Fundamentals of Web Programming [a]

JavaScript: Client Programming and Example Scripts

Teodor Rus

rus@cs.uiowa.edu

The University of Iowa, Department of Computer Science

[a] Copyright 2009 Teodor Rus. These slides have been developed by Teodor Rus using material published by R.W.Sebesta, *Programming the World Wide Web* , Addison Wesley 2009. They are copyrighted materials and may not be used in other course settings outside of the University of Iowa in their current form or modified form without the express written permission of the copyright holder. During this course, students are prohibited from selling notes to or being paid for taking notes by any person or commercial firm without the express written permission of the copyright holder.

Content

- Control Statements;
- Object Creation and Modification;
- Arrays and Constructors;
- An Example;
- Functions;
- Pattern Matching Using Regular Expressions;
- Error and Scripts.

Control Statements

- Selection statement;
- Switch statement;
- Loop statements;

Selection Statements

- `if Expr then Statement`
- `if Expr then Statement else Statement`
- **Example:**

```
if (a > b)
  document.write("a is greater than b <br />");
else
  {
   a = b;
   document.write("a was not greater than b <br />");
   document.write("Now they are equal <br />");
  }
```

Example: `controlS.html`

Switch Statement

```
witch (expression)
     {
      case value 1:
       // statement(s) [break];
      case value 2:
       // statement(s) [break];
      ...
      [default:
       // statement(s) [break];
     }
```

Semantics of Switch

- The expression is evaluated;
- The value of the expression is compared with case values. If one matches control is transferred to its statement(s);
- Execution continues through the remaining cases; If this is not intended, case statements need to be terminated with break (as in C);
- If no case value match, default (if present) is executed.

Switch Demo

```
<?xml version = "1.0" encoding = "utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- switch.html: a demo of the switch -->

<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> JavaScript Switch Demo </title> </head>
   <body>
     <script type = "text/javascript" src = "bordersize.js" >
     </script>
   </body>
</html>
```

Loop Statements

1. **While loop:** syntax:

```
while (ControlExpression)
     { Statement }
```

   Semantic: the same as C while loop;

2. **For loop:** syntax:

```
for (Initialize; ControExpression; Increment)
   { Statement }
```

   Semantic: the same as C for loop.

3. **Do-while loop:** syntax

```
do {Statement} while ControlExpression
```

   Example:
   do count++; sum = sum + (sum * count); while (count <= 50);
   Semantic: the same as C do-while loop

More Loops

4. **For-in loop:** syntax

```
for (identifier in Object)
   Statement

Example:
for (var prop in myCarObject)
   document.write("Name: ",prop,";Value: ",myCar[prop],"<br />");
```

lists all properties and values of object myCarObject

Example Loop

Printing dates sand checking time needed to print date.

```
<?xml version = "1.0" encoding = "utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- date.html: a demo of the for loop and Date object -->

<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> JavaScript Switch Demo </title> </head>
   <body>
     <script type = "text/javascript" src = "date.js" >
     </script>
   </body>
</html>
```

Object Creation and Modification

- Objects are created by a `new` expression that implies a call to Object constructor.

  Note: in JavaScript `new` operator creates the object and initializes properties;

  (**Note:** in Java `new` initializes members but does not create them);

- Properties of an object are accessed using dot notation in which first is the object name followed by dot followed by property/method;

- Number of properties in a JavaScript object is dynamic: at any time during interpretation, properties can be added to or deleted from an object.

Example Object Creation

```
// Create an Oject object
var myCar = new Object();
// Create and initialize the make propery
myCar.make = "Ford";
// Create and initialize model
myCar.model = "Contour SVT";
```

Note: an object and its properties can be created by an abbreviated statement that contains no new operator nor call to the constructor.

Example:

```
var myCar = {make: "Ford", model: "Contour SVT"}
```

Objects Are Nested

One can create a new object that is a property of a created object.

Example:

```
myCar.engine = new Object();
myCar.engine.config = "V6";
myCar.engine.hp = 200;
```

Property names of an object can be accessed as if they were the elements of an array, using property name (a string literal) as subscript.

Example:

```
var prop1 = myCar.make; // using dot notation
var prop2 = myCar["make"]; // using array notation
```

Note: array notation is used in for-in loop.

Accessing a property that does not exits return "undefined"

Arrays

- In JavaScript arrays are object that have special functionality.
- Array elements can be primitive values or references to other objects, including other arrays.
- JavaScript arrays have dynamical length.

