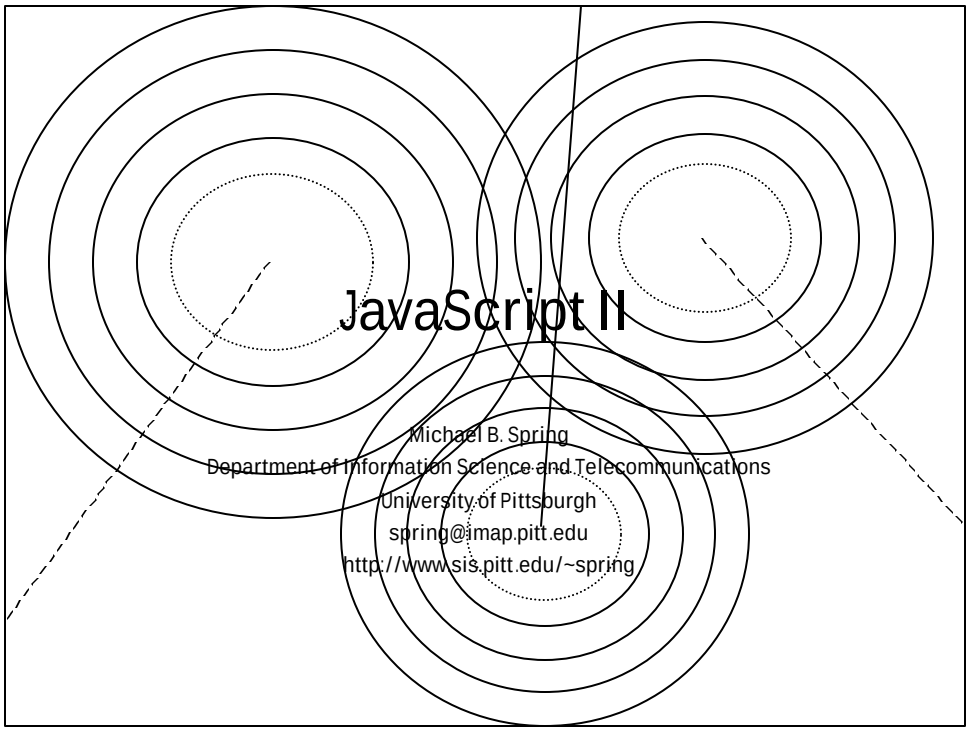
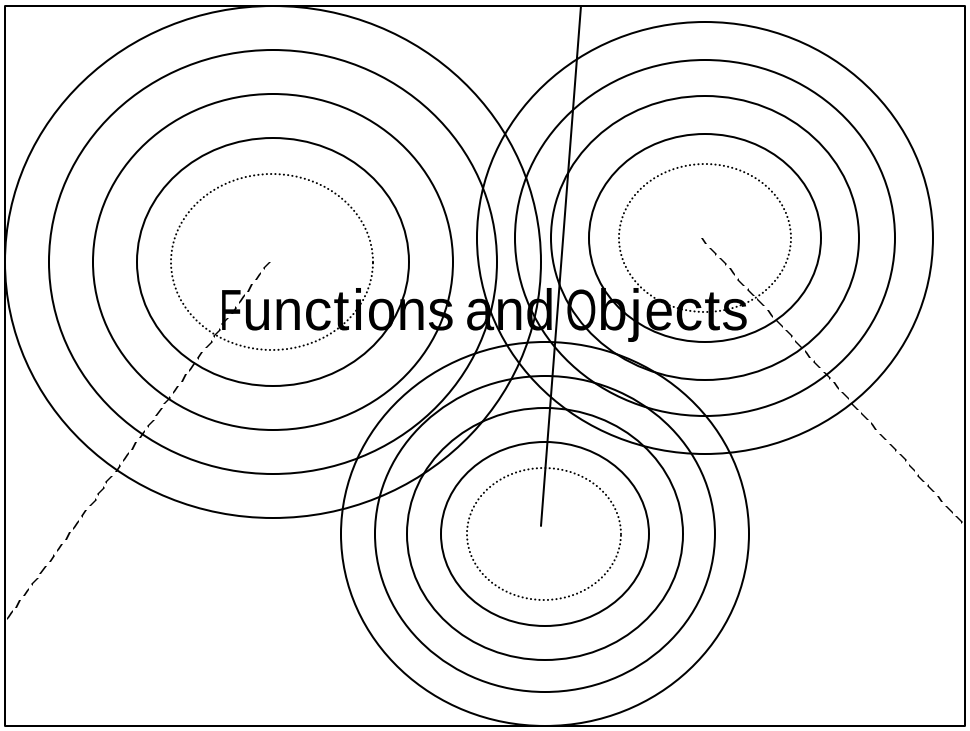


JavaScript II

Michael B. Spring
Department of Information Science and Telecommunications
University of Pittsburgh
spring@imap.pitt.edu
http://www.sis.pitt.edu/~spring

Overview

- Functions
- Objects
- Image Manipulation
- Windows and Dialogs
- Javascript capability
- The Document Object
- Cookies and State
- Security



Functions

- Defined by the “function” keyword followed by:
  - the name of the function
  - a comma-separated list of arguments in parentheses
  - JavaScript statements comprising the body of the function contained in curly braces. “{...}”

```
function print(msg) {
        document.write(msg, “<BR>“);
        }
```

- JavaScript does not check number of arguments
- Functions may or may not return a value.

