JavaScript

What is JavaScript?

- JavaScript is one of a new breed of Web languages called *scripting languages*.
- Simple language that can be used to add extra features to make dynamic Web pages.
- Java is intended for programmers, scripting languages make it easy for nonprogrammers to improve a Web page.
- JavaScript was originally developed by Netscape Corporation for use in its browser, Netscape Navigator.

What is JavaScript?

- A convenient syntax, flexible variable types, and easy access to the browser's features.
- Run on the browser without being compiled; the source code can be placed directly into a Web page.
- Can program in JavaScript easily; no development tools or compilers are required.
- Can use the same editor you use to create HTML documents to create JavaScript, and it executes directly on the browser (currently, Netscape or Microsoft Internet Explorer).

What is JavaScript?

- JavaScript was originally called LiveScript, and was a proprietary feature of the Netscape browser.
- JavaScript has now been approved by Sun, the developer of Java, as a scripting language to complement Java.
- Can work directly with HTML elements in a Web page, something Java can't handle.

JavaScript vs Java

- JavaScript can be combined directly with HTML.
- The JavaScript language structure is simpler than that of Java.
- The JavaScript interpreter is built into a Web browser.
- JavaScript is supported on more platforms than Java.

Combining JavaScript with HTML

- Java applets are compiled and stored on the server as byte codes, but JavaScript programs are simple ASCII text files.
- JavaScript functions can been kept as separate files or included within an HTML page.
- The *<SCRIPT>* tag, an extension of HTML supported by Netscape, enables JavaScript functions to be included in the page.

Example

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
  <TITLE>First JavaScript Page</TITLE>
</HEAD>
<BODY>
<H1>First JavaScript Page</H1>
<SCRIPT TYPE="text/javascript">
<!--
document.write("<HR>");
document.write("Hello World Wide Web");
document.write("<HR>");
// -->
</SCRIPT>
</BODY>
</HTML>
```


Example

Simplified Language Structure

- JavaScript is interpreted rather than compiled.
- Changing a script is as simple as changing an HTML page.
- Rather than creating objects and classes, existing objects in JavaScript can be easily accessed.
- Variables are *loosely typed*: Variables need not be declared before their use, and most conversions (such as numeric to string) are handled automatically.

Simplified Language Structure

- Event handlers enable a JavaScript function to execute whenever an action is performed on part of the HTML document. For example, a form's input field can be checked for certain values whenever it is modified.

Web Browser Integration

- JavaScript is an *object-oriented* language.
- JavaScript also includes objects that enable you to access features of the browser directly. These objects represent actual elements of the browser and the Web page, such as windows, documents, frames, forms, links, and anchors.
- Information about the links, anchors, and form elements in the current page can be accessed by the JavaScript.

Web Browser Integration

- JavaScript can e used to control the browser. For example, it's possible to include a "back" button on your page that will send the user back to the previous page-just like the browser's back-arrow button.

Uses for JavaScript

- **Including Dynamic Information**
  - JavaScript can be used to add a bit of life to a Web page by making some of the text dynamic.
- **Validating Forms**
  - The user enters data into the form, then presses the Submit button, and the server, using a CGI program, responds to the information. This is useful, but it isn't very interactive-before you can receive a response, you have to wait for the data to travel to the server and back.

Uses for JavaScript

- **Making Pages Interactive**
  - **Finally, JavaScript can be used to remove some of the drudgery from a normal Web page by giving the user some control of the page.**
    - **For example, you could have a background on/off button to control the display of the background, or a button to toggle the display of a table of contents.**

Receiving Web Page

1. Enter a URL into browser, or select a bookmark.
2. The browser sends an HTTP request for the URL to the appropriate server (known).
3. The server sends back the contents of the Web page at the URL.
4. The browser sends additional requests for each of the graphics page.
5. After receiving enough information about the graphics to devote the correct amount of space to them, the browser displays the page.

Processing Script

- If the script is included in the header, it is ignored unless a script later calls it.
- If the script is included directly in the body, its output will be included in the Web page-thus, it can affect the display of the page.
- If the script is an event handler for a specific part of the page, it will be processed only when the event happens.

Limitations

- Working with Entire HTML Pages
  - JavaScript program cannot replace part of the currently loaded HTML page
- Lack of Network Communication
  - JavaScript has no facilities for communicating between the Web browser and the HTTP server. This means that although JavaScript can manipulate data the user enters in a form, the resulting data cannot be sent back to the server.

Limitations

- Limited Graphics and Multimedia Capabilities
  - JavaScript is a scripting language, so it it cannot be used to crate a multimedia application.
- Limited Size of Scripts and Speed
  - The early implementations of JavaScript (in Netscape Navigator) required that the scripts you use for a Web page be included in the HTML for that page. This means that there was a practical limitation of about 32K for the page and all scripts, because the browser must download the entire page before executing the script.

JavaScript Structure

