JavaScript

Session 1

What/Why/How is JS?

- JavaScript is an interpreted, **object-based** scripting language modeled after C++.
- **Interpreted languages**: an interpreter is needed to translate a program's statement into machine code. JavaScript Interpreter is built into Web browsers.
- **Object-based** language: most of client-side JavaScript's objects come from Web pages such as images, forms, and form elements.
- **Scripting languages**: are easy to learn, easy to use, excellent for small routines and applications, and developed to serve a particular purpose such as PERL, PHP.
- JavaScript is **loosely typed** language, which means that variables do not need to have a type specified.

Client-Side JavaScript sample

```
<html>
<head>
      <title>JavaScript sample</title>
</head>
<body>
<h1>Client-side JavaScript sample</h1>

<script language="javaScript" type="text/javascript">
      document.write("Hello World! ");
</script>
</body>
</html>
```

Lexical Structure/Character Set

- The **Lexical Structure** of a programming language is the set of **elementary rules** that specifies how you were program in that language.
- JavaScript programs are written using the **Unicode character set**.
- The 7-bit ASCII encoding is useful only for English.
- The 8-bit ISO Latin-1 encoding is useful only for English and major Western European languages.
- The 16-bit Unicode encoding can represent virtually every written language in common use of the planet.
- The **ECMAScript** v3 **standard** allows Unicode characters anywhere in a JavaScript program, version 1 and 2 of the standard allow Unicode characters only in comments and quoted string literals; all elements are restricted to the ASCII character set.

JavaScript is Case Sensitive

- JavaScript is a case-sensitive language:

```
Document.Write("Hello!");          // NOT OK
document.write("Hello!");          // OK
```

- JavaScript ignores “Whitespaces” (spaces, tabs, and newlines) that appear between token in programs.
- JavaScript automatically inserts semicolons before a line break.

```
var mytext = "abc";       // OK
var mytext = "abcd
                          efgh ";      //NOT OK – a line break inserted into a string
```

- JavaScript statements are end with semicolon (;). (optional)
- Omitting semicolon is **not a good** programming **practice**.
- **Single line comment**: any text **between a // and the end of a line is** treated as a comment and is ignored by JavaScript.
- Multiple lines comment: any text between the **characters /* and */ is also** treated as a comment.

JavaScript Syntax

Set of rules for how JavaScript programs are built

JavaScript uses most of the usual instructions and syntax that many programming languages use

Variables, Expressions, Arrays, Objects, Loops, Conditionals, Comparisons, Switches, Functions

Literals/Identifiers

- A literal is a data value that appears directly in a program.
- 12 //the number twelve (number literal)
- 1.2 //the number one point two (number literal)
- "hello world” //a string of text (string literal)
- 'Hi' //another string (string literal)
- true //a Boolean value (Boolean literal)
- null //absence of an object (special value)
- An identifier is simply a name.
- Identifiers are used to name variables and functions, and to provide labels for certain loops in JavaScript code.
- Identifier rules
- The first character must be a letter, an underscore (_).
- Subsequent characters can be a letter, a digit, an underscore.

The <script> tag

- When Netscape introduced JavaScript in Netscape 2, it included support for a new tag: the <script> tag.

  <script language="JavaScript" type="text/javascript">   </script>
- The **<script>** tag has several attributes, two of which you should always set: language and type.
- The language attribute specifies which scripting language you are using.

  language="JavaScript/JavaScript1.1/…"
- The type attribute specifies the MIME type of the text contained within the <script> tag or the file referenced by the <script> tag. **Type="text/javascript"**
- The **src attribute**: you only need to set this attribute when you attach an external JavaScript file.
- An external JS file is just a simple text document with a **.js extension**.
  - Only contain JS statements
  - <script> tags are unnecessary
  - You may use either a relative or an absolute path to indicate the location of your external JS file

Integrating JS into your web documents

- In the <head> of an HTML document
- In the <body> of an HTML document
- Inline with HTML as an event handler
- In an external JavaScript file
- Inline using the javaScript pseudo-protocol

Placing JS statements in the HTML <head> section

- Is the perfect place for any statements that need to be read and **executed before the contents** of your Web document (in the <body> tag) load.
- Good place to **declare user-defined functions** (then call it from the <body> area)
- Good place to **declare global variables**