Array Object Creation

Array objects can be created in two ways:

1. The usual way using new operator and a call to Array constructor:

   ```
   var myList = new Array(1, 2, "three", four");
   var yourList = new Array(100);
   ```

   First statement creates and initializes an array object with 4 elements;
   Second statement creates an array object of length 100, no elements.

2. Second way to create an array object is using a literal array:

   ```
   var myList2 = [1, 2, "three", "four"];
   ```

   creates an array object with the same elements as `myList`.

Characteristics

JavaScript array objects have the following characteristics:

1. The lowest index of every JavaScript array is zero (0);
2. Array element access is specified with numeric expression in brackets;
3. Length of an array is the highest subscript to which a value was assigned plus 1;
4. Length of an array is read/write accessible through `length` property;

More Characteristics

5. Only the assigned elements of the array actually occupy space. Hence, the length property of an array is not necessarily the number of its allocated elements;
6. All array elements are allocated dynamically from the heap. Assigning a value to an array element that did not previously exists creates that element.

Example Array

```
<?xml version = "1.0" encoding = "utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- array.html: a demo of the array object -->

<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> JavaScript Array Demo </title> </head>
   <body>
     <script type = "text/javascript" src = "array.js" >
     </script>
   </body>
</html>
```

Array Methods

1. `join()`: converts all elements of an array to strings and catenate them into a string. If a string parameter is provided it is used as separator in the new string.

   Example:

   ```
   var names = new Array ("Mary", "Murray", "Murphy", "Max");
   var nameString = names.join(":"):
   ```

   value of nameString is: `"Mary:Murray:Murphy:Max"`

2. `reverse()`: reverses element order of the Array object.

   Example:

   ```
   var newarray = names.reverse();
   value of newarray is ["Max", "Murphy", "Murray", "Marry"]
   ```

More Methods

3. `sort()`: coerce array elements to strings if they are not strings) and sort them alphabetically.

   Example `names.sort();` make value of names array:

   `["Mary", "Max", "Murphy", "Murray"]`

   Note: if one needs to sort something other than strings, or to sort in another order, the comparison operation, say $\prec$, must be supplied as parameter.

   (a) $elem1 \prec elem2$ must return negative if $elem1, elem2$ are in the desired order;

   (b) $elem1 \prec elem2$ must return 0 if $elem1 = elem2$;

   (c) $elem1 \prec elem2$ must return positive if they need be interchanged.

More Methods

4. `concat()`: catenate its argument to the end of the Array object.

   Example:

   ```
   var newNames = names.concat("Moo", "Meow");//newNames value is:
   //["Mary" "Murray", "Murphy", "Max", "Moo", "Meow"]
   ```

5. `slice()`: returns the part of the array specified by parameters.

   Example:

   ```
   var list1 = [2, 4, 6, 8,10];
   var list2 = list1.slice(1,3) // list2 is [4, 6, 8]
   ```

   If slice() is given only one parameter the result is the array string with the element specified by the given index;

6 `toString()`: has the same effect as `join()`;

More Methods

7. push(), pop(), unshift(), shift() methods allows easy implementation of stack and queues in arrays.
   (i) pop() and push() remove and add (respectively) elements.

   Example:

   ```
   var list = ["Dasher", "Dancer", "Donner", "Blitzen"];
   var deer = list.pop(); // deer is "Blitzen"
   list.push("Blitzen"); // puts "Blitzen" back
   ```

   (ii) shift() and unshift() remove and add an element to the beginning of an array.

   Example:

   ```
   var deer = list.shift() :// deer is now "Dasher"
   list.unshift(); // puts "Dasher" back
   ```

Two Dimensional Array

A two-dimensional array is implemented in JavaScript as an array of arrays.

```
<?xml version = "1.0" encoding = "utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- nestedArray.html: a demo of array of arrays -->

<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> JavaScript Nested Array Demo </title> </head>
   <body>
     <script type = "text/javascript" src = "nestedArray.js" >
     </script>
   </body>
</html>
```

Functions

JavaScript functions are similar to C functions.

- Function definition:

```
function funcName(FuncParameters)
   {
    Function Body
   }
```

  where Function Body may or may not include a return statement

- Function call:
  `funcName(Arguments);` or
  `var = funcName(Arguments);`

- JavaScript functions are objects. Variables that reference functions can be treated as object references.

