

## תרגיל 1

הגשה: יום שישי, 09/03/2007, שעת סגירת רוס

\*\*\* את התרגיל יש להגיש אלקטרונית

בשלב זה אין צורך לבדוק חוקיות הקלט  
הגדירו פונ' עזר במקומות בהם אתם רואים לנכון  
הוסיפו תיעוד במקומות בהם אתם רואים לנכון

פורים שמח

\*\*\* מבנה קובץ הבדיקות TESTS אותו עליכם להגיש:

```
(= (<func_name> <argumentes>) <expected value>)
(= (my_add 2 3 4) 9)
```

לדוגמא:

\*\*\* קבצים שיש להגיש

• TESTS

• ex1.scm

• README - אופציונלי

מקובצים ל ex1.tar ע"י { tar -cvf ex1.tar ex1.scm TESTS README }

1. סדרת פיבונצ'י (1,1,2,3,5,8,13) מוגדרת ע"י

$$a_1=1 \quad a_2=1 \quad a_n=a_{n-1}+a_{n-2}$$

• ממשו את הפונ' (fibon &lt;index&gt;)

• ממשו את הפונ' (fibonSum <n>) המחזירה את סכום n האיברים הראשונים בסדרת פיבונצ'י

2. • ממשו את הפונ' (my\_power <x(real)> <n(integer - might be negative)>) המחזירה את  $x^n$

• ממשו את הפונ' (n\_square <x(real)> <n(integer)> <tolerance(real)>) המחזירה קירוב ל  $\sqrt[n]{x}$  (באופן דומה ל my\_sqrt שהוצג בכיתה)

$$a_k = \frac{(n-1)a_{k-1} + \frac{x}{(a_{k-1})^{n-1}}}{n} : \sqrt[n]{x}$$

נוסחת הקירוב האיטרטיבית ל  $\sqrt[n]{x}$  :  
את הניחוש הראשוני העבירו כמספר real (עם נקודה עשרונית)  
אחרת התוצאה תהיה נכונה אך לא קריאה.

- ממשו את הפונקציות `(third_square <x(real)>)`, `(second_square <x(real)>)`, `(forth_square <x(real)>)`, המחזירות קירוב ל  $\sqrt{x}$ ,  $\sqrt[3]{x}$ ,  $\sqrt[4]{x}$  עד כדי טעות של 0.001

3. לימדו את הפונקציה `remainder` לחישוב שארית חלוקה

- ממשו את הפונקציה הבוליאנית `(prime <n>)` לבדיקת ראשוניות
- ממשו את הפונקציה הבוליאנית `(co_prime <n1> <n2>)` לבדיקת זרות (אי קיום מחלק משותף מלבד 1)
- 
- ממשו את הפונקציה `(phi <n>)` המחזירה את מספר המספרים הקטנים מ- $n$  וזרים לו (פונקציה אוילר)

4. ממשו את הפונקציה `(mul_from_to <n1> <n2>)` המחזירה את מכפלת המספרים בין  $n_1$  ל  $n_2$   
 $n! = (\text{mul\_from\_to } 1 \ n)$

5. ממשו את הפונקציה `(det.2.2 <a1> <a2> <a3> <a4>)` את הדטרמיננטה  
 $a_1 * a_4 - a_2 * a_3 =$

6. מספרי קטלן Catalan מוגדרים ע"י  $\{ c_0=1 \ c_n=\sum_{k=1}^n c_{k-1}c_{n-k} \}$  ומונים את מספר הביטויים החוקיים ב- $n$  סוגריים ימניים וב- $n$  סוגריים שמאליים.

למידע נוסף על מספרי קטלן אתם יכולים להסתכל ב

[http://en.wikipedia.org/wiki/Catalan\\_number](http://en.wikipedia.org/wiki/Catalan_number)

ממשו את הפונקציה `(catalan <index>)` המחזירה את  $c_n$

## תרגיל 1

הגשה: יום חמישי, 15/03/2007, שעת סגירת רוס

\*\*\* את התרגיל יש להגיש אלקטרונית בשלב זה אין צורך לבדוק חוקיות הקלט הגדירו פונ' עזר במקומות בהם אתם רואים לנכון הוסיפו תיעוד במקומות בהם אתם רואים לנכון הגדירו פונ' רק בעזרת lambda כפי שהגדרנו בכיתה אתם רשאים (ומומלץ) להשתמש בפונ' שתגדירו בסעיף מסוים בסעיפים אחרים. מובן שבמקרה כזה מומלץ לוודא שהפונ' הבסיסיות לא כוללות טעויות.

