


Javascript

INF340 Web Design Concepts

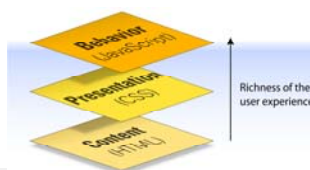
The 3 Layers of a Web Page

- An HTML page can be thought of as having three distinct layers:
 - Content: HTML
 - Presentation: CSS
 - Behavior: Javascript**



Building a Web Page Properly

- Building a web page from the bottom up enhances the richness of the experience
 - HTML first
 - Styling second
 - Interaction third



Interaction through Javascript

- Javascript is designed to add interactivity and behavior to a styled, semantic HTML document
- It should be used to enhance the browsing experience but can be turned off by the user
- While a standard version exists, it does not always work the same way across computers

Example of Behaviors

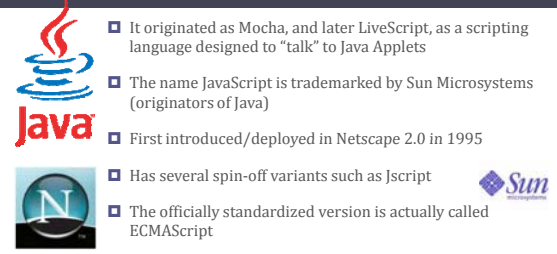
```

<style>
.p-blue { color: blue; }
.bold { font-weight: bold; }
.a { color: red; }
.a:hover { color: purple; }
</style>
<p class="p-blue p-bold">
  This is a blue, bold content area
  in the page using proper classes.
  <a onclick="alerter();">CLICK</a>
</p>
<script type="text/javascript">
function alerter() {
  alert("THIS IS AN ALERT");
}
</script>
    
```

➤ This example show proper HTML and styling mixed with an event-driven behavior enhanced by Javascript

A Little History

- It originated as Mocha, and later LiveScript, as a scripting language designed to "talk" to Java Applets
- The name JavaScript is trademarked by Sun Microsystems (originators of Java)
- First introduced/deployed in Netscape 2.0 in 1995
- Has several spin-off variants such as Jscript
- The officially standardized version is actually called ECMAScript



Modern Javascript

- Modern Javascript (ECMAScript-262, aka Javascript 1.8) is extremely flexible and powerful
- It supports structured programming (uses conditionals and loops), multiple variants (data types) and more
- It can be used sequentially, procedurally, or as an object-oriented language

Features of Javascript

- Dynamic
 - Dynamic typing (allows cross-casting)
 - Objects as associative arrays
 - Run-time evaluation
- Functional
 - First-class functions (functions are objects)
 - Inner functions and closures (can write functions inside of other functions)

Features of Javascript +

- Prototyping
 - Uses prototypes (ability to simulate class-based programming through use of prototypes)
 - Functions as object constructors or methods (functions themselves can be used in a flexible environment and for multiple purposes)
- More
 - Run-time environment (works without compiling)
 - Variadic functions (indefinite parameters to functions)
 - Regular expressions supported

Security Issues in Javascript



- Javascript works inside the "sandbox" – a region of the browser's security that prevents use outside of basic browser functions – it cannot write files!
- Javascript is constrained by the "same origin policy" so scripts from one source cannot interact with scripts from another source (they must both come from the same domain or hostname)
- Vulnerabilities do exist and are often the source of malicious exploits

Data Types (Simple)

- Strings
 - Character data of any type
- Numerics
 - Numbers, including integers and floating numbers
- Boolean
 - True or false
- Arrays
 - A collection of other data types contained within a single container

Data Types (Complex)

- Date
 - An object capable of comprehending date and time references in various formats
- Math
 - Allows performance of specific complex math tasks
- RegExp
 - Regular expression – a means of expressing strings by reference

So What's Javascript Good For?