Example Function

```
function fun()
  { document.write("This surely is fun! <br />");}

// Testing a function
refFun = fun(); // refFun referes to fun object
fun();  // A call to fun()
refFun; // Another call to fun()
```

Example: `funcFun.html`

Calling a Function

- From a link:
  Define the function in the document head;
  Call it from a link in the document body using the form:

  ```
  <a href = "JavaScript:functionName()">
  <big> click here for function call </big>
  </a>
  ```

  See the example `linkcall.html`.

- From an event:
  Define function in the head, call it from an event. See the example `eventcall.html`.

- From a Java Script:
  Define function in the document head. Call it from a JavaScript defined in the document body. See the example `jscall.html`.

Observation

- In all cases, function can be defined in a file, say `funcall.js`, and path to this file can be used in the document using the function;
- **Example:** assuming that `funcall.js` is in the directory `JSscripts` located in the same directory with the XHTML document, the reference to the function definition would be:

```
<script type = "text/javascripti" src = "SJscripts/funcall.js">
```

Facts

1. Because JavaScript functions are objects, their references can be properties in other objects, in which case they act as methods.
2. Interpreter must see a function definition before its call. Consequently function definitions are placed in the head of XHTML document, either explicitly or implicitly.
3. Normally (not always) calls to functions appear in the document body.

Scope of Variables

Scope: text of the program in which a definition holds!

- Variables that are implicitly declared (by the interpreter not with var), even if implicit declaration occur in a function definition, have global scope (they are visible in the entire XHTML document);
- Variables that are explicitly declared outside function definition have global scope;
- Explicitly declared variables in function definition have local scope;
- If a variable that is both local and global in a function, the local variable has precedence (hides the global);
- Do not use nested functions.

Parameters

Formal parameters: are variable that appear in function definition;
Actual parameters: (arguments) variables that appear in function call.

- JavaScript uses pas-by-value parameter passing method;
- Because references can be passed as actual parameters for objects, the calling function has access to the objects and can change them, thus providing pass-by-reference mechanism;
- If a reference to an object is passed and the function changes the corresponding formal parameter (rather than the object it references), it has no effect on the actual parameter.

Example

Suppose an array passed as parameter to a function, as follows:

```
function fun1(myList)
  {
   var list2 = new Array (1, 3, 5);
   myList[3] = 14;
   myList = list2;
  }
var list = new Array(2, 4, 6, 8);
fun1(list);
```

Function operation:

```
myList[3] = 14;// replaces the 8 in list;
myList = list2;// myList refers to a different object;
```

Facts

1. Because of JavaScript dynamic typing there is no type checking of parameters. The calling function can check the parameter type by `typeof`. `typeof` cannot distinguish between different objects;
2. Number of parameter in a function is not checked against the number of formal parameters. Excess parameters passed are ignored; excess formal parameters are set to undefined.
3. All parameters are communicated through the property array `arguments`. By accessing `arguments.length` a function can determine the number of actual parameter passed.
4. Because `arguments` array is accessible directly, all actual parameters specified in the call are available, including actuals that do not correspond to formals.

Example

```
<?xml version = "1.0" encoding = "utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- params.html: a demo of parameter passing -->

<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> JavaScript Parameter Passing Demo </title> </head>
   <body>
     <script type = "text/javascript" src = "JScripts/params.js" >
     </script>
   </body>
</html>
```

Passing by Reference

The elegant way for passing primitive values by references is:

- Put the primitive values in an array and pass the array;
- Because arrays are objects, this works;
- Example: byReference.html

The Demo

```
<?xml version = "1.0" encoding = "utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- byReference.html: a demo of parameter passingi by reference -->

<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> Parameter Passing by Referenbce </title> </head>
   <body>
     <script type = "text/javascript" src = "JScripts/byReference.js" >
     </script>
   </body>
</html>
```

Another Example

Computing the median of an array of numbers.

```
<?xml version = "1.0" encoding = "utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- median.html: a demo for function and array  -->

<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> Function and Array Operations </title> </head>
   <body>
     <script type = "text/javascript" src = "JScripts/median.js" >
     </script>
   </body>
</html>
```

Constructors

JavaScript constructors are special methods that create and initialize the properties of newly created objects.

- Constructors are called by `new` operator follower by object name;
- Variable `this` is predefined reference variable to the object on which constructor works. When the constructor is called `this` is a reference to the newly created object. Example:

```
function car(newMake, newModel, newYear)
   {
    this.make = newMake;
    this.model = newModel;
    this.year = newYear;
   }
```

This function can be used as a constructor:

myCar = new car("Ford", "Capri", "1990");

Object Methods

If a method is to be included in an object it is initialized in the same way as properties are.

