```
//List operations
function remove_duplicates(lst){
    if(is_empty_list(lst)){
        return [];
    }else{
        var h = head(lst);
        return pair(h,
               accumulate(function(a,b){
                       return a!==h?pair(a,b):b;
                      },[],remove_duplicates(tail(lst))));
    }
}
function are_equal_sets(set1,set2){
    var r = map(function(e){return !is_empty_list(member(e,set1));},set2);
    var b = accumulate(function(a,b){return a&&b;},true,r);
    if(length(set1)===0&&length(set2)===0){
        return true;
    }else if(length(set1)===length(set2)&&b){
        return true;
    }else{
        return false;
    }
}
function mergeB(xs, ys) {
  if (is_empty_list(xs) && is_empty_list(ys)) {
    return [];
  } else if (is_empty_list(xs)) {
    set_tail(ys, mergeB(xs, tail(ys)));
    return ys;
  } else if (is_empty_list(ys)) {
    set_tail(xs, mergeB(tail(xs), ys));
    return xs;
  } else if (head(xs) <= head(ys)) {
    set_tail(xs, mergeB(tail(xs), ys));
    return xs;
  } else {
    set_tail(ys, mergeB(xs, tail(ys)));
    return ys;
  }
}
//Trees
function accumulate_tree(op,init,tree){
    if(is_empty_list(tree)){
      return init;
    }else if(!is_list(head(tree))){
      return op(head(tree),accumulate_tree(op,init,tail(tree)));
    }else{
      return op(accumulate_tree(op,init,head(tree)),
               accumulate_tree(op,init,tail(tree)));
    }
}

function count_leaves(tree) {
  return (is_empty_list(tree))
         ? 0
         : (is_list(head(tree))
            ? count_leaves(head(tree))
            : 1)
           + count_leaves(tail(tree));
}

function longest_path(tree){
    if(!is_list(tree)){
```

```
        return 1;
    }else
    if(is_empty_list(tree)){
        return 1;
    }else if(!is_list(head(tree)))
    {
        return 1+longest_path(tail(tree));
    }else{
        return 1+Math.max(longest_path(head(tree)),longest_path(tail(tree)));
    }
}
//Somewhat related
function cc(amount, kinds) {
  if (amount === 0) {
    return 1;
  } else if (amount < 0 || kinds === 0) {
    return 0;
  } else {
    return cc(amount, kinds - 1) + cc(amount - highest_denom(kinds), kinds);
  }
}
//Permutations
function permutations(s) {
  if (is_empty_list(s)) {
    return list([]);
  } else {
    return accumulate(
          append, [],
            map(function(x) {
                return map(function(p) {
                        return pair(x,p);
                      },
                       permutations(remove(x,s)));
              },
            s)
            );
  }
}
function permutations_r(s,r) {
    if (r===0||is_empty_list(s)) {
        return list([]);
    } else {
        return accumulate(append, [],
                                  map(function(x) {
                                          return map(function(p) {
                                                          return pair(x, p);
                                                      },
                                              permutations_r(remove(x, s),r-1));
                                  }, s));
    }
}
function combinations(xs,k){
  if(k===0){
    return list([]);
  }else if(is_empty_list(xs)){
    return [];
  }else{
    var s1 = combinations(tail(xs),k-1);
    var s2 = combinations(tail(xs),k);
    var x = head(xs);
    var has_x = map(function(s){return pair(x,s);},s1);
    return append(has_x,s2);
  }
}
```

```
//Power set
function power_set(s){
  if(is_empty_list(s)){
    return list([]);
  }else{
    var rest = power_set(tail(s));
    return append(rest,map(function(x){return pair(head(s),x);},rest));
  }
}
//Subsequence
function all_subsequences(xs){
  if (is_empty_list(xs)){
    return list([]);
  }else{
    var ys = all_subsequences(tail(xs));
    return append(ys,
           map(
             function(y){
              return pair(head(xs),y);
             },
             ys));
  }
}
//Tower of Hanoi
function move_tower(size, from, to, extra) {
  if (size === 0) {
    ;
  } else {
    move_tower(size - 1, from, extra, to);
    display("move from " + from + " to " + to);
    move_tower(size-1, extra, to, from);
  }
}
//OOP
function Class(arg1,arg2){
  this.field1 = arg1;
  this.field2 = arg2;
  //or
  //SuperClass.call(this,arg1,arg2);
}
Class.Inherits(Class);
Class.prototype.method1 = function(a,b){
  //Superclass.prototype.methodname.call(a,b);
}
//Tream
var e = function(){return [];};
var a = pair(2,function(){return pair(4,e);});
var b = pair(3,function(){return pair(5,e);});
var t = pair(a,function(){return pair(b,e);});

function tream_map(f, t){
  if (is_empty_list(t)){
    return [];
  } else {
    var x = function() {
      return tream_map(f, stream_tail(t));
    };
    if(is_pair(head(t))){
      return pair(tream_map(f, head(tree)), x);
    }else{
      return pair(f(head(tree)), x);
    }
  }
}
```

```
//Loop
var count_pair= (function(seen){
  function fn(x){

    if(! is_pair(x)){
      return 0;
    }else if(!is_empty_list(member(x,seen))){
      return 0;
    }else{
        //display(x); //For debugging
      seen = append(list(x),seen);
      //display("What is in seen is "+seen);
      return count_pair(head(x))+count_pair(tail(x))+1;
    }
  }
  return fn;
})([]);
var has_loop = (function(seen){
  function fn(x){
    if(is_empty_list(x)){
      return false;
    }else if(!is_empty_list(member(x,seen))){
      return true;
    }else{
      seen = append(list(x),seen);
      return has_loop(tail(x));
    }
  }
  return fn;
})([]);
function has_loop(lst){
    function next(xs){
        if(equal(xs,false)||is_empty_list(xs)||is_empty_list(tail(xs))){