Placing JS statements in the <body> section

- This is the **best**, and only, place to write statements that actually produce content for the inclusion in an HTML document.
- Calls to functions that declared in the <head>

Custom Greeting

```
<html>
<head><title>Custom Greeting</title>
<script>
        /* Global variable */
        var visitor = prompt("What is your name?", "");
</script>
</head>

<body>
<h1>Custom Greeting</h1>

<script>
document.write("<h1>Welcome, ",  visitor,  "</h1>");
</script>

</body>
</html>
```

Writing JS statements inline as event handlers

- An event handler is one or more JS statements called in response to a particular event.

```
<html>
<head>
    <title>JS inline as event handler</title>
</head>

<body onload="alert('Welcome!');">

    <h1>JS inline with event handler</h1>

</body>
</html>
```

Object Oriented Concepts

- **Object**: is **an item**, thing. It has attributes/properties that describe it. It also **has methods** which are actions that you can perform with the object or on the object
- JavaScript uses **dot notation** to refer to an object and its associated properties and methods.
- For example, pen is an object
- To reference ink color property, we use the name of the object followed by a dot (period) and the name of the property.

```
pen.inkColor
```

- To change the value of an object's property.

```
pen.inkColor = "blue";
```

- To reference the object's method, we reference the name of the object first, followed by a dot and the method name, we end with a set of parentheses. The parentheses are meant to hold any arguments or parameters that provide data to or modify the performance of a method.

```
pen.write("Hello");
```

JavaScript Output

- **console.log(‘some value’)** – Prints to console in browser or terminal
- **window.alert()** – Displays in an alert box in the browser
- **document.write()** – Display within <script> tags in the html
- **innerHTML** – Access an html element using document.getElementById() and output to it

Using the write method

```
<html>
<head><title>Using the write method</title>
</head>
<body>
<h1>Using the write method</h1>

<script language="JavaScript" type="text/javascript">
document.write("Hello World!!");
</script>

</body>
</html>
```

Using the write method to write HTML data

```
<html>
<head>
      <title>Using the write method</title>
</head>
<body>
      <h1>Using the write method to write HTML data</h1>

<script language="JavaScript" type="text/javascript">

      document.write("<h1 style='text-align: center;'>Hello World!!</h1>");

</script>

</body>
</html>
```

The document.write method

- The document’s write method can accept multiple string parameters separated with commas

```
document.write(string1, string2, …)
```

- The following statement would cause an error. You have two choices to fix it: keep the text all on one line or break the string into two strings

```
document.write("long text long text long
               text long text long text");
```

Summary of Object Oriented Concepts

	Description	Real-world example	JavaScript example
Object	An item or thing	pen	document
Properties	An attribute that describes an object	pen.inkColor	document.bgColor
Method	An action that can be performed with or on an object	pen.write()	document.write()

Where does JS Objects come from

- Built into the language, like Math, String, Date and Array (Core JS)
- Come from Web documents and are made available to Client-Side JS **via the Document Object Model (DOM)**
- Come from the browser, such as navigator, location, and history objects also made available to Client-Side JavaScript by the DOM
- Programmers create our own custom objects

Variables

Variables are used to store values

JavaScript uses the **“var”** keyword to declare variables and an equal sign to assign values

```
var x;
x = 100;
```

Same as

```
var x = 100;
```

- Variables ARE case sensitive
- Variables can contain letters, numbers, underscores (_) and dollar signs ($)
- Variables MUST begin with a letter, underscore(_) or dollar sign ($)

Expressions

- An expression is a combination of values, variables and operators which computes a value

2 * 5
x * 5
“Hello”+ “ ” + “World”

Arrays

- JavaScript arrays allow us to store multiple values in a single variable

```
var names = [‘Bob’, ‘Jim’, ‘Jose’, ‘Paula’];

console.log(names[0]);
```

Loops

Execute a block of code as long as a condition is true and repeat

For Loop:

```
for(i = 0; Ii < 10;i++){
        console.log(i)
}
```

While Loop:

```
while(i < 10){
        console.log(i);
        i++’
}
```

Objects

Almost everything in JS can be considered an “object”.
Objects have properties and methods (functions)

```
// Assign Properties
var person = {name:”Mike”, age:33, hairColor:”Brown”}
// Access Property
console.log(person.name);
// Accessing Method
person.getName()
```