```
<SCRIPT LANGUAGE="JavaScript">
<!--
   document.write("Hello!");
   window.alert("Hello again!");
// -->
```


JavaScript Structure

- Statements
  - *Statements* are simply commands that perform a certain purpose.
  - The term *statement* is also used to refer to any line of JavaScript code.
- Functions
  - A *function* accepts *parameters* and returns a value.
  - Built-in functions
  - User-defined functions

JavaScript Structure

- Variables
  - A *variable* is simply a name given to a value.
- Expressions
  - An *expression* is something that JavaScript interprets before using it in a statement.
- Objects, Properties, and Methods

Comments

- A single-line comment begins with two slashes (//) and ends at the end of the line.
- A multiple-line comment begins with the /* delimiter and ends with the */ delimiter. This type of comment can include any number of lines. These work like the comments in the C++ language.

Data Types and Variables

The four fundamental data types used in JavaScript are the following:

- *Numbers*, such as 3, 25, or 1.4142138. JavaScript supports both integers and floating-point numbers.
- *Boolean*, or logical values. These can have one of two values: true or false.
- *Strings*, such as "This is a string". These consist of one or more characters of text.
- *The null value*, represented by the keyword null. This is the value of an undefined variable.

Data Types

- JavaScript is a *loosely typed* language.
- This means that a variable is declared without type specified.
- Any allowable type of data can be stored in a variable. Conversions between different types of data happen automatically.

Integers

- An *integer* is simply a number that does not include a decimal.
- *Integers* in JavaScript can be only positive numbers.
- An integer as a literal in JavaScript simply a number.
  - For example, this statement prints the number 47:

```
document.write(47);
```

- JavaScript considers any number without a leading zero to be a decimal (base 10) number.

Integers

- Different types of numbers :
  - Decimal: no leading zero (57, 5000)
  - Hexadecimal: prefix with 0x (0x56, 0xFE)
  - Octal: leading 0 (045, 013)

Floating Point

- Floating-point values can be either positive or negative.
- Scientific notation can be used to refer to floating-point numbers.

Boolean

- Boolean values can contain one of two values: true or false.
- Numbers 0 and 1 can be used as synonyms for false and true in many cases in JavaScript; however, it's best to treat Boolean values as strictly Boolean.

Strings

- A *string* is simply a group of characters
- Strings as literals in JavaScript are enclosing in double or single quotation marks.
  - `"This is a string."`
  - `'A'`
  - `'25 pounds'`
  - `"200"`

Special Characters

- `\a`: Alert (bell) character (produces a bell sound)
- `\b`: Backspace character (moves the cursor back one character)
- `\f`: Form-feed character (indicates a new page on a printer)
- `\n`: New line charcter (indicates a new line of text)
- `\r`: Carriage return character (moves the cursor to the beginning of the line)
- `\t`: Tab character (advances the cursor to the next tab stop)

Examples

```
document.write(" This is line 1.\n This
  is line 2.\n This is line 3.");

document.write("To output double
  quotations, \"escape character,\" is
  needed.");

document.write("The DOS files are in the
  C:\\DOS\\FILES directory.");
```


Arrays

- JavaScript does not support *arrays* as variables.
- They are objects.
- An array can be created by using the *Array* object.

```
TestScores = new Array(5);

TestScores[0]=89;
TestScores[3]=TestScores[0];
```


Numbers and Text

```
quantity = 3;
   //quantity declared as integer
description = "•red juicy watermelons";
   //description declared as string
quantity += 1.5;
   //quantity becomes floating-point -- 4.5
quantity +=.5;
   //quantity becomes integer -- 5
quantity = quantity + description;
   //quantity becomes string -- "5 red juicy
     watermelons";
```


Naming and Declaring Variables

Rules for JavaScript Variable Names

- Can include letters of the alphabet, both upper- and lowercase including underscore character.
- Can also include the digits 0-9.
- Spaces or any other punctuation character are not allowed.
- The first character of the variable name must be either a letter or the underscore character.
- Variable names are case-sensitive; totalnum, Totalnum, and TotalNum are different variable names.
- There is no official limit on the length of variable names, but they must fit within one line.

Assignment Statement