            return false;
        }else{
            return tail(xs);
        }
    }
    function fn(p1,p2){
        display(p1);
        display(p2);
      if(equal(p1,false)||equal(p2,false)){
        return false;
      }else{
        if(p1===p2){
          return true;
        }else{
          return fn(next(p1),next(next(p2)));
        }
      }
    }
    return fn(next(lst),next(next(lst)));
}
function memoize(f) {
  var table = make_table();
  return function(x) {
    if (has_key(x, table)) {
      return lookup(x, table);
    } else {
      var result = f(x);
      insert(x, result, table);
      return result;
    }
  }
```

```
}
function memo_fun(fun) {
  var already_run = false;
  var result = undefined;
  return function() {
        if (!already_run) {
          result = fun();
          already_run = true;
          return result;
        } else {
          return result;
        }
      };
}
//Mutable List
function mutable_reverse(xs) {
// VERSION 1
  if (is_empty_list(xs)) {
    return xs;
  } else if (is_empty_list(tail(xs))) {
    return xs;
  } else {
    var temp = mutable_reverse(tail(xs));
    set_tail(tail(xs), xs);
    set_tail(xs, []);
  return temp;
// VERSION 2
  function helper(prev, xs) {
    if (is_empty_list(xs)) {
      return prev;
    } else {
      var rest = tail(xs);
      set_tail(xs, prev);
      return helper(xs, rest);
    }
  }
  return helper([], xs);
}
//Stream
function inverse_unit_series(s){
  return pair(1,function(){return scale_series(-1,mul_series(stream_tail(s),
    inverse_unit_series(s)));});
}
function div_series(s1,s2){//s1 / s2 check divisibility before dividing
  var constant_term = head(s2);
  if(constant_term===0){
    return "Error: divided by 0";
  }else{
    return scale_stream(1/constant_term,function(){return mul_series(s1,inverse_unit_series(
    scale_series(1/constant_term,s2)));});
  }
}

function stream_append_pickle(xs, ys) {
  if (is_empty_list(xs)) {
    return ys();
  } else {
    return pair(head(xs),
          function() {
            return stream_append_pickle(stream_tail(xs),
                        ys);
          });
  }
}
```

```
318
319 function sieve(s) {
320   return pair(head(s),
321     function() {
322       return sieve(stream_filter(function(x) {return !is_divisible(x, head(s));},stream_tail(
    s)));
323     });
324 }
325 var primes = sieve(integers_from(2));
326 //Intepreter
327 //Question 2
328 function is_variable_assignment(stmt){
329     return stmt.tag==="assignment";
330 }
331 //Question 2
332 function is_variable_assignment(stmt){
333     return stmt.tag==="assignment";
334 }
335 function set_variable_value(variable,val, env) {
336     function env_loop(env) {
337         if (is_empty_environment(env)) {
338             error("Unbound variable: " + variable);
339         } else if (has_binding_in_frame(variable.name, first_frame(env))) {
340             first_frame(env)[variable.name] = force(evaluate(val,env));
341             return undefined;
342             /*
343             var val = force(first_frame(env)[variable]);
344             first_frame(env)[variable] = val;
345             return val;
346             */
347         } else {
348             return env_loop(enclosing_environment(env));
349         }
350     }
351     return env_loop(env);
352 }
353 //Question 3
354 function is_boolean_operation(stmt){
355     return stmt.tag==="boolean_op";
356 }
357 function evaluate_boolean_operation(val1, val2, operator, env){
358     var v1 = force(evaluate(val1,env));
359     var v2 = force(evaluate(val2,env));
360     if( operator==="||"){
361         return v1||v2;
362     }else{
363         return v1&&v2;
364     }
365 }
366 //Question 4
367 function is_while_statement(stmt){
368     return stmt.tag==="while";
369 }
370 function evaluate_while_statement(predicate,list_of_statements,env){
371     function loop(remaining_statements){
372         if(is_empty_list(remaining_statements)){
373             return evaluate_while_statement(predicate,list_of_statements,env);
374         }else{
375             force(evaluate(head(remaining_statements),env));
376             loop(tail(remaining_statements));
377         }
378     }
379     if(force(evaluate(predicate,env))===true){
380         loop(list_of_statements);
```

```
    }else{
        return undefined;
    }
}
//For loop
function is_for_loop(stmt){
    return stmt.tag==="for";
}
function evaluate_for_loop(initialiser,predicate,finaliser,statements,env){
    function loop(remaining_statements){
        if(is_empty_list(remaining_statements)){
            force(evaluate(finaliser,env));
            if(force(evaluate(predicate,env))===true){
                return evaluate_for_loop(undefined,predicate,finaliser,statements,env);
            }else{
                return undefined;
            }
        }else{
            var h = head(remaining_statements);
                force(evaluate(head(remaining_statements),env));
                loop(tail(remaining_statements));
        }
    }
    if(initialiser===undefined){
        ;
    }else{
        force(evaluate(initialiser,env));
    }
    loop(statements);
    return undefined;
}

//Reference
build_list(n, f) //Makes a list with n elements
           by applying the unary function f to the numbers 0 to n - 1.
for_each(f, xs)  //Applies f to every element of the list xs, and then returns true.
remove(x, xs)  remove_all(x, xs)
enum_list(start, end)
         // Returns a list that enumerates numbers starting from start
         // using a step size of 1, until the number exceeds (>) end.
```

All da best tomorrow!