JavaScript
Basic Syntax – Part 2

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/ajax.html

For additional materials, please see http://www.coreservlets.com/. The JavaScript tutorial section contains complete source code for all examples in the entire tutorial series, plus exercises and exercise solutions for each topic.

For customized training related to JavaScript or Java, email hall@coreservlets.com
Marty is also available for consulting and development support

Taught by lead author of Core Servlets & JSP, co-author of Core JSF (4th Ed), and this tutorial.

Available at public venues, or custom versions can be held on-site at your organization.

• Courses developed and taught by Marty Hall
  – JavaScript, jQuery, Ext JS, JSF 2.3, PrimeFaces, Java 8 programming,
  Spring Framework, Spring MVC, Android, GWT, custom mix of topics
  – Courses available in any state or country.
  – Maryland/DC companies can also choose afternoon/evening courses.

• Courses developed and taught by coreservlets.com experts (edited by Marty)
  – Hadoop, Hibernate/JPA, HTML5, RESTful Web Services

Contact hall@coreservlets.com for details
Topics in This Section

• Array basics
• Strings
• Regular expressions
• Array methods

For additional materials, please see http://www.coreservlets.com/. The JavaScript tutorial section contains complete source code for all examples in the entire tutorial series, plus exercises and exercise solutions for each topic.
### Array Basics

- **One-step array allocation**
  
  ```javascript
  var primes = [2, 3, 5, 7, 11, 13];
  var names = ["Joe", "Jane", "John", "Juan"];
  ```

- **No trailing comma after last element**

- **Accessing array elements**
  
  ```javascript
  primes[0] \rightarrow 2
  primes[5] \rightarrow 13
  names[0] \rightarrow "Joe"
  names[3] \rightarrow "Juan"
  ```

- **Replace entry at index 3**
  ```javascript
  names[3] = "Jill";
  ```

- **The length property**
  
  ```javascript
  var names = ["Joe", "Jane", "John", "Juan"];
  names.length \rightarrow 4
  ```

- **Notice that if the length is 4, the index of the last entry is 3**
- **This is true in general: array indexes run from 0 to length-1**

### Looping Down Arrays

- **Traditional-style for loop**
  
  ```javascript
  for(var i=0; i<someArray.length; i++) {
    var value = someArray[i];
    doSomethingWith(value);
  }
  ```

- **JavaScript-specific for loop (“the Lindsay loop”)**
  
  ```javascript
  for(var i=0, value; value=someArray[i]; i++) {
    doSomethingWith(value);
  }
  ```
Looping Down Arrays

• **for-in loop**

```javascript
for(var i in array) {
  doSomethingWith(i);
}
```

• **Usually reserved for objects (covered later)**
  
  Not recommended for looping down normal arrays
  
  • The values of i above are the indexes, not the array values
  
  • JavaScript has “array-like objects” that you normally treat as arrays, but that can have extra properties other than the indexes, and these extra properties will show up as values for i above.

Two-Step Array Allocation

• **Idea**
  
  – First build empty array, then fill in the elements
  
  – Often used in real life, because you frequently do not know the array elements or even the array size until after doing some calculations, so one-step array allocation will not work

• **Simple example**

```javascript
var names = new Array(4);
names[0] = "Joe";
names[1] = "Jane";
names[2] = "John";
names[3] = "Juan";
```

• **More typical example**

```javascript
var names = new Array(4);
for(var i=0; i<names.length; i++) {
  names[i] = someCalculation();
}
```
More on Arrays

• Arrays can be sparse
  
  ```javascript
  var names = new Array();
  names[0] = "Joe";
  names[100000] = "Juan";
  ```

• Arrays can be resized
  
  – Regardless of how arrays is created, you can do:
    ```javascript
    myArray.length = someNewLength;
    myArray[anyNumber] = someNewValue;
    myArray.push(someNewValue)
    ```
  
  – These are legal regardless of which way myArray was made

More on Arrays (Continued)