Basic Object Features

- Use a function to construct an object
    - function car(make, model, year) {
       this.make = make;
       this.model = model;
       this.year = year; }
- Objects have prototypes, can be changed
    - var c = new car(“Ford”,”Taurus”,1988);
    - car.prototype.print = function () {
     return this.year + “ “ + this.make + “ “ + this.model;}
    - c.print();

Conditionals / If Statements

Runs a block of code if something is true

```
var x = 10;
if(x > 5){
		console.log('Yes');
}
```

```
if(x > 5){
		console.log('Yes');
} else {
		console.log('No');
}
```

Functions

- Functions are objects with method called "( )"
  - A property of an object may be a function (=method)
    - function max(x,y) { if (x>y) return x; else return y;};
    - max.description = "return the maximum of two arguments";
  - Local declarations may appear in function body
- Call can supply any number of arguments
  - functionname.length : # of arguments in definition
  - functionname.arguments.length : # arguments in call
  - Basic types are passed by value, objects by reference
- "Anonymous" functions
  - (function (x,y) {return x+y}) (2,3);

Examples of Functions

- Curried functions
  - function CurriedAdd(x) { return function(y){ return x+y} };
  - g = CurriedAdd(2);
  - g(3)
- Variable number of arguments
  - function sumAll() {

    ```
    var total=0;
    for (var i=0; i< sumAll.arguments.length; i++)
        total+=sumAll.arguments[i];
    return(total); }
    ```
  - sumAll(3,5,3,5,3,2,6)

Anonymous Functions

- Anonymous functions very useful for callbacks
  - `setTimeout(function() { alert("done"); }, 10000)`
  - Evaluation of `alert("done")` delayed until function call
- Simulate blocks by function definition and call
  - `var u = { a:1, b:2 }`
  - `var v = { a:3, b:4 }`

  -
    ```
    (function (x,y) {
        var tempA = x.a; var tempB =x.b;  // local variables
        x.a=y.a; x.b=y.b;
        y.a=tempA; y.b=tempB
    }) (u,v)   // Works because objs are passed by ref
    ```

How to run JavaScript?

JavaScript in URL (JavaScript Pseudo-Protocol)

- copy the following JS code and paste it in the URL, then hit the enter key.

```
javascript: var now=new Date(); document.write("<h1> the time is: </h1>" + now);
```

JavaScript inside the <body> section

- Directly code the JS inside the body element. (Normally use the document.write() function to display the output)

- Inside the <body> element:

```
<script language="JavaScript" type="text/javascript">
var a = 10;
document.write("Resut is: " + a);
</script>
```

Call a JavaScript function

- From the body element, directly call a function defined inside the head element. (Normally use the alert() function to display output)

- Inside the head element: (defines the function)

```
<script language="JavaScript" type="text/javascript">
function f1() {
var a = 10;
alert("Result is " + a);
}
</script>
```

- Inside the body element: (call the function)

```
<a href="javascript: f1();"> Click to run JS</a>
```

Exercise: Greeting card

```
<head>
<script language="JavaScript" type="text/javascript">
            var YourName = prompt("May I know your name?", "");
</script>
</head>
<body>
<section>
<h1>Custom Greeting</h1>
<script language="JavaScript" type="text/javascript">
document.write("<p style='text-align: center;'>Hello, ", YourName);
</script>
</section>
<footer>
```

DOM Model

Session 2

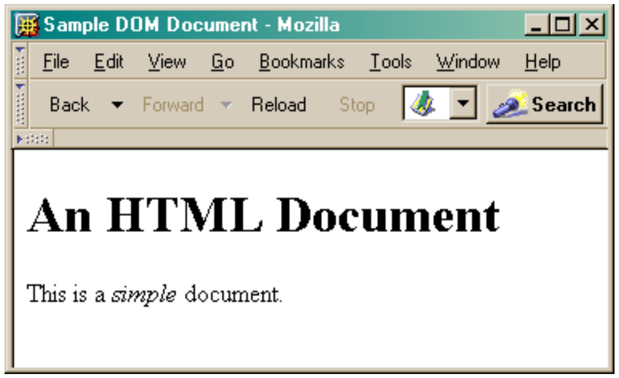
What is the DOM?

- Document Object Model
  - Your web browser builds a *model* of the web page (the *document*) that includes all the *objects* in the page (tags, text, etc)
  - All of the properties, methods, and events available to the web developer for manipulating and creating web pages are organized into objects
  - Those objects are accessible via scripting languages in modern web browsers

This is what the browser reads (sampleDOM.html).