\*\*\* מבנה קובץ הבדיקות TESTS אותו עליכם להגיש:

```
(= (<func_name> <argumentes> <expected value>)
   (= (my_add 2 3 4) 9))
```

\*\*\* קבצים שיש להגיש

- ex2.scm

- TESTS

- README - אופציונלי

מקובצים ל ex2.tar ע"י { tar -cvf ex2.tar ex2.scm TESTS README }

1. ממשו את הפונ'

- (sum\_function <function> <function>)
- (composed\_function <function> <function>)

המקבלות שתי פונ' אונריות מספריות f,g ומחזירות את הפונ' האונריות המספריות  $f+g, f \circ g = f(g(x))$

2. נמדל את המספרים המרוכבים ע"י הפונ'

```
(define complex (lambda (a b) (cons a b)))
a+bi → (complex a b)
```

ממשו את הפונ'

- `(complex_add <complex number> <complex number>)`
- `(complex_mul <complex number> <complex number>)`

המחזירות מספר מרוכב שהינו תוצאת החיבור והמכפלה בהתאמה.

3. ממשו את הפונ' `(series <a(function)> <n(natural)>)` המחזירה את הרשימה  
 $a(0), a(1), a(2), a(3), \dots, a(n)$

4. ממשו את הפונ' `(add_8_to_all <list>)` המוסיפה 8 לכל אחד מאברי רשימת הקלט ומחזירה את התוצאה.

5. ממשו את הפונ' הבוליאנית `(forall <list> <condition>)` המחזירה #t אם ם התנאי מתקיים לכל אחד מאברי הרשימה

6. ממשו את הפונ'

`(list_min_max <list>)` המחזירה את האיבר המינימלי והאיבר המקסימלי ברשימה כ pair

- `(list_product <list>)` המחזירה את מכפלת האיברים ברשימה
- `(list_average <list>)` המחזירה את ממוצע האיברים ברשימה

7. ממשו את הפונ' `(even_indexes <list>)` המחזירה את סדרת האיברים שבמ-קומות הזוגיים

שימו לב שפונ' זו שונה מהפונ' `select_even` שראינו בכיתה.

8. ממשו את הפונ'

• `(filter <list> <boolean function>)` המחזירה את אברי הרשימה עליהם הפונ' מחזירה #t

• `(multi_filter <list> <boolean function list>)` המחזירה את אברי הרשימה עליהם כל הפונ' מחזירות #t

9. ממשו את הפונ' `(item_no <index> <list>)` אשר בהינתן סדרה  $a_0, a_1, a_2, a_3, \dots, a_k$  ואינדקס  $i$  ( $-(k+1) \leq i \leq k$ ) מחזירה את  $a_i$  וכאשר  $i$  שלילי מחזירה את האיבר ה- $i$  מהסוף.

10. ממשו את הפונ' `(change_item_no <value> <index> <list>)` אשר בהינתן סדרה  $a_0, a_1, a_2, a_3, \dots, a_k$ , אינדקס  $i$  ( $-(k+1) \leq i \leq k$ ) וערך  $v$  מחליפה את  $a_i$  ב- $v$  וכאשר  $i$  שלילי מחליפה את האיבר ה- $i$  מהסוף.

11. ממשו את הפונ' `(monotone <list>)` אשר בהינתן סדרה  $a_0, a_1, a_2, a_3, \dots$  מ-זירה את הסידרה  $b_0, b_1, b_2, b_3, \dots$  המקיימת

$$b_1 = a_1 \bullet$$

$$\forall i : b_i = a_i \text{ ונמצא אחריו } b_{i-1} \bullet$$

לדוגמא:

$$\bullet (monotone '(1 2 4 3 7)) \rightarrow '(1 2 4 7)$$

(monotone '(1 9 2 3 4)) → '(1 9) •

(monotone '(1 9 9 9 2 3 4)) → '(1 9) •

12. ממשו את הפונקציה `(interleave <list> <list>)` אשר בהינתן סדרות  $a_0, a_1, a_2, a_3, \dots$  ו- $b_0, b_1, b_2, b_3, \dots$  מחזירה את הסידרה  $a_0, b_0, a_1, b_1, a_2, b_2, \dots$  , לדוגמא:

(interleave '(1 2 3 4) '(a b)) → '(1 a 2 b 3 4) •

(interleave '(1 2 3) '(a b c d e)) → '(1 a 2 b 3 c d e) •

(interleave '() '(a b c d e)) → '(a b c d e) •

## תרגיל 3

הגשה: יום שישי, 30/03/2007, 12:00

\*\*\* את התרגיל יש להגיש אלקטרונית בשלב זה אין צורך לבדוק חוקיות הקלט הגדירו פונ' עזר במקומות בהם אתם רואים לנכון ( השתמשו בlet ) הוסיפו תיעוד במקומות בהם אתם רואים לנכון הגדירו פונ' רק בעזרת lambda כפי שהגדרנו בכיתה אתם רשאים (ומומלץ) להשתמש בפונ' שתגדירו בסעיף מסוים בסעיפים אחרים. מובן שבמקרה כזה מומלץ לוודא שהפונ' הבסיסיות לא כוללות טעויות.