• Arrays have methods
  
  – push, pop, concat, slice, reverse, sort, forEach, map, filter, reduce
  
  • See upcoming slides

• Regular objects can be treated like arrays
  
  – You can use numbers (indexes) as object properties
    
    • More on this when we cover objects
Strings

String Basics

• You can use double or single quotes
  ```javascript
  var names = ["Joe", 'Jane', "John", 'Juan'];
  ```

• Strings have length property
  ```javascript
  "foobar".length → 6
  ```

• Numbers can be converted to strings
  - Automatic conversion during concatenations.
    ```javascript
    var val = 3 + "abc" + 5;  // Result is "3abc5"
    ```
  - Conversion with fixed precision
    ```javascript
    var n = 123.4567;
    var val = n.toFixed(2); // Result is 123.46 (not 123.45)
    ```
String Basics (Continued)

- Strings can be compared with ==
  
  "foo" == 'foo'
  // returns true

- Strings can be converted to numbers

  var i = parseInt("37 blah");
  // Result is 37 - ignores blah

  var d = parseFloat("6.02 blah");
  // Result is 6.02 - ignores blah

Core String Methods

- Simple methods
  
  - charAt, indexOf, lastIndexOf, substring, toLowerCase, toUpperCase

    "hello".charAt(1); \rightarrow "e"

    "hello".indexOf("o"); \rightarrow 4  // Returns -1 if no match

    "hello".substring(1,3); \rightarrow "el"

    "hello".toUpperCase(); \rightarrow "HELLO"

- Methods that use regular expressions

  - match, replace, search, split

- HTML methods

  - anchor, big, bold, fixed, fontcolor, fontsize, italics, link, small, strike, sub, sup

    "test".bold().italics().fontcolor("red")
    \rightarrow '<font color="red">\<i\><b\>test</b></i></font>'

  - These are technically nonstandard methods, but supported in all major browsers

    - But I prefer to construct HTML strings explicitly anyhow
Regular Expressions: Overview

- **You specify a regexp with /pattern/**
  - *Not* with a String as in Java and many other languages
- **Most special characters same as in Java/Unix/Perl**
  - `^, $, .` – beginning, end of string, any one char
  - `\` – escape what would otherwise be a special character
  - `*, +, ?` – 0 or more, 1 or more, 0 or 1 occurrences
  - `{n}, {n,}` – exactly n, n or more occurrences
  - `[]` – grouping
  - `\s, \S` – whitespace, non-whitespace
  - `\w, \W` – word char (letter or number), non-word char
- **Modifiers**
  - `/pattern/g` – do global matching (find all matches, not just first one)
  - `/pattern/i` – do case-insensitive matching
  - `/pattern/m` – do multiline matching
String Methods that Use Regular Expressions

- **replace**
  - Replaces all places that match the regular expression with a replacement string
    
    "axbxxcxxxxd".replace(/x+/g, "q") → "aqbqcqd"

- **match**
  - Returns array of parts of the String that match the regular expression
    
    "axbxxcxxxxd".match(/x+/g) → ["x", "xx", "xxx"]

- **split**
  - Returns array of all parts of the String that are in between the regular expressions
    
    "axbxxcxxxxd".split(/x+/) → ["a", "b", "c", "d"]

- **search**
  - Returns the position of the first place that matches the regular expression
    
    "axbxxcxxxxd".search(/x+/) → 1

Regular Expression: Examples

```javascript
>>> var firstString = "axbxxcxxxxd";
>>> firstString.split("x");
["aa", "bb", ",", ",", "ccc", "ddd"]

>>> firstString.split(/x/);

>>> firstString.split(/\w+/);
["aa", "bb", "ccc", "ddd"]

>>> var secondString = "foo123bar321baz223boo";
>>> secondString.split("123");
["foo", "bar321baz223boo"]

>>> secondString.split(/[123]+/);
["foo", "bar", "baz", "boo"]

>>> var thirdString = "foo <blink>bar</BLINK> baz";
>>> thirdString.replace(/\</blink>/g, "")
"foo bar baz"
More Information on Regular Expressions

- Online API references given earlier
  (See RegExp class)
  - http://www.w3schools.com/jsref/jsref_obj_regexp.asp

- JavaScript Regular Expression Tutorials

---

coleservlets.com – custom onsite training

Array Methods
Big Idea

• In JavaScript, arrays can have methods
  – Not functions to which you pass arrays, but methods of arrays
    ```javascript
    var nums = [1,2,3];
    nums.reverse(); \rightarrow [3,2,1]
    ```
  ```javascript
    [1,2,3].reverse(); \rightarrow [3,2,1]
    ```

