# CSCI 2910
# Client/Server-Side Programming

Topic: Intro to JavaScript



## Today's Goals

With a little luck at the conclusion of this lecture, we should be able to:

- Define what type of programming language JavaScript is
- Describe the syntax of JavaScript
- Insert a JavaScript program into an XHTML file
- Hack into someone else's JavaScript

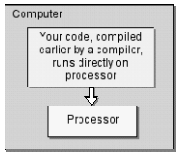


## What is JavaScript?

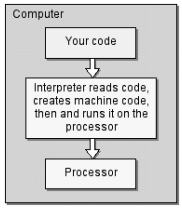
- JavaScript is a high-level language
- JavaScript is an interpreted language
  - A program called an interpreter runs on the client's computer.
  - One instruction at a time, the interpreter figures out what to do by reading your ***text*** program file.
  - Interpreted programming language can run on any platform or O/S as long as there is an interpreter that runs on that particular setup.
  - Interpreted languages are usually slower.
  - Speed may not be a big factor because programs written in JavaScript are not usually that complex



## What is JavaScript? (continued)




**Compiled Code Block Diagram**




**Interpreter Block Diagram**



## What is JavaScript? (continued)

- JavaScript runs through a web browser
  - developed by Netscape in order to give dynamic operation and control to web pages
  - executed on the user's or client's computer, i.e., the interpreter is part of the browser, not the server software
- JavaScript is an object-oriented language
  - program elements are organized into "objects"
- JavaScript is case sensitive
- JavaScript is NOT Java
  - Java and JavaScript share many characteristics, but Java is a compiled language and it has a number of capabilities, instructions, and libraries not available to JavaScript.



## JavaScript Syntax

- JavaScript programs are created with basic text editors just like your HTML web pages.
- The JavaScript interpreter that will be running your code will look line-by-line through the code to see what to do.
- In order for the interpreter to understand what you've written, there are some rules you must follow, i.e., syntax.

## JavaScript Syntax – Terminating a Line

- In general, each instruction may cross more than one line on the screen. In other words, white space such as tabs, spaces, or carriage returns do not usually affect the flow of the program.
- Because of this, we need to have a way to indicate that we have reached the end of an instruction.
- In JavaScript, we do this with a semicolon (;). For example:

```
A = B + C;
```



## JavaScript Syntax – Grouping Lines

- The program needs to know how to group sections of code together.
- Grouping might be required for things such as:
  - functions
  - loops
  - if-blocks
- We need to have a syntax for doing this. In JavaScript, we use the curly brackets ({ and }). For example:

```
{
  Lines of code grouped together
}
```



## JavaScript Syntax – Comments

- All programming languages allow the programmer to add comments to their code that are "invisible" to the execution of the program.
- In JavaScript, there are two kinds. (Actually there are three, but one of them is sort of unofficial.) They are:
  - Block comments
  - End of line comments
  - Comments to force old browsers to ignore JavaScript (the unofficial one)



## Block Comments

```
/* This is a block comment. It
is surrounded by the
slash/asterisk and
asterisk/slash that indicate the
beginning and ending of a
comment respectively. A block
comment may span multiple lines.
*/
```



## End of Line Comments

```
// This is an end of line comment.
// Everything from the double
// slashes to the end of the line
// is ignored.
// To use this method over
// multiple lines, each line must
// have its own set of double
// slashes.
```



## Comments for Old Browsers

- This third method of commenting is not an official method. It is simply a fix so that JavaScript can be commented out with HTML comment tags so that older browsers will ignore the script.

```
<!-- The HTML comment character
<!-- acts just like double slashes
```

- It is important to note that "- ->" doesn't do anything and is not a JavaScript comment character.

## Flow Control

- Flow control consists of a number of reserved words combined with syntax to allow the computer to decide which parts of code to execute, which to jump over, and which to execute multiple times.
- For the most part, all computer languages share the same basic reserved words for control, so what we discuss here will be the same for many different languages.



## Flow Control – "If"

- As an example, assume you want to print "Student's grade = A" when the value of the variable grade is above 93.
- The code below uses an *if-statement* to do this:

```
if (grade > 93)
     write("Student's grade = A");
```

- If grade was 93 or below, the computer would simply skip this instruction.



## Flow Control – "If" (continued)

- The programmer can also group multiple instructions to execute based on the result of the if-statement using the curly brackets. For example:

```
if (grade > 93)
{
   write("Student's grade is an A");
   honor_roll_value = True;
}
```



## Flow Control – "If" (continued)

- The programmer can string together if-statements to allow the computer to select from one of a number of cases using *else if* and *else*. For example:

```
if (grade > 93)
  write("Student's grade is an A");
else if (grade > 89)
  write("Student's grade is an A-");
else
  write("Student did not get an A");
```



## Flow Control – "Switch"

- A *switch* statement can be used to replace long sequences of if and else if statements. A sample of the JavaScript *switch/case* statement is shown here. →

```
write("Today is ");
switch(day_of_week)
{
     case 0:
          write("Sunday");
          break;
     case 1:
          write("Monday");
          break;
     case 2:
          write("Tuesday");
          break;
     and so on...
}
```



## Flow Control – "While"

- Another type of program flow control is to make a section of code execute multiple times.
- One way this is done is with the *while-loop*.
- This format also uses a "condition" placed between two parenthesis
- As long as the condition is true, the program continues to execute the code between the curly brackets in a round-robin fashion.
- Once the condition is false, execution goes to the next section of code.

