CSE 341, Autumn 2010
Assignment #9 - JavaScript
Due: Friday, December 10, 2010, 11:30 PM

This four-part assignment is our introduction to programming in JavaScript for solving relatively small programming problems. Include your answers in a file named `hw9.js`. You will need `underscore.js` from the course web site.

Part A - Basic JavaScript Functions:

Define each of the following JavaScript functions in the global scope. Unless otherwise specified, your functions may assume that all expected parameters are passed and that their values are of the expected types. Unless otherwise specified, the behavior of each function when passed insufficient parameters or parameters of wrong types is undefined. For full credit, do not define any global symbols other than the ones specified here.

1. `repl`

Write a function `repl` that accepts a string and an integer *n* and returns the string repeated *n* times. For example, `repl("hello", 3")` should return `"hellohellohello"`. If *n* is less than or equal to zero, return an empty string. The *n* parameter should be treated as an optional parameter; if it is not passed, assume an *n* of 1.

2. `isPrime`

Write a function `isPrime` that accepts an integer and returns `true` if the integer is a prime number and `false` otherwise. A prime number is defined to be a positive integer whose only factors are 1 and itself. 2 is considered to be the smallest prime number. For example, `isPrime(53)` should return `true` and `isPrime(24)` should return `false`.

3. `box`

Write a function `box` that draws a text figure of a rectangular box. The function should accept a single argument: an object that stores various attributes of the box as its properties (this is referred to as an "object as argument specifier" in the lecture notes). Your `box` function should use the following properties of the argument object, if they are present, to guide its drawing of the box. If any property is absent, instead use the default value shown. If no parameters are passed whatsoever (no object argument specifier is specified), use the default for all properties.

- `width`: number of characters wide for the box (default 10); you may assume it is at least 2 if it exists
- `height`: number of characters wide for the box (default 5); you may assume it is at least 2 if it exists
- `border`: character to use to draw around the outside of the box (default `"*"`); assume it is a single character
- `filling`: character to use to draw in the interior of the box (default `"."`); assume it is a single character

Here are several calls to the function and their expected outputs:

```
box();        // or box({});
```
```
**********
*........*
*........*
*........*
**********
```

```
box({width: 22, height: 4, border: "#", filling: "x"});
```
```
######################
#xxxxxxxxxxxxxxxxxxxx#
#xxxxxxxxxxxxxxxxxxxx#
######################
```

```
box({height: 8, filling: " "});
```
```
**********
*        *
*        *
*        *
*        *
*        *
*        *
**********
```

```
box({border: "+", width: 4});
```
```
++++
+..+
+..+
+..+
++++
```

Note that JavaScript's `print` function prints an entire line, so you will have to print entire lines in one call rather than being able to write loops that print single characters.

4. `pigLatin`

Add a method `pigLatin` that accepts a string and that returns a new string with the same characters as the original, but with the text converted to "Pig Latin", according to the following rules (which aren't quite the same as "real" Pig Latin):

- All words that begin with vowels should be removed from the string.
- For each word that begins with a consonant, move the word's first character to the word's end, preceded by "-" and followed by "ay". For example, `"hello"` becomes `"ello-hay"` and `"STRONG"` becomes `"TRONG-Say"`.

You may assume that words are separated by a single space. For example, if the following variable has been declared:

```
var s = "Seattle Mariners are a great team eh?";
```

The call of `pigLatin(s)` would return `"eattle-Say ariners-May reat-gay eam-tay"`.

To get full credit, you are required to solve this problem using **higher-order functions** such as map/filter/reduce. You should not use any loops or recursion in your solution. Recall that you can break apart a string using its `split` method.

Part B - Augmenting Existing JavaScript Types:

In each of the following problems, add functionality to existing JavaScript data types by modifying their prototypes.

5. Number `squared`

Add a method `squared` to all numbers that accepts no parameters and returns the square of that number, i.e., the number multiplied by itself. For example, if the following variable has been declared:

```
var n = 7;
```

Then the call of `n.squared()` should return `49`.