• Most important methods
  – push, pop
  – sort
  – forEach
  – map
  – filter
  – reduce

Many more details at

push, pop, join

• push
  ```javascript
  var nums = [1,2,3];
  nums.push(4);
  nums; \rightarrow [1,2,3,4]
  ```

• pop
  ```javascript
  var val = nums.pop();
  val; \rightarrow 4
  nums; \rightarrow [1,2,3]
  ```

• concat
  ```javascript
  var nums2 = nums.concat([4,5,6]);
  nums2; \rightarrow [1,2,3,4,5,6]
  nums; \rightarrow [1,2,3]
  ```
• With no arguments (default comparisons)
  – Note the odd behavior with numbers: they are sorted lexicographically, not numerically

```javascript
["hi","bye","hola","adios"].sort();
⇒ ["adios","bye","hi","hola"]
[1,-1,-2,10,11,12,9,8].sort();
⇒ [-1,-2,1,10,11,12,8,9]
```

• With function as argument
  – Function returns negative if first of two compared items should go first, positive if second should go first, zero if they are tied. More on functions in upcoming lecture.

```javascript
var nums = [1,-1,-2,10,11,12,9,8];
function difference(n1,n2) { return(n1-n2); }
function reverseDifference(n1,n2) { return(n2-n1); }
nums.sort(difference);
⇒ [-2, -1, 1, 8, 9, 10, 11, 12]
nums.sort(reverseDifference);
⇒ [12, 11, 10, 9, 8, 1, -1, -2]
```
### Sorting: Java 8 vs. JavaScript

#### Java 8

```java
String[] testStrings = {"one", "two", "three", "four"};
Arrays.sort(testStrings,
            (s1, s2) -> s1.length() - s2.length());
Arrays.sort(testStrings,
            (s1, s2) -> s1.charAt(s1.length() - 1) -
                         s2.charAt(s2.length() - 1));
```

#### JavaScript

```javascript
var testStrings = ["one", "two", "three", "four"];
testStrings.sort(function(s1, s2) {
    return(s1.length - s2.length);});
testStrings.sort(function(s1, s2) {
    return(s1.charCodeAt(s1.length - 1) -
           s2.charCodeAt(s2.length - 1));
});
```

First variation of each sorts by length, second variation sorts by last character.

### forEach

#### Big idea

- Calls function on each element of array. Cannot break “loop” partway through
  - Lacks option to run in parallel that Java 8 has

#### Examples

```javascript
[1,2,3].forEach(function(n) { alert(n); });
// Pops up alert box in page 3 times showing each number

[1,2,3].forEach(alert);
// Same as above. Explained in later section on functions.
```

- Summing an array (but reduce can also be used)
  ```javascript
  var nums = [1,2,3];
  var sum = 0;
  nums.forEach(function(n) { sum += n; });
  sum; // 6
  ```
**map**

- **Big idea**
  - Calls function on each element, then accumulates result array of each of the outputs. Returns new array; does not modify original array.
  - Like the Java 8 “map” method, but not as powerful since the JavaScript version does not support lazy evaluation or parallel operations.

- **Examples**
  
  ```javascript
  function square(n) { return(n * n); }
  [1,2,3].map(square);
  \rightarrow [1, 4, 9]
  ```

**filter**

- **Big idea**
  - Calls function on each element, keeps only the results that “pass” (return true for) the test. Returns new array; does not modify original array.
  - Like the Java 8 “filter” method, but not as powerful since the JavaScript version does not support lazy evaluation or parallel operations.