Functions, Objects, and Arrays

- User defined functions may be assigned to objects as new methods.

```
function square(x) { return x*x; }
o = new Object;
o.sq = square;
y = o.sq(16);  // y now contains 256
```

- Any variable may be equated to a function.

```
a = new Array(10);
a[0] =  square(x);
a[1] = 20;
a[2] = a[0](a[1]);  // a[2] contains 400
```

- Functions may be used like any other data type.


```
// define some functions
function add(x,y)  { return x+y; }
function subtract(x,y)  { return x-y; }
function multiply(x,y)     { return x*y; }
function divide(x,y){ return x/y; }

// define a function that takes one of the above
// functions and invokes it on two operands

function operate(operator, operand1, operand2)
{
       return operator(operand1, operand2);
}

// Could be invoked to calculate (2+3) + (4*5)

var i = operate(add, operate(add, 2,3),
                 operate(multiply, 4,5));
```

```
// store the functions in an associative array
var operators = new Object();
operators[“add”] = add;
operators[“subtract”] = subtract;
operators[“multiply”] = multiply;
operators[“divide”] = divide;

// Create a function to look up an operator by name
// and invokes it on the supplied operands
function operate2(op_name,operand1, operand2){
if(operators[op_name] == null)
       return “undefined operator”
else
       return operators[op_name](operand1, operand2);
}

//  Could also be used with predefined functions

var k = operate2(“pow”, 10, 2);
```

The Function Object

- Functions are automatically converted to objects when used in an object context.
- Remember -- arguments and caller properties are only defined while the function is being executed
- In order to refer to an objects properties from within the object, the object must refer to itself.
  - JavaScript doesn’t support “this”
  - Self reference uses the functions own name.

```
function f() { return f.arguments[0] * f.arguments[1];}
```

More on Functions

- The arguments[] array
  - refers to an array containing the complete set of arguments passed to a function
- The caller property
  - refers to the function that invoked the current one.
  - Cannot be used to inspect the caller or arguments of caller function
- The Function() Constructor
  - defines functions without the function keyword

```
var f = new Function("x", "y", "return x*y;");
```


Event Handler Functions

- Event handler functions are designed to handle events associated with various elements on a web page.
- Not usually invoked by a JavaScript program.
- Generally invoked by the browser itself whenever certain "events" occur.

```
<FORM>
  <INPUT TYPE="submit" VALUE="Click me!"
        onClick="var sum=1+2; alert(sum);">
</FORM>
```

Creating Objects with Constructors

- Objects are created in ad hoc fashion. Simply create a constructor function and invoke it with the new operator
- A constructor is a function that:
  - Is responsible for appropriate initialization. (It is passed a reference to the new "empty" object, this)
  - Does not return a value.

```
// constructor for a rectangle object
function Rectangle(w, h)
{
   this.width = w;
   this.height = h;
}
```


Object Properties

- Each piece of data in an object is a property
  - generally accessed with the '.' operator
  - // Read a property value
  - w = image.width
  - // set a property value
  - window.location = "http://foo.bar.bin/index.html"
- Define a property by setting its value
- Reading a property that doesn't exist returns the special JavaScript value undefined.
- Once defined, a property cannot be undefined

Methods

- A method is a JavaScript function invoked through an object. Methods allow use of this

```
function compute_area()
{
return this.width * this.height
}
```

- Create a new Rectangle object using constructor

```
var rect = new Rectangle(8.5,11);
```

- Define a method by assigning the function as a value

```
rect.area = compute_area;
```

- Invoke the new method
- a=rect.area();

Defining Methods in a Constructor

```
// define some function to be used as methods
function Rectangle_area() { return this.width * this.height; }
function setSize(w,h) { this.width = w; this.height = h; }
function enlarge(f) { this.width *= f; this.height *= f; }
function shrink(f) { this.width /= f; this.height /= f; }
// define a constructor for Rectangle objects
function Rectangle(w,h){
// initialize properties
   this.width = w;
   this.height = h;
// define methods for the object
this.area = Rectangle_area;
this.size = setSize;
this.enlarge = enlarge;
this.shrink = shrink;
}
r = new Rectangle(2,2);
a = r.area();
r.enlarge(3);
```

