## Question 1: Theoretical Questions

**Q1.1** Give an example for each of the following categories in L3:

- Primitive atomic expression: 3, #t, <
- Non-primitive atomic expression: x
- Non-primitive compound expression: (+ 3 5), (if #t 3 4)
- Primitive atomic value: the numerical value of 3, the logical value of false
- Non-primitive atomic value: the symbol ‘green in L3
- Non-primitive compound value: the list (1, 2, 3)

**Q1.2** What is a special form? Give an example
A special form is an expression that is evaluated in a non-standard way [= not a procedure application].
Example: (if #t 5 6)

**Q1.3** What is a free variable? Give an example [2 points]
A variable x occurs free in an expression E if and only if there is some use of x in E that is not bound by any declaration of x in E.
Example: y occurs free in (lambda (x) (+ x y))

**Q1.4** What is Symbolic-Expression (s-exp)? Give an example
S-expression is an hierarchical structures (tree) of values.
We met two kind of s-exp in the course:
A tree of program tokens, which is the input for the parser: [‘<’, [‘+’, ‘3’,’4’], [‘/’, ‘10’, ‘2’]]
A tree of expressions / values: ‘(1 2 (3 4) / (1 2 (3 4))

**Q1.5** Give an example for syntactic abbreviation.
*cond* is a syntactic abbreviation of *if*, *let* is a syntactic abbreviation of *lambda*:

```
(cond (((> x 3) 4)
       ((> y 8) 5)
       (else 6))
⇒
(if (> x 3)
   4
   (if (< y 8)
      5
      6))
```

```
(let
  ((a 1) (b 2))
  (*a b))
⇒
(
  (lambda  (a b)
    (* a b))
  1 2
)
```

**Q1.6** Let us define the L0 language as L1 excluding the special form *define*. Is there a program in L1 which cannot be transformed to L0? Explain or give a contradictory example.

If we forbid (in the interpreter) the appearance of the defined variable in its defined values (which has no meaning for a language without user procedures and recursion calls), any var reference in L1 can be replaced by its defined value expression.

**Q1.7** Let us define the L20 language as L2 excluding the special form *define*. Is there a program in L2 which cannot be transformed to L20? Explain or give a contradictory example.

Yes. Since L2 support user procedure which may contain recursion calls, a variable which is defined by lambda may appear in the body of the lambda as a recursion call. So the var references cannot be replaced by their defined value.

**Q1.8** In practical session 5, we dealt with two representations of primitive operations: *PrimOp*, *Closure*. Denote an advantage for each of the two methods.

PrimOp: no need for environment lookup at any operator application.

Closure: no need for interpreter modification for new defined primitive operations.

**Q1.9** In class, we implemented *map* in L3, where the given procedure is applied on the first item of the given list, then on the second item, and so on. Would other implementation which applies the procedure in opposite order (from the last item to the first one), while keeping the original order of the items in the returned list, be equivalent? Would this be the case also for *reduce*.

The order of the procedure application for the case of *map*, since the procedure gets only the list item, and since L3 is functional with no side effects. This implies that the procedure applications on the list items are independent.

In contrast, the procedure application in reduce gets beside the list item the aggregated value of the previous items, so the order of the application matters.
For example: (reduce \ 1 ‘(1 2 3)) would return 1/6 for one order, and 3/2 for the opposite order.

