JavaScript

What is JavaScript?

- Client side interpreted embedded programming language used to enhance websites
  - ECMAScript language standard implementation
  - ECMA-262
- No relation to Java
- Can manipulate HTML
- Event driven

To use or not to use

- Helpful for:
  - Dynamic content
  - Adding logic to HTML
  - Make changes without refreshing the page
  - Form validation
  - Ease processing on server
- Not helpful for:
  - Accessing resources
  - Do anything that requires privacy
    - JavaScript is shown publicly

Adding JavaScript to HTML

- Inline:
  - <script language="javascript"> \\code </script>
- External:
  - <script language="javascript" src="code.js" />
- Typically this is placed in between the <head> tags
- Case sensitive (unlike HTML)

Variables

- Variables are untyped
  - No need to declare type (int, bool, etc)
  - Doesn't care about type until called
- Example:
  - var name = "Andrew";
  - var num = 24;
  - var name = num;
  - name will now return 24

Arrays

- Strings are arrays
  - var x = "Andrew"; x[3] returns "r"
- Array is also it's own data structure and doesn't require all items to be of the same type.
  - var x = new Array();
    - x[0] = "Rawr";
    - x[1] = 9001;
  - var x = new Array("Rawr", 9001);
  - var x = ["Rawr", 9001];
- Arrays have a length:
  - x.length

If..Else / Switch.

- Similar to what you'd expect
  - if (<condition>) { \\code }
  - if (<condition>) { \\code } else { \\more }
  - if (<condition>) { \\code } else if { \\more } else { \\finally }

```
switch (<variable>) {
case <match>:
   \\code
   break;
case <match2>:
   break;
default:
}
```

Loops

- For:
  - for (<var>; <condition>; <do>) { \\code }
  - for (<var> in <list>) { \\code }
- While:
  - while (<condition>) { \\code }
  - do { \\code } while (<condition>);
- break and continue work as expected

Functions

- Similar to any other languages' functions and methods
- Can have nested functions
  - Variables have scope similar to nested loops
- Used by events
  - Runs only when called
- Example:
  - function <name> (<parameters>) { \\code }

Popups

- Used for displaying information to the user outside of the page.
  - Typically as a warning or when an error has occured
  - Ask user for additional information
  - Confirmation

- alert(<text>); - Exits via an okay button
- confirm(<text>); - Returns true or false (ok or cancel)
- prompt(<text>, <default>); - Returns user's input

Try it

Create your own html file and using a text editor create.

- Create an Array
- Create a function that:
  - Use a for loop to loop through the array
  - Print the contents of each element using document.write();
  - Use a prompt() to ask for a username
  - Using an if statement, take the input and if it is 4 characters long, print "Yes", else print "No".
- Use <body onload="f()"> to execute function on page load

Exceptions and Try/Catch

- Exceptions are easily thrown in a function by adding the following line. This will exit the function, returning the exception text.
  - throw <text>;
    - Example: throw "Error1";
- Try/Catch is as expected:
  - try { \\code } catch (<error>) { \\more }

Document Object Model (DOM)

- Standard way to access/manipulate HTML documents
- Hierarchy of objects in HTML
- Examples of objects:
  - window, location, document, anchors, body
  - images, forms, elements, tables
- Code example:
  - document.write("This is displayed.");

Cookies

- Stored in text file on client
- Can store multiple values (";" delimited)
- Limited
  - 300 per browser
  - 20 per web server
  - 4KB per cookie
- Default: Read only by originating webpage
  - Can be read by others using:
    - path - multiple sites
    - domain - multiple servers
- Remove by setting the expiration to current time or a past time.

Cookies (cont)

Example:

```
document.cookie("username=Andrew;expires=2011-01-11");
var aCookie = document.cookie;
var items = aCookie.split(";");
var expires = items[1].split("=")[1];
```

- The use of split returns an array of substrings
- After the first go we have "username=Andrew" and "expires=2011-01-11"
- After the second go we have "expires" and "2011-01-11"
- The variable "expires" now equals "2011-01-11"

Date()

- Date(); returns the current date.
- Date(<milliseconds>); returns the date since 1970/01/01
- Date(<date_string>); returns date given by string
- Date(y,m,d,h,m,s,ms); returns the date based on which variables are filled in
  - Date(2011, 6, 17); = 6/17/2011
  - Date(2011, 6, 17, 13, 5); = 6/17/2011 13:05

Try it

Goto:

http://www.w3schools.com/js/tryit.asp?filename=tryjs_cookie_username

- Look at the code to create a cookie.
- In your browser go to where your cookies are stored.
- Find the "username" cookie for www.w3schools.com
- Notice the fields and when it expires.
- Try running the code again
  - The cookie hasn't expired yet!