6. Array `shuffle`

Add a method `shuffle` to all arrays that accepts no parameters and rearranges the elements of the array into a randomly chosen order with equal probability. For example, if the following variable has been declared:

```
var a = [10, 20, 30, 40, 50, 60, 70, 80, 90];
```

Then the call of `a.shuffle();` might randomly rearrange the elements of `a` to be the following:

```
[40, 30, 80, 10, 50, 20, 70, 90, 60]
```

Use the following version of the classic Fisher-Yates shuffling algorithm, which shuffles elements properly and gives each value an equal chance of appearing at each index of the array:

```
To shuffle an array a of n elements:
    for each i from n - 1 down through 1, do:
        j := random integer in range 0 ≤ j ≤ i.
        exchange a[j] and a[i].
```

7. String `toTitleCase`

Add a method `toTitleCase` to all strings that accepts no parameters and returns a new string where each *word* of the original string has its first letter in uppercase and the rest of that word's letters in lowercase. For example, if the following variable has been declared:

```
var s = "QUICK Fox JUmPs ovEr LazY DOg";
```

Then the call of `s.toTitleCase()` should return `"Quick Fox Jumps Over Lazy Dog"`. You may assume that words in the string are separated by exactly one space; this is helpful because you can reliably break the string apart into words. If called on the empty string, your method returns `""`.

8. String `toAlternatingCase`

Add a method `toAlternatingCase` to all strings that returns a new string that contains the same characters as the original, but alternating between upper and lowercase. By default, the first character should be uppercase, the second lowercase, the third uppercase, and so on. But if an optional boolean parameter of `true` (or any truthy value) is passed, instead make the first character lowercase, the second uppercase, the third lowercase, and so on. For example, using the variable `s` declared above, `s.toAlternatingCase()` should return `"QuIcK FoX JuMpS OvEr lAzY DoG"` and `s.toAlternatingCase(true)` should return `"qUiCk fOx jUmPs oVeR LaZy dOg"`. Spaces, numbers, and other non-alphabetic characters are included even though they do not distinguish between upper/lowercase. The behavior has nothing to do with words; capitalize every other character, regardless of the number of words or their lengths.

9. String `toLeetSpeak`

Add a method `toLeetSpeak` to all strings that returns a new string that contains the same characters as the original, but with various substitutions to convert the text to "Leet Speak". Substitute 4 for A, 3 for E, 1 for I, 0 (zero) for O, and Z for S. Also convert the string to alternating case, starting with the first character capitalized. For example, using the variable `s` declared above, `s.toLeetSpeak()` should return `"Qu1cK F0X JuMpZ 0v3r l4zY D0G"`.

Note that the method's behavior has nothing to do with words; you are to capitalize every other character, regardless of the number of words in the string or their lengths. For example, `"xx xx xx xxx xxx xxxx xxxx".toLeetSpeak()` returns `"Xx xX Xx xXx xXx xXxX XxXx"`. Also note that you are to perform the replacements described above regardless of the initial case of the character in the original string. For example, the call of `"aaAAeeEEiiIIooOOuuUUssSS".toLeetSpeak()` returns `"444433331111000OUuUuZzZz"`.

Note: This problem is easier to solve if you use the String object's `replace` method, but by default, it will replace only the first occurrence of a given character or pattern. To perform a global replacement, use a *regular expression*, which is indicated as a sequence of characters between `/` slashes, with a `g` character after the closing slash to indicate "global". For example, `"HELLO CLASS".replace(/L/g, "x")` returns `"HExxO CxASS"`.

