CHAPTER 4: JAVASCRIPT

Note

- All example files for chapter 4 are available at

https://swe.umbc.edu/~zzaidi1/is448/chapter4.2/

4.8 Arrays

- Arrays are lists of elements indexed by a numerical value
- Array indexes in JavaScript begin at 0
- Arrays can be modified in size even after they have been created

4.8 Creating arrays

- Arrays can be created using the **new** operator
  - calling the new operator with one parameter creates an empty array of the specified number of elements
    - **var** your_list = **new Array**(10)
  - calling the new operator with two or more parameters creates an array with the specified parameters as elements
    - **var** your_list = **new Array**(10, 20)
- Literal arrays can be specified using square brackets to include a list of elements
    - **var** alist = [1, "ii", "gamma", "4"];
- Elements of an array do not have to be of the same type

Creating Arrays

- Three ways to initialize an array

```
var stooges = new Array();
stooges[0] = "Larry";
stooges[1] = "Moe";
stooges[2] = "Curly";
```

```
var stooges = new Array("Larry", "Moe", "Curly")
```

```
var stooges = ["Larry", "Moe", "Curly"];
```

Array methods

- Methods
  - pop, push
    - Remove (return) and add from end
  - shift, unshift
    - Remove (return) and add from front
  - concat, join, reverse, slice, sort, splice, toString
- Properties
  - length: returns the length of the array
- Example: insert_names.js, insert_names.html

Using Array Methods

```
var a = [“Joey", “Chandler"];
a.push(“Monica");
a.unshift(“Ross");
a.pop();
a.shift();
a.sort();
```

- Example: friends_names.js, friends_names.html

Lab: Using arrays

- Write a program, cars.js, that
  - defines an array with three car names in it. the names of the cars are Honda, BMW, Toyota
  - and prints the array to the screen
    - hint: use the 'length' property of the Array object and a loop to print contents of an array to the screen
- Download cars.html and add the necessary code to reference the Javascript program cars.js

4.9 Function Fundamentals

- Function definition syntax
  - A function definition consist of a header followed by a compound statement

```
function function-name(optional-formal-parameters){
        Statements;
}
```

- return statements
  - A return statement causes a function to cease execution and control to pass to the caller
  - A return statement may include a value which is sent back to the caller
    - This value may be used in an expression by the caller
  - A return statement without a value implicitly returns undefined
- Function call syntax
  - Function name followed by parentheses and any actual parameters
  - Function call may be used as an expression or part of an expression
- Functions must defined in the page header before use

Function Definition

Syntax

```
function name() {
   statement ;
   statement ; ...
   statement ;
}
```

Example

```
function sum(a, b) {
   var sumOfTwo = a + b;
   return(sumOfTwo);
}
```

- Parameter types and return types are not specified
  - i.e., **var** is not written in parameter declarations
- Any variables declared in the function are local (only exist in that function)
- Example: usingfunctions.html usingfunctions.js

Calling Functions

Syntax

```
function-name(parameterValue, ..., parameterValue);
```

Example Usage

```
var number1 = 10; var number2 = 20;
var computedSum = sum(number1, number2);
```

- If the wrong number of parameters are passed,
  - too many: extra ones are ignored
  - too few: remaining ones get an undefined value

4.9 Local Variables

- "The ***scope*** of a variable is the range of statements over which the variable is visible"

- A variable not declared using var has global scope
  - visible throughout the page,
  - Even if first used inside a function definition

- A variable declared with var outside a function definition has global scope

- A variable declared with var inside a function definition has local scope, visible only inside the function
  - If a global variable has the same name, it is hidden inside the function definition

Variable Scope

```
1  var count = 1;
2  document.write(count);
3  increment();
4  product();
5  document.write(count);
6
7  function increment()
8  {
9      count = count + 1;
10     temp = 10;
11     var temp2 = 7;
12     document.write(count);
13  }
14
15 function product()
16 {
17     var count = 5;
18     count = count * 10;
19     document.write(count);
20     temp = temp * 9;
21     document.write(temp);
22     temp2 = temp2 * 2;
23     document.write(temp2);
24 }
```

- Variable count at iine 1
  - Global or local?
- What gets printed at line 2 for count?
- Variable temp at line 10
  - Global or local?
- Variable temp2 at line 11
  - Global or local?
- What gets printed at line 12 for count?
- Variable count in line 17
  - Global or local?
- What gets printed at line 19 for count?
- What gets printed at line 21 for temp?
- What gets printed at line 23 for temp2?
- What gets printed at line 5 for count?