\*\*\* מבנה קובץ הבדיקות TESTS אותו עליכם להגיש:

```
(= (<func_name> <argumentes> <expected value>)
   (= (my_add 2 3 4) 9))
```

\*\*\* קבצים שיש להגיש

- ex3.scm
- TESTS
- README - אופציונלי

מקובצים ל ex3.tar ע"י { tar -cvf ex3.tar ex3.scm TESTS README }

1. המירו את הפו'

```
(n_root <x(real)> <n(integer)> <tolerance(real)>)
(catalan <index>)
```

שכתבתם בתרגיל בראשון כך שתשתמשנה ב let ולא בפונ' עזר חיצונית.

שימו לב ששם הפונ' תוקן מ n\_square ל n\_root

2. סידרה רקורסיבית (פרמטרית)

```
 $a_0(x), a_1(x), a_2(x), a_3(x), \dots$ 
```

מוגדרת ע"י הפונ'

```
(first-elem x)
```

(next-elem n x prev)

לדוגמא  $x^n!$  מוגדר ע"י

$\text{first-elem}(x) = 1$

$\text{next-elem}(n,x,\text{prev}) = x^n \cdot \text{prev}$

כאשר  $n$  הוא האינדקס של האיבר הנוכחי.

ממשו את הפונ'

• (k-elem x first-elem-func next-elem-func)

המחזירה פונ' שבהנתן  $k$  מחזירה את  $a_k$

• (compute-k-sum x first-elem-func next-elem-func)

המחזירה פונ' שבהנתן  $k$  מחזירה את  $\sum_{i=0}^k a_i$

• (compute-inf-sum x first-elem-func next-elem-func)

המחזירה פונ' שבהנתן  $\text{tolerance}$  מחזירה קירוב ל  $\sum_{i=0}^{\infty} a_i$  ( הניחו שהסדרה מקימת  $|a_n| \rightarrow 0$  )

תנאי העצירה של הפונ'  $|a_k| \leq \text{tolerance}$  ובמקרה זה מחזירה  $\sum_{i=0}^k a_i$

• (my-sin x tolerance)

המקרבת את  $\sin(x)$  לפי הנוסחא

$$\sin(x) = \sum_{n=0}^{\infty} (-1)^n \cdot \frac{x^{2n+1}}{(2n+1)!}$$

והשימוש ב  $\text{tolerance}$  כפי שהוגדר בסעיף הקודם.

עליכם להשתמש ב  $\text{compute-inf-sum}$  מהסעיף הקודם.

3. ממשו את הפונ'

• (list-prefix list1 list2)

המחזירה האם  $\text{list1}$  היא רישא של  $\text{list2}$

• (sub-list list1 list2)

המחזירה האם  $\text{list1}$  היא תת-סידרה (ברציפות) של  $\text{list2}$

שתי הפונ' מוגדרות עבור סדרות מספריות בלבד.

לדוגמא:

(list-prefix '(1 2 3 4 5 6 7 8) '(1 2 3 4)) → #f

(sub-list '(1 2 3 4 5 6 7 8) '(1 2 3 4)) → #f

(list-prefix '(1 2 3 4 5 6 7 8) '(2 3 4 5)) → #f

(sub-list '(1 2 3 4 5 6 7 8) '(2 3 4)) → #f

(list-prefix '(1 2 3 4) (1 2 3 4 5 6 7 8)) → #t

(sub-list '(1 2 3 4) (1 2 3 4 5 6 7 8)) → #t

(list-prefix '(2 3 4 5) (1 2 3 4 5 6 7 8)) → #f

(sub-list '(3 4 5 6 7) (1 2 3 4 5 6 7 8)) → #t

(sub-list '(1 2 5) '(1 2 3 4 5 6 7 8)) → #f

4. ממשו את הפונקציה `(satisfy predicate a1 a2 a3 ...)` המחזירה את מספר האיברים בקלט המקיימות את הפרדיקט לדוגמא:

בהנחה כי קיים פרדיקט `pos` המחזיר האם הקלט חיובי ממש.