Objects as Associative Arrays

- Object properties may be accessed via [ ] syntax

  object.property object[“property”]

- Allows for runtime manipulation of properties

```
var addr = “”;
for(i = 0; i < 4; i++) {
  addr += customer[“address” + i]
}

value = 0;
for (stock_name in portfolio) {
value += get_share_value(stock_name) *
  portfolio[stock_name];
}
```


Arrays

- Different elements of a single array may be of different types.
- JavaScript arrays are sparse.
- Arrays and Objects are the same thing
  - create with the new operator
  - write your own constructor
  - Navigator 3.0 and Explorer 3.0 predefine Array();
    - a = new Array();
    - a = new Array(10);
    - a = new Array(5,4,3,2,1, “testing”,”testing”);

Arrays in Navigator 3.0+

- the Length property
  - use to loop through array values
- Array.join()
  - converts all elements to a string and concatenates them
- a = new Array(1,2,3); s = a.join(); // s == “1,2,3”
- s = a.join(“+“); // s == “1+2+3”
- Array.reverse()
  - reverses “in place” all elements of an array
- Array.sort()
  - sorts elements according to passed method
- function numOrder(a,b) { return a-b; }

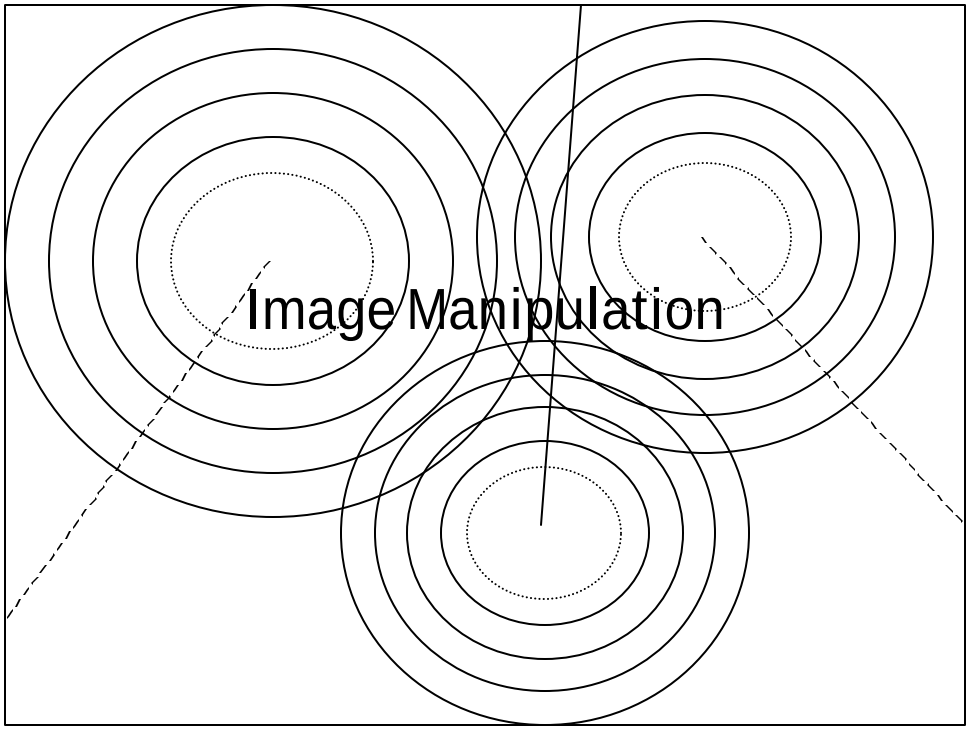


Image Replacement

- The Image.src Property
  - This property is read/write.
  - Can be used to make the browser load a new image in the same space as the one currently displayed.
  - New image must be the same size as the current image
- Image Caching
  - Creating an off-screen image forces it to cache.
  - Caching dramatically speeds the loading of an image.
  - To replace an image set the src property of the desired onscreen image to the URL of the desired image