Variable Scope

```
1  var count = 1;
2  document.write(count);
3  increment();
4  product();
5  document.write(count);
6
7  function increment()
8  {
9      count = count + 1;
10     temp = 10;
11     var temp2 = 7;
12     document.write(count);
13  }
14
15 function product()
16 {
17     var count = 5;
18     count = count * 10;
19     document.write(count);
20     temp = temp * 9;
21     document.write(temp);
22     temp2 = temp2 * 2;
23     document.write(temp2);
24 }
```

- variable *count* in line 1 is global, i.e., can be seen by all functions
- variable *temp* in line 10 is global, i.e., can be used by any function within Javascript file
- variable *temp2* in line 11 is local, i.e., can be used by only increment function
- variable *count* in line 17 is local to the product function
- both the functions increment and product can use and modify *count* (what is printed in lines 2, 5, 12, 19 for *count*?)
- Example: see scope.html

Java script Errors

- Most common student errors
  - My program does nothing. (most errors produce no output)
  - It just prints undefined. (many typos lead to undefined variables)

Debugging Tools in Browsers

- If using Internet Explorer,
  - look for a yellow triangle at the bottom of the browser window (sometimes it will show up as a separate window)
  - Click on the triangle to see the error (shows line number of error also!!)
  - However, first, the option to check for errors must be enabled in IE
    - Tools -> Internet Options -> Advanced, Uncheck 'Disable Script Debugging' checkbox and check 'Display a notification about every script error' box
- If using Firefox use the 'Web Developer' plugin
  - Installed on school machines
  - Go to Tools -> Web Developer -> Select Web Console
  - Can also reach the 'Web Console' by clicking on the three horizontal lines on the right of the Firefox window
  - Window pane shows up at bottom which shows the error and line number
- If using Chrome, enable the 'Developer Tools' option
  - Can reach the 'Developer Tools' by clicking on the three horizontal lines on the right of the Chrome window, then, 'More Tools', then 'Developer Tools' (may be slightly different in different version of Chrome)

Steps to finding Javascript errors

- Step 1: Is your Javascript program being called? Put an alert statement as the very first statement in your Javascript code and see if an alert box is displayed
  - If yes, move on to Step 2
  - If no, there is a problem with how you are calling the Javascript code—check the statements where you call the Javascript code or statements that include external Javascript files for errors
- Step 2: Open the page in Firefox or Chrome. Enable the Web Developer plugin in Firefox or Developer Tools in Chrome and see if any errors are reported by these tools
  - If yes, click on error to see details of the error. Go to corresponding line number and fix the error
  - If no, go to Step 3

Steps to finding Javascript errors

- Step 3: comment out everything in your code, except for two or three statements at a time. Start with the first 3 statements. Repeat Step 2 to see if any error is shown in these statements.
  - If yes, continue to uncomment next 3 statements and repeat Step 3
  - If no, an error is in these statements–fix it. Once fixed, uncomment next 3 statements and repeat Step 3

4.12 Using Regular Expressions

- Regular expressions are used to specify patterns in strings
- JavaScript provides two ways to use regular expressions in pattern matching
  - methods of the String class
  - RegExp objects (not covered in the class)
- A literal regular expression pattern is indicated by enclosing the pattern in slashes,
  - E.g., /bits/

4.12 Characters and Character-Classes

- Meta-characters: characters that have special meaning in regular expressions
  - `\ | ( ) [ ] { } ^ $ * + ? .`
  - These characters may be used literally by escaping them with \
  - Example: if you want to search for "{abc}", your pattern should be written as /\{abc\}/
- Other characters represent themselves
- A period matches any single character
  - /f.r/ matches for and far and fir but not fr and not faaar
  - /snow./ matches snowy and snowe and snowed
- A character class matches one of a specified **set** of characters
  - To define a character class, place desired characters in square brackets: [character set]
  - List characters individually: [abcdef]
  - Give a range of characters: [a-z]
  - ^ at the beginning negates the class: [^abcdef] means to search for other than these six characters

4.12 Predefined character classes

Name	Equivalent Pattern	Matches
\d	[0-9]	A digit
\D	[^0-9]	Not a digit
\w	[A-Za-z_0-9]	A word character (alphanumeric)
\W	[^A-Za-z_0-9]	Not a word character
\s	[\r\t\n\f]	A whitespace character
\S	[^ \r\t\n\f]	Not a whitespace character

Example using predefined character classes:

/\d\.\d\d/ Matches a digit, followed by a period, followed by two digits

/\D\d\D/ Matches a single digit

/\w\w\w/ Matches three adjacent word characters

4.12 Repeated Matches

- A pattern can be repeated a fixed number of times by following it with a pair of curly braces enclosing a count
- Examples
  - /xy{4}z/
    - matches the string “xyyyyz”
  - /\(\d{3}\)\d{3}-\d{4}/ might represent a telephone number
    - matches the string “(555)555-1111”

4.12 Repeated Matches

- A pattern can be repeated by following it with one of the following special characters
  - * indicates zero or more repetitions of the previous pattern
  - + indicates one or more of the previous pattern
  - ? indicates zero or one of the previous pattern

- Example:
  - /[A-Z]\w*/
    - matches a string of the form:
      - an upper-case letter ([A-Z])
      - followed by a zero or more letters, digits or underscores (\w)

