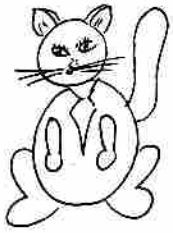
① 27.02.07
PLLAB



מרצה: אילן ilan_h@cs.huji.ac.il
pllab@cs.huji.ac.il
news group local.course.pllab.ta/stud

מבחן - אין
תרגילים - יש. בדוק את עצמך. מאוישים אוקטרונית. בדוק
עבודת register רבי ראשים תרגילים.
הגשה - קובץ קוד
קובץ בפיקוח
קריאותי

יש לקרוא את ההבדלאות
שמופיעות באתר.

תכנות פונקציונלי: זה משהו שונה מאשר שאפנו עד כה.
בתכנות פונקציונלי יש ביטויים אבל אין זרם. פחות מסתבכים
אל ניכון. בשפה יש ערכים שמתחלפים.
השפה שנלמד איתה היא scheme. יש שונה לתוכנה באתר.

ב scheme יש 4 טיפוסים של מספרים: $\frac{2}{3}$, 3.4, 3e4

טיפוסים בוליאניים - #t, #f

את הפעולות רושמים כ- prefix. למשל 5+4 תהיה +54.
scheme עושה DFS על העץ של הביטוי ועל פעם מתלף הרכבה
בביטוי אטומי.
על מקום שכותבים בו ביטוי אטומי אפשר לרשום פונקציה כי יהיה זה ערך
שאפשר לשרשר וכך להמשיך הלאה.
כאן יש שפה שמרבים את עצמה יש התניה שה סינטקס שלה הוא לה:

(if <exp> <exp> <exp>)

למשל (if (< 1 2) 47 4). במקרה לה הפלט הוא 4
ואולי 47 לא משנה כרגע.
ב scheme תמיד חייבים להיזהר את התנאי של ה else כי אחרת תהיה
בעיה לשרשר.

יש את הפקודה שמזכה ל- switch :

```
cond ( ((...)(...))
       ((...)(...))
       (else (...)) )
```

אפשר להגדיר קבועים (define my-pi 3.14)

ואת אפשר להגדיר פונקציות (define (square x) (* x x))

עין בה שום טיפוסיות. אפשר לרשום גם (square 2) וגם (square 3.14)

בניגוד שבאיפון פונקציה נקלה לעשות רקורסיה

```
(define (sum_from_one n)
  (if (= n 1) 1
      (+ n (sum_from_one (- n 1)))))
```

אולם כך מסתבר זה האופן שיעדוק ? הוא משומש רק כשכבר אנחנו צריך לעשה אותו- למשל כשהוא יקבל קלט מסביבה. אלא שאז תהיה הקרטוה איטופוית אלא הוא "פרנה" שמזמן שמדבר אפי הקלט.

אפשר להגדיר מחדש את = עושה. לדוגמא

```
(define (= a b)
    (or a b)
)
```

אפשר להגדיר פונקציות כפרמטר, פשוט מעבירים את השם של הפונקציה ומפנים אפשר להשתמש בה כרצוך.

אפשר להגדיר פונקציה בלי לתת לה שם. `((lambda (x) (* x x)) 5)`
ואת זה אפשר להגדיר פונקציה. ההבדל בין זה לבין הדוגמה מאחרונות שהגדרה `define` זה כאן ההבדל בין `(define pi 3.14)` לבין סתם הערך 3.14.

אחרשון הדוגמא הנכונה להגדיר פונקציות הוא

```
(define square
    (lambda (x) (* x x))
)
```

נאו שאפשר לכתוב פונקציה אפשר יותר להתאיר פונקציה.

```
(if (> y 2)
    (lambda (x) (+ x 1))
    (lambda (x) (+ x 3))
)
```

מה שמתאר בין פונקציה שמחזירה בעצם הכיוון של מה שמשתנה שאין בו צורך.
אפשר להגדיר את הפונקציה ויש להפעיל אותה על מספר.

```
((lambda (x) (* x x)) 5)
```

מבנה הנתונים הכי פשוט שיש בה רשימה.

pair זה בעצם של שני ערכים (•,•). מיצגים אותו ע"י construct : (cons 1 3) משמש לשתי הפעולות האומר הראשון משתמשים ב- car והשני משתמש ב- cdr.

