סדנת תכנות פונקציונלי

01/03/2007 23:00
עידכון אחרון:03/03/2007 22:00

תרגיל 1

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

*** את התרגיל יש להגיש אלקטרונית
בשלב זה אין צורך לבדוק תוקיות הקלט
הגדירו פו' עזר במקומות בהם אתם רואים לנכון
הוסיפו תיעוד במקומות בהם אתם רואים לנכון
פו רי ם ש מת

*** מבנה קובץ הבדיקות TESTS אותו עליכם להגיש:
(= (<func_name> <argumentes>) <expected value>)
לדוגמא: (= (my_add 2 3 4) 9)

*** קבצים שיש להגיש

- TESTS
- ex1.scm
- README - אופציונלי

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

1. סדרת פיבונצי (13,8,5,3,2,1,1) מוגדרת ע"י
$a_1=1$ $a_2=1$ $a_n=a_{n-1}+a_{n-2}$

- ממשו את הפו' (fibon <index>)
- ממשו את הפו' (fibonSum <n>) המחזירה את סכום n האיברים הראשונים בסדרת פיבונצי

2.
- ממשו את הפו' (my_power <x(real)> <n(integer - might be negative)>) המחזירה את x^n
- ממשו את הפו' (n_square <x(real)> <n(integer)> <tolerance(real)>) המחזירה קירוב ל $\sqrt[n]{x}$ (באופן דומה ל my_sqrt שהוצג בכיתה)
נוסחת הקירוב האיטרטיבית ל $\sqrt[n]{x}$: $a_k = \frac{(n-1)a_{k-1}+\frac{x}{(a_{k-1})^{n-1}}}{n}$
את הניחוש הראשוני העבירו כמספר real (עם נקודה עשרונית)
אחרת התוצאה תהיה נכונה אך לא קריאה.

- ממשו את הפו' (second_square <x(real)>) , (third_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 הפו' remainder לחישוב שארית חלוקה

- ממשו את הפו' הבוליאנית (prime <n>) לבדיקת ראשוניות
- ממשו את הפו' הבוליאנית (co_prime <n_1> <n_2>) לבדיקת זרות (אי קיום מחלק משותף מלבד 1)
-
- ממשו את הפו' (phi <n>) המחזירה את מספר המספרים הקטנים מ-n וזרים לו (פו' אוילר)

4. ממשו את הפו' (mul_from_to <n_1> <n_2>) המחזירה את מכפלת המספרים בין n_1 ל n_2
   n! = (mul_from_to 1 n)

5. ממשו את הפו' (det_2_2 <a_1> <a_2> <a_3> <a_4>) את הדטרמיננטה $\begin{vmatrix} a_1 & a_2 \\ a_3 & a_4 \end{vmatrix}$
   $= a_1 * a_4 - a_2 * a_3$

6. מספרי קטלן Catalan מוגדרים ע"י $\{ c_0=1 \; c_n=\Sigma_{k=1}^{n} c_{k-1} c_{n-k} \}$ ומונים את מספר הביטוים החוקיים בn סוגריים ימניים ובn סוגריים שמאליים.
   למידע נוסף על מספרי קטלן אתם יכולים להסתכל ב
   $http://en.wikipedia.org/wiki/Catalan\_number$
   ממשו את הפו' (catalan <index>) המחזירה את c_n

סדנת תכנות פונקציונלי

07/03/2007 23:15
עידכון אחרון:11/03/2007 21:35

תרגיל 1

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

*** את התרגיל יש להגיש אלקטרונית
בשלב זה אין צורך לבדוק תוקיות 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>)

לדוגמא: (= (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 הפו'

- (sum_function <function> <function>)
- (composed_function <function> <function>)

המקבלות שתי פו' אוניות מספריות f,g ומחזירות את הפו' האוניות המספריות f+g,f∘g=f(g(x))

2. נמדל את המספרים המרוכבים ע"י הפו'

(define complex (lambda (a b) (cons a b)))

a+bi → (complex a b)

ממשו את הפו'

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

המחזירות מספר מרוכב שהינו תוצאת החיבור והמכפלה בהתאמה.