- Interactivity
 - Providing error handling in form completion
 - Block element swapping
 - Ajax
- Animation
 - Used in conjunction with styling, Javascript provides a solid, rudimentary animation engine
- Calculations
 - Performing in-page calculations

Where Do I Put Scripts?

```
<html>
<head>
<script type="text/javascript">
alert('this is inline scripting');
</script>
<script type="text/javascript"
src="external.js"></script>
</head>
</html>
```

```
//this is an external script called
//external.js
alert('this is external scripting');
```

- Scripts can be
 - Inline (also usually called "embedded")
 - External
- External is better
 - Globalized – can hit all pages
 - Modularized – can be re-purposed

Statements

```
alert('this is a statement')
alert('and this is another one')

if(condition exists)
do some action
else
do some other action
```

```
alert('this is a statement');
alert('and this is another one');

if(condition exists){
do some action;
}else{
do some other action;
}
```

- Each step of a program is called a statement
- A statement tells the browser to do something
- A series of statements creates a program
- Statements can be separated by a semi-colon or a new line

Statements

```
//Bad way
alert('hi'); if(some condition){ /* do
some action */ } else { /* do some
other action */ } var p = prompt('can I
even tell what is going on?')
```

```
alert('hi');
if(some condition){
/* do some action */
} else {
/* do some other action */
}
var p = prompt('can I even tell what is
going on?');
```

- In general using both new lines and semi-colons is preferred
 - Easier to find the end of statements
 - Easier to debug
 - More common practice
- Indenting is also a helpful common practice

Variables

- Variables are containers for storing data
- It gives a name to a value (much like algebra)
- The keyword **var** is used to **declare** a variable
 - Variable names must be declared
 - Values can be assigned at time of declaration
- Once a name is declared it can be re-declared
 - Re-initializes the variable
 - This is unlike most languages

Variables

- Variable naming rules
 - Letters and numbers in name only, no spaces or symbols except `_`
 - `$` is also usable in special cases
 - Cannot start with a number
 - Case sensitive
 - `FOO` is not `foo` is not `Foo` is not `fOO`
- Common practices
 - Lower case or camel casing is common
 - Hungarian and Bulgarian notation is not common
 - (why do you think not)

Operators

- ▣ Operators are symbols that perform (generally) mathematical functions to variables
 - ▣ + is addition
 - ▣ - is subtraction
 - ▣ * is multiplication
 - ▣ / is division
 - ▣ % is modulation

Escape

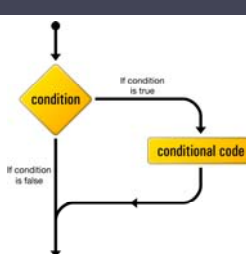
```
document.write "The cow says \"moo\"";
alert('This is an alert.\n\nThis will appear two lines below the line "This is an alert."');
```

- ▣ The back-slash \ is called the escape character
 - ▣ If you are in a situation where you need quotations that are within a string demarked by the same quotation symbol
 - ▣ Also provides for some special sequences

Conditionals

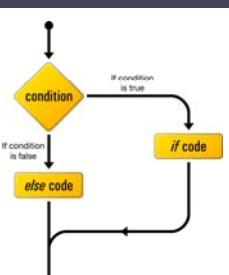
- ▣ Programs are statements whose logic comes based on decisions
- ▣ In programming, decisions occur based on conditions
- ▣ Conditionals are simply a reference to the state in which a condition exists

if Conditional



- ▣ The simplest conditional
- ▣ Only performs action if the single condition is true
- ▣ Uses comparison operators (e.g. > or ==)
- ▣ Can have complex (multiple) conditions within the single statement

