1.1 Is a function-body with multiple expressions required in a pure functional programming? In which type of languages is it useful? What about L3? [2 points]

לא, זה שימושי רק כאשר יש side-effects, ע"פ הסמנטיקה הערך של התוכנית
הוא ערכו של הביטוי האחרון, כאשר אין side-effects אזי ערכו של הביטוי האחרון
אינו מושפע מהביטויים הקודמים.
במקרה של L_3, יש צורך לחשב רק את ה- define ולאחריהם ניתן לקבל לביטוי
האחרון.

Q1.2

a. Why are special forms required in programming languages? Why can't we simply define them as primitive operators? Give an example [3 points]
b. Can the logical operation 'or' be defined as a primitive operator, or must it be defined as a special form? Refer in your answer to the option of shortcut semantics. [3 points]

a. Special forms are required in case the semantics of the evaluation does not follow the default 'procedure application semantics, i.e, special forms have unique evaluation rule, and they allow control over the evaluation of their arguments, which primitive operators can't do.

For example: 'if' special form, doesn't required pre-evaluation of the 'then' and 'else' sub-expressions.

If 'if' were a premitive operator, both 'then' and 'else' would be evaluated regardless of the condition, the evaluation of this expression:
if(> 2 5) 10 (\ 8 0), would case a division by zero error, because premitive operator would attempt to evaluate both '10' and (/8 0) even though '2 > 5' is 'false.

Q1.2.b :

The logical operation 'or' must be defined as special form to enable shortcut semantics.

Shortcut semantics stop evaluation as soon as a true value is found, if 'or' were primitive operator, all arguments would be evaluated regardless of their values, potentially causing errors or unnecessary computations.

Q1.3

a. What is the value of the following L3 program? Explain. [2 points]

```
(define x 1)
(let ((x 5)
      (y (* x 3)))
  y)
```

The value is 3, explaination:

- Global definition: 'x' is defined as '1'.
- let expression: 'x' is locally set as 5
  'y' is calculated using the global 'x' whis is 1, so $y = 1 * 3 = 3$
- Body evaluation: The body of 'let' expression returns 'y' which is 3

Therefore the value of the program is 3.

b. Read about let* here.
What is the value of the following program? Explain. [2 points]

```
(define x 1)
(let* ((x 5)
       (y (* x 3)))
  y)
```

The value is 15, explaination:

The 'let*' form sequentially binds variables, allowing each subsequent binding to reference the ones before it, In the program, 'x' is locally bound to '5' and 'y' as (*x 3), resulting in 15.

c. Define the let* expression above as an equivalent let expression [2 points]
d. Define the let* expression above as an equivalent application expression (with no let) [2 points]

c.
```
(let ((x 5)
      (y (* x 3)))
  y)
```

d.
```
((lambda (x)
   ((lambda (y) y)
    (* x 3)))
 1)
```

Q.4 [6 points]

a. In L3, what is the role of the function valueToLitExp?
b. The valueToLitExp function is not needed in the normal evaluation strategy interpreter (L3-normal.ts). Why?
c. The valueToLitExp function is not needed in the environment-model interpreter. Why?

a. The role is to turn back values of type value into expressions. It's needed because in the substituting model, applecation are computed by substituting computed values into the body of closures, which are expressions. Therefore, this convertion is necessary to ensure compatibility of types and facilitate the computational process.

b. because argument are not evaluated before the substitution, so there is no types problem.

c. because in this model no substitution is performed, so there is no types problem.

Q.5 [4 points]

a. What are the reasons that would justify switching from applicative order to normal order evaluation? Give an example.

b. What are the reasons that would justify switching from normal order to applicative order evaluation? Give an example.

a. Reasons:

1. Avoid unnecessary computations until needed.

2. Error prevention: prevent errors or infinity loops by delaying evaluation.

Example:

```
(
 (lambda (x y z) (if x y z))
 #t 3 (sqrt 2)
)
```

אפליקטיב: כל שלושת האופרנדים מחושבים מראש

(if true 3 1.412…)

נורמל: האופרנדים מוצבים כפי שהם, החישוב נדחה לזמן בו האופרנד נדרש

(if #t 3 (sqrt 2))

במקרה זה נורמל יעיל יותר, אין צורך מעשי לחשב את הביטוי של ה (alt (sqrt 2

b. Reasons:

1. Applicative order can avoid redundant computation, as all arguments are evaluated once upfront.

2. Avoid redundant side effects.

Example:

```
)
 (lambda (x) (+ x x))
 (* 2 3)
)
```

אפליקטיב: נחשב תחילה את (* 2 3) ונציב

(6 6 +)

נורמל: נציב (* 2 3) מבלי לחשבם בשלב זה. הם יחושבו מאוחר יותר כאשר זה יידרש עבור הפעלת האופרטור הפרימיטיבי +.

(+ (* 2 3) (* 2 3))

בדוגמא זו, אפליקטיב יעיל יותר, בנורמל החישוב של (* 2 3) יבוצע פעמיים.

Q.6 [4 points]

a. Why is renaming not required in the environment model?
b. Is renaming required in the substitution model for a case.where the term that is substituted is "closed", *i.e.*, does not contain free variables? explain.

a. because each function has it's own separate space (enviroment) for variables, This keep variables from different functions or enviroments from interfering with each other.

b. Renaming isn't needed when substituting a closed term because there are no outside variables that could cause conflicts. the term is self contained, so it doesn't interfere with other variables.

in other words In the substitution algorithm when encountring a procEXP, the bounded variables are filtered out, so once free variable are out of the picture, no renaming is required, without renaming the variable filtering is imperative.

Q2.d :

- Convert the ClassExps to ProcExps (as defined in 2.c)

```
(define pi 3.14)
  (define square (lambda (x) (* x x)))
    (define circle
      (lambda (x y radius)
        (lambda (msg)
          (if (eq? msg 'area)
              ((lambda () (* (square radius) pi)))
              (if (eq? msg 'perimeter)
                  ((lambda () (* 2 pi radius)))
                  #f)))))
  (define c (circle 0 0 3))
  (c 'area)
```

- List the expressions which are passed as operands to the L3applicativeEval function during the computation of the program, (after the conversion), for the case of substitution model.

- 3.14
- (lambda (x) (* x x))
- (lambda (x y radius)
  (lambda (msg)
  (if (eq? msg 'area)
  ((lambda () (* (square radius) pi)))
  (if (eq? msg 'perimeter)
  ((lambda () (* 2 pi radius)))
  #f)))))
- (circle 003)
- 0
- 0
- 3
- (lambda (msg)
  (if (eq? msg 'area)
  ((lambda () (* (square radius)) pi)))
  (if (eq? msg 'perimeter)
  ((lambda () (* 2 pi radius
  #f)))))
- 'area

```
(if (eq? msg 'area)
    ((lambda () (* (square radius) pi)))
    (if (eq? msg 'perimeter)
        ((lambda () (* 2 pi radius)))
        *f )))))
```

- (lambda () (* (square radius) pi))
- (* (square radius) pi)
- (square radius)
- radius
- (square 3)
- 3
- (* 3 3)
- 9
- (* 9 pi)

output 28.26

- Draw the environment diagram for the computation of the program (after the conversion), for the case of the environment model interpreter.