```
quantity = 3;

description = "•red juicy watermelons";

quantity = quantity + description;

Type?
```


Scope of Variables

- A variable declared within a function with the var keyword is *local* to that function; other functions cannot access it. Variables used to store the function's parameters are also local to the function.
- A variable declared outside a function, or without the var keyword in a function, is *global*; all JavaScript statements in the current HTML or script file can access the value.

Example

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
   <TITLE>Variable Decelarations</TITLE>
<SCRIPT LANGUAGE="javascript">
var boys = 20;
var men = 10;
girls = 15;
function init()
{
   var women = 25;
   children = boys + girl;
}
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```


Expression and Operators

- *Assignment operators* are used to assign a value to a variable, or change its current value.
- *Arithmetic operators* are used to perform addition, subtraction, and other math on numeric literals or variables.
- *String operators* are used to manipulate string values.
- *Logical operators* deal with Boolean values, and are mainly used for testing conditions.

Expression and Operators

- *Bitwise operators* are used to manipulate individual bits of a numeric value in a Boolean fashion.
- *Comparison operators* are used to test the relationship between two numeric values.

Assignment Operators

- += adds the number on the right to the variable on the left.
- –= subtracts the number on the right from the variable on the left.
- *= multiplies the variable by the number on the right.
- /= divides the variable by the number on the right.
- %= uses the modulo operator, described in the next section.

Arithmetic Operators

- Basic operators: +, –, ×, ÷
- % modulo operator
- + concatenation for strings
- Logical operator
  - && (And) returns true if both of the operands are true.
  - || (Or) returns true if either of the operands is true.
  - ! (Not) returns the opposite of the variable it prefixes.

Bitwise Operators

- And (&) returns one if both of the corresponding bits are one.
- Or (|) returns one if either of the corresponding bits is one.
- Xor (Exclusive Or) (^) returns one if either, but not both, of the corresponding bits is one.
- Left shift (<<) shifts the bits in the left operand a number of positions specified in the right operand.
- Right shift (>>) shifts to the right, including the bit used to store the sign.
- Zero-fill right shift (>>>) fills to the right, filling in zeros on the left.

Comparison Operators

- Less than (<)
- Greater than (>)
- Greater than or equal to (>=)
- Less than or equal to (<=)
- Equal to (=)
- Not equal to (≠)

Precedence of Operators

- Comma (,), used to separate parameters
- Assignment operator (=)
- Conditional operators (?, :)
- Logical OR (||)
- Logical AND (&&)
- Bitwise OR (|)
- Bitwise XOR (^)
- Bitwise AND (&)
- Equality (==) and inequality (!=)
- Comparison operators (<, <=, >, >=)
- Bitwise shift (<<, >>, >>>)
- Addition and subtraction (+, -)
- Multiplication, division, and modulo (*, /, %)
- Negation (!, ~, -), increment (++), and decrement (--)
- Function call or array index ((), [])

Converting and Evaluating Variables and Expressions

- **The parseInt Function**
  - `The parseInt()` function looks for an integer number as the first part of the string.
  - It ignores the decimal portion, if found.

```
a = parseInt("39 steps");
```

  - An optional second parameter of `parseInt()` specifies the base of number in the string.
  - This can be 8 (octal), 10 (decimal), or 16 (hexadecimal) you can use any numeric base.

Converting and Evaluating Variables and Expressions

- `a = parseInt(text,16);`
  `parseInt()` returns a special value, "NaN", (Not a Number) if it encounters a non-numeric character at the beginning of the string.

parseFloat() Function

- The `parseFloat()` function is similar to `parseInt(),` but works with floating-point values.
- It finds a decimal floating-point number in the string, and returns it.

```
a = "2.7178 is the base of a natural
logarithm.";
```

  - This statement can handle all the components of a floating-point number: signs, decimal points, and exponents.

eval() Function

- The `eval()` function has many uses in sophisticated programming techniques,
- `eval()` looks for any valid JavaScript expression.

```
a = eval("20 + 1 + 4");
```

- Any JavaScript expression can be used in the string passed to eval()-numeric operations, comparison operations, statements, multiple statements, and even entire JavaScript functions.

eval() Function

```
var text = "Fred";
var statement = text + "= 31";
eval(statement);
```