(satisfy pos 1 2 3 4) → 4

(satisfy pos 1 -2 -3 4) → 2

(satisfy pos) → 0

(satisfy) → לא יבדק

5. ממשו את הפונקציה `(my-reverse list)` המקבלת רשימה ומחזירה את הרשימה בסדר הפוך ומשתמשת ברקורסיה זנב

6. ממשו את הפונקציה `(my-unique list)` המקבלת רשימה מספרית ממוינת ומחזירה רשימה ממוינת בה כל איבר מופיע בדיוק פעם אחת. (שימו לב שאינכם צריכים להניח דבר על מפתח המיון)

7. ממשו את הפונקציה `(maximizer list1 list2)` המקבלת רשימת תווים ורשימת strings ומחזירה את ה string הכולל את המספר המקסימלי של תווים מהרשימה (עם חזרות). (case sensitive)

(במקרה ויש מספר ממקסימים החזירו את הראשון) לדוגמא:

(maximizer '(#\a #\b #\c) ("abc" "ccc" "ab cd" "aaaa")) → "aaaa"

(maximizer '(#\a #\b #\c) ("abc" "ccc" "ab cd" "bb b" "aa aa" "aaaa")) → "aa aa"

8. ממשו את הפונקציה `(substring string1 string2 <optional bool>)` המחזירה את ה string הראשון הינו תת-string של ה string השני.

הפרמטר האופציונלי-האם הבדיקה הינה (case sensitive) (ברירת המחדל - false)

לדוגמא:

(substring "aaa" "baaab" #t) → #t

(substring "aaa" "baaab" #f) → #t

(substring "aAa" "baaab" #t) → #f

(substring "aAa" "baaab" #f) → #t

(substring "aAa" "baaab") → #t

9. ממשו את הפונקציה `(concatenate string1 string2 ...)` המקבל מספר strings ומשרשר אותם.  
 לדוגמא:

`(concatenate "aaa" "baaab") → "aaabaaab"`  
`(concatenate "aaa" "baaab" "222") → "aaabaaab222"`  
`(concatenate) → "Empty string" *`  
 הפלט הוא ה-string "Empty string"

10. מתוך סכום השעור types.pdf :

`sum-nested-list(ls)` is

case ls of  
 () ⇒ 0  
 cons(x,ls') ⇒  
 case x of  
 number ⇒ x + sum-nested-list(ls')  
 list ⇒ sum-nested-list(x) + sum-nested-list(ls')

ממשו את הפונקציה `(sum-nested-list list)` המממשת את הפסאודו-קוד הנ"ל.  
 לדוגמא:

`(sum-nested-list '(1 (2 3 a) (a b c) (a (b 6)))) → 12`

## תרגיל 4

הגשה: יום שישי, 20/04/2007, 23:59

\*\*\* את התרגיל יש להגיש אלקטרונית  
 אין צורך לבדוק חוקיות הקלט  
 הגדירו פונ' עזר במקומות בהם אתם רואים לנכון ( השתמשו ב let )  
 הוסיפו תיעוד במקומות בהם אתם רואים לנכון  
 הגדירו פונ' רק בעזרת lambda כפי שהגדרנו בכיתה  
 אתם רשאים (ומומלץ) להשתמש בפונ' שתגדירו בסעיף מסוים בסעיפים אחרים.  
 מובן שבמקרה כזה מומלץ לוודא שהפונ' הבסיסיות לא כוללות טעויות.

\*\*\* מבנה קובץ הבדיקות TESTS אותו עליכם להגיש:  
 (= (<func\_name> <argumentes> <expected value>)  
 לדוגמא: (= (my\_add 2 3 4) 9)

\*\*\* קבצים שיש להגיש

- ex4.scm
- TESTS
- אופציונלי - README

מקובצים ל ex4.tar ע"י { tar -cvf ex4.tar ex4.scm TESTS README }

העתיקו את המימוש לתוך ex4.scm של הפונ'

- stream-nil (value)
- stream-null?
- stream-cons
- stream-car
- stream-cdr

1. • ממשו את הפונ' (tri-uncurry f) אשר בהנתן פונ' המקבלת שלושה ארגומ-  
 נטים 'אחד אחד' מחזירה פונ' שקולה המקבלת שלושה ארגומנטים.