Example:

```
function dispalyCar()
   {
    document.write("Car make: ", this.make, "<br />");
    document.write("Car model: " this.model, "<br />");
    document.write("Car year: ", this.year, "<br />");
   }
```

If `this.display = displayCar;` is added to car constructor, then myCar.display() would behave as expected.

Facts

1. Collection of objects created using the same constructor have the same set of properties and methods;
2. They can however diverge through user code;
3. There is no convenient way to determine in a script whether two objects have the same properties and methods.

Pattern Matching

- There are two approaches to pattern matching in JavaScript:
  1. Based on `RegExp` object;
  2. Based on `String` object.

  In both patterns are regular expressions of Perl PL.
- The simplest method is `search()` which take a pattern as parameter and return its position in String object through which it was called, and -1 if no entry.

Note: only string matching is analyzed in the textbook.

Creating Regular Expressions

Regular expression: is a pattern of characters enclosed in forward slashes.

- The literal way: assigns a regular expression to a variables.

  ```
  var variableName = /regular expression/optinons
  ```

  Options are: i, used to ignore case, g used for global (all occurrences) match, m used to match over multiple lines.

  Example:

  ```
  var myReg = /rabit/;
  var regobj = /Iowa City/ig
  ```

- Using RegExp() constructor:

  ```
  var myReg = new RegExp("Regular Expression", "Options");
  ```

Example

One can also use operations on string objects to manipulate patterns.

Examples:

```
var str = "Rabbits are furry";
var position = str.search(/bits/);
if (position >= 0)
  document.write("’bits’ does appear in position", position, "<br />");
else
  document.write("’bits’ does not appear in ", str, "<br />");
```

Methods of RegExp Objects

There are two methods that can be used to test for a match in a string:

- test(): For a regular expression object RE, RE.test("string"); searches for the pattern RE in the string "string" and returns true or false. Example: `pattern1.html`;
- exec(): Like test() but returns null if no match is found, or an array that contains the string that matched the RE. Example: `pattern2.html`.

Class Properties RegExp

Property	Meaning
input	Represents input string being matched.
lastMatch	Represents the last matched characters.
latsParen	Represents the last parenthesized substring pattern match.
leftContext	Represents the substring preceding the most recent pattern match.
RegExp.$*	The same as multi-line option (m).
RegExp.$&	Represents the last matched characters.
RegExp.$_	Represents the string input that is matched.
RegExp.$‘	Substring preceding the most recent match (left Context).
RegExp.$’	Substring following the most recent match (right Context).
RegExp.$+	Last parenthesized substring match.
RegExp.$1,$2,$3. . .	Captures the substrings of matches.
rightContext	Substring following the most recent match.

Instance Properties of RegExp

Property	Meaning
global	boolean, to specify if the g option was used.
ignoreCase	boolean, to specify if the i option was used.
lastIndex	with g option, char position following last match by exec() or test().
multiline	boolean, to specify if the m option was used.
source	text of the regular expression.

`pattern3.html` illustrates Class and instance properties.

String Methods

String object provides the following methods that work with regular expressions:

Method	What it does
match(regex)	return matched substrings in an array (as exec() does)
replace(regex,replacement)	substitute regex with replacement
search(regex)	find starting position of regex in string
split(regex)	removes regex from string for each occurrence.

Examples: `pattern4.html`, `pattern5.html`, `pattern6.html`, `pattern7.html`.

Characters and Patterns

Pattern meta-characters: $\backslash|()[]\hat{}\$ * +?.$ they can be matched by preceding them by a backslash.
Character classes:

1. Period (.) matches any character;
2. Classed of characters are specified by placing them in brackets.
   ([abc] matches ’a’, ’b’, or ’c’);
3. If a circumflex $(\hat{})$ is the first character of a class it inverts the specifying set.
   Example: $[\hat{}\texttt{aeiou}]$ specifies all consonants.

Predefined Classes

Name	Equivalent pattern	Matches
\ d	$[0-9]$	a digit
\ D	$[\hat{}0-9]$	not a digit
\ w	$[A-Za-z_0-9]$	a word character (alphanumeric)
\ W	$[\hat{}A-Za-z_0-0]$	not a word character
\ s	$[\backslash r\backslash t\backslash n\backslash f]$	a white space character
\ S	$[\hat{}\backslash r\backslash t\backslash n\backslash f]$	not a white space character

Example:

```
/\d\.\d\d/ //Matches a digit followed by a period, followd by 2 digits
/\D\d\D/  // Matches a signle digit
/\w\w\w/  // Matches three adjacent word characters
```

Pattern Specification

- Repetition: $/\texttt{xy}\{4\}\texttt{z}/$ allows y to be repeated 4 times;
- Quantifiers: $(*)$ means zero or more repetitions, $(+)$ means one or more repetitions, $(?)$ means one or none repetitions.