## Flow Control – "While" (continued)

- Syntax:

```
while(condition)
{
  // statements to execute
}
```

- Example:

```
while(temperature > 72)
  air_conditioner_value = On;
```



## Flow Control – "For"

- It is also possible to create a loop that uses a counter or index. A *for-loop* is a single statement that:
  - initializes a index,
  - performs a operation on that index at the end of the loop, and
  - defines a condition on which to stop executing the loop.
- Its syntax is shown below:

```
for (initialization; terminating
       condition; operation)
{
    // statements to execute
}
```



## Flow Control (continued)

- This is ***NOT*** a comprehensive list of flow control formats.
- For more information on elements of flow control in JavaScript, check out one of the on-line resources:

http://www.devguru.com/technologies/javascript/11470.asp
http://www.hscripts.com/tutorials/javascript/switch-case.php
http://www.javascriptkit.com/javatutors/primer1.shtml



## Operators

- As with any programming language, there are operators that are used to processes information. These include:
  - Arithmetic operators
  - Bitwise operator
  - Assignment operators
  - Comparison operators
  - Logical operators
  - String operators



## Arithmetic Operators

- \+ Addition operator: adds two values
  Example: A + B
- – Subtraction operator: subtracts one number from another
  Example: A – B
- * Multiplication operator: multiplies two numerical values
  Example: A * B
- / Division operator: divides one number into another
  Example: A/B
- ++ Increment operator: adds one to its operand
  Example: A++
- -- Decrement operator: subtracts one from its operand
  Example: A--
- % Modulus operator: returns the integer remainder of a division of the second value into the first
  Example: A % B



## Bitwise Logical Operations

- ~ Bitwise NOT operator: Inverts each bit of the single operand placed to the right of the symbol
- & Bitwise AND: Takes the logical-bitwise AND of two values
- | Bitwise OR operator: Takes the logical-bitwise OR of two values
- ^ Bitwise XOR: Takes the logical-bitwise exclusive-OR of two values

## Bitwise Shift Operations

- << Left shift: Shifts the left operand left by the number of places specified by the right operand filling in with zeros on the right side.
- >> Sign-propagating right shift: Shifts the left operand right by the number of places specified by the right operand filling in with the sign bit on the left side.
- >>> Zero-fill right shift operator: Shifts the left operand right by the number of places specified by the right operand filling in with zeros on the left side.



## Assignment Operators

- The equal sign, '=', sets the element on the left side equal to the value of the element on the right side
- There are some short-hand uses of the equal sign:
  - a += b is equivalent to a = a + b
  - a -= b is equivalent to a = a - b
  - a *= b is equivalent to a = a * b
  - a /= b is equivalent to a = a / b
  - a %= b is equivalent to a = a % b
  - a &= b is equivalent to a = a & b
  - a |= b is equivalent to a = a | b
  - a ^= b is equivalent to a = a ^ b
  - a <<= b is equivalent to a = a << b
  - a >>= b is equivalent to a = a >> b
  - a >>>= b is equivalent to a = a >>> b



## Comparison Operators

- > Returns true if the first value is greater than the second
- >= Returns true if the first value is greater than or equal to the second
- < Returns true the first value is less than the second
- <= Returns true if the first value is less than or equal to the second
- == Returns true if first value is equal to second
- != Returns true if first value is not equal to second



## Logical Operators

- ! Returns true if its operand is zero or false
- && Returns false if either operand is zero or false
- || Returns false if both operands are zero or false



## Strings

- Strings are identified using double quotes, e.g., "This is a JavaScript string."
- Two strings can be concatenated using the '+', e.g., "David " + "Tarnoff" equals "David Tarnoff"
- Some characters are interpreted as white space or act as control characters and require an alternative method of being entered. These are called ***escape characters***.



## Escape Characters

\n newline
\t tab
\r carriage return
\\ backslash
\" double quote
\' single quote

## Functions

- JavaScript allows for the definition of functions in the case of repeated operations or operations that are too large to be embedded into a tag (We'll discuss embedding functions into a tag later).
- There are three important things you need to include with a function:
  - the function's name,
  - any parameters passed to the function, and
  - the code to execute.
- The general format is shown below.

```
function functionName(passed parameters)
{
      // statements to execute
}
```



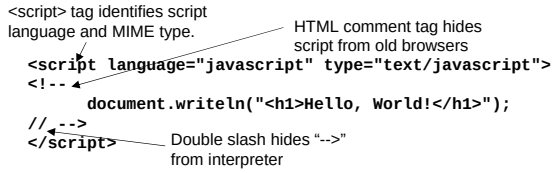
## Function (continued)

- The word "function" above indicates that we are creating a function.
- The string *functionName* is the name of the function. We need this so we know what to call when we are trying to execute the function.
- The items between the parenthesis should be a list of the variables being passed to the function. The items should be separated with commas.
- The curly brackets ({ and }) surround the code we are going to be using to create the function.



