JavaScript

CS 4640
Programming Languages for Web Applications
How HTML, CSS, and JS Fit Together

Content layer
The HTML gives the page structure and adds semantics

Presentation layer
The CSS enhances the HTML page with rules that state how the HTML content is presented

Behavior layer
The JS controls how the page behaves, adding interactivity
JavaScript: Some History

- JavaScript was *introduced* as part of the Netscape 2.0 browser
- Microsoft soon released its own version called *Jscript*
- ECMA developed a standard language known as *ECMAScript*
- *ECMAScript Edition 6* is widely supported and is what is commonly called “JavaScript”
- JavaScript = ECMAScript + DOM API

![JavaScript Timeline](image-url)
Developing JavaScript Software

• **Writing** JavaScript code
  • Any text editor (e.g., Vim, Notepad, Emacs)
  • Specialized software (e.g., MS Visual InterDev)

• **Executing** JavaScript
  • Load into browser (need HTML document)
  • Browser detects *syntax* and *run-time* errors
    • Firefox: JavaScript console lists errors
      • Window (Ctrl-Shift-J)
      • Mac (Command-Shift-J)
    • IE: Exclamation icon and pop-up window (**Bottom left**)
JavaScript

- JavaScript is **not** Java

- JavaScript is a **scripting** language:
  - **Embedded** in HTML
  - **Interpreted** as the page is loaded
  - Can **manipulate** the HTML page

- Primary purpose is for client-end processing of HTML documents
  - Netscape’s Livewire, first server-side JS engine, allows JS to be used for form processing on the server
  - We will not discuss server side JS

- **No type checking**
JavaScript Characteristics

• **JavaScript** does not need to be compiled
  • JS is an *interpreted* language
  • A JS interpreter is software that runs inside a browser that reads and executes JavaScript

• Interpreted vs. compiled languages :
  • Advantage: *simplicity*
  • Disadvantages: *efficiency, maintainability, scalability, reliability*
Why and Why Not JavaScript?

• What can be done with JS on the client and cannot be done with other techniques on the server?
  • Monitor user events and take action
  • Some dynamic effects

• What can be done on both client and server, but are better with JS?
  • Build HTML dynamically when page is loaded
  • Interactive web pages
  • Communicate with the server asynchronously (Ajax)

• What are the drawbacks of JS?
  • Platform dependent
  • Can be turned off
  • Performance
  • Security
  • Hard to write reliable and maintainable JS
Embedding JS in HTML

Use the `<script>` tag to embed JS code in `<head>` or `<body>`

```html
<script type="text/javascript">
    // code goes here
</script>
```

- Functions and code that may execute multiple times are typically placed in the `<head>` section
  - These are only interpreted when the relevant function or event-handler are called

- Code that needs to be executed only once, when the document is first loaded is placed in the `<body>` section
Embedding JS in HTML

• JS code may be written in a separate file. The external file can be included in an HTML file using the `src` attribute of a `<script>` tag

```html
<script type="text/javascript" src="path/to/file.js"></script>
```

• JS code is **visible** in the client browser
  • Do not “hardcode” anything that you don’t want the client to see
Script Calls

Some script calls may be embedded in the HTML tags

```html
<select name="country" onchange="jmp(url)">

or

<a href="javascript:newWindow('resources/JSPWebResources.html')">JSP resources</a>
```

```javascript
function newWindow(url)
{
    hWnd = window.open (url,"HelpWindow","width=410,height=180,resizable=yes,scrollbars=yes");
}
```

Script is evaluated once encountered by browser
First Example

```html
<!DOCTYPE html>
<html>
<head>
<title>First JavaScript Example</title>
</head>
<body>
<h2>This line is straight HTML</h2>
<h3>
<script type = "text/javascript">
    document.write("These lines are produced by<br/>");
    document.write("the JavaScript program<br/>");
    alert("Hey, JavaScript is fun!");
</script>
</h3>
<h2>More straight HTML</h2>
<script type = "text/javascript" src="file.js"></script>
</body>
</html>
```

[see jex1.html]
Variables

• Variables are **loosely** typed

• Type is **determined dynamically** based on the value stored
  • The `typeof` operator can be used to check type of a variable

• Declarations are made using the `var` keyword
  • Variables declared but not initialized have the value `undefined`
  • Variables used and not declared or initialized have value `Null`

• Names start with letters, $, or _ followed by any sequence of letters, $, _, or digits