- ממשו את הפונ' `(uncurry f)` אשר בהנתן פונ' המקבלת מספר כלשהו של ארגומנטים 'אחד אחד' (לפחות ארגומנט אחד) מחזירה פונ' שקולה המקבלת את הארגומנטים 'בבת אחת' הערה: הפונ' יכולות להשתמש זו בזו. לדוגמא:

```
(define muladd
  (lambda (x)
    (lambda (y)
      (lambda (z)
        (+ (* x y) z))))))
```

```
((muladd 3) 4) 5) → 17
((tri-uncurry muladd) 3 4 5) → 17
((uncurry muladd) 3 4 5) → 17
(((uncurry muladd) 3 4) 5) → 17
((((uncurry muladd) 3) 4) 5) → 17
```

2. ממשו את הפונ'

- `(elem k stream)`
- `(tail k stream)`

המחזירות את האיבר ה-  $k$  -י והסיפא החל מהאיבר ה-  $k$  -י (כולל) בהתאמה. (האיבר הראשון יסומן באינדקס 0) לדוגמא:

```
(elem 3 naturals) → 3
(tail 4 naturals) → <4 , 5 , 6 , 7 , 8 , ...>
```

3. ממשו את הפונ'

- `(list->stream list)` אשר בהנתן רשימה מחזירה stream שקול.
  - `(stream->list stream)` אשר בהנתן stream סופי מחזירה רשימה שקולה.
4. ממשו את הפונ' `(make-power-stream n)` אשר בהנתן מספר מחזיר את סדרת החזקות שלו. לדוגמא:

```
(make-power-stream 3) → <3 , 9 , 27 , 81 , ...>
(make-power-stream 2) → <2 , 4 , 8 , 16 , ...>
(make-power-stream 1) → <1 , 1 , 1 , 1 , ...>
(make-power-stream -1.5) → <-1.5 , 2.25 , -3.375 , 5.0625 , ...>
```

5. ממשו את הפונ'

(stream-add stream stream) •

(stream-convolution stream stream) •

כאשר בהנתן  $a_0, a_1, a_2, a_3, \dots$  ו  $b_0, b_1, b_2, b_3, \dots$

הסכום מוגדר איבר איבר

$$a_0 + b_0, a_1 + b_1, a_2 + b_2, a_3 + b_3, \dots$$

והקונבולוציה מוגדרת

$$c_0, c_1, c_2, c_3, \dots$$

$$c_k = \sum_{i=0}^k a_i * b_{k-i}$$

פשר מקובל: טורי חזקות  $\langle a_0, a_1, a_2, a_3, \dots \rangle \Rightarrow \sum_{i=0}^{\infty} a_i x^i$

ותחת פשר זה קונבולוציה הינה כפל.

שימו לב: כל אחד מה streams יכול גם להיות סופי ובמקרה זה נתיחס אליו כבעל סיפא שכולה אפסים.

התוצאה צריכה תמיד להיות סידרה אינסופית.

לדוגמא:

$$(\text{stream-add } (\text{tail } 3 \text{ naturals}) (\text{tail } 5 \text{ naturals})) \rightarrow \langle 8, 10, 12, 14, \dots \rangle$$

$$(\text{stream-convolution } \langle 1, 1, 1, \dots \rangle \langle 1, -1 \rangle) \rightarrow \langle 1, 0, 0, 0, \dots \rangle$$

$$(\text{stream-convolution } \langle 1, 1, 1, \dots \rangle \langle 1, -1, 0, 0 \rangle) \rightarrow \langle 1, 0, 0, 0, \dots \rangle$$

## תרגיל 5

הגשה: יום רביעי, 06/06/2007, 23:59

\*\*\* את התרגיל יש להגיש אלקטרונית  
 ודאו ששמות הפונקציות המוגשות זהות לשמות המופיעים בתרגיל  
 אין צורך לבדוק חוקיות הקלט  
 הגדירו פונ' עזר במקומות בהם אתם רואים לנכון ( השתמשו ב let )  
 הוסיפו תיעוד במקומות בהם אתם רואים לנכון  
 הגדירו פונ' רק בעזרת lambda כפי שהגדרנו בכיתה  
 אתם רשאים (ומומלץ) להשתמש בפונ' שתגדירו בסעיף מסוים בסעיפים אחרים.  
 מובן שבמקרה כזה מומלץ לוודא שהפונ' הבסיסיות לא כוללות טעויות.  
 \*\*\* עליכם לכתוב פונקציות יעילות ככל האפשר ובפרט להשתמש ברקורסית זנב  
 במקומות שאפשר ויעיל.