## Inserting JavaScript into XHTML

- Except for some special cases which will be discussed later, JavaScript should be inserted into an XHTML file between <script>...</script> tags.






## JavaScript in Web Pages

There are two ways JavaScript scripts are executed in web pages

- As the page is loaded, embedded scripts are executed and the script's output is inserted where the script occurred in the page
- If an event occurs that is associated with a script, the script is executed



## Where to Insert JavaScript

There are 5 places where scripts may be inserted:

- In the head of the page: between <head> tags
  - Scripts in the header can't create output within the HTML document, but can be referred to by other scripts.
  - Header is often used for functions.
- In the body of the page: between <body> tags
  - Output from scripts contained in the body is displayed as part of the HTML document when the browser loads the page



## Where to Insert JavaScript (continued)

- After the body of the page (after </body>)
  - In this case, the script can see all XHTML objects in a body tag
  - Example: Used to place a cursor in a field
- Within an XHTML tag, such as <body>, <a>, <input>, <img> or <form>
  - This is called an event handler and allows the script to work with HTML elements.
  - In this case, no <script> tag is used.
- In a separate file

## Events

- Sometimes, you will need to execute a section of JavaScript code in response to an event.
- Events are things that happen to an object.
  - For example, assume we have an object "person".
  - An event might be that their eyes become dry. What would they do? Blink!

```
person.onDryEyes = blink();
```

  - The object in this example is "person".
  - "blink()" is a method or function.
  - The event is "onDryEyes".



## Events (continued)

- Examples of events for your HTML pages include:
  - onLoad
  - onClick
  - onMouseOver
  - onMouseOut
- Each of these events can execute a script.
- Events must be placed in a tag. For example:

```
<a href="somesite.htm" onClick =
"javascript:function();">link text</a>
```



## Object Models

JavaScript provides access to a number of different components on the client's side:

- HTML elements
- Browser information
- JavaScript-specific objects



## Object Models (continued)

- As stated earlier, JavaScript is an object-based language. Can you name some objects related to the client viewing a page?
- To support standard implementation across all browsers and applications, a set of object models has been created for use with JavaScript.
  - Browser Object Model (BOM)
  - Document Object Model (DOM)



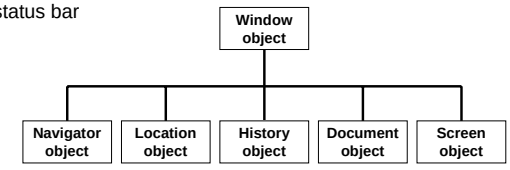
## Browser Object Model

- The BOM defines the components and hierarchy of the collection of objects that define the browser window.
- For the most part, we will only be working with the following components of the BOM.
  - window object
  - location object
  - history object
  - document object
  - navigator object
  - screen object



## Window Object

- Top level in the BOM
- Allows access to properties and method of:
  - display window
  - browser itself
  - methods thing such as error message and alert boxes
  - status bar



## Document Object

- Top of the Document Object Model (DOM).
- This is probably the one you'll use most.
- Allows access to elements of the displayed document such as images and form inputs.
- The root that leads to the arrays of the document: forms[ ] array, links[ ] array, and images[ ] array.
- Least compliance to standard here – Netscape 6, Opera 6, and Mozilla 1.0 are the best.



## Navigator Object

- Provides access to information and methods regarding the client's browser and operating system.
- Commonly used to determine client's browser capabilities so page can be modified real time for best viewing.
- Example: A script may check the browser type in order to modify CSS styles.



## History Object

- Provides access to the pages the client has visited during the current browser session.
- Methods such as back() and forward() can be used to move through the history.
- Can also be used to jump to any point in the history.
- As with any browser history, it only allows for a single path.



## Other BOM Objects

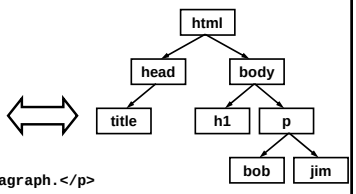
- Location Object – Provides access to and manipulation of the URL of the loaded page.
- Screen Object – Provides access to information about the client's display properties such as screen resolution and color depth.
- More information can be found at:
  http://javascript.about.com/library/bltut22.htm
  http://www-128.ibm.com/developerworks/web/library/wa-jsdom/



## Document Object Model

- Document is modeled as a tree.
- DOM Changes based on page displayed. Example:

```
<html>
<head>
  <title>My Page</title>
</head>
<body>
  <h1>My Page</h1>
  <p name="bob" id="bob">
    Here's the first
    paragraph.</p>
  <p name="jim" id="jim">
    Here's the second paragraph.</p>
</body>
</html>
```




- Another example can be found at:
  http://oopweb.com/JavaScript/Documents/jsintro/Volume/part2/part2.htm