later:

  -
    ```
    var course = new Object();
    course.number = "CIT597";
    course.teacher = "Dr. Dave";
    ```
- You can write and use a constructor:

  -
    ```
    function Course(n, t) {   // best placed in <head>
        this.number = n;
        this.teacher = t;
    }
    ```
  - `var course = new Course("CIT597", "Dr. Dave");`

Arrays and objects

- Arrays *are* objects
- `car = { myCar: "Saturn",  7: "Mazda" }`
  - `car[7]` is the same as `car.7`
  - `car.myCar` is the same as `car["myCar"]`
- If you *know* the name of a property, you can use dot notation: `car.myCar`
- If you *don't know* the name of a property, but you have it in a variable (or can compute it), you *must* use array notation: `car.["my" + "Car"]`

The with statement

- with (*object*) *statement* ; uses the *object* as the default prefix for variables in the *statement*
- For example, the following are equivalent:

  -
    ```
    with (document.myForm) {
        result.value = compute(myInput.value) ;
    }
    ```

  -
    ```
    document.myForm.result.value =
        compute(document.myForm.myInput.value);
    ```
- One of my books hints at mysterious problems resulting from the use of with, and recommends against ever using it

Functions

- Functions should be defined in the `<head>` of an HTML page, to ensure that they are loaded first
- The syntax for defining a function is:
  `function` *name*(*arg1*, ..., *argN*) { *statements* }
  - The function may contain `return` *value*; statements
  - Any variables declared within the function are local to it
- The syntax for calling a function is just
  name(*arg1*, ..., *argN*)
- Simple parameters are passed *by value,* objects are passed *by reference*

Working with Event Handlers

- Events are controlled in JavaScript using **event handlers** that indicate what actions the browser takes in response to an event.
- Event handlers are created as attributes added to the HTML tags in which the event is triggered.
- The general syntax is:

```
< tag onevent = “JavaScript commands;”>
```

  - ***tag*** is the name of the HTML tag
  - ***onevent*** is the name of the event that occurs within the tag
  - ***JavaScript commands*** are the commands the browser runs in response to the event

No-class objects via constructor functions

```
function MyObject(param1, param2) {
   “use strict”;
   this.property1 = param1; // public attribute
   this.property2 = param2; // public attribute
   this.doSomething = function(…) { // public method
     // function body goes here
   }
}
var x = new MyObject(x,y); // creation
```

- Looks like a regular function
- **Always use a capital letter for the function name**
- No formal attribute declarations
  - Use of “this” automatically creates a public attribute
    - Be careful; typos may introduce unwanted attributes

Making attributes and methods private

```
function MyObject(param1, param2) {
   "use strict";
   var property1 = param1; // private attribute
   this.property2 = param2; // public attribute
   this.doSomething1 = function(…) { // public method
     // function body goes here
   }
   var doSomething2 = function(…) { // private method
     // function body goes here
   }
}
var x = new MyObject(x,y); // creation
```

Literal objects

```
var roscoe = {
  firstName: “Roscoe”, // public attr
  lastName: “Raider”, // public attr
  getFullname: function(){ // public method
     return this.firstName +
     this.lastName;
  };
}
```

- **This is a kind of Singleton for Javascript**

JS6 class approach

```
class MyObject {
   "use strict";
  constructor(param1, param2) {
     var property1 = param1; // private attribute
     this.property2 = param2; // public attribute
     this.doSomething1 = function(…) { // public method
        // function body goes here
  } // end constructor

  var doSomething2 = function(…) { // private method
     // function body goes here
   }
} // end class
var x = new MyObject(x,y); // creation (same as JS5)
```

Using a Javascript constructor to create objects (same approach whether class-based or not)

```
var x = new MyObject(“arg1”, “arg2”);
x.setXXX(“arg1b”);
var y = x.getXXX();
```

Always use ‘**`new`**’; otherwise the effect will be to simply call MyObject as a normal function.

- No instance of MyObject would be created.
- The attributes would be added to the “window” object instant

JavaScript “core API” defines only a few native objects – the remainder come from the hosting environment (i.e. the browser)

- String – similar to the Java String class
- Array – generic container/collection class
- Math - like the Java Math class
- Number, Boolean – wrapper classes similar to Java wrapper classes (Integer, Double etc)
  - var x = 123; // x is treated as a Number
  - var y = “123”; // y is treated as a String
  - var z = new Number(“123”); // z is a Number
- Date – represents dates and times