Part C - Using JavaScript Libraries:

To solve each of the following problems, download the Underscore library from the link on the course web site. We have not covered this library in any real detail, but you just have to use it to make a few short calls in your code. The goal is to get a bit of practice using a JavaScript library and taking advantage of its useful functionality. *(If you like, you may use Underscore on other problems on this assignment, but you are not required to do so.)* You should link Underscore to your program by writing the following line at the top of your `hw9.js` file:

```
load("underscore.js");
```

See the following URL (linked on the class web page) for more documentation/information about how to use Underscore:

- http://documentcloud.github.com/underscore/

10. `isPrime2`

Define a function `isPrime2` that is identical to your previous `isPrime` function, except that it performs memoization to be more efficient. The Underscore library can help you do this with little effort. (Do not implement your own memoization from scratch.) *Hint:* Recall that you can define a function using `var` instead of `function` if you like. Our solution to this problem is a 1-liner.

11. `primesInRange`

Define a function `primesInRange` that accepts a minimum and maximum value as integers and returns an array containing all prime numbers that appear in that range (inclusive). If the minimum is greater than the maximum, return an empty array. For example, the call of `primesInRange(15, 59)` should return `[17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59]`. For full credit, use **higher-order functions** instead of loops. *Hint:* Underscore can supply you with a range of numbers to process. Our solution to this problem has a 2-line body.

Part D - Objects and Prototypes:

Define a constructor and prototype called `Rectangle` for objects representing 2-dimensional rectangles in the x/y plane. A `Rectangle` object is defined through the following constructor and has the properties listed below. Define each of the given methods only once within the rectangles' **prototype** so that their code is not replicated in each `Rectangle` object.

For all of the following, you may assume that the client will pass all expected parameters to each function and that they will be of the appropriate type(s). The behavior is undefined otherwise. Keep in mind that, as with all computer graphics, **the y-axis is inverted**; (0, 0) is the top-left corner of the screen and increasing y values go downward from there.

- `left`, `right`, `top`, `bottom`
  Four integers representing the left/right-most x-coordinate and top/bottom-most y-coordinate occupied by this rectangle, inclusive.
- `Rectangle(`***`left`***`, `***`top`***`, `***`right`***`, `***`bottom`***`)`
  A constructor that accepts the four corner coordinate values and uses them to initialize a newly created rectangle object. The order of the parameters is the same as if we were passing two points, (x_1, y_1), (x_2, y_2), where the first represented the top/left corner of the rectangle and the second represented its bottom/right corner. You may assume that the left value passed is less than or equal to the right value passed and that the top value passed is less than or equal to the bottom value passed.
- `toString()`
  Returns a string representation of the rectangle in exactly the following format. For example, given the rectangles declared below, the call of `r1.toString()` returns `"{left=3,right=10,top=5,bottom=19}"`. The method should not modify `this` rectangle.
- `union(`***`r`***`)`
  Returns a new rectangle of the smallest bounding box that entirely contains both this rectangle and the given rectangle. For example, given the rectangles below, the call of `r1.union(r2)` returns a rectangle whose `toString` is `{left=3,right=11,top=2,bottom=19}`. This method is commutative; `r2.union(r1)` yields the same result. The method should not modify `this` rectangle or the rectangle passed in.
- `intersect(`***`r`***`)`
  Returns a new rectangle representing the largest rectangular area that is contained within both this rectangle and the given other rectangle. For example, given the rectangles declared below, the call of `r1.intersect(r2)` returns a rectangle whose `toString` representation is `{left=4,right=10,top=5,bottom=8}`. By nature this method is commutative; `r2.intersect(r1)` yields the same result. The call of `r3.intersect(r1)` would return `{left=6,right=9,top=8,bottom=11}`. If the rectangles do not overlap at all, such as `r1.intersect(r4)`, your `intersect` function should return `null`. Rectangles do include their borders; `r1.intersect(new Rectangle(10,19,13,24))` returns `{left=10,right=10,top=19,bottom=19}`. The method should not modify `this` or the rectangle passed in.
- `contains(`***`obj`***`)`
  Returns `true` (or any truthy value) if the given point/rectangle lies entirely inside of this rectangle, and `false` (or any falsy value) otherwise. The parameter might be a single point, or it might be another rectangle. Any object passed in that has both an `x` and `y` property is considered to be a point; any other object is treated as a rectangle. For example, given the rectangles declared below, `r1.contains(r2)` returns `false`, but `r1.contains(r3)` returns `true`. `r1.contains(p1)` returns `true`, but `r1.contains(p2)` returns `false`. A rectangle does contain the points around its four borders, and it is considered to contain itself. The method should not modify `this` rectangle or the object passed in.

```
var r1 = new Rectangle(3, 5, 10, 19);
var r2 = new Rectangle(4, 2, 11, 8);      // interesting union/intersection w/ r1
var r3 = new Rectangle(6, 8, 9, 11);      // contained within r1
var r4 = new Rectangle(20, 19, 30, 32);   // far away from the others
var r5 = new Rectangle(3, 5, 10, 19);     // equal to r1

var p1 = {x: 7, y: 8};                    // a point contained within r1
var p2 = {x: 1, y: 2};                    // a point not contained within r1
```