```
<html>
  <head>
    <title>Sample DOM Document</title>
  </head>
  <body>
    <h1>An HTML Document</h1>
    <p>This is a <i>simple</i> document.
  </body>
</html>
```

This is what the browser displays on screen.


Sample DOM Document - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop Search

An HTML Document

This is a *simple* document.

This is a drawing of the model that the browser is working with for the page.

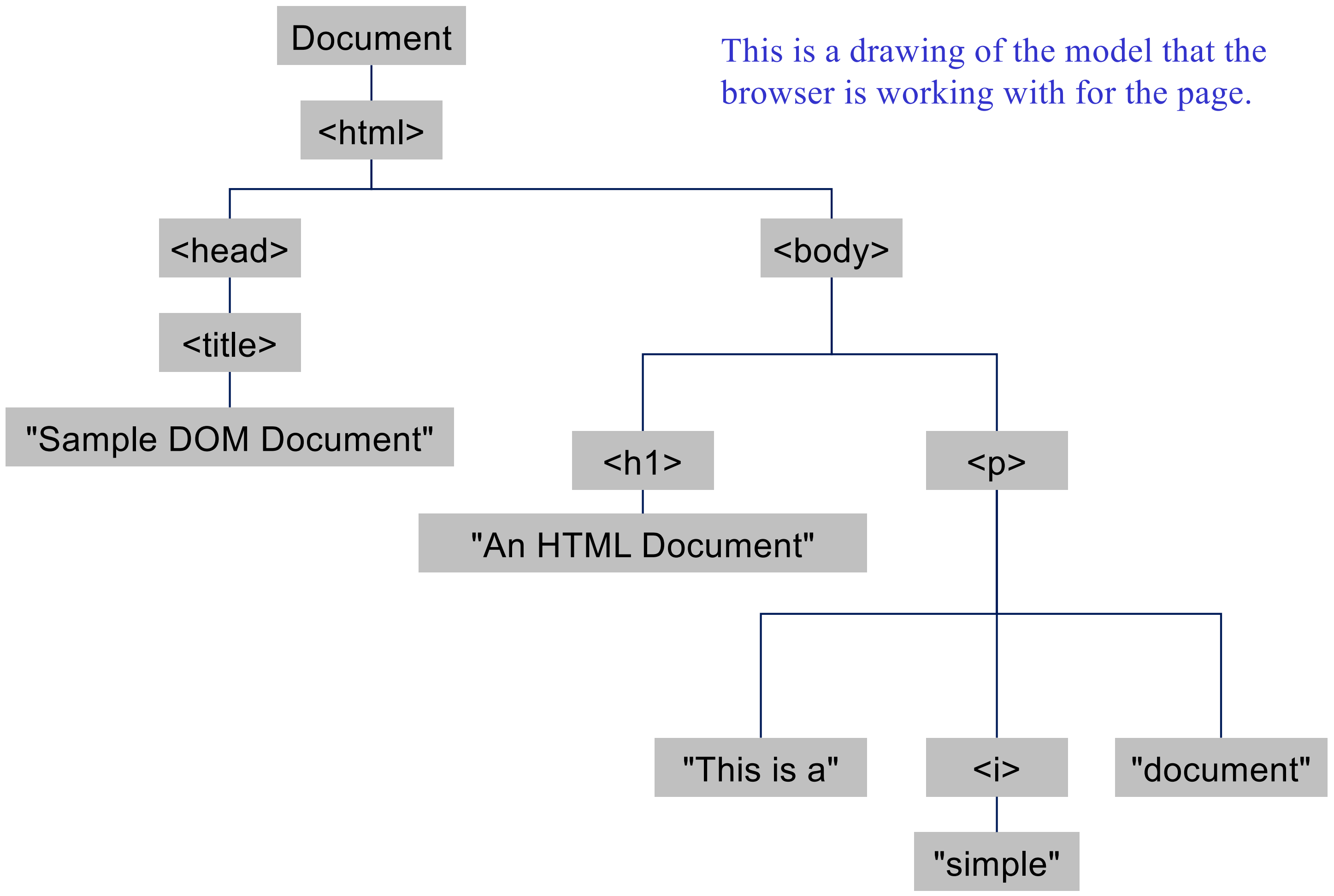


Figure 17-1. The tree representation of an HTML document
Copied from JavaScript by Flanagan.

Why is this useful?

- Because we can access the model too!
  - the model is made available to scripts running in the browser, not just the browser itself
    - A script can find things out about the state of the page
    - A script can change things in response to events, including user requests
  - We have already used this capability in the GUI programming that we've done

Recall our simple example

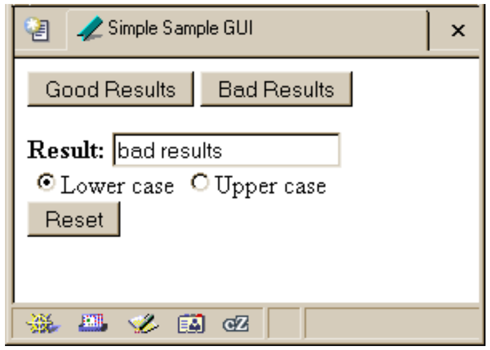
This has several simple controls.

Two buttons to control the results

One text field to display the results

One pair of radio buttons to control the display

One button to reinitialize



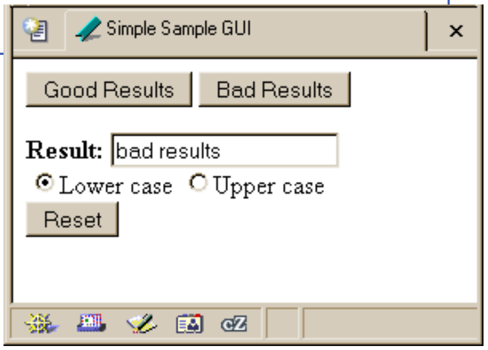
Result: bad results

(●) Lower case ( ) Upper case

http://www.cs.washington.edu/education/courses/100/04au/slides/16-dom/gui.html

setResults(resultString)