## Question 2: Programing in L3

```
;Q2.1
; Signature: empty?(lst)
; Type: [Any -> Boolean]
; Purpose: query for list emptiness
; Pre-conditions: true
; Tests: (empty? '()) => true, (empty? '(1)) => false
(define empty?
   (lambda (lst)
      (eq? lst '())))


;Q2.2
; Signature: list?(lst)
; Type: [Any -> Boolean]
; Purpose: type predicate for lists
; Pre-conditions: true
; Tests: (list? '(1 2)) => true, (list? (cons 1 2)) => false
(define list?
   (lambda (lst)
      (if (empty? lst)
          #t
          (if (pair? lst)
             (if (list? (cdr lst))
                #t
                #f)
              #f)
      )
   )
)

;Q2.3
; Signature: equal-list?(lst1,lst2)
; Type: [List<Any>*List<Any> -> Boolean]
; Purpose: equality operator for lists
; Pre-conditions: true
; Tests:(equal-list? '(1 2) ‘(1 2)) => true, (equal-list? ‘(1) ‘(2)) => false
```

```
(define equal-list?
    (lambda (lst1 lst2)
      (if (and (pair? lst1) (pair? lst2))
          (and (or (eq? (car lst1) (car lst2))
                   (equal-list? (car lst1) (car lst2)))
               (equal-list? (cdr lst1) (cdr lst2)))
          (and (empty? lst1) (empty? lst2))
      )
    )
)



;Q2.4
; Signature: append(list1,list2)
; Type: [List(Any)*List(Any) -> List(Any)]
; Purpose: concatenate list1 and list2
; Pre-conditions: true
; Tests: (append '(1 2 3) '(4 5 6)) => '(1 2 3 4 5 6)
(define append
  (lambda (list1 list2)
    (if (empty? list1)
        list2
        (cons (car list1)
              (append (cdr list1) list2)))))

;Q2.5
; Signature: append3(list1,list2,num)
; Type: [List(Any)*List(Any)*Any -> List(Any)]
; Purpose: concatenate list1, list2 and num
; Pre-conditions: true
; Tests: (append3 '(1 2 3) '(4 5 6) 7) => '(1 2 3 4 5 6 7)
(define append3
  (lambda (list1 list2 num)
    (if (empty? list1)
        (append list2 (list num))
        (cons (car list1)
              (append3 (cdr list1) list2 num))
    )
 )
)

; Q2.6
```

```
; Signature: pascal(n)
; Type: [Number -> List(Number)]
; Purpose: return nth row of Pascal's triangle
; Pre-conditions: n is a natural number
; Tests:(pascal 5) => '(1 4 6 4 1)
(define pascal (lambda (n)
                 (if (= n 1)
                     (list 1)
                     (append3 (list 1) (list_of_sums (pascal (- n 1))) 1))))

(define list_of_sums (lambda (lst)
                       (if (empty? lst)
                           lst
                           (if (equal-list? lst (list 1))
                              '()
                               (append
                                       (list (+ (car lst) (car (cdr lst))))
                                       (if (empty? (cdr lst))
                                          '()
                                          (list_of_sums (cdr lst))
                                       )
                               )
                           )
                      )
                        )
)
```