Image Event Handlers

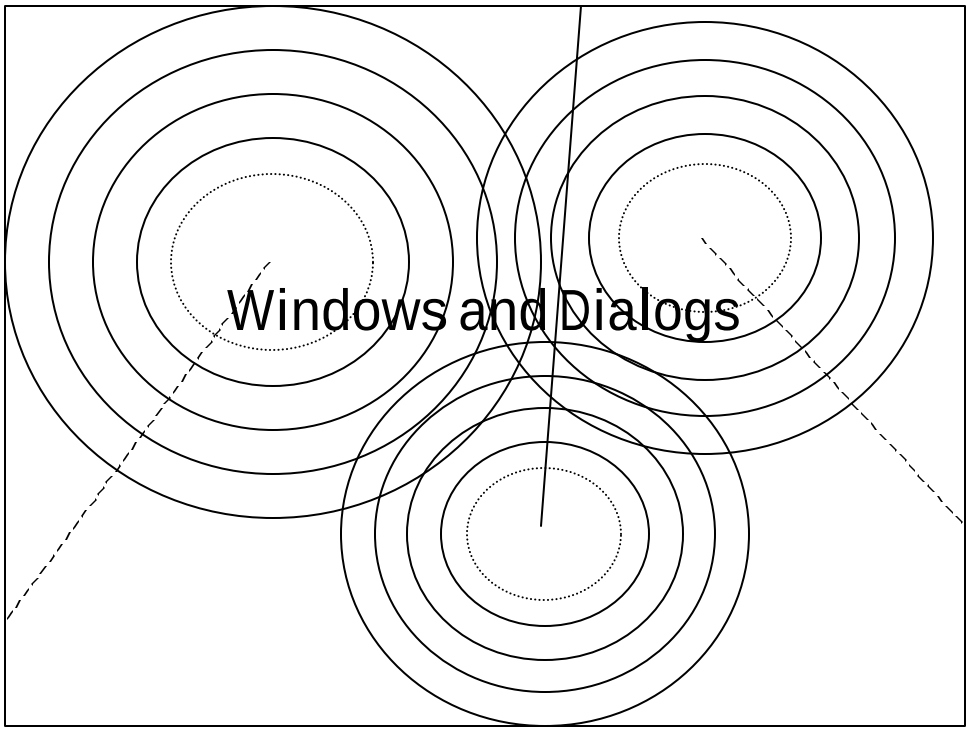
- Both the <IMG> tag and the Image() constructor have an onLoad() event handler.
  - invoked when the image is completely loaded.
  - Use to automatically start animations.
- onError
  - invoked if an error occurs during an image load
- onAbort
  - invoked if the user aborts an image load.
  - example: clicking the “stop” button.
- For any image ONE and only one of these handlers will be called

Image Animation Script

```
<IMG SRC="images/0.gif" NAME="animation"
<SCRIPT>
images= new Array(10);
for(var i=0; i<10; i++) {
  images[i] = new Image();
  images[i].src = "images/" + i + ".gif";  }

function animate(){
  document.animation.src = images[frame].src;
  frame = (frame+1)%10;
  timeout_id = setTimeout("animate()", 250); }

var frame = 0;
var timeout_id = null;
</SCRIPT>
```


Form to Control Script

```
<FORM>
   <INPUT TYPE=button Value'"Stop"
   onClick="if(timeout_id == null) &&
       num_loaded_imagesa == 10) animate()">
   <INPUT TYPE=button Value'"Start"
      onClick="if(timeout_id) clearTimeout(timeout_id);
       timout)id = null;">
</FORM>
```



Windows, Objects, Variables in the JavaScript Name Space

- In reality, variables are nothing more than properties of the current window.

```
var hitcount = 0;
parent.frames[1].hitcount;
```

- The browser window is represented by a Window object. Formally, this object has no name, but the following all work:

```
window.alert("The URL is: " + window.location);
alert("The URL is: " + location);
alert("The URL is: " + self.location);
```

- Under the window, there are a set of additonal objects that may be accessed

Simple Dialogs

- alert();

```
function warn_on_submit(){
    alert("This may take some time....");
}
```

- confirm();

```
var msg = "Are you sure your network can handle this?"
if(confirm(msg))
    location.replace("highcap.html");
else
    location.replace("lowcap.html");
```

- prompt();

```
n = prompt("What is you name?","");
document.write("Welcome to my homepage " + n + "<hr>");
```


Additonal Windows

- Create an explicit reference to another window by creating that window.

```
var newwin = window.open("sitemap.html",  "site_map_window");
```

- Window.open() returns an explicit reference to the newly created window

```
newwin.defaultStatus = "Site Map. Click map for details";
```

- Close windows with window.close();

```
window.close(site_map_window)
window.close(self)
```

Windows and Frames

- Each frame in a browser is represented with a window object
  - Each Window object has a frames[] array property.
  - Each Window object has a parent property.
- Explict names can by used as Target attributes

  <A HREF="chapter01.html" TARGET="mainwin">

  Chapter 1, Introduction

  </A>

  <FRAME NAME="toc" SOURCE="toc.html">

  parent.toc

  parent.frames[1]

Opening and Closing Windows

- The open() method features argument
  - string that contains a comma-seperated list of "features" for the new window.

```
smallwin = window.open("", "small",
"location,status, width=400,height=300");
```