Math

- JavaScript has it's own math functions built in
  - abs(x), random(x), round(x), sin(x), etc
- Also has constants defined
  - PI, E, LN2, LOG10E, SQRT2
- To access these, just call Math then the function/constant directly
  - Math.abs(x)
  - Math.PI

Stand back…I know RegEx

- JavaScript also has an easy way of doing regular expressions
- There is the functional way:
  - var pattern = new RegEx(<pattern>, <modifiers>);
- Or a simplified way:
  - var pattern = /<pattern>/modifiers;
- Use test() with RegEx to see if a string matches:
  - pattern.test("Hey there");
    - Will return true or false
- User exec() to find a matching string and return the results.

Objects

- Similar to classes in other languages
  - Can have variables and methods
- var myObject = new Object();
  - myObject.name = "Andrew";
  - myObject.number = 42;
- var myObject = {name: "Andrew", number: 42};
  - myObject.tired = "Yes"

Objects - Functions

- Functions can create objects, effectively being constructors.

```
function dude (name, age) {
this.name = name;
this.age = age;
}

dude.setAge = function (x) { this.age = x; };

var guy = new dude("Andrew", 24);
guy.setAge(42)
```

Objects - Singletons

If an object will only exist in a single instance, you can do the following:

```
var myObject = {firstmethod: function (x,y) { \\code} };
myObject.firstmethod(5,"A");
```

Try it

In an HTML file:

- Create an object with a few variables
  - one contains a string
  - one contains a number
  - one contains a function
- In the function, use alert(); to display the object's string
- Using the "for...in" loop, print each of the object's variable name, alongside with the value.
  - for (x in obj) { print x : obj[x] } //Pseudo code
- Call the object's function

Asynchronous JavaScript and XML

- Fast and Dynamic web pages
- Perform behind the scenes to update portions of a webpage without having to reload the whole page.
- Based on:
  - XMLHttpRequest object - communicate with server
  - JavaScript/DOM - display/manipulate information
  - CSS - Style it to make it look nice
  - XML - Format data for transfering

AJAX - XMLHttpRequest

- Create object
  - var xmlrequest = new XMLHttpRequest();
- Send a request
  - open(httpMethod, targetURL, async);
    - xmlrequest.open("GET", "example.asp", true);
  - send();
    - xmlrequest.send();

XMLHttpRequest - GET

- Simple, fast, good for cached data.
- Simple:
  - open("GET", "demo.asp", true)
  - Can return cached data
- Fresh:
  - open("GET", "demo.asp?t="+Math.random(), true)
  - Unique id prevents cached data from appearing
- Send information:
  - open("GET", "demo.asp?username=Andrew&age=24", true)

XMLHttpRequest - POST

- For database accessing, sending large amounts of data, can't work with caching, need for security/robust transfer
- Simple:
  - open("POST", "demo.asp", true)
- Send form information:
  - open("POST", "demo.asp", true)
  - setRequestHeader("Content-type", "application/x-www-form-urlencoded")
  - send("name=Andrew&age=24");

XMLHttpRequest - Server Response

- If response is not XML
  - request.responseText
  - Returns the text from the server
- If it is XML
  - request.responseXML
  - Returns an XML file from the server
  - Probably will need to be parsed and then use

AJAX - readyState

- Holds the status of the XMLHttpRequest object
- Perform actions based on the readyState
- onreadystatechange even is triggered when readyState changes
- onreadystatechange stores a defined function to occur and process the readyState upon change

AJAX - readyState (cont)

- readyState statuses:
  - 0: request not initialized
  - 1: server connection established
  - 2: request received
  - 3: processing request
  - 4: request finished and response is ready

- status:
  - 200 = "OK"
  - 404 = Page not found


The End