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

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

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

6. ממשו את הפו'

   (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 הפו' (even_indexes <list>) המחזירה את סדרת האיברים שבמ-קומות הזוגיים

   שימו לב שפו' זו שונה מהפו' select_even שראינו בכיתה.

8. ממשו את הפו'

   - (filter <list> <boolean function>) המחזירה את אברי הרשימה עליהם הפו' מחזירה #t
   - (multi_filter <list> <boolean function list>) המחזירה את אברי הרשימה עליהם כל הפו' מחזירות #t

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

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

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

    - $b_1 = a_1$
    - האיבר הראשון בסדרת הקלט הגדול מ b_{i-1} ונמצא אחריו $\forall i : b_i =$

    לדוגמא:

    - (monotone '(1 2 4 3 7)) → '(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 הפו' (interleave <list> <list>) אשר בהינתן סדרות $a_0, a_1, a_2, a_3, \ldots$, $b_0, b_1, b_2, b_3, \ldots$, מחזירה את הסידרה $a_0, b_0, a_1, b_1, a_2, b_2, \ldots$

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)

סדנת תכנות פונקציונלי

21/03/2007 01:00
עידכון אחרון:23/03/2007 08:00

תרגיל 3

הגשה:יום שישי, 30/03/2007, 12:00

*** את התרגיל יש להגיש אלקטרונית
בשלב זה אין צורך לבדוק תוקיות 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>)

לדוגמא: (= (my_add 2 3 4) 9)

*** קבצים שיש להגיש

- ex3.scm
- TESTS
- אופציונלי - README

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 בפו' עזר חיצוניות.
שימו לב ששם הפו' תוקן מ n_square ל n_root

2. סידרה רקורסיבית (פרמטרית)

$a_0(x), a_1(x), a_2(x), a_3(x), \ldots$

מוגדרת ע"י הפו'

(first-elem x)

(next-elem n x prev)

לדוגמא $x^n n!$ מוגדר ע"י

first-elem(x) = 1

next-elem(n,x,prev) = x*n*prev

כאשר n הוא האינדקס של האיבר הנוכחי.

ממשו את הפו'

- (k-elem x first-elem-func next-elem-func)
  המחזירה פו' שבהנתן k מחזירה את a_k
- (compute-k-sum x first-elem-func next-elem-func)
  המחזירה פו' שבהנתן k מחזירה את $\Sigma_{i=0}^{k} a_i$
- (compute-inf-sum x first-elem-func next-elem-func)
  המחזירה פו' שבהנתן tolernace מחזירה קירוב ל $\Sigma_{i=0}^{\infty} a_i$ (הניחו שהסדרה מקימת $|a_n| \to 0$)
  תנאי העצירה של הפו' $|a_k| \leqslant tolerance$ ובמקרה זה מחזירה $\Sigma_{i=0}^{k} a_i$
- (my-sin x tolerance)
  המקרבת את sin(x) לפי הנוסחא
  $\sin(x)=\Sigma_{n=0}^{\infty}(-1)^n \cdot \frac{x^{2n+1}}{(2n+1)!}$
  והשימוש בtolerance כפי שהוגדר בסעיף הקודם.
  עליכם להשתמש בcompute-inf-sum מהסעיף הקודם.

3. ממשו את הפו'

- (list-prefix list1 list2)
  המחזירה האם list1 היא רישא של list2
- (sub-list list1 list2)
  המחזירה האם list1 היא תת-סידרה (ברציפות) של list2

שתי הפו' מוגדרות עבור סדרות מספריות בלבד.