• Case sensitive
Variables

• Variable is a container that hold things for later use

• String
  ```javascript
  var strVar = 'Hello';
  ```

• Number
  ```javascript
  var num = 10;
  ```

• Undefined
  ```javascript
  var undefined;
  ```

• Null
  ```javascript
  var nulled = null;
  ```

• Objects (including arrays)
  ```javascript
  var intArray = [1, 2, 3];
  ```

• Symbols
  ```javascript
  var sym = Symbol('Description of the symbol');
  ```

• Functions
  ```javascript
  function setFocus()
  {
    document.focus.firstName.focus();
  }
  ```
Variables

• Loose typing means that JS figures out the type based on the value

```javascript
var x;       // type: Undefined
x = 2;       // type: Number
x = 'Hi';    // type: String
```

• Variables have block scope.
  • Declarations outside of any function are global
  • Declarations within a function are local to that function
Expressions

- If operator is + and an operand is string, it treats the + as a string concatenation operator and coerce other operand to string

  ```javascript
  var x = 'Hello';
  var y = 4;
  var result = x + y;  // 'Hello4'
  ```

- If operator is arithmetic, and string value can be coerced to a number, it will do so

- If string is non-numeric, result is NaN (NotaNumber)

- String can be explicitly converted to a number using parseInt and parseFloat

[see jex2.html]
Using Arithmetic Operators

- Let’s initialize, \( y = 4 \)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Resulting ( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>( x = y + 5 )</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( x = y + &quot;5&quot; )</td>
<td>“45”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( x = &quot;Four&quot; + y + &quot;4&quot; )</td>
<td>“Four44”</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>( x = y - 2 )</td>
<td>2</td>
</tr>
<tr>
<td>++</td>
<td>Increment</td>
<td>( x = y++ )</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( x = ++y )</td>
<td>5</td>
</tr>
<tr>
<td>--</td>
<td>Decrement</td>
<td>( x = y-- )</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( x = --y )</td>
<td>3</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>( x = y * 4 )</td>
<td>16</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>( x = 10 / y )</td>
<td>2.5</td>
</tr>
<tr>
<td>%</td>
<td>Modulo</td>
<td>( x = y % 3 )</td>
<td>1</td>
</tr>
</tbody>
</table>
Using Assignment Operators

- Let’s initialize, \( x = 10 \)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent arithmetic operators</th>
<th>Resulting x</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>( x = 5 )</td>
<td>( x = 5 )</td>
<td>5</td>
</tr>
<tr>
<td>+=</td>
<td>( x += 5 )</td>
<td>( x = x + 5 )</td>
<td>15</td>
</tr>
<tr>
<td>-=</td>
<td>( x -= 5 )</td>
<td>( x = x - 5 )</td>
<td>5</td>
</tr>
<tr>
<td>*=</td>
<td>( x *= 5 )</td>
<td>( x = x * 5 )</td>
<td>50</td>
</tr>
<tr>
<td>/=</td>
<td>( x /= 5 )</td>
<td>( x = x / 5 )</td>
<td>2</td>
</tr>
<tr>
<td>%=</td>
<td>( x %= 5 )</td>
<td>( x = x % 5 )</td>
<td>0</td>
</tr>
</tbody>
</table>
Applying Comparison and Conditional operators

• Let’s initialize, \( x = 10 \)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example 1</th>
<th>Example 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal to (value only)</td>
<td>( x == 8 )</td>
<td>( x == 10 )</td>
<td>false</td>
</tr>
<tr>
<td>===</td>
<td>Equal to (value and type)</td>
<td>( x === 10 )</td>
<td>( x === &quot;10&quot; )</td>
<td>true</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal (value only)</td>
<td>( x != 5 )</td>
<td>( x != &quot;10&quot; )</td>
<td>true</td>
</tr>
<tr>
<td>!==</td>
<td>Not equal (value and type)</td>
<td>( x !== &quot;10&quot; )</td>
<td>( x !== 10 )</td>
<td>true</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>( x &gt; 5 )</td>
<td>( x &gt;= 10 )</td>
<td>true</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
<td>( x &gt;= 10 )</td>
<td>( x &lt; 5 )</td>
<td>false</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>( x &lt; 5 )</td>
<td>( x &lt;= 10 )</td>
<td>true</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>( x &lt;= 10 )</td>
<td>( x &lt;= 10 )</td>
<td>true</td>
</tr>
</tbody>
</table>
Chaining Multiple Comparisons with Logical Operators