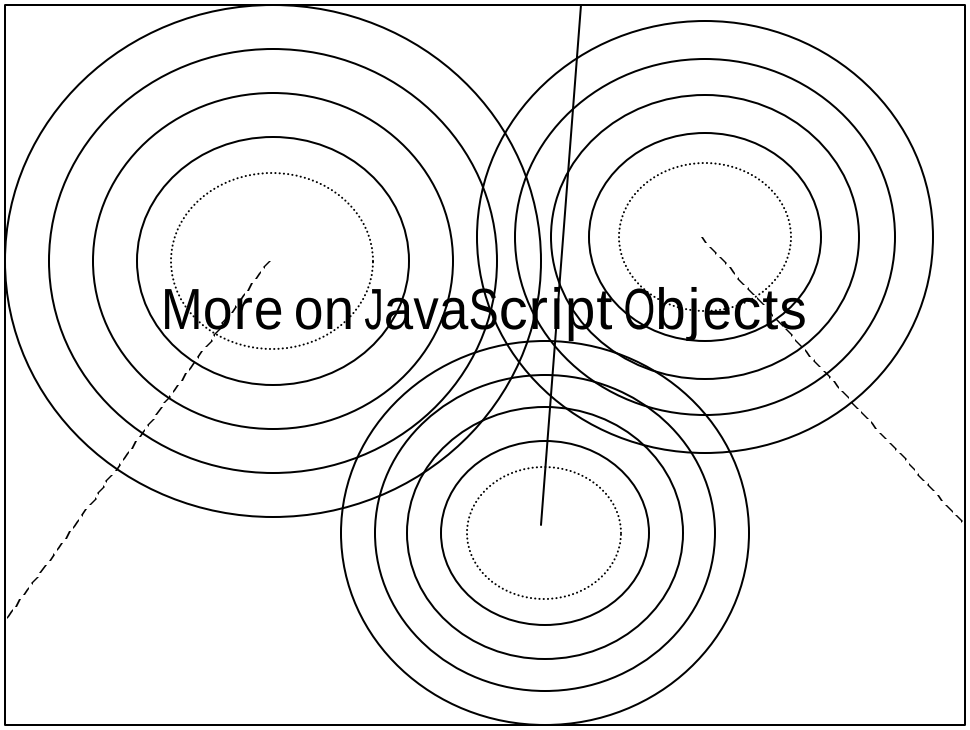
- Close with window.close();
- Window.onerror()
  - event handler invoked whenever any kind of JavaScript error is detected by the browser.
- The Status Line

```
<A HREF="SiteMap.html"onMouseover="status"='Go to
      SiteMape'; return true;"> Site Map </A>
```

Frame Programming Techniques

- In complex framesets create “global” properties of the top-level browser that refer to each of the frames in your program.
- Handling multiple dynamic frames
  - Invisible frames
  - Explicitly create a frame at a location greater than 100% of the frame width or height.

```
<frameset rows="50%,50%,*">
  <frame name="dynamic_frame_1">
  <frame name="dynamic_frame_2">
  <frame name="invisible_frame"
  src="program.html">
</frameset>
```

The JavaObject Object

- serves as a wrapper around Java objects.
  allows JavaScript programs to read / write the public fields of Java objects and invoke Java methods.
- Array and Object Access Operators
  - document.lastModified frames[0].appName
  - document["lastModified"] data["val" + i]

Browser Objects

- The Navigator Object
  - provides version and configuration information about the browser.
  - appName, appVersion, userAgent, appCodename
- The Location Object
  - specifies the URL currently being displayed, and allows JavaScript to load new URLs.
  - protocol, host, pathname, search, reload(), replace()
- The History Object
  - contains information about the URLs that have been previously displayed in the window.
  - back(), forward(), go() / / go is VERY buggy !!

The Document Objects

- Most document properties correspond to the attributes of the <BODY> tag in HTML.
  - lastModified is a string that specifies the date and time of the most recent modification
  - referrer specifies the URL of the document that contained the link that brought the user to this page

```
<SCRIPT>
  if (document.referrer == "" ||
  document.referrer.indexOf("mysite.com" == -1)
       {
       window.location= "javascript:'You can't get
              there from here!'";
       }
</SCRIPT>
```

