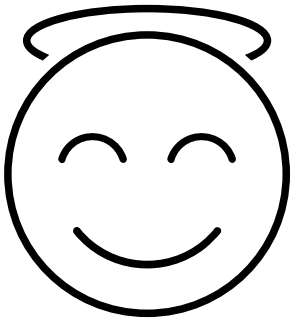
תרגיל בית 1

שם	ת.ז	מייל
עבידאת מוחמד	319053823	obedat@campus.technion.ac.il
מוחמד חטיב	206890998	kh.mohammad@campus.technion.ac.il



שאלה 1)

התבקשנו לממש פונקציה Cat שמקבלת שתי רשימות ומחזירה שרשורן

נשתמש בשתי פונקציות עזר :

```
(defun cat (l1 l2)(cat_help(swap l1 ()) l2))

(defun cat_help (li1 li2)(cond
((Eq li1 nil) li2); if li1 =nil => li2( maybe li2 = nil)
((eq li2 nil) li1); if li2 = nil => li1 (li1 != nil)
((eq (cdr li1) nil) (cons(car li1) li2))
(t(cat_help(cdr li1) (cons (car li1) li2)))))

(defun swap (x y)(cond
((Eq x nil)nil)
((eq (cdr x) nil) (cons (car x) y))
(t(swap (cdr x) (cons (car x) y)))))
```

שאלה 2)

1. בדיקה מדוקדקת תגלה הבדל נוסף בין שני המימושים של הפונקציה exists חוץ מאשר עליו מצביע הדיון. אתר הבדל זה, והסבר מדוע הוא אינו משנה את הסמנטיקה של המימוש.

מימוש ראשון :

```
(defun exists (x xs) ; determine whether atom x is in list xs
  (cond ; three case to consider
    ((eq xs nil) nil) ; (i) list of xs is exhausted
    ((eq x (car xs)) t) ; (ii) item x is first in xs
    (t (exists x (cdr xs))))) ; (iii) otherwise, search recursively for x on rest of xs
```

מימוש שני :

מימוש exists באמצעות null יראה אם כן כך

```
(defun exists (x xs) ; determine whether atom x is in list xs
  (cond ; Three cases to consider:
    ((null xs) xs) ; (i) list of xs is exhausted
    ((eq x (car xs)) t) ; (ii) item x is first in xs
    (t (exists x (cdr xs))))) ; (iii) otherwise, recurse on rest of xs
```

ההבדל בין שני המימושים הוא בבדיקה האם הרשימה XS ריקה , במימוש הראשון בודקים את זה ישירות אם הרשימה ריקה אז מחזירים nil אבל במימוש שני עושים זאת באמצעות פונקציית עזר קוראים לה null שבפועל כל מה שהיא עושה בודקת האם XS ריקה , אם היא ריקה כלומר כפי שהגרנו NIL=() אז הפונקציה NULL תחזיר T .

```
(defun null(x) (eq x nil))
```

ואז התנאי הראשון מתקיים ונחזיר כפי שכתוב XS ,
אבל נשיים לב ש NIL=XS כי בפועל קיבלנו T מהפעלת
הפונקציה NULLלכן שני המימושים עושים אותה
סמנטיקה .

בנוסף הפונקציה eq היא פונקציה אטומית לעומת null
שהיא פונקצית ספריה.

2. הפונקציה member היא פונקצית ספריה סטנדרטית ברוב המימושים של ליספ. הפונקציה מקבלת אטום ורשימה, ומחזירה את זנב הרשימה הארוך ביותר שהאיבר הראשון שבו הוא האטום. אם האטום אינו מצוי ברשימה, הפונקציה מחזירה nil. ממש פונקציה זו במיני ליספ, והסבר מדוע ניתן להשתמש להחליף כל קריאה ל-exists בקריאה דומה ל-member עם אותם פרמטרים בדיוק.

```
(defun member (x xs)
(cond
((null xs) xs) ; xs=nil => nil
((eq x (car xs)) xs) ; return xs*********
(t ( member x (cdr xs)))))
```

ניתן להשתמש בפונקציה MEMBERבמקום EXIST
מכיוון ש Member מחזירה את אורך תת הרשימה
הארוך ביותר המתחיל מאטום נתון , כלומר אם לא חזר
Nil מהפעלת הפונקציה נוכל להסיק שקיים האטום

הזה כלומר , נוכל להשתמש בפונקציה Member במקום Exist כך שכל מה שעלינו לעשות לבדוק ערך ההחזרה . אם הוא NIL אז לא קיים האטום , אחרת האיבר הוא כן אטום.

שאלה 3)

3. ממש במיני ליספ את הפונקציה הרקורסיבית equal אשר משווה שני ביטויי S אם הם זהים, כלומר אם הטופולוגיה שלהם כעצים בינאריים מלאים זהה, והאטומים המצויים בעלים זהים גם כן. הפונקציה מחזירה t במקרה של זהות, ו-nil אחרת.

שני ביטוי S בנויים מאטומים , לכן עלינו לבדוק כל אטום בנפרד .

```
(defun equal (x y)
(cond
 ((null x) null b); if x=nil  return null (b) (if b = nil return t
else return nil)
((null b) b); if b=nil => nil (x != nil)

((eq (car a) (car b)) equal (cdr a) (cdr b))
((t) nil)
))
```

שאלה 4)

במימוש שלנו אנחנו נשתמש בפונקציה badd המקבלת שתי רשימות אשר מיצגות מספר בינארי בסדר הפוך ומחזירה רשימה המכילה את התוצאה.
נגדיר פונקצית עזר:

```
(defun digitMultiply(digit l2) (
Cond
((eq (car digit) 'Z) '(Z))
(t (bnormalize l2))))
```

```
(defun bmul(l1 l2) (
cond
((null (bnormalize l1)) (bnormalize l2))
((null (bnormalize l2)) (bnormalize l1))
((andalso (eq (car l1) 'Z) (null(cdr l1)) )'Z)
((andalso (eq (car l2) 'Z) (null(cdr l2)) )'Z)
((andalso (eq (car l1) 'O) (null(cdr l1)) )l2)
((andalso (eq (car l2) 'O) (null(cdr l2)) )l1)
( t (badd (digitMultiply  l2 l1) (cons 'Z (bmul
l1 (cdr l2)))))))
```

שאלה 5)

1. If S_expression is an atom, look it up in the a-list, and return the value parts that corresponds to it.
2. Else if the first arg in the S_expression list is an primitive function do the following:
   2.1. Divide the S_expression into 2 parts the name of the primitive function and the actual parameters of his functions and check:
      2.1.1. If the primitive function is cond, don't evaluate actual
      2.1.2. Else evaluate the actuals by checking if the name of the function matches a primitive function, else call primitive function error with an error message.
3. Find the lambda exp, and the actual parameters and do:
   3.1. Divide the lambda exp into 3 parts: the tag, the list of formal parameters and the body.
   3.2. Evaluate the body part by adding the names (the formals) of the parameters and their matching value (actual) to the a-list, in this manner:

3.2.1. If the lambda tag is the atom nlambda don’t evaluate the acutal parameters, and link them together in the a-list.

3.2.2. If the tag is lambda evaluate actual parameters recursively and then link.

3.2.3. Call error primitive function with the message unknown lambda and the lambda tag.