\*\*\* מבנה קובץ הבדיקות TESTS אותו עליכם להגיש:

```
(= (<func_name> <argumentes> <expected value>)  

  (= (my_add 2 3 4) 9)
```

\*\*\* קבצים שיש להגיש

- ex5.scm

- TESTS

- README - אופציונלי

מקובצים ל ex5.tar ע"י { tar -cvf ex5.tar ex5.scm TESTS README }

העתיקו את המימוש לתוך ex5.scm של הפו'

- stream-nil (value)
- stream-null?
- stream-cons
- stream-car
- stream-cdr

קחו לתשומת לבכם שלא אהיה זמין לדואר במהלך השבוע שבין 25/05 ל 02/06

1. ממשו את הפר' (sum <int>)

ממנה נגזרות פר' אונריות  $f_0, f_1, f_2, f_3, \dots$  באופן הבא

sum בהינתן קלט שיסומן  $a_0$  מחזירה את  $f_0$

$f_i$  בהינתן קלט מספרי שיסומן  $a_{i+1}$  מחזירה את  $f_{i+1}$

בנוסף  $f_i$  על קלטים מסוג symbol:

על הקלט 'SUM' מחזיר את

$$\sum_{k=0}^i a_k$$

על הקלט 'LAST' מחזיר את  $a_i$

לדוגמא

(define a (sum 3))

(define b (a 3))

(define c (b 4))

(define d (c 5))

(define e (d 6))

(define f (e 7))

(define g (f 8))

(define f2 (d 7))

(define g2 (d 8))

(a 'SUM) → 3

(e 'SUM) → 21

(g 'SUM) → 36

(a 'LAST) → 3

(e 'LAST) → 6

(g 'LAST) → 8

(g2 'SUM) → 23

(g2 'LAST) → 8

2. ממשו את הפר' (long-func1 f n)

המחשבת בהנתן  $n > 0$  ופר' המקבלת שני ארגומנטים את

$(f\ 0\ (f\ 1\ (\dots\ (f\ <n-1\ >\ n))))$

ממשו את הפר' (long-func2 f n)

המחשבת בהנתן  $n > 0$  ופר' המקבלת שני ארגומנטים את

$(f\ n\ (f\ <n-1\ >\ (\dots\ (f\ 1\ 0))))$

התנהגות הפונקציות על  $n = 0$  אינה מוגדרת אך הפונקציות צריכות לצאת באופן מסודר ולא לעוף.

לדוגמא

$(\text{long-func1 } (\text{lambda } (x\ y) (+\ x\ y))\ 6) \rightarrow 21$   
 $(\text{long-func2 } (\text{lambda } (x\ y) (+\ x\ y))\ 6) \rightarrow 21$   
 $(\text{long-func1 } (\text{lambda } (x\ y) (*\ x\ y))\ 6) \rightarrow 0$   
 $(\text{long-func2 } (\text{lambda } (x\ y) (*\ x\ y))\ 6) \rightarrow 0$   
 $(\text{long-func1 } (\text{lambda } (x\ y) (\text{if } (\text{or } (= x\ 0) (= y\ 0))\ 1\ (*\ x\ y)))\ 6) \rightarrow 720$   
 $(\text{long-func2 } (\text{lambda } (x\ y) (\text{if } (\text{or } (= x\ 0) (= y\ 0))\ 1\ (*\ x\ y)))\ 6) \rightarrow 720$   
 $(\text{long-func1 } (\text{lambda } (x\ y) (-\ x\ y))\ 6) \rightarrow 3$   
 $(\text{long-func2 } (\text{lambda } (x\ y) (-\ x\ y))\ 6) \rightarrow 3$   
 $(\text{long-func1 } (\text{lambda } (x\ y) (-\ x\ y))\ 5) \rightarrow -3$   
 $(\text{long-func2 } (\text{lambda } (x\ y) (-\ x\ y))\ 5) \rightarrow 3$

3. ממשו את הפר'  $(\text{split-at } \langle i:\text{int} \rangle \langle x:\text{stream of numbers/list of numbers} \rangle)$

המחזירה pair  $(i \geq 0)$  בו

האיבר הראשון הינו list באורך  $i$  והינו הרישא של  $x$

והאיבר השני הינו מאותו הטיפוס כמו  $x$  והינו הסיפא של  $x$

שימו לב ש  $i$  יכול להיות גדול מאורכו של  $x$  ו  $x$  יכול להיות ריק.

לדוגמא

הערה: בסימון  $\langle \text{list1} \rangle . \langle \text{list2} \rangle$  כוונתי  $(\text{car } \dots) \rightarrow \langle \text{list1} \rangle$   
 $(\text{cdr } \dots) \rightarrow \langle \text{list2} \rangle$