If-else Conditional



- ▣ The simplest conditional
- ▣ Performs one or the other action
- ▣ Uses comparison operators (e.g. > or ==)
- ▣ Can have complex (multiple) conditions within the single statement

if and if-else Syntax

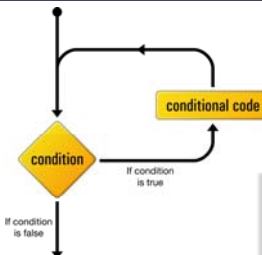
```
var x = 721;
var y = 20;
var z = 22;
//simple if conditional
if(x>y) {
}
//compound conditional with else
if(x>y&&y<z&&x != z){
  alert('all conditions are true');
} else {
  alert('at least one of the conditions is false');
}
//secondary assessment with else if
if(x>y&&y<z&&x != z){
  alert('all conditions are true');
} else if (x<y){
  alert('only occurs if x less than y');
}
```

- ▣ Additional assessment using else if in place of the else – provides for secondary evaluation

Loops

- Automates repetitive tasks
- Performs a repeated set of actions
- Fundamentally all loop conditions work the same
- All work so long as the condition remains *true*

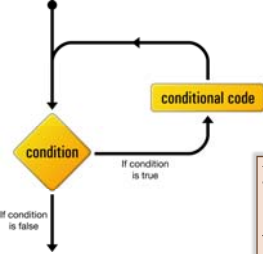
while Loop



- Performs the loop so long as, upon encountering the conditional check, it finds the condition to be true
- Needs incremter (or decremter inside loop)

```
var x = 5
while (x > 0) {
  alert(x);
  x--; //the decremter
}
//outputs 5 alerts as 5, 4, 3, 2, 1
```

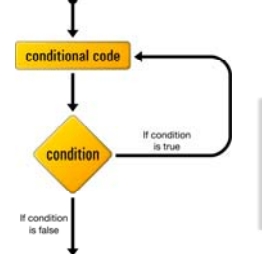
do-while Loop



- Same as while loop except that the while condition is placed after the loop
- Will always execute at least once, even if false

```
var x = 5
do {
  alert(x);
  x++; //the decremter
}
while(x > 25);
//outputs 1 alerts as 5
```

for Loop



- Provides the starting and ending values as well as the incremter

```
var x = 2;
var z;
for(y=1; y<4; y++){
  z = x * y;
  alert(z);
}
//outputs 3 alerts as 2, 4, 6
```

Functions

- Functions are small sequences of code - routines - that wait to be executed by command
- Javascript has built-in functions, such as **alert()**
- Functions are initialized by the keyword **function**
- A function can be executed infinite times
- You can use any word as a function name except any keyword or existing function

Function Arguments

```
doMyThing(1,2,3);
function doMyThing(a,b,c){
  var x = (a + b) * c;
  alert(x); //9
}

doSomething("foo", "bar");
function doSomething(){
  alert(arguments[0]); //foo
  alert(arguments[1]); //bar
}
```

- Arguments are variables that functions use to perform its actions
- Your functions can have as many arguments as you want them to
- Arguments passed in are themselves an array

Return Functions

```

var y = doMyThing(1,2,3);
echoist(y);

function doMyThing(a,b,c){
  var x = (a + b) * c;
  alert(x); //9
  return x; //sends x back out
  alert(x); //ignored
}

function echoist(z){
  var d = z * z;
  alert(d); //81
}
    
```

- Most functions, rather than output, simply return a value
- The return is always the last action performed - anything that follows is ignored

Variable Scope

```

var z = 10;
var a = 2;

function myMethod(aa,zz){
  return (zz/aa); //sends 5 back out
}

var b = myMethod(a,z);
alert(b); //5
alert(g); //undefined
    
```

- Global scope** variables are declared outside the function and can pass in and out of functions
- Local scope** variables are only usable inside the function

Accessing Nodes

- Nodes can be accessed in several ways:
 - By ID
 - By Element (Tag)
 - By Restricted Element (Child Tag)
 - By Class Name

Accessing Nodes by ID

```

<style>
#some-paragraph { color: blue; }
</style>

<script>
var t = document.getElementById("some-paragraph");
alert(t.nodeName); //returns p
alert(t.innerHTML); //returns Some...
</script>
    
```

```

<p id="some-paragraph">
Some text within the paragraph
</p>
    
```

