Assignment 2

שאלה 1:

1.1. צורות מיוחדות (special forms) דרושות בשפות תכנות מכיוון שניתן בעזרת הוספה של כל צורה כזאת להעשיר את מבנה השפה בדרך שלא הייתה קיימת עד אותו זמן. לדוגמא בהוספת הצורה המיוחדת if נוספת האפשרות שפעולה מסוימת תתבצע רק בהינתן תנאי מסוים, בהוספת הצורה המיוחדת lambda ניתן לקחת קטע קוד ולהפוך אותו לפונקציה ובכך לקרוא לה בהמשך, בהוספת הצורה המיוחדת define ניתן לשמור ערך של משתנה וכדומה.
נבחין כי לא ניתן להגדירן כאופרטורים פרימיטיביים בשפה מכיוון שהדרך בה מחשבים פרוצדורה אשר מסתמכת על אופרטורים פרימיטיביים היא: חישוב האופרטור (הביטוי השמאלי ביותר), חישוב כל אחד מהפרמטרים שמימינו והפעלת האופרטור על אותם הפרמטרים. דרך פעולה זו אינה תעבוד במקרה של ה-special forms, מכיוון שמקרים אלו דרך הפעולה להפעלת הפרוצדורה תהיה ספציפית לאותה הצורה המיוחדת, לכל צורה מיוחדת יש אופן פעולה ייחודי.
דוגמא לכך: בצורה המיוחדת define, המילה השמורה define מזוהה בהתחלה, כתוצאה מכך מתבצע חישוב הביטוי אשר נמצא בחלק הימני ואז מתבצע binding בין הביטוי המחושב (הערך שלו) לבין המשתנה הרשום משמאלו. אילו היינו מפעילים את define כאופרטור פרימיטיבי אז מלבד זאת שהוא לא היה ממלא את ייעודו, התוכנית גם לא הייתה מתקמפלת.

1.2. תוכנית ב-L1 אשר מציאת הערך שלה יכול להתבצע במקביל:

```
(* (+ 4 5) (- 3 1))
(= 5 (+ 2 3))
```

תוכנית ב-L1 אשר מציאת הערך שלה לא יכול להתבצע במקביל:

```
(define x 2)
(define y (+ x 3))
(+ y x)
```

1.3. כל תוכנית שניתן לכתוב ב-L1, ניתן להמירה לתוכנית שקולה ב-L0. מה שפעולת הdefine מאפשרת בשפה L1 היא הפשטה לביטויים. היא גורמת לתוכנית להיראות יותר ברורה, היא גורמת לכך שיהיו פחות טעויות אשר נוצרות על ידי המתכנת, אך היא לא הכרחית בשפה L1, מכיוון שבכל תוכנית ניתן להמיר את המשתנה אשר הוגדר בdefine במה שהמשתנה מייצג ובכך כל תוכנית ב-L1 תהיה שקולה לתוכנית ב-L0.

1.4. קיימות תוכניות ב-L2 שלא ניתן להמירן לתוכנית שקולה ב-L-20. לדוגמא:

```
(define factorial
 (lambda (n)
  (if (< n 2)
    1
    (* n (factorial (- n 1))))))
```

1.5. map- סדר הפעלת הפרוצדורה על איברי הרשימה יכול להיות מבוצע באופן מקבילי.
הפונקציה map עוברת על אברי הרשימה ומבצעת את אותה הפעולה על כל אחד מהאיברים ללא תלות בפעולות על שאר האיברים.
filter- סדר הפעלת הפרוצדורה על איברי הרשימה יכול להיות מבוצע באופן מקבילי.
הפונקציה filter עוברת על אברי הרשימה ומבצעת את אותה הפונקציה הבוליאנית על כל אחד מהאיברים ומחזירה כל אחד מהאיברים שצלחו את אותה הפונקציה וזאת ללא תלות בפעולות על שאר האיברים.
reduce- סדר הפעלת הפרוצדורה על איברי הרשימה חייב להיות מבוצע באופן סדרתי, אחרת ייתכן שנקבל פלטים שונים על אותו קלט.
דוגמא: נניח כי פונקציית הreduce ממומשת באופן הבא (כפי שנראה בהרצאה):

```
(define reduce
 (lambda (reducer init lst)
  (if (empty? lst)
     init
     (reducer (car lst)
              (reduce reducer init (cdr lst))))))
```

נניח כי היא מקבלת כקלט את הרשימה [1,2] עם ערך התחלתי 1 ומתבצעת פעולת החילוק
נראה שני תרחישים המובילים לקבלת ערך שונה:

א) היא תעבור תחילה על האיבר הראשון ברשימה ולאחר מכן על האיבר השני ברשימה.
חישוב זה יוביל לערך הבא:

$$1/(reduce\ /\ 1'(1\ 2))\rightarrow 1/(1/(reduce\ /\ 1\ '(2)))\rightarrow 1/(1/(2(reduce\ /\ 1\ '()))\rightarrow 1/(1/2)\rightarrow 2$$