- Let’s initialize, \( x = 10 \) and \( y = 5 \)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>And</td>
<td>((x == 10 &amp;&amp; y == 5))</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>((x == 10 &amp;&amp; y &gt; x))</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>((x &lt; 10 &amp;&amp; y &gt; x))</td>
<td>false</td>
</tr>
<tr>
<td>!</td>
<td>Not</td>
<td>!(x == y)</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>!(x &gt; y)</td>
<td>false</td>
</tr>
<tr>
<td>Mix</td>
<td></td>
<td>((x &gt;= 10 &amp;&amp; y &lt; x</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(((x &lt; y</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( !(x == y) &amp;&amp; y &gt;= 10)</td>
<td>false</td>
</tr>
</tbody>
</table>
Control Structures (if Statement)

```java
if (x == 5) {
    do_something();
}

if (x == 5) {
    do_something();
} else {
    do_something_else();
}

if (x < 5) {
    do_something();
} else if (x < 10) {
    do_something_else();
} else {
    do_nothing();
}
```
Control Structures (switch Statement)

switch (expression) {
    case value1:
        // code to execute
        break;
    case value2:
        // code to execute
        break;
    default:
        // code to execute if not value1 or value2
}
Looping (while Loop)

```javascript
while (condition) {
    // code to execute
    // update to end the loop
}

var i = 1;
while (i < 5) {
    // code to execute
    i++;
}
```
Looping (do-while Loop)

do {
    // code to execute
} while (condition);

var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
var i = 0;
do {
    var day = days[i++];
    console.log("It's " + day);
} while (day != "Wednesday");
Looping (for Loop)

```javascript
for (assignment; condition; update;) {
    // code to execute
}

var cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];
var text = "";
var i;
for (i = 0; i < cars.length; i++) {
    text += cars[i] + "<br>";
}
```
Looping (for-in Loop)

```javascript
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]; for (var idx in days) {
    console.log("It's " + days[idx] + "<br />");
}
```
Array Objects

• More relaxed version of Java arrays
  • Size can be changed and data can be mixed
  • Cannot use arbitrary keys as with PHP arrays - index

• Creating arrays
  • Using the `new` operator and a constructor (`Array`) with multiple arguments
    ```javascript
    var A = new Array("hello", 2, "you");
    ```
  • Using the `new` operator and a constructor (`Array`) with a single numeric argument, assigned the size of array
    ```javascript
    var B = new Array(50);
    ```
  • Using square brackets to make a literal
    ```javascript
    var C = ["we", "can", 50, "mix", 3.5, "types"];
    ```
    ```javascript
    var D = [50];
    ```
Array Predefined Operations

- **Concat**: two arrays into one – copies to end
- **Join**: array items into a single string (commas between)
- **Push, pop, shift, unshift**: easy implementation of stacks and queues
  - Push and pop are a “right stack” – add/remove from end
  - Shift and unshift are a “left stack” – add/remove from beginning queue
- **Sort**: 
  - Sort by default compares using alphabetical order
  - To sort using numbers we pass in a comparison function defining how the numbers will be compared
- **Reverse**: the items in an array

[see jex4.html]
Functions

- Similar to Java functions but header is somewhat different
  
  ```javascript
  function add(num1, num2) {
    return num1 + num2;
  }
  
  var num = add(4, 6);
  ```

- Return type not specified (like PHP, since JS has dynamic typing)
- Parameter types also not specified
- Functions execute when they are called, just as in any language
- To allow this, function code should be in the `<head>` section
Anonymous Functions and Function Expressions

• Functions can be assigned to variables

```javascript
var magic = function(num1, num2) {
    return num1 + num2;
}
var myNum = magic(4, 6);  // “Function expression”
```

• Variables declared in a function are local to the function

• Parameters are all value
  • No parameter type-checking

[see `jex3.html`]
Immediately Invoked Function Expressions

• Anonymous functions can be executed once as the interpreter comes across them

```javascript
var magic = (function(num1, num2) {
  return num1 + num2;
}());
```

Parentheses tell the interpreter to call the function immediately

Grouping operators tell the interpreter to treat this as an expression
Functions and Default Values (ES6)

```javascript
function add(num1=10, num2=45) {
    return num1 + num2;
}
var r = add(); // 55
r = add(40); // 85
r = add(2, 6); // 8
```
Global and Local Scopes

// show size of the building plot
function showPlotSize(width, height) {
  return 'Area: ' + (width * height);
}
var msg = showPlotSize(3, 2);

// show size of the garden
function showGardenSize(width, height) {
  return width * height;
}
var msg = showGardenSize();

Naming collision
  • Two JavaScript files, both have a global variable with the same name