CS 331 – Assignment 1

Due: 11:59pm, Friday, February 14

Use the *ASSIGNMENT1* dropbox on D2L to submit *one file* named *assign1.js* with all of your functions in it.

Write each of the six functions specified below, putting their definitions in your *assign1.js* file. You must name each function precisely as indicated, or it will crash when we test it, resulting in zero credit on that problem. You should also put the definitions of any helper functions you need in that file. Do not put any of your test cases in that file since we will load our own test cases. Here is precisely the way we will test your submitted functions. We will locate ourselves in the directory where your *assign1.js* is located. Then . . .

```
$ node
> .load /shared/naps/cs331/init.js                                  Initialize underscore library
⋮
> .load /shared/naps/cs331/little-schemer-12-primitives-for-assign1.js    Load Little Schemer's 12 primitives
⋮
> .load assign1.js                                                  Load your assignment file
⋮
> duple(6,5)                                                        We start our test cases
[ 5, 5, 5, 5, 5, 5 ]
>
⋮
                                          Many more devious test cases for all your functions
```

The six functions you need to write are:

1. Write a function

   var duple = function(n, x)

   that returns a list containing n copies of x. Example usage:

   ```
   > duple(6, 5)
   [ 5, 5, 5, 5, 5, 5 ]
   ```

2. Write a function

   var reverse = function (list)

   that reverses the list it is given. Example usage:

   ```
   > reverse([4, 5, 6, 7, 8])
   [ 8, 7, 6, 5, 4 ]
   ```

3. Write a function

   var flatten = function (list)

   that returns a list of the numbers in *list* in the order that they occur when *list* is printed. That is, flatten removes all the inner parentheses from its argument *list*. Example usage:

   ```
   > flatten([1, 2, [3, 4, 5], 6])
   [ 1, 2, 3, 4, 5, 6 ]
   ```

4. Write a function

   var down = function (l)

   that wraps list brackets around each top-level element of the list *l* that it is given as its only argument. Example usage:

   ```
   > down([1,2,3])
   [ [ 1 ], [ 2 ], [ 3 ] ]

   > down([[1],[2],[4]])
   [ [ [ 1 ] ], [ [ 2 ] ], [ [ 4 ] ] ]

   > down([1,[2,3],4])
   [ [ 1 ], [ [ 2, 3 ] ], [ 4 ] ]
   ```

```
> down([1, [2, [3]], 4])
[ [ 1 ], [ [ 2, [Object] ] ], [ 4 ] ]

> require('util').inspect(down([1, [2, [3]],4]), false, 10)
'[ [ 1 ], [ [ 2, [ 3 ] ] ], [ 4 ] ]'
```

Note that, if the output returned from the function does not display to the depth you need, you can use node's utility inspect function to display the output to a greater depth. Above a depth of 10 was used in the final example. In that example, 3 only has one set of brackets immediately enclosing it because the specifications for this problem state that list brackets are only wrapped around each top-level element.

5. Write a function

   var up = function (l)

   that removes a pair of brackets from each top-level element of the list *l* that it is given. If a top-level element is not a list, then it is included in the result as is. Example usage:

```
> up ([[1,2],[3,4]])
[ 1, 2, 3, 4 ]
> up ([[1, [2]], 3])
[ 1, [ 2 ], 3 ]
```

6. Suppose that we have a nested list representation of a binary search tree as indicated by the following *tree_test* var.

```
var tree_test = [14, [7, [], [12, [], []]],
                     [26, [20, [17, [], []],
                          [] ],
                          [31, [], []]]]
```

   Write a function

   var path = function (n, bst)

   where *n* is a number and *bst* is a binary search tree (using the representation above) that contains the number *n*. *path* should return a list of 0's and 1's showing how to find the node containing *n*, where 1 indicates "go right" and 0 indicates "go left". If *n* is found at the root, the empty list is returned. Example usage:

```
> var tree_test = [14, [7, [], [12, [], []]],
                     [26, [20, [17, [], []],
                          [] ],
                          [31, [], []]]]
... ... ... undefined
> tree_test
[ 14,
  [ 7, [], [ 12, [], [] ] ],
  [ 26, [ 20, [Object], [] ], [ 31, [], [] ] ] ]
> path(17, tree_test)
[1, 0, 0]
```