```
<script type="text/javascript">
function setResults(resultString) {
  var tempString = resultString;
  if (document.getElementById("radioLC").checked) {
    tempString = tempString.toLowerCase();
  } else if (document.getElementById("radioUC").checked) {
    tempString = tempString.toUpperCase();
  }
  document.getElementById("resultField").value = tempString;
}
</script>
```



Result: bad results
[✓ Lower case] Upper case

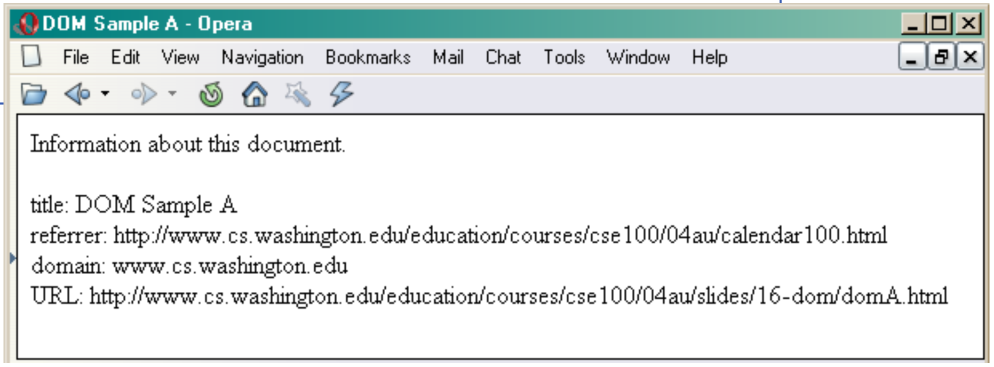
the highlighted script above makes reference to several objects in the document object model

`document.getElementById("radioLC").checked`

- Reference to several nodes in the model of the page that the browser constructed
- **`document`**
  - The root of the tree is an object of type `HTMLDocument`
  - Using the global variable `document`, we can access all the nodes in the tree, as well as useful functions and other global information
    - title, referrer, domain, URL, body, images, links, forms, ...
    - open, write, close, getElementById, ...

Some information from a document