לדוגמא:

(list-prefix '(1 2 3 4) '(1 2 3 4 5 6 7 8)) → #f
(sub-list '(1 2 3 4) '(1 2 3 4 5 6 7 8)) → #f
(list-prefix '(2 3 4 5) '(1 2 3 4 5 6 7 8)) → #f
(sub-list '(2 3 4) '(1 2 3 4 5 6 7 8)) → #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 הפו' (satisfy predicate a_1 a_2 a_3 ...) 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 הפו' (my-reverse list) המקבלת רשימה ומחזירה את הרשימה בסדר הפוך ומשתמשת ברקורסית זנב

6. ממשו את הפו' (my-uniqe 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 הפו' (maximizer list1 list2) המקבלת רשימת תוים ורשימת strings ומחזירה את ה string הכולל את המספר המקסימלי של תוים מהרשימה (עם חזרות). (case sensitive)

(במקרה ויש מספר ממקסמים 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 הפו' (substring string1 string2 <optional bool>) המחזיר האם 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 הפו' (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 הפו' (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

סדנת תכנות פונקציונלי

14/04/2007 23:30
עידכון אחרון:16/04/2007 00:17

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 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>)
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 הפו'

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

1. • ממשו את הפו' (tri-uncurry f) אשר בהנתן פו' המקבלת שלושה ארגומנטים 'אחד אחד' מחזירה פו' שקולה המקבלת שלושה ארגומנטים.

- ממשו את הפו' (uncurry f) אשר בהנתן פו' המקבלת מספר כלשהו של ארגומנטים 'אחד אחד' (לפחות ארגומנט אחד) מחזירה פו' שקולה המקבלת את הארגומנטים 'בבת אחת'

הערה: הפו' 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 הפו'

- (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 הפו'

- (list->stream list) אשר בהנתן רשימה מחזירה stream שקול.
- (stream->list stream) אשר בהנתן stream סופי מחזירה רשימה שקולה.

4. ממשו את הפו' (make-power-stream n) אשר בהינתן 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 הפו'

- (stream-add stream stream)
- (stream-convolution stream stream)

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

הסכום מוגדר איבר איבר

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

והקונבולוציה מוגדרת

$$c_0, c_1, c_2, c_3, \ldots$$

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

פשר מקובל: טורי חזקות $< a_0, a_1, a_2, a_3, \ldots > \Rightarrow \Sigma_{i=0}^{\infty} a_i x^i$

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:

(stream-add (tail 3 naturals) (tail 5 naturals) → <8,10,12,14,...>

(stream-convolutiom <1,1,1,...> <1,-1> → <1,0,0,0,0,0...>

(stream-convolutiom <1,1,1,...> <1,-1,0,0,0> → <1,0,0,0,0,0...>

סדנת תכנות פונקציונלי

20/05/2007 23:45
עידכון אחרון:24/05/2007 21:45

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 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.

*** 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>)
לדוגמא: (= (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 הפו' (sum <int>)
ממנה נגזרות פו' אונריות $f_0, f_1, f_2, f_3, \ldots$ 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 הפו' (long-func1 f n)
המחשבת בהנתן $n > 0$ ופו' המקבלת שני ארגומנטים את
$(f\ 0\ (f\ 1\ (\ldots\ (f\ <n-1>\ n))))$
ממשו את הפו' (long-func2 f n)
המחשבת בהנתן $n > 0$ ופו' המקבלת שני ארגומנטים את

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

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

לדוגמא

(long-func1 (lambda (x y) (+ x y)) 6) → 21
(long-func2 (lambda (x y) (+ x y)) 6) → 21
(long-func1 (lambda (x y) (* x y)) 6) → 0
(long-func2 (lambda (x y) (* x y)) 6) → 0
(long-func1 (lambda (x y) (if (or (= x 0) (= y 0)) 1 (* x y))) 6) → 720
(long-func2 (lambda (x y) (if (or (= x 0) (= y 0)) 1 (* x y))) 6) → 720
(long-func1 (lambda (x y) (- x y)) 6) → 3
(long-func2 (lambda (x y) (- x y)) 6) → 3
(long-func1 (lambda (x y) (- x y)) 5) → -3
(long-func2 (lambda (x y) (- x y)) 5) → 3