- **Examples**
  
  ```javascript
  function isEven(n) { return(n % 2 == 0); }
  [1,2,3,4].filter(isEven);
  \rightarrow [2, 4]
  ```
Notes on map and filter

- **Cheaper if you combine mapping operations**
  
  ```javascript
  var singleCost = someArray.map(combinedFunction);
  var doubleCost = someArray.map(funct1).map(funct2);
  ```

- **Cheaper if you combine filtering tests**
  
  ```javascript
  var singleCost = someArray.filter(combinedTest);
  var doubleCost = someArray.filter(test1).filter(test2);
  ```

- **Wasteful on large arrays if you need only one result**
  
  ```javascript
  var firstResult = largeArray.filter(test).map(funct)[0];
  ```

- **Points seem obvious, but none are true in Java 8**
  
  - Two calls to map vs. one call with a combined function: same cost
  - Two calls to filter vs. one call with a combined test: same cost
  - Finding first element of result of series of mapping and filtering operations: cost depends only on location of first match, not on size of original array
  
  - For more detail, see Java 8 tutorial at coreservlets.com

reduce

- **Big idea**
  
  - Takes function and starter value. Each time, passes accumulated result and next array element through function, until a single value is left.
  
  - Like the Java 8 “reduce” method, but not as powerful since the JavaScript version does not support lazy evaluation or parallel operations.

- **Examples**
  
  ```javascript
  function add(n1,n2) { return(n1 + n2); }
  function multiply(n1,n2) { return(n1 * n2); }
  function bigger(n1,n2) { return(n1> n2 ? n1 : n2); }

  var nums = [1,2,3,4];
  var sum = nums.reduce(add, 0);   // 10
  var product = nums.reduce(multiply, 1);  // 24
  var max = nums.reduce(bigger, -Number.MAX_VALUE); // 4
  ```
Notes on reduce

• **Backward args from Java 8 and some other languages**
  – In Java 8, reduce takes starter value (identity) first, combiner function second
  – JavaScript takes combiner function first, starter value second

• **There is one-arg version**
  – Both the JavaScript and Java 8 versions of reduce let you omit the starter value, but then you have to worry about what to do if there are no values in the array

• **reduceRight method**
  – Goes in opposite order: from highest index to lowest

• **Other names**
  – Some other languages call this “fold” or “inject” instead of “reduce”

More Array Methods

• **concat**
  – Concatenates arrays
    ```javascript
    [1,2,3].concat([4,5,6]); → [1,2,3,4,5,6]
    ```

• **slice**
  – Returns sub-array
    ```javascript
    [9,10,11,12].slice(0, 2); → [9,10]
    [1,2,3].slice(0); → [1,2,3] // Makes copy of array
    ```

• **reverse**
  – Reverses array (returns it, but also changes original)
    ```javascript
    [1,2,3].reverse(); → [3,2,1]
    ```

• **indexOf**
  – Finds index of matching element
    ```javascript
    [9,10,11].indexOf(10); → 1
    [9,10,11].indexOf(12); → -1
    ```
Wrap-up

Summary

- **JavaScript arrays**
  - One step allocation
    ```javascript
    var nums = [1, 2, 3];
    ```
  - Looping down arrays
    ```javascript
    for(var i=0; i<nums.length; i++) { doSomethingWith(nums[i]); }
    ```
  - Two-step allocation
    ```javascript
    var nums = new Array(12);
    for(var i=0; i<nums.length; i++) { nums[i] = someCalculation(); }
    ```
  - There are useful array methods, especially push, pop, sort, map, filter, and reduce

- **Strings**
  - Either single or double quotes are legal. There are some useful String methods

- **Regular expressions**
  - Used for comparing to patterns
Questions?

For additional materials, please see http://www.coreservlets.com/. The JavaScript tutorial section contains complete source code for all examples in the entire tutorial series, plus exercises and exercise solutions for each topic.