  Example:

  ```
  /\d+\.\d*/ //Matches the dot notation of a decimal numbner
  /[A-Za-z]\w*/ // A letter followed by zero or more letters
  ```
- Boundary position between a latter $(\backslash w)$ and a non-letter $(\backslash W)$ is pacified by $\backslash$ b. Example:

  ```
  /\bis\b? //Matches is in "A tulip is a flower"
           // Does not match is in "A frog isn't"
  ```

Anchors

An anchor allows us to specify that a pattern must match at a given position.

- Matching at the beginning of the string is specified by prefixing the pattern by a circumflex anchor $(\hat{})$.

  Example:

  ```
  /^pearl/ //Does not match "My pearls are white
  ```

- Matching at the end of the string is specified by appending $ to the patter.

  Example:

  ```
  /gold$/ // Does not match "golden"
  ```

Facts

1. Anchor characters match positions not characters;
2. When a circumflex $(\hat{})$ appears in a pattern at a position other than the beginning, it has no special meaning, it matches itself;
3. Same for the dollar ($) sign.

Pattern Modifiers

- The ’i’ modifier: an ’i’ after a pattern makes letters of the pattern case insensitive.

```
/Apple/i // Matches "APPLE", "apple", "aPPle", etc
```

- The ’x’ modifier allows white space to appear in a pattern. Example:

```
/\d+\s [A-Z[a-z]+/x // Matches "123 Street"
```

- The ’g’ modifier makes a pattern replacement be global.

Methods

1. `replace()`: takes two parameters: a pattern and a replacement string. If the replacement is to global, ’g’ modifier needs be added. The matched substrings are available in the predefined variables $1, $2, etc.

   Example:

   ```
   var str = "Fred, Freddie, and Frederica were siblings";
   str.replace(/Fre/g, "Boys);
   ```

   sets str to the value:

   "Boyd, Boyddie, and Boydarica were siblings"

2. `match()`: takes a single parameter, a pattern, and returns an array of matched strings. If ’g’ modifier is used the returned array has all matched strings; is no ’g’ modifier is used the array has first string matched and remaining elements are matched of parenthesized parts of the pattern.

Examples match()

```
var str = "Having 4 apples is better than having 3 oranges";
var matches = str.match(/\d/g);
```

makes matches be [3, 4]

```
var str = "I have 428 dollars, but I need 500";
var matches = str.match(/(\d+)([ˆ\d]+)(\d+)/;
document.write(matches, "<br />");
```

prints the following array:

```
["428 dollars, but I need 500", "428",
"dollars, but I need ", "500"]
```

`428 dollars` and `but I need 500`, is the match;

`428`, `but I need`, `500` are the parts of the string that match the parenthesized parts of the pattern.

More Methods

3. `split():` takes as argument a string pattern and split the string into substring separated by the pattern parameter.

```
var str = "grapes:apples:oranges";
var fruit = str.split(":");
```

makes fruits be ["grapes", "apples", "oranges"]

Check Forms

This example uses a function to check that a given string contains a phone number.

```
<?xml version = "1.0" encoding = "utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- formCheck.html: a demo for phone number check -->

<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> JavaScript Form Check Demo </title> </head>
   <body>
     <script type = "text/javascript" src = "formCheck.js" >
     </script>
   </body>
</html>
```

Challenge: make it interactive!

Errors in IE7 Scripts

To debug IE7 (and higher) scripts use the following method:

- select *Internet Option* from Tools menu and chose *Advanced* tab; this open a checkbox window;
- Uncheck *Disable script debugging* box;
- Check *Display a notification about every script error*;
- Press `Apply`

JavaScript errors will cause browser to open a display window with script error.

Errors in FX2 Scripts

To debug FX2 (and higher) scripts use the following method:

- Select *Tool* and then *Error Console* which open e debugging window;
- Script errors are displayed in this window;
- Clear the window before you try a script again.
- Download and install debuggers.

Note: the debugging window should be opened and cleared before using the browser and should remain opened during the browser usage.