```
<html>
  <head>
    <title>DOM Sample A</title>
  </head>
  <body>
    Information about this document.<br>
    <script type="text/javascript">
    document.write("<br>Title: ",document.title);
    document.write("<br>Referrer: ",document.referrer);
    document.write("<br>Domain: ",document.domain);
    document.write("<br>URL: ",document.URL);
    </script>
  </body>
</html>
```


DOM Sample A - Opera
File Edit View Navigation Bookmarks Mail Chat Tools Window Help

```
Information about this document.

title: DOM Sample A
referrer: http://www.cs.washington.edu/education/courses/cse100/04au/calendar100.html
domain: www.cs.washington.edu
URL: http://www.cs.washington.edu/education/courses/cse100/04au/slides/16-dom/domA.html
```

`document.getElementById("radioLC").checked`

- `getElementById("radioLC")`
  - This is a predefined function that makes use of the `id` that can be defined for any element in the page
  - An `id` must be unique in the page, so only one element is ever returned by this function
  - The argument to `getElementById` specifies which element is being requested

Some information about elements

```
<html>
  <head>
    <title>DOM Sample B</title>
    <script type="text/javascript">
    function showInfo() {
      var element = document.getElementById("opener");
      var buffer = element.id + " tag is " + element.tagName;
      alert(buffer);
      element = document.getElementById("actionItem");
      buffer = element.id + " tag is " + element.tagName;
      buffer += ", type is "+element.type;
      alert(buffer);
    }
    </script>
  </head>
  <body>
    <p id="opener">The id attribute is very helpful.</p>