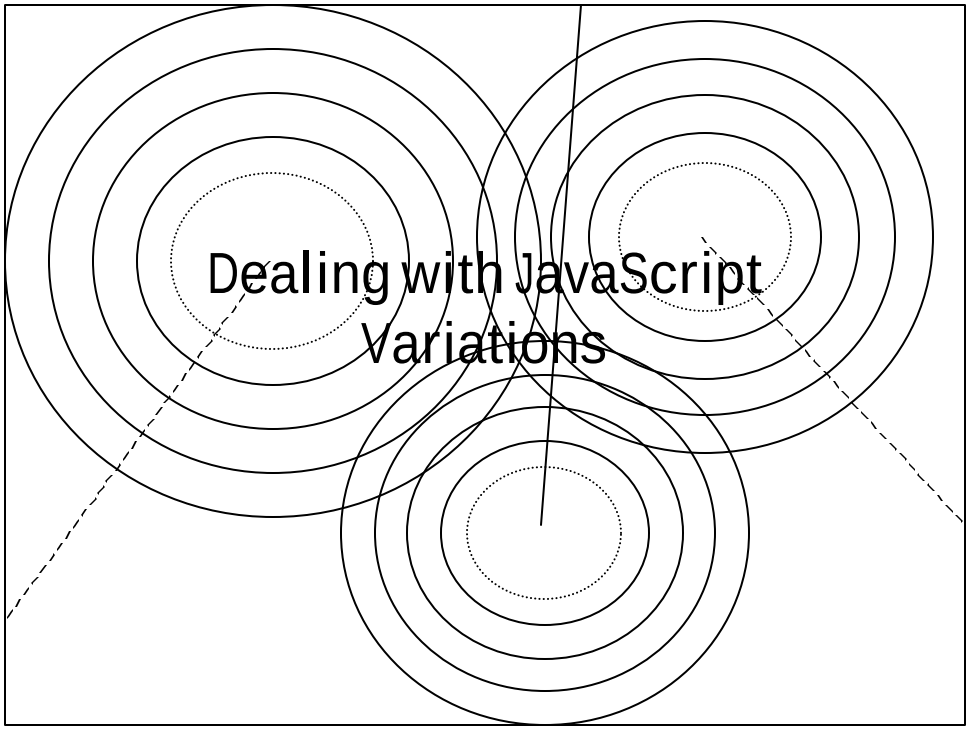

The write() Method

- Most important feature of the Document object.
- Allows the dynamic generation of web page content from JavaScript programs.
  - using write() in current window will overwrite currently displayed content!
- Useful in generating content for other windows

```
<SCRIPT>
  parent.frames[0].document.open();
  parent.frames[0].document.write
       ("<HR>Hello from your sibling frame!<HR>");
  parent.frames[0].document.close();
</SCRIPT>
```

Write and Non-HTML Documents

- Browser assumes the open() method create an HTML document (MIME type “text/html”)
- Different MIME types can be specified as an argument to open(); document.open(image/jpg);
- Not supported in Internet Explorer 3.0 !



Hiding Scripts from Old Browsers

- Browser that recognize <SCRIPT> but not JavaScript will ignore anything between the tags.
- Browser that don't recognize <SCRIPT> treat JavaScript code as HTML content!!
- To avoid this use HTML style comments

```
<SCRIPT LANGUAGE="JavaScript">
<!-- begin HTML comment that hides the script

              JavaScript code

// end HTML comment that hides the script -->
<SCRIPT
```


The <NOSCRIPT> Tag

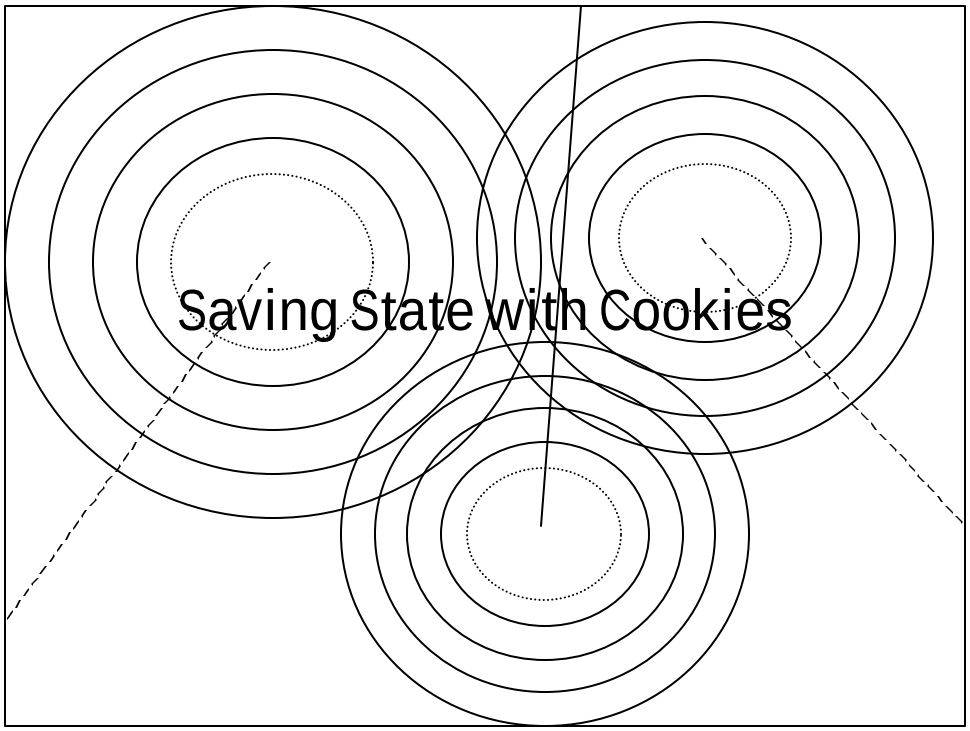
```
<HTML>
<HEAD><TITLE>JavaScript1.1 Page</TITLE></HEAD>
<BODY>
<H1>JavaScript 1.1</H1>
<NOSCRIPT>
      This page needs JavaScript1.1 <BR>
      which your browser doesn't support!
</NOSCRIPT>
<SCRIPT LANGUAGE="JavaScript1.1"> <!--
      JavaScript code
// -->
</SCRIPT>
</BODY>
</HTML>
```