## Question 3: Syntactic Transformation

```
/*
Purpose: Transform L3 AST to L30 AST
Signature: l3ToL30(l3AST)
Type: [Parsed | Error] => [Parsed | Error]
*/
export const l3ToL30 = (exp: Parsed | Error): Parsed | Error  =>
    isError(exp) ? exp :
    isCExp(exp) ? l3ToL30CExp(exp) :
    isDefineExp(exp) ? makeDefineExp(exp.var, l3ToL30CExp(exp.val)) :
    isProgram(exp) ? makeProgram(map(l3ToL30, exp.exps)) :
    exp;

const l3ToL30CExp = (exp: CExp): CExp =>
```

```
    isAtomicExp(exp) ? exp :
    isLitExp(exp) ? listLit2Cons(exp) :
    isIfExp(exp) ? makeIfExp(l3ToL30CExp(exp.test),
                             l3ToL30CExp(exp.then),
                             l3ToL30CExp(exp.alt)) :
    isAppExp(exp) ? listApp2Cons(exp) :
    isProcExp(exp) ? makeProcExp(exp.args, map(l3ToL30CExp, exp.body)) :
    isLetExp(exp) ? makeLetExp(
        zipWith(makeBinding,
                map((binding)=> binding.var.var,exp.bindings),
                map((binding)=> l3ToL30CExp(binding.val),exp.bindings)),
        map(l3ToL30CExp,exp.body)) :
    exp;


const listApp2Cons = (exp: AppExp): LitExp | AppExp =>
    (isPrimOp(exp.rator) && exp.rator.op === 'list') ?
    (
        exp.rands.length === 0 ? makeLitExp(makeEmptySExp()) :
        exp.rands.length === 1 ?
            makeAppExp(makePrimOp('cons'),
              [l3ToL30CExp(first(exp.rands)), makeLitExp(makeEmptySExp())]) :
            makeAppExp(makePrimOp('cons'),
              [l3ToL30CExp(first(exp.rands)),
               listApp2Cons(makeAppExp(makePrimOp('list'),rest(exp.rands)))])
    ) :
    makeAppExp(l3ToL30CExp(exp.rator), map(l3ToL30CExp, exp.rands));

const listLit2Cons = (exp: LitExp): LitExp | AppExp =>
    isEmptySExp(exp.val) ? exp :
    isCompoundSExp(exp.val) ?
     (isQuoteExp(exp.val.val1) ?
        (isEmptySExp(exp.val.val2) ?
            makeAppExp(makePrimOp('cons'),
              [makeLitExp(exp.val.val1), makeLitExp(makeEmptySExp())]) :
            makeAppExp(makePrimOp('cons'),
              [makeLitExp(exp.val.val1),
               listLit2Cons(makeLitExp(exp.val.val2))])
        ) :
       (isEmptySExp(exp.val.val2) ?
        makeAppExp(makePrimOp('cons'),
           [listLit2Cons(makeLitExp(exp.val.val1)),
            makeLitExp(makeEmptySExp())]) :
```

```
        makeAppExp(makePrimOp('cons'),
         [listLit2Cons(makeLitExp(exp.val.val1)),
          listLit2Cons(makeLitExp(exp.val.val2))])
      )
     )
     : exp;

const isQuoteExp = (exp: SExp): boolean =>
 isCompoundSExp(exp) && isSymbolSExp(exp.val1) && exp.val1.val ==='quote';
```

## Question 4: Code translation

```
/*
Purpose: Transform L2 AST to Python program string
Signature: l2ToPython(l2AST)
Type: [Parsed | Error] => [string | Error]
*/
export const l2ToPython = (exp: Parsed | Error): string | Error =>
    isError(exp) ? exp.message :
    isProgram(exp) ? map(l2ToPython,exp.exps).join("\n") :
    isBoolExp(exp) ? (exp.val ? 'True' : 'False') :
    isNumExp(exp) ? exp.val.toString() :
    isVarRef(exp) ? exp.var :
    isDefineExp(exp) ? exp.var.var + " = " + l2ToPython(exp.val) :
    isProcExp(exp) ? "(" + "lambda " +
                         map((p) => p.var, exp.args).join(",") + ": " +
                         l2ToPython(exp.body[exp.body.length-1]) +
                     ")" :
    isIfExp(exp) ? "(" + l2ToPython(exp.then) +
                        " if " +
                        l2ToPython(exp.test) +
                        " else " +
                        l2ToPython(exp.alt) +
                   ")" :
    isAppExp(exp) ?
        (isPrimOp(exp.rator) ?
            primOpApp2Python(exp.rator, exp.rands) :
            l2ToPython(exp.rator) + "(" +
                       map(l2ToPython, exp.rands).join(",") + ")") :
```

```
    Error("Unknown expression: " + exp.tag);


const primOpApp2Python = (rator : PrimOp, rands : CExp[]) : string =>
    rator.op === "not" ? "(not " + l2ToPython(rands[0]) + ")" :
    rator.op === "and" ? "(" + map(l2ToPython,rands).join(" && ") + ")" :
    rator.op === "or" ? "(" + map(l2ToPython,rands).join(" || ") + ")" :
    "(" + map(l2ToPython,rands).join(" " +
                      (rator.op === '=' ? '==' : rator.op) + " ") + ")"
```