ב) היא תעבור תחילה על האיבר השני ברשימה ולאחר מכן על האיבר הראשון ברשימה.
חישוב זה יוביל לערך הבא:

$$1/(reduce\ /\ 1\ '(2\ 1))\rightarrow 1/(2/(reduce\ /\ 1\ '(2)))\rightarrow 1/(2/(1(reduce\ /\ 1\ '()))\rightarrow 1/(2/1)\rightarrow 1/2$$

all- סדר הפעלת הפרוצדורה על איברי הרשימה יכול להיות מבוצע באופן מקבילי.
הפונקציה all עוברת על אברי הרשימה ומבצעת את אותה הפונקציה הבוליאנית על כל אחד מהאיברים ומחזירה תשובה עבור כל אחד מהאיברים הללו וזאת ללא תלות בפעולות על שאר האיברים.
compose- סדר הפעלת הפרוצדורה על איברי הרשימה חייב להיות מבוצע באופן סדרתי כי אחרת ייתכן שנקבל פלטים שונים על אותו קלט.
דוגמא: הפעלת פונקציית הcompose על רשימת הפרוצדורות [map,filter] כאשר הפונקציה map מעלה את אברי הרשימה ב-1 והפונקציה filter מחזירה את האיברים הזוגיים ברשימה.
נניח כי ישנה הרשימה (1,2,3) אז ישנם שני תרחישים:

א) קודם תופעל על הרשימה הפונקציה map ותוחזר הרשימה (2,3,4), לאחר מכן תופעל הפונקציה filter ותוחזר הרשימה (2,4).

ב) קודם תופעל על הרשימה הפונקציה filter ותוחזר הרשימה (2), לאחר מכן תופעל הפונקציה map ותוחזר הרשימה (3).

1.6. הערך שיתקבל בתוכנית הוא 9.
במהלך התוכנית מתבצעות פעולות define כך ש-b מקבל את הערך 1, c מקבל את הערך 2, pair מקבל ביטוי מטיפוס classExp ו-p34 ביטוי מטיפוס pair.

כעת כאשר מחושב הביטוי (5 ((lambda (c) (p34 'f)), התוכנית מזהה את p34 כpair, וכאשר היא מגיעה לשורה (((f (lambda () (+ a b c) היא רואה ש-a ו-b מוגדרים כפרמטרים בpair עם הממספרים 3 ו-4 בהתאמה ומציבה אותם בהתאם. לעומת זאת, c אינו מופיע כפרמטר בp34 ולכן מכיוון ש-c הוא varRef התוכנית ניגשת לסביבה על מנת למצוא את ערכו, כך שבמקרה זה לlambda(c) עם הערך 5 אין משמעות ו-c מקבל את הערך 2 שהוגדר ב define בתחילת התוכנית, כלומר מתבצע החישוב 3+4+2=9.

שאלה 2:

1) $; Signature: append\ (lst1\ lst2)$
$; Type: [List * List \rightarrow List]$
$; Purpose: Procedure\ which\ gets\ two\ lists\ and\ returns\ their\ concatenation$
$; Pre-conditions: none$
$; Tests: (append\ '(1\ 2)\ '(3\ 4)) \rightarrow '(1\ 2\ 3\ 4)$

2) $; Signature: reverse\ (lst)$
$; Type: [List \rightarrow List]$
$; Purpose: Procedure\ which\ gets\ a\ list\ and\ reverses\ it$
$; Pre-conditions: none$
$; Tests: \left(reverse\ '(1\ 2\ 3\ 4)\right) \rightarrow '(4\ 3\ 2\ 1)$

3) $; Signature: duplicate-items\ (lst\ dup-count)$
$; Type: [List * List(Number) \rightarrow List]$
$; Purpose: Procedure\ which\ gets\ two\ lists: lst\ , dup-count\ and\ duplicates\ each\ item\ of\ lst\ according\ to\ the\ number\ defined\ in\ the\ same\ position\ in\ dup-count.$
$; Pre-condition: dup-count\ is\ not\ empty\ and\ all\ the\ numbers\ there\ are\ natural\ numbers$
$; Tests: (duplicate-items\ '(1\ 2\ 3)'(2\ 1\ 0\ 1\ 0\ 2)) \rightarrow '(1\ 1\ 2)$

4) $; Signature: payment\ (n\ coins-lst)$
$; Type: [Number * List(Number) \rightarrow Number]$
$; Purpose: Procedure\ which\ gets\ a\ sum\ of\ money\ and\ list\ of\ available\ coins, and\ returns\ the\ number\ possible\ ways\ to\ pay\ with\ these\ coins$
$; Pre-conditions: n\ and\ the\ numbers\ in\ coins-lst\ are\ non-negative\ numbers$
$; Tests: \left(payment\ 5\ '(1\ 1\ 1\ 2\ 2\ 5\ 10)\right) \rightarrow 3$

5) $; Signature: compose-n\ (f\ n)$
$; Type: [(T \rightarrow T) * Number \rightarrow (T \rightarrow T)]$
$; Purpose: Procedure\ which\ an\ unfary\ function\ f\ and\ a\ number\ n\ and\ returns\ the\ closure\ of\ the\ n-th\ self\ composition\ of\ f$
$; Pre-conditions: n\ is\ a\ natural\ positive\ number$
$; Tests: (define\ mul8\ (compose-n\ (lambda\ (x)\ (*\ 2\ x))\ 3))$
$(mul8\ 3) \rightarrow 24$