For Very Old Browsers

```
<! -- commented out in both JavaScript and HTML -->
<!-- --> commented out only in JavaScript

<SCRIPT LANGUAGE="JavaScript">
<!-- --><HR><H1>This Page Requires JavaScript</H1>
<!-- -->Your browser does not support JavaScript so
<!-- -->you will not be able to display this page.

<!-- being hiding JavaScript from old browsers

             JavaScript Code Goes Here

 // end hiding -->
</SCRIPT>
```


Version Control and Information

```
<SCRIPT LANGUAGE="JavaScript"> <!--
      _version = 10 // --> <SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.1"> <!--
      _version = 11 // --> <SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.2"> <!--
      _version = 12 // --> <SCRIPT>
<SCRIPT LANGUAGE="JavaScript"> <!--
  if(_version < 11) {
   document.write('This page requires JavaScript1.1');
   document.write('Your browser uses JavaScript1.0');
  }
// --> </SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
<!-- Start hiding JavaScript Code
      JavaScript code goes here
// -->
</SCRIPT>
```



Cookies

- A cookie is a small amount of named data stored by the web browser and associated with a particular web page or web site.
- Generally used to save/share state information
- Cookies can be transient or stored on the client
- Document.cookie
  - a string property that allows you to access and manipulate the cookie(s) associated with a web page.
  - setting the property creates a new cookie
  - reading the cookie returns a list of all cookies that apply to the current page.

- Cookie strings take the form of name=value pairs.
  - Browsers are not required to store more than 300 cookies total and not more than 20 per web server.
  - cookies are generally limited to 4K in size
- Cookies are transient by default
  - set an expiration date to save a cookie between sessions
  - name=value; expires=date;
- By default a cookie is associated with and accessible to the web page that created it and any other web page in the same directory.
  - specify a path to make it accessible beyond the dir.
  - specify a domain to make it accessible across servers.

Reading Cookies

- the cookie property returns a string containing all cookies associated with the current document.
- String is a semicolon + blank space separated list of name=value pairs.
- use String.indexOf() and String.substring() to determine the value of the cookie you are interested in.
- The value of a cookie must not contain any semicolons, commas, or whitespace.
  - use escape to encode cookie values prior to storing them
  - use unescape to decode cookie values on retrieval

```
function GetCookie(name) {
   var result = null;
   var myCookie = " " + document.cookie + ";";
   var searchName = " " + name + "=";
   var startOfCookie = myCookie.indexOf(searchName);
   var endOfCookie;
   if(startOfCookie != -1) {
       // skip past cookie name
       startOfCookie += searchName.length;
       endOfCookie =
              myCookie.indexOf(";", startOfCookie);
       result =
   unescape(myCookie.substring(startOfCookie,endOfCookie)
   );
   }
   return result;
}
```

Storing Cookies

- Just set the cookie property to a string of the form name=value

```
function EZsetCookie(name,value) {
  document.cookie = name + "=" + escape(value);
}
```

- The name=value pair can be anything.
  - FavoriteColor=Blue
  - CurrStat=1:2:1:0:0:1:0:3:3:1
- Setting an expiration date
  - expires =date
  - Dates are in the form
  - Wdy, DD-Mon-YY HH:MM:SS GMT
  - Mon, 08-Jul-96 03:18:20 GMT

Setting a cookie to expire in one week

```
var name="foo";
var value="bar";
var oneWeek = 7 * 24 * 60 * 60 * 1000;
var expDate = new Date();
expDate.setTime (expDate.getTime() + oneWeek);
document.cookie = name + "=" + escape(value) +
  "
  expires=" + expDate.toGMTString();
```