Suggested Development Strategy:

Douglas Crockford's **JSLint** tool (linked from the course web site) can help you find common JavaScript bugs. Since this is your first JavaScript program, you will probably encounter frustrating problems. If so, paste your code into JSLint to look for errors or warnings.

Rhino does have a **debugger**, but we have not discussed how to use it. You might prefer to insert `print` statements to display values within your code. To launch the debugger, type this command into a terminal in your program's directory:

```
java -classpath rhino.jar org.mozilla.javascript.tools.debugger.Main hw9.js
```

For more information about Rhino's JS debugger, visit this URL:

- http://www.mozilla.org/rhino/debugger.html

If you produce any test cases or **testing code** for this assignment, you are welcome to share it with your classmates using our course discussion forum. Any shared testing code may test for external behavior but may not test any aspects of the internal correctness of the student's Scheme source code itself. The instructor and TAs reserve the right to remove testing code that we feel is inappropriate for any reason.

Grading and Submission:

Submit your finished `hw9.js` file electronically using the link on the course web page. For reference, our solution is roughly 105 "substantive" lines long according to the course Indenter page. You don't need to match this number or even come close to it; it is just a rough guideline.

Your program should not produce syntax or runtime **errors** when executed. You may lose points if you name your functions or other global symbols incorrectly, even if their behavior is correct.

Your code should work for both basic and advanced cases. Perform your own **testing**, and remember to test edge cases. As always, if the solution to one problem is useful in helping you solve a later problem, you should call the earlier function from the later one.

You are expected to use good programming **style**, such as naming, indentation/spacing, and avoiding redundancy when possible. Avoid long lines of over 100 characters in length; if you have such a line, split it into multiple lines. Recall that our Indenter tool is able to re-indent your program for you.

Place a descriptive **comment** header at the top of your program along with a comment header on each function, including a description of the meaning of its parameters, its behavior/result, and any preconditions it assumes. If you declare a non-trivial or non-obvious inner helper function, also briefly comment the purpose of that helper in a similar fashion. If a function's code is large or elaborate, you should put comments within the function explaining the complex code. All comments you write should be in your own words and not copied directly from this document.

Efficiency on a fine level of detail is not crucial on this assignment. But code that is unnecessarily computationally inefficient (such as code that performs an algorithm that should be O(n) in O(n^2 log n) time) might receive a deduction.

Redundancy, such as recomputing a value unnecessarily or unneeded recursive cases, should be avoided.

For full credit, your code should pass the **JSLint** tool with no errors reported, with (only) the following options checked:

- Assume Rhino
- Disallow undefined variables
- Require Initial Caps for constructors
- Require parens around immediate invocations

JSLint is picky and can be hard to pass. Here are a few common errors and workarounds:

- *Missing radix parameter*: instead of `parseInt(expr)`, call `parseInt(expr, 10)`, indicating base-10.
- *'_' is not defined*: JSLint doesn't know about the Underscore library. Write `var _;` at the top of your file.
- *Unnecessary semicolon* or *missing semicolon*: Remember, `function foo() {}` should not have a semicolon, but `Foo.prototype.foo = function() {};` and `var foo = function() {};` should.