    <p id="closer">This is the closing paragraph.</p>
    <form>
    <button id="actionItem" type="button" onclick="showInfo()">Show Info</button>
    </form>
  </body>
</html>
```

`document.getElementById("radioLC").checked`

- **`checked`**
  - This is a particular property of the node we are looking at, in this case, a radio button
  - Each type of node has its own set of properties
    - for radio button: `checked, name, ...`
    - refer to the HTML DOM for specifics for each element type
  - Some properties can be both read and set

Some specific properties

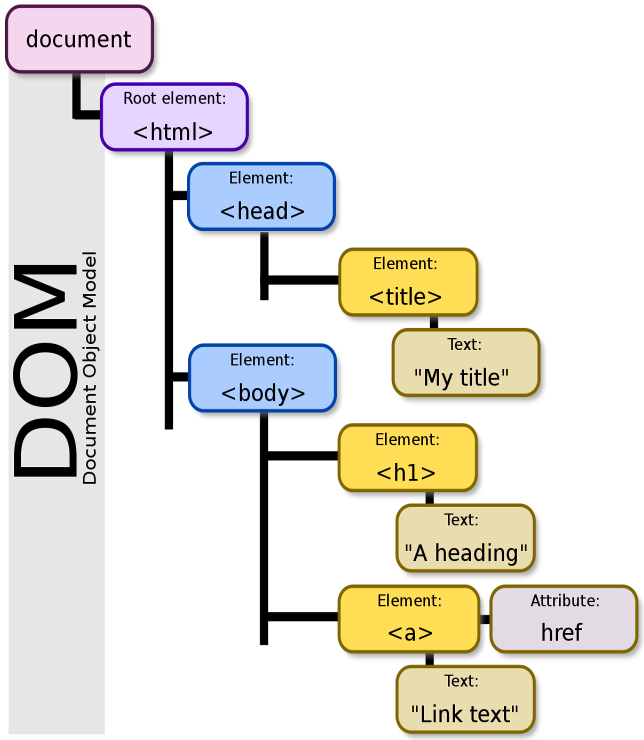
```
<head>
<title>Simple Sample GUI</title>
<script type="text/javascript">
function setResults(resultString) {
  var tempString = resultString;
  if (document.getElementById("radioLC").checked) {
    tempString = tempString.toLowerCase();
  } else if (document.getElementById("radioUC").checked) {
    tempString = tempString.toUpperCase();
  }
  document.getElementById("resultField").value = tempString;
}
</script>
</head>
```

Just the tip of the DOM

- The HTML Document Object Model is a standard for structuring data on a web page
  - The field is advancing rapidly as people recognize the benefits of standardized structure and access
  - The DOM is steadily improving to cover general purpose data structuring requirements
- XML (Extendible Markup Language) also uses the Core DOM to specify its structured data
  - similar to HTML but more carefully defined

Getting vs. Setting

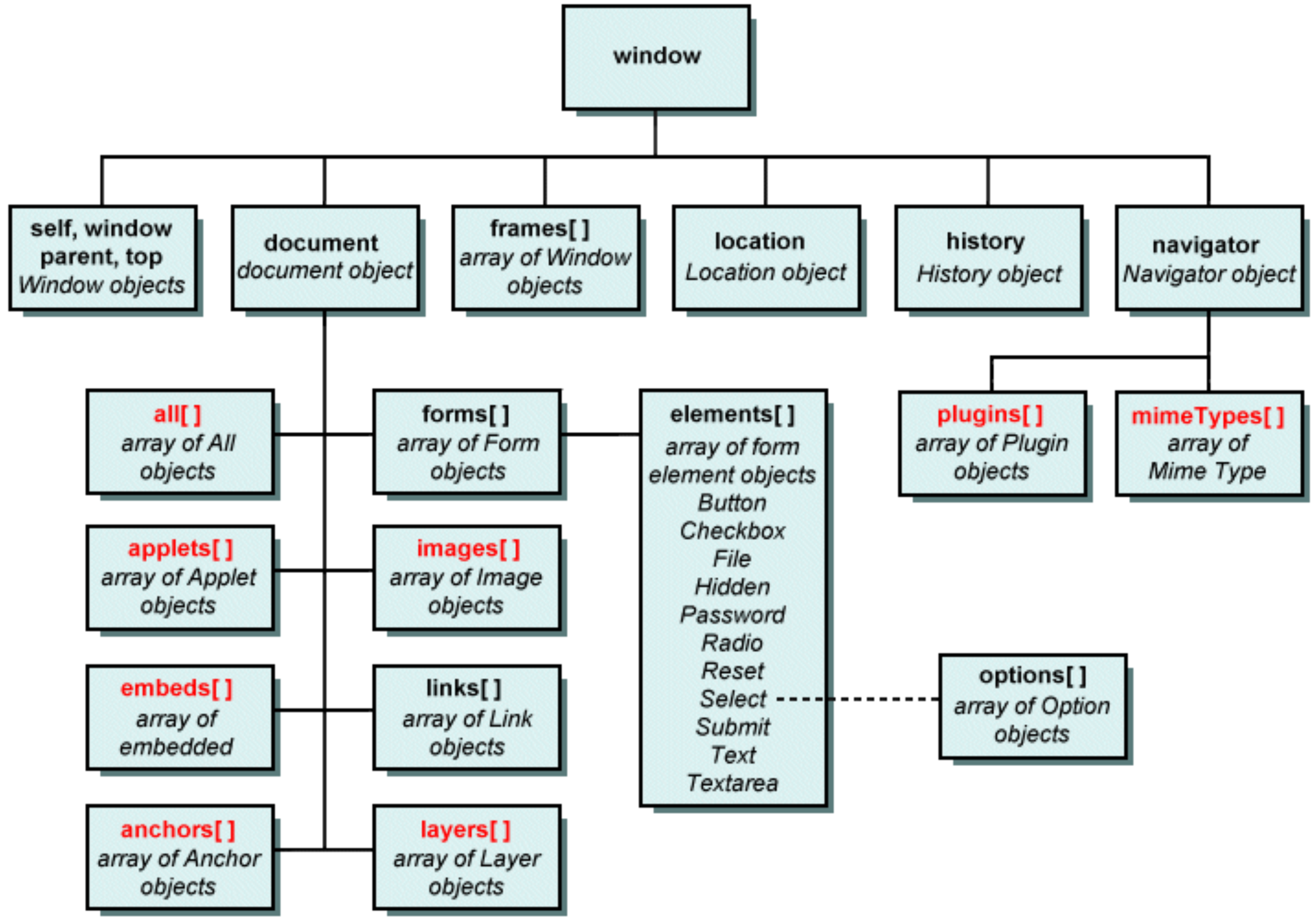
```
var oldvalue = document.getElementById("resultField").value;