3. ממשו את הפו' (split-at <i:int> <x:stream of numbers/list of numbers>)

המחזירה pair בו $(i \geqslant 0)$

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 (<list1> . <list2>) כוונתי (car ...) → <list1>
(cdr ...) → <list2>

(split-at 2 '(0 1 2 3 4 5)) → ((0 1) . (2 3 4 5))
(car (split-at 2 '(0 1 2 3 4 5))) → (0 1)
(cdr (split-at 2 '(0 1 2 3 4 5))) → (2 3 4 5)
(split-at 0 '(0 1 2 3 4 5)) → (() . (0 1 2 3 4 5))
(split-at 9 '(0 1 2 3 4 5)) → ((0 1 2 3 4 5) . ())
(split-at 2 naturals) → ((0 1) . < 2345... >)
(split-at 0 <1 1>) → (<> . <1 1>)
(split-at 9 <1 1>) → (<1 1> . <>)

4. ממשו את הפו' (fib-generator)

המחזירה stream הכולל את מספרי פיבונצ'י $1, 1, 2, 3, 5, 8, \ldots$

רמז: השתמשו ב stream-add שממשתם בתרגיל הקודם.

5. ממשו את הפו' (partial-sums <stream:infinite/finite/empty>)

המקבלת stream

ומחזירה את ה stream של הסכומים החלקיים.

לדוגמא

(partial-sums <1,3,2,1,...>) → <1,4,6,7,...>

(partial-sums <1,1,1>) → <1,2,3>

(partial-sums <>) → <>

6. ממשו את הפו' (is-length <stream/list> r)

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

7. ממשו את הפו' (stream-merge <list of infinite streams>)

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

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

לדוגמא

(stream-merge <2,4,6,8,...> <1,3,5,7,...>) → <1,2,3,4,5,...>

(stream-merge <2,4,6,8,...> <3,6,9,12,...>) → <2,3,4,6,8,9,10,12,...>

$\forall stream\ s$ (stream-merge s) → s

$\forall stream\ s$ (stream-merge s s) → s

$\forall stream\ s$ (stream-merge s s s) → s

סדנת תכנות פונקציונלי

16/06/2007 23:45
עידכון אחרון:20/06/2007 22:30

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 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.

*** 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>)

לדוגמא: (= (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 הפו' לתוך ex6.scm

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

1. ממשו את הפו' (expr2func expression)

המקבל ביטוי במחרוזת ומחזירה פו' 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-הביטוים לבין האופרטור

(expr1 + expr2)

(expr1 - expr2)

(expr1 * expr2)

לדוגמא

((expr2func "5")) → 5

((expr2func "(5 + 13)")) → 18

((expr2func "(x + 13)") 1) → 14

((expr2func "(x - y)") 1 2) → -1

((expr2func "(x - x)") 1) → 0

((expr2func "((z * z) - (y * y))") 5 4) → 9

2. ממשו את הפו' (add-monitor func)

המקבלת פו' f (מספרי אשר מקבלת ארגומנט אחד) ומחזירה פו' חדשה שנסמנה כ $f2$

$f2$ בהיקראה עם ארגומנט מספרי מחזירה כמו f

$f2$ בהיקראה עם הארגומנט $TIMES'$ מחזירה את מספר הפעמים שנקראה עד כה עם ארגומנט מספרי

$f2$ 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) → 0

(a 'LAST) → 0

(a 53) → 56

(a 'TIMES) → 1

(a 'LAST) → 56

(a 5) → 8

(a 'TIMES) → 2

(a 'LAST) → 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 ושוים על הפונקציה להחזיר $\#t$

אם הינם מאותו טיפוס ושונים על הפונקציה להחזיר $\#f$

רשימות תחשבנה שוות אם"ם 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>)
הבודק 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 הפו' (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 הפו' על האלמנטים ה i של הקלטים

לדוגמא

(stream-zip cons primes naturals) $\Rightarrow <(1.1)(2.2)(3.3)(5.4)(7.5)...>$