$(\text{split-at } 2 '(0\ 1\ 2\ 3\ 4\ 5)) \rightarrow ((0\ 1) . (2\ 3\ 4\ 5))$   
 $(\text{car } (\text{split-at } 2 '(0\ 1\ 2\ 3\ 4\ 5))) \rightarrow (0\ 1)$   
 $(\text{cdr } (\text{split-at } 2 '(0\ 1\ 2\ 3\ 4\ 5))) \rightarrow (2\ 3\ 4\ 5)$   
 $(\text{split-at } 0 '(0\ 1\ 2\ 3\ 4\ 5)) \rightarrow ((). (0\ 1\ 2\ 3\ 4\ 5))$   
 $(\text{split-at } 9 '(0\ 1\ 2\ 3\ 4\ 5)) \rightarrow ((0\ 1\ 2\ 3\ 4\ 5) . ())$   
 $(\text{split-at } 2 \text{ naturals}) \rightarrow ((0\ 1) . \langle 2345\dots \rangle)$   
 $(\text{split-at } 0 \langle 1\ 1 \rangle) \rightarrow (\langle \rangle . \langle 1\ 1 \rangle)$   
 $(\text{split-at } 9 \langle 1\ 1 \rangle) \rightarrow (\langle 1\ 1 \rangle . \langle \rangle)$

4. ממשו את הפר'  $(\text{fib-generator})$

המחזירה stream הכוללת את מספרי פיבונצ'י  $1, 1, 2, 3, 5, 8, \dots$

רמז: השתמשו ב  $\text{stream-add}$  שמשתם בתרגיל הקודם.

5. ממשו את הפונ'  $(\text{partial-sums } \langle \text{stream:infinite/finite/empty} \rangle)$

המקבלת stream

ומחזירה את ה stream של הסכומים החלקיים.

לדוגמא

$$(\text{partial-sums } \langle 1,3,2,1,\dots \rangle) \rightarrow \langle 1,4,6,7,\dots \rangle$$

$$(\text{partial-sums } \langle 1,1,1 \rangle) \rightarrow \langle 1,2,3 \rangle$$

$$(\text{partial-sums } \langle \rangle) \rightarrow \langle \rangle$$

6. ממשו את הפונ'  $(\text{is-length } \langle \text{stream/list} \rangle r)$

המחזירה האם אורך הארגומנט הראשון הוא  $r$  ( $r \geq 0$ )

7. ממשו את הפונ'  $(\text{stream-merge } \langle \text{list of infinite streams} \rangle)$

המקבלת רשימה לא ריקה של streams אינסופיים מספריים ממוינים בהם כל מספר מופיע לכל היותר פעם אחת.

הפונ' מחזירה את stream האיחוד שאף הוא לא כולל עותקים כפולים של אותו המספר.

לדוגמא

$$(\text{stream-merge } \langle 2,4,6,8,\dots \rangle \langle 1,3,5,7,\dots \rangle) \rightarrow \langle 1,2,3,4,5,\dots \rangle$$

$$(\text{stream-merge } \langle 2,4,6,8,\dots \rangle \langle 3,6,9,12,\dots \rangle) \rightarrow \langle 2,3,4,6,8,9,10,12,\dots \rangle$$

$$\forall \text{stream } s \ (\text{stream-merge } s) \rightarrow s$$

$$\forall \text{stream } s \ (\text{stream-merge } s \ s) \rightarrow s$$

$$\forall \text{stream } s \ (\text{stream-merge } s \ s \ s) \rightarrow s$$

## תרגיל 6

הגשה: יום חמישי, 05/07/2007, 23:59

\*\*\* את התרגיל יש להגיש אלקטרונית  
 ודאו ששמות הפונקציות המוגשות זהות לשמות המופיעים בתרגיל  
 אין צורך לבדוק חוקיות הקלט  
 הגדירו פונ' עזר במקומות בהם אתם רואים לנכון ( השתמשו ב let )  
 הוסיפו תיעוד במקומות בהם אתם רואים לנכון  
 הגדירו פונ' רק בעזרת lambda כפי שהגדרנו בכיתה  
 אתם רשאים (ומומלץ) להשתמש בפונ' שתגדירו בסעיף מסוים בסעיפים אחרים.  
 מובן שבמקרה כזה מומלץ לוודא שהפונ' הבסיסיות לא כוללות טעויות.  
 \*\*\* עליכם לכתוב פונקציות יעילות ככל האפשר ובפרט להשתמש ברקורסית זנב  
 במקומות שאפשר ויעיל.

\*\*\* מבנה קובץ הבדיקות TESTS אותו עליכם להגיש:

```
(= (<func_name> <argumentes> <expected value>)
   (= (my_add 2 3 4) 9))
```

\*\*\* קבצים שיש להגיש

- ex6.scm
- TESTS

• README - אופציונלי

מקובצים ל ex6.tar ע"י { tar -cvf ex6.tar ex6.scm TESTS README }

העתיקו את המימוש של הפונ' לתוך ex6.scm

- stream-nil (value)
- stream-null?
- stream-cons
- stream-car
- stream-cdr

1. ממשו את הפונקציה `(expr2func expression)` המקבל ביטוי במחרוזת ומחזירה פונקציה המחשבת אותו (סדר הארגומנטים כסדר ההופעה במחרוזת) סינטקס המחרוזת:

כל מספר הוא מחרוזת חוקית  
 כל אות היא מחרוזת חוקית ומיצגת משתנה  
 בדיוק רווח יחיד בין תתי-הביטויים לבין האופרטור

$$(expr1 + expr2)$$

$$(expr1 - expr2)$$

$$(expr1 * expr2)$$

לדוגמא

$$((expr2func "5")) \rightarrow 5$$

$$((expr2func "(5 + 13)")) \rightarrow 18$$

$$((expr2func "(x + 13)")) 1 \rightarrow 14$$

$$((expr2func "(x - y)")) 1 2 \rightarrow -1$$

$$((expr2func "(x - x)")) 1 \rightarrow 0$$

$$((expr2func "((z * z) - (y * y))")) 5 4 \rightarrow 9$$

2. ממשו את הפונקציה `(add-monitor func)` המקבלת פונקציה  $f$  (מספרי אשר מקבלת ארגומנט אחד) ומחזירה פונקציה שנסמנה  $f^2$  כהיקראה עם ארגומנט מספרי מחזירה כמו  $f$   $f^2$  בהיקראה עם הארגומנט `'TIMES` מחזירה את מספר הפעמים שנקראה עד כה עם ארגומנט מספרי  $f^2$  בהיקראה עם הארגומנט `'LAST` מחזירה את התוצאה האחרונה (על ארגומנט מספרי) או 0 אם לא נקראה עדיין על ארגומנט מספרי.

לדוגמא

```
(define a (add-monitor add3))
(a 'TIMES) \rightarrow 0
(a 'LAST) \rightarrow 0
(a 53) \rightarrow 56
(a 'TIMES) \rightarrow 1
(a 'LAST) \rightarrow 56
(a 5) \rightarrow 8
(a 'TIMES) \rightarrow 2
(a 'LAST) \rightarrow 8
```

(a 3) → 6  
(a 4) → 7  
(a 'TIMES) → 4  
(a 'LAST) → 7

3. ממשו את הפו' (my-equal? a b)  
המשוה בין

- מספרים (בכל היצוגים)
- מחרוזות (case sensitive)
- symbols
- characters (case sensitive)
- רשימות (מקוננות) של הנ"ל

על זוג ארגומנטי קלט - אם אינם מאותו טיפוס על הפונקציה להחזיר #f  
אם הינם מאותו טיפוס ושויים על הפונקציה להחזיר #t  
אם הינם מאותו טיפוס ושונים על הפונקציה להחזיר #f  
רשימות תחשבנה שוות אם הן באותו האורך ושוות בכל איבריהן (עם חשיבות לסדר)  
אם הקלט או אחד מאיברי הרשימות אינו מאחד הטיפוסים הנ"ל על הפו' להחזיר את המחרוזת "Unexpected type"  
אתם יכולים להשתמש בפו' ההשוואה של Scheme עבור טיפוסים אטומים בלבד אך לא עבור רשימות.

4. ממשו את הפו' (uniform-list? <nested list>)

המקבל רשימה מקוננת ומחזיר #t רק אם כל האברים האטומים של הרשימה מאותו טיפוס (או הרשימה ריקה)  
טיפוסים שיש לבדוק

- boolean
- מספרים
- מחרוזות
- symbols
- characters

5. ממשו את הטיפוס set עבור קבוצה מספרית. כלומר אין חשיבות לסדר וכל איבר מופיע פעם אחת.  
ממשו את הפו'

(empty-set)  
מחזירה קבוצה ריקה

`(set-is-empty? <set>)`  
`(set-is-in? <set> <element>)`  
 הבדוק שייכות  
`(set-add <set> <element>)`  
 המחזיר קבוצה לאחר הוספת האיבר  
`(set-union <set1> <set2>)`  
 המחזיר את קבוצת האיחוד  
`(set-intersect <set1> <set2>)`  
 המחזיר את קבוצת החיתוך  
`(set-length <set>)`  
 המחזיר את גודל הקבוצה  
`(set-map <set> <unary-func>)`  
 המחזיר את תמונת הפונקציה על הקבוצה הנתונה

6. ממשו את הפונ' `(stream-zip <zipping-func> <stream1> <stream2>)` המקבלת שני *streams* ומחזירה *stream*.

אם אחד מהקלטים קצר יותר אזי הפלט הינו באורך הקצר יותר. האלמנט ה-*i* של הפלט מתקבל מהפעלת הפונ' על האלמנטים ה-*i* של הקלטים לדוגמא

`(stream-zip cons primes naturals) ⇒ <(1.1)(2.2)(3.3)(5.4)(7.5)... >`