4.12 Anchors

- Anchors in regular expressions match positions rather than characters
  - Anchors are 0 width and may not take multiplicity modifiers
- The $ symbol is used to anchor a pattern to the end of the string
- The ^ symbol is used to anchor a pattern to the beginning of the string
- The \b symbol is used to match at word boundaries

Anchor at beginning and end

- Anchor at beginning of the string
  - ^ at the beginning of a pattern matches the beginning of a string
  - Example: the pattern /^pearl/
    - matches the string “pearls are pretty”
    - but does not match the string “My pearls are pretty”

- Anchor at the end of the string
  - $ at the end of a pattern matches the end of a string
  - Example: the pattern /gold$/
    - matches the string “I like gold”
    - but does not match the string “You are golden”

Anchors symbols in middle of pattern

- What if the '$' character or the '^' character appear in the middle of a pattern?
- The $ in the pattern /a$b/ matches a $ character
  - $ does not have special meaning if it appears in middle of pattern
- The ^ in the pattern /2^3/ matches the ^ character itself
  - If the ^ symbol appears anywhere other than beginning of pattern, it has no special meaning

4.12 Anchors

- Anchoring at a word boundary
  - \b matches the position between a word character and a non-word character or the beginning or the end of a string

- Example: /\bthe\b/
  - will match the string "the"
  - will also match 'the' in the string "one of the best"
  - but will not match with the string "theatre"

4.12 Pattern Modifiers

- Pattern modifiers are specified by characters that follow the closing / (forward slash) of a pattern

- Modifiers modify the way a pattern is interpreted or used

4.12 Pattern Modifiers

- The i modifier causes letters in pattern to match either uppercase or lowercase letters in string

- Example: pattern /Apple/i matches the strings
  - “APPLE”
  - “AppLE”
  - “apple”

4.12 Pattern Modifiers

- The x modifier causes whitespace in the pattern to be ignored
  - This allows better formatting of the pattern
  - \s which indicates space still retains its meaning

- Example: pattern **/\d+[A-z][a-z]+/x**
  - matches the string "555Hilltop Circle"
  - matches the string "555Hilltopcircle"

4.12 Example Using *test* method

```
var statement = "Rabbits are furry";
var pattern = /bits/;
var result = pattern.test(statement);
if (result)
        document.write("'bits' appears in statement <br />");
else
        document.write("'bits' does not appear in statement <br />");
```

- The **test** method is used to look for a pattern in a larger string
- The test method returns true if pattern is found in string, else returns false
- When calling the test method, we need to
  - call the search method on the pattern that we are looking for
  - Also pass the string in which the pattern is being searched as a parameter
- In the above example, we are looking for the pattern 'bits' in the larger string given in the variable *statement*
- The output of this code is:

  'bits' appears in statement
- Example: See rabbits2.html, rabbits2.js

4.12 Example Using search

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

- The **search** method is used to search for a pattern in a larger string
- When calling the search method, we need to
  - call the search method on the string that we are searching, and
  - Also pass the pattern to be matched within the two forward slashes
- The **search** method returns
  - the position of a match if pattern found in string,
  - or -1 if no match was found
- In the above example, we are searching for the pattern ‘bits’ in the larger string given in the variable *statement*
- The output of this code is:

  'bits' appears in position 3
- Example: See rabbits.html, rabbits.js

4.12 Other Pattern Matching Methods

- The replace method takes a pattern parameter and a string parameter
  - The method replaces a match of the pattern in the target string with the second parameter
  - A g modifier on the pattern causes multiple replacements
- Example

```
var str = “Fred, Freddie, Frederica were siblings”;
str2 = str.replace(/Fre/g, “Boy”);
```

Result: str2 is set to “Boyd, Boydie, Boyderica were siblings”

4.12 Other Pattern Matching Methods

- The match method takes one pattern parameter
  - Without a g modifier, the return is an array of the match and parameterized sub-matches
  - With a g modifier, the return is an array of all matches
- Example

```
var str = "I have 400 dollars but I need 500";
var matches = str.match(/\d/g);
```

Result: matches array is set to [4,0,0, 5,0,0]

4.12 Other Pattern Matching Methods

- The split method splits the object string using the pattern to specify the split points

- Example

```
var str = “apples:oranges:bananas”;
var fruit = str.split(“:”);
```

Result: fruit array is set to [apples, oranges, bananas]

Lab: Using Functions and Regular Expressions

- Download regexp.html from the examples folder
- This file prompts user to enter a name and should check whether user entered only letters or included numbers as well in their name
- Modify the file by adding code to the Javascript function check_name to do the following:
  - To check if name entered by user has only letters
    - Hint: use the test method (or the search method)
    - and, the function check_name should say 'Welcome *username*' if user entered name correctly, otherwise, it should tell the user that they made a mistake and that the user should reload the page and try again
- Remember to enable the appropriate debugging option in your browser so you can see any errors