האומרים לנו חיובים להיות בעצם של בוליאן או ביט של ביט ומספר.

אפשר לייצר מבנה נתונים של רשימה מקושרת ע"י בוליאנים אחד מחזיק השני, ומסוף רשימה ניקח.

(cons 1 (cons 2 (cons 3 ( ))))

יש ערכים יותר קצרים להגדיר רשימה: (quote (1 2 3 4 5))

(1 2 3 4 5)!

(list 1 2 3 4 5)

ההפרש בין list לבין השניים הקודמים הוא ש list גם מפעיל פעולות על הפרמטרים אחרי quote משאיר אותם כמו שהם.

(quote (1 (+ 1 1) 3 4 5)) = (1 (+ 1 1) 3 4 5)

(list 1 (+ 1 1) 3 4 5) = (1 2 3 4 5)

פונקציה שכדאי לממש ברשימות.

reverse - הופך רשימה

append - משרשר רשימות (לוקח שני בוליאנים, אחד הופך את הרשימה כרשימה אחת שהוא השרשור)

map - מחזירה רשימת ערכים של פעולה פונקציה על כל איברי הרשימה.

③ 13.3.07
Pllab

הקדמה אפיונים סטרוקטורים: התפרסם באתר תאריך למבחן.
אין מבחנים עבר להכנס ולחלק

ב scheme יש אפשרות שפונה למשתנים.
נניח שיש פונקציה מהצורה:

(define complex_func (lambda (s) ((s)

ויש פונקציה אחרת שמשתמשת ב-complex_func. אם היא משתמשת בה לראשונה נוכל לא יעלם את הינו נדרשים להגדיר משתנה שמקשר את ערך הפונקציה ואת עובדים איתן.
לשים binding באופן זמני לרשימת שמות

(let ((x1 e1)
(x2 e2)
⋮
(xn en)
)
e) ← הביטוי שמשתמשים

איך משתמשים בזה?

(define list-proc
(lambda (ls)
(let ((ans (complex_func ls))
)
(cond ((=0 ans)
⋮
)
)
)

סדר הביצוע: - משתכנים את n הביטויים
- מושיבים אותם לשמות
- משתכנים את הביטוי e
כה חשוב רשים תלויות בין הביטויים.

מבנה דומה אפשר להגדיר פונקציות פנימיות שלהן מוכרות מתוך let שמותן הוגדרו.

define זה הגדרות משותפות! let זה לקישור מקומי.

יש מופן שלרוך אתי שמו משתנים ואין מקשרים איך. ואיך למשל זה יהיה חוקי:

```
(let*
   ((x 5)
    (y x)
   )
   (+ x y)
)
```

כי y מכיר את x הקודם.

לכן תמיד עדיף להשתמש ב-let* למשל את הלך הבא:

```
(define x 7)
(define y 8)
(let ((x y)
      (y x))
   (+ x y)
)
```

מתייחס אל x ו-y בפשוט

שאלתון היה בשנים דרך להשפיע על let.

יש מדיה זה פונקציות בקונטקסט כי בכניסה השנייה לתוך הפונקציה הנסה שלה כבר לא מוגדר בשבילה יש את let ואז אפשר להגדיר פונקציות בקונטקסט שקוראות אחת לשנייה. סדר הקוד ב let הוא חלוי אימפלמנטציה אבל עדיף לא להשתמש כי אחרי פונקציות בקונטקסט.

20.3.04
Plab

אפשר להגדיר רשימה של סמבולים - (point 2 line)'
וכך מתייחס לכל מה שבפנים מתוך סימנף.
ואפשר גם ככה: (point' 2 line' list) ואז הוא מתייחס
ל-2 רגל זרק נותרו ו- line ו- point הם סימבולים.

אפשר לתפוס אותו משהו וזה סימבול ע"י symbol? והשווה
לשורת ע"י eqv?.

הפונקציה eval משערכת. אם נתנים לה סימבול היא משתמשת
בו במובן הרגיל. היא מוציאה לאור מה שקשור ומציגה את
הערך. אם יש לה נתונים לה סתם משתנים רשלום אין היא
פשוט מתשבחת אותם וכו'.

למשל מה קