שאלה 1:

1) לא, לא נדרש כי אנו יכולים להשתמש בסוג יחיד כדי להפעיל את הפונקציה לדוגמה (x>4?f(x):g(x)) ,אבל חשוב לדעת שאנו לא יכולים להשתמש במשתנים שהוגדרו מחוץ לפונקציות כי זה יגרום לטות או שגיה , וזה עוזר הרבה בשפות פרוצדוראליות

2) **א)** כי כל תוכנית לניתוח חייבת להיות בעלת ידע מיוחד על כל סוג של צורה מיוחדת כי לכל צורה מיוחדת יש תחביר אידאוסינקרטי משלה (אם מגדירים אותם בצורה רגילה למשל define x 10 אז ל x חייב מוערך שזה יכול לגרום ל טעות) לדוגמה (if (> x 3) #t #f) שהדוגמה אומרת שאם ה x הוא גדול מ 3 אז יחזיר נכון אחרת יחזיר לא נכון

ב) בעולת or חייב שתוגדר כ special form כי בפעולת or אנו לא צרכים לבדוק את כל התנאים כי אם התקיים אחד התנאים אז לא צריך לבדוק את התנאים האחרים כי יש תנאי שהוא נכון , כלומר לדוגמה כמו פעולת if אם התקיים ה test אין צורך לבדוק ה else וכך פעולת ה or אם התקיים אחד התנאים אין צורך למשייך לבדיקות האחרות לכן פעולת or צריך שתוגד כ special form

3) **syntactic abbreviation** : היא פרושה לקטין את הכתיבה של הקוד בהגדרת תחברים חדשים בשפה– כלומר במקום לכתוב קוד כל כך ארוך ומסובך להבנה באמצעות המשתנים הפרימיטיביים הקיימים כבר בשפה ,אפשר להשתמש בפקודות שנקראות syntactic sugar שמקטינות את אורך הקוד וגורמות שהקוד יהיה יותר קרי וגם יותר מובן למשתמש לדוגמה: if condition , lambda, define, let

4) **א)** **(define x 1) (let ((x 5) (y (* x 3))) y)**: ה let מחזירה את y והערך של y ב let הוא 3 לכן תשובה של הביטוי הזה היא 3

הספר: מכיוון ש x הוגדר ב define אז מתקיים ש x הוא ערך גלובלי ולפי הגדרת let מתקיים שבהגדרת הערך y ה x שמוגדר בסוגריים של ה let עדיין הוא לא מוגדר בזיכרון (x שב define לא אותו x של let) לכן מתקיים שהערך של המשתנה y הוא 3 וה let מחזירה את y

ב) **(define x 1) (let* ((x 5) (y (* x 3))) y)**: ה let* מחזירה את y והערך של y ב let* הוא 15 לכן תשובה של הביטוי הזה היא 15

הספר: לפי הגדרת let* כל ביטוי שמוגדר ב bindings היא עוברת ביטוי אחרי ביטוי ומגדירה אותו בזיכרון לאחר מכן עוברת לביטוי השני וכך הלה, אז מתקיים שבהתחלה x שהוגדר let* יוגדר בזיכרון ואז כאשר מגיעים ל y מכיוון ש x ו y בתוך אותו סוגריים ו x מוגדר אז ה x שמגדיר את y זה יהיה ה x שבאותו סוגריים לא שהוגדר ב define (הן לא אותו x כי יש להן כתובות שונות) לכן מתקיים שהערך של y הוא y=3*5=15

ג) נכתוב את הכתובות של הקוד הבא :

```
(define x 2)

(define y 5)

(let

    ((x 1) (f (lambda (z) (+ x y z))))
```

```
            (f x))

(let*
        ((x 1) (f (lambda (z) (+ x y z))))
            (f x))
```

אז הכתובות:

```
(define x 2)
(define y 5)
(let
        ((x 1) (f (lambda (z) ([+:free] [x:free] [y:free] [z:0 0]))))
            ([f:0 1] [x: 0 0]))
```

```
(let*
        ((x 1) (f (lambda (z) ([+:free] [x:1 0] [y:free] [z:0 0]))))
            ([f:0 1] [x:0 0]))
```

ד) נגדיר את *let באמצעות let בביטוי הזה (let* ((x 1) (f (lambda (z) (+ x y z)))) (f x)) אז

(let* ((x 1) (f (lambda (z) (+ x y z)))) (f x)) = (let (x 1) (let (f (lambda(z) (+ x y z))) (f x)))

ה) נגדיר את *let באמצעות ביטויים (לא let) בביטוי הזה (let* ((x 1) (f (lambda (z) (+ x y z)))) (f x)) אז

(let* ((x 1) (f (lambda (z) (+ x y z)))) (f x))= (define f (lambda(z) (+ 1 y z)))

שאלה 2 ברק 2:

הפונקציות:

1) make-ok:
    a) Signature: make-ok(val)

b) Type: [T->list(T)]
c) Purpose: encapsulates the val as an ok structure of type result
d) Pre-conditions: val is an atomic
e) Tests: make-ok(4)->[tag: “ok” , value: 4]

2) Make-error:
   a) Signature: make-error(msg)
   b) Type: [string->list(T)]
   c) Purpose : gets an error string and encapsulates it as an ‘error’ structure of type result
   d) Pre-conditions: msg is a string value
   e) Tests: make-error(“not right”)-> [tag: Failure, value: ”not right”]
3) ok?
   a) Signature : ok?(res)
   b) Type: [any->boolean]
   c) Purpose: to know if the res is an Ok list or not
   d) Pre-conditions: none
   e) Test : ok? (make-ok(4))-> #t
4) error?
   a) Signature: error? (res)
   b) Type:[any->boolean]
   c) Purpose: to know if the res is an error list or not
   f) Pre-conditions: none
   d) Test: error? (make-error(“not a number”))-> #t
5) result?
   a) Signature: result? (res)
   b) Type: [any->Boolean]
   c) Purpose: to know if the res is a list or it not
   d) Pre-conditions: none
   e) Test: result? 5->#f

6) result->val
   a) signature: result->val(res)
   b) Type:[any->T]
   c) Purpose: return the value(the value is a T type) of the result or to return a make-error massage
   d) Pre-conditions: none
   e) Test: result->val (make-ok(4))-> 4

7) Bind
   a) Signatures: bind(f)
   b) Type: [T->[T->T]]
   c) Purpose: if the res is a list so run the f on the cdr of the res else return an error msg
   d) Pre-conditions: f is a function

   e) Test: (bind (lambda (x) (make-ok (* x x)))-> make-Ok(x*x)
8) make-dict
   a) Signatures: make-dict()
   b) Type: [any->list]
   c) Purpose: returns a new empty dictionary(list)
   d) Pre-conditions: none
   e) Test: make-dict()-> (list)
9) dict?
   a) Signatures: dict?(e)
   b) Type:[any->Boolean]
   c) Purpose: to know if the e is an dictionary or not
   d) Pre-conditions: none
   e) Test:dict('())->#t

10) Get
   a) Signatures: get(dict k)
   b) Type:[list*number-> T]
   c) Purpose: returns the value in dict assigned to the given key as an ok result. In case the given key is not defined in dict, an error result should be returned.
   d) Pre-conditions: dict is an dictionary (list) and the k is an number
   e) Test: get ((3 4) 3)-> 4
11) Put
   a) Signatures: put(dict k v)
   b) Type: [list(T)*number*T->list(T)
   c) Purpose: returns a result of a dictionary with the addition of the given key-value. In case the given key already exists in the given dict, the returned dict should contain the new value for this key.
   d) Pre-conditions: dict is an dictionary(list) ,k is an number and v as an genere type
   e) Test: put((2 3) 2 5)->(2 5)
12) map-dict
   a) Signatures: map-dict(dict f)
   b) Type: [list(T)*function->list(T)]
   c) Purpose: it applies the function of the values in the dictionary, and returns a result of a new dictionary with the resulting values.
   d) Pre-conditions: dict is an dictionary and f is an function
   e) Test: map-dict((1 3.4) (lambda(x)->(x*x)))-> (1 9.16)
13) filter-dict
   a) Signatures: filter-dict(dict prad)
   b) Type:[list(T)*filter-function->list(T)]
   c) Purpose: returns a result of a new dictionary that contains only the key-values that satisfy the filter-function
   d) Pre-conditions: dict is an dictionary and Prad is an filter-function

e) Test:filter-dict((1 2.3) (lambda(x)->(x%2===1)))->(1.3)