National University of Singapore
School of Computing
CS1101S: Programming Methodology (JavaScript)
Semester I, 2012/2013

Discussion Group Exercises 8

For this discussion group, you should make use of the Scratch Interpreter project, found on the course website. The link for this is *http://www.comp.nus.edu.sg/c̃s1101s/misc/scratch_interpreter.zip*.

Problems:

1. Recall back in Mission 1 you evaluated various JavaScript statements so as to get familiar with JavaScript. Now, at the end of CS1101S, we can use our interpreter to get a better understanding of how these statements might be evaluated by a JavaScript interpreter.

   If we evaluate "*var x = 3; x * 4;*", a sketch of how our interpreter evalutes this program might be:

   - calls *evaluate_sequence*, since we have a sequence of statements.
     - calls *evaluate* on the first statement, which calls *evaluate_Var_definition* to define *x* as 3.
     - Within *evaluate*, we call:
       - look up value of variable *.
       - look up value of variable *x*.
       - 4 is self-evaluating.

       and finally *apply* these operands with * to get our result.

   Show, similarly, how our interpreter evaluates each of the following statements:

```
==> 42;

==> 0000;

==> "the force!";

==> 6 * 9;

==> "2 + 3";

==> 1 / 0;

==> var red = 44;

==> var green = 43;

==> red - green;

==> var blue = green;

==> var purple = blue + green;
==> green(5);

==> (function(x, y, z){ return x; });

==> (function(x, y){ return x + y; })(4, 3 + 4);


==> (function(wow, it, works){ return wow(it, works); })
==> (function(a, b){ return a - b; }, 7, 5);

==> (function(wow, it, works){ return works(wow, it); })
==> (5, 7, function(a, b){ return a - b; });

==> var x = 2;
==> (function(x){ return x + x; })(5);

==> (function(rune){ return rune * rune; })(5);

==> rune;
```

2. One powerful feature you have learned in JavaScript is the ability to make use of closures. That is, functions can make use of variables which are not local to the function, even when the function has been returned from that scope which defined these variables.
   We use this closure property for *make_account* which you have seen previously.

   Explain how our JavaScript interpreter handles this closure functionality, using this example:

```
function makeAdder(k){
    return function(x){
            return x + k;
        };
}

var add1 = makeAdder(1);
var x = add1(5);
```

3. We can experiment with *lib/interpreter/interpreter.js*, and try to add more functionality. Try to add, or explain how you would add:
   - Assignment of global variables from within a function. For example, the program:

     ```
     function f(){
         x = 5;
     }
     f();
     x;
     ```

     Should evaluate to 5.

   - Short-circuit Evaluation of Boolean Expressions:
     In JavaScript, if we evaluate the expressions "$true \mid\mid nonsense$", or "$false\ \&\&\ nonsense$", where '$nonsense$' is not a defined variable, we get $true$ and $false$, respectively. This is because JavaScript evaluates these Boolean expressions in a "short circuit" manner, and will not evaluate the second operand if the first operand is enough to decide the Boolean expression.

At the moment, the interpreter will try to evaluate '*nonsense*', and will run into an error as '*nonsense*' is not defined.
The interpreter should then be able to correctly evaluate the expressions above.

4. Richard wants to play around with the interpreter to experiment with new language features for JediScript. He thinks he has a cool idea for a new language feature.
Richard would like to add a 'history' feature to the language our interpreter interprets, which would give access to the previous value of a variable. He gives the following example for how he thinks it should behave:

```
var x = 3;
x; // evaluates to 3
history(x); // evaluates to undefined, since we didn't have a value previously
x = 4;
x; // evalutes to 4
history(x); // evaluates to 3, as this was the previous value x had
```

Richard thinks this would be useful for situations such as:

```
// Swap values of x and y
x = y;
y = history(x); // set y to previous value of x.
```

or

```
// Get last item in list.
while(!is_empty_list(lyst)){
    lyst = tail(lyst);
}
lyst = history(lyst); // since lyst was the empty list, lyst now refers to the last item.
```

Explain whether, without needing to make changes to the syntax file, Richard could change our interpreter.js to simulate this functionality, and if he can, which changes Richard would need to make, and if he cannot, explain what limitation prevents him from doing this.