- Best when node IDs are known
- Though maybe not type (nodeName property)
- Assumes uniqueness

Accessing Nodes by Element

```

<style>
p { color: red; }
</style>

<script>
var t = document.getElementsByTagName("p");
alert(t.style.color); //returns red
alert(t.innerHTML); //returns Some...
</script>
    
```

```

<p id="some-paragraph">
Some text within the paragraph
</p>
    
```

- Best when node IDs are not known
- Accesses tags (if only one) or tag groups
 - If tag groups, puts into an array
 - Properties can transverse from styling

Accessing by Restricted Tag

```

<script>
var t =
document.getElementsByTagName("ul");
var t1 = t[0];
var x = t1.getElementsByTagName("li");
alert(x[1]); //returns Item A2
</script>
    
```

```

<ul>
<li>Item A1</li>
<li>Item A2</li>
</ul>
<ul>
<li>Item B1</li>
<li>Item B2</li>
</ul>
    
```

- For accessing children of parent tag
 - Assumes all items are within an array structure
 - Allows for looping of child trees
 - Works best when there are dependent children (such as li or td)

Accessing by Class Name

```

var arrE = [];
if (typeof document.all!="undefined"){
  arrE = document.all;
}else{
  arrE =
  document.getElementsByTagName("");
}
var arrT = [];
var pattern = new RegExp("(^| )" +
theClass+"( |$)");
for (var i=0;i < arrE.length;i++){
  if (pattern.test(arrE[i].className)){
    arrT[j] = arrE[i].innerHTML;
    j++;
  }
}
for(p=0; p<arrT.length; p++){
  alert(arrT[p]); //AAA then BBB
}
    
```

- First create an array of all elements then subset it by creating a new array whose elements have a specific class
- Also works for any attribute

```

<hr class="blue"/>
<p class="red">AAA</p>
<p class="red">BBB</p>
<hr class="black"/>
    
```

Accessing Attributes

```

<script>
var t = document.getElementById("lnk");
var tAtt = t.getAttribute("href");
alert(tAtt); //returns foo.html
</script>
    
```

- The `getAttribute` method extracts specific attribute values
- Good when you need to check a reference for comparison
- Assumes some alternative method of node access first

```

<a href="foo.html" id="lnk">
  This is a test link
</a>
    
```

Setting Attributes

```

<script>
var t = document.getElementById("lnk");
t.setAttribute("href", "goo.html");
</script>
    
```

- Once accessed, the attribute can be modified using `setAttribute` method
- Again, assumes some alternative method of node access first

```

<a href="foo.html" id="lnk">
  This is a test link
</a>

Becomes

<a href="goo.html" id="lnk">
  This is a test link
</a>
    
```

Setting Styles

```

<script>
var t =
document.getElementsByTagName("a");
t.style.color = "red";
</script>
    
```

- Likewise, styling can be modified by using the `.style.property` sequence
- Works under rules of cascaded precedence
- Applies to node or node group accessed

```

<a href="foo.html" id="link1"
style="color:blue;">
  This is a test link
</a>
<a href="foo.html" id="link2"
style="color:purple;">
  This is another test link
</a>
    
```

Setting Styles

```

<script>
function changeBody() {
  var body = document.getElementsByTagName("body")[0];
  body.style.backgroundColor = "#000000";
  body.style.color = "#FFFFFF";
}
</script>
<body style="background-color: #FFFFFF; color: #000000;">
</body style="background-color: #000000; color: #FFFFFF;">
    
```

Setting Styles With Classes

```

<script>
function changeBody() {
  document.getElementsByTagName("body")[0].className = "black";
}
</script>
<body class="white">
<body class="black">
    
```

- Always smarter to change classes rather than style directly - easier to manage