Deleting a cookie

- there is no explicit delete cookie function
- delete by setting the date to the past.

```
function clearCookie(name)
{
  var ThreeDays = 3 * 24 * 60 *60 * 1000;
  var expDate = new Date();
  expDate.setTime (expDate.getTime() - ThreeDays);
  document.cookie = name + "=ImOutOfHere; expire="
      + expDate.toGMTString();
}
```

- When deleting a cookie the value doesn't matter

Other Ways of Saving State

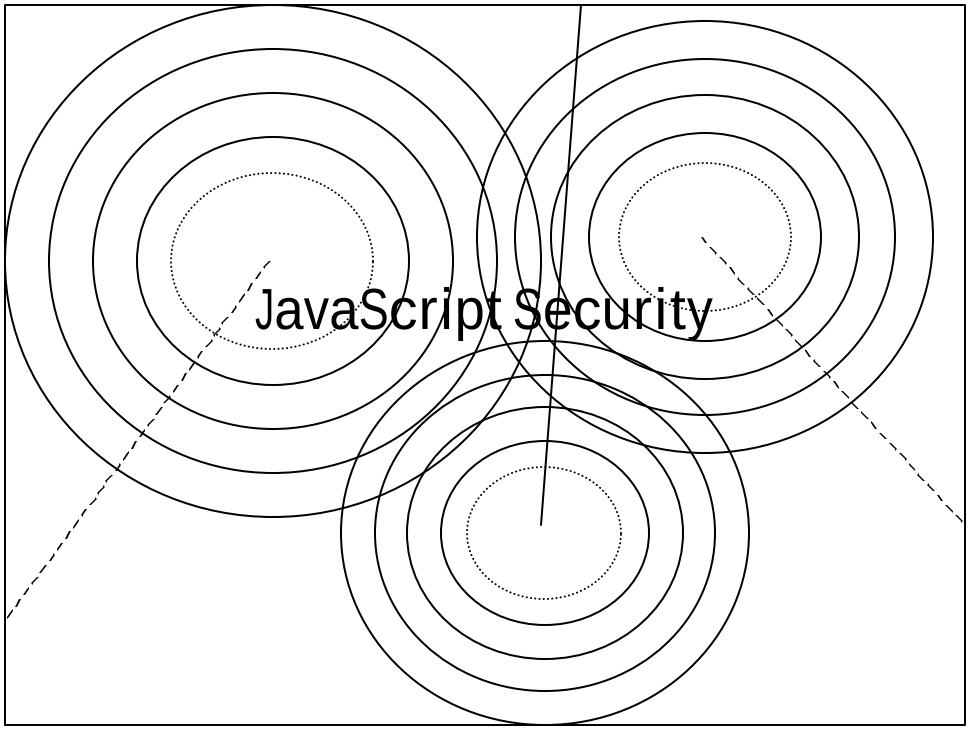
- Query String
  - You can add state information to hypertext links
  - seperate state information from URL with a "?"
  - seperate pieces of information with an "&"

```
<A HREF = "http://www.pitt.edu/mypage.html?
color=blue&size=extra+large">XL Blue</A>
```

- Hidden Fields
  - Pass state information between forms and web pages

```
<INPUT TYPE="hidden: NAME="hiddenfield" VALUE="value">
```

- Niether of these methods require JavaScript



JavaScript and Security

- JavaScript does not support certain capabilities conducive to malicious activity.
  - There is no File object or file access functions.
  - There are no networking primitives.
- Biggest JavaScript security concern is privacy.
  - Do not allow a script to export private user information.
    - Browser version v. Browser history.
  - If private information can be stolen it will be stolen.

Security Holes and Hobbles

- JavaScript has used identify-and-patch.
- A security hobble restricts the capabilities of a scripting language so that a security hole cannot be exploited.
  - The History object cannot access the elements of the history[] array.
    - back(), forward(), go().
- Not all holes are readily identified.
  - Netscape 2.0 about:cache URL & links[] array.
  - file:/ / / URL discovered the contents of the root directory.

The Data-Tainting Security Model

- Rather than preventing the reading of private data, prevent its exportation.
  - All JavaScript data values are given a flag.
  - Flag indicates if a value is “tainted” (ie private)
  - Untainted values may be exported arbitrarily.
  - Any value may be manipulated (tainted or not)
  - Tainting is inherited.
- This is the theory and it works great in Perl but . . .

JavaScript Data-Tainting

- The flag is more of an accumulator.
  - There are several types of tainting in Navigator.
    - History may not be exported anywhere.
    - Form values in a document loaded from server abc.xyz.com may only be exported to server abc.xyz.com.
- Data-Tainting only prevents tainted data from being exported automatically.
  - When an export rule is violated the user is prompted.
- JavaScript functions and methods may be tainted.
  - Return values are automatically tainted regardless of the taint state of the functions arguments.