document.getElementById("resultField").value = "new value";
```



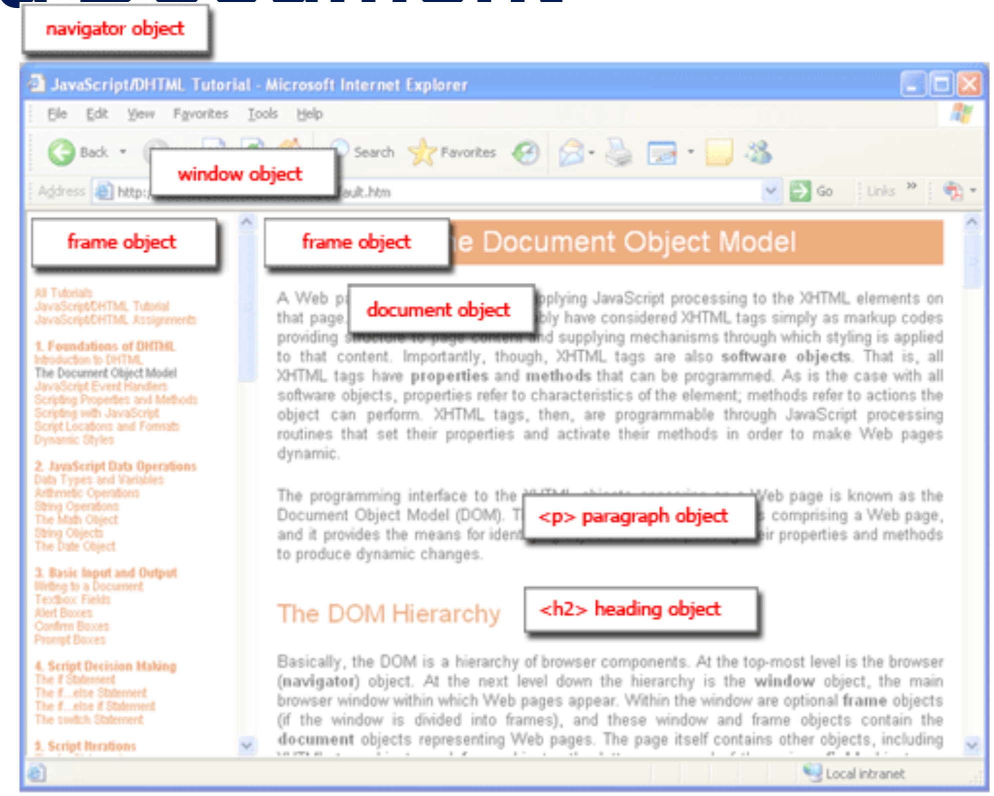
Top DOM tree nodes are directly available as document properties
<html> : document.documentElement
<body>:document.body
<head>: document.head

To target an arbitrary element of the page.
document.getElementById(*id-name*) — Find an element by element id. **document.getElementsByTagName(tag-*name*) —** Find elements by tag name.
document.getElementsByClassName(class-*name*) — Find elements by class name.
Most powerful and commonly used Method is
document.querySelector(CSS-selector) — The most versatile method .it select the first element of the selected CSS selector.
document.querySelectorAll(CSS-selector) — to select all the elements of targeted type.

DOM tree structure:
According to the Document Object Model (DOM), every HTML tag is an object. Nested tags are “children” of the enclosing one. The text inside a tag is an object as well. All these objects are accessible using JavaScript, and we can use them to modify the page.


window
self, window parent, top Window objects
document document object
frames[ ] array of Window objects
location Location object
history History object
navigator Navigator object
all[ ] array of All objects
forms[ ] array of Form objects
elements[ ] array of form element objects Button Checkbox File Hidden Password Radio Reset Select Submit Text Textarea
plugins[ ] array of Plugin objects
mimeTypes[ ] array of Mime Type
applets[ ] array of Applet objects
images[ ] array of Image objects
embeds[ ] array of embedded
links[ ] array of Link objects
options[ ] array of Option objects
anchors[ ] array of Anchor objects
layers[ ] array of Layer objects

Browser and Document Structure



e Document Object Model

A Web pa pplying JavaScript processing to the XHTML elements on that page. bly have considered XHTML tags simply as markup codes providing structure to page content and 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 Web page is known as the Document Object Model (DOM). T comprising a Web page, and it provides the means for iden 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 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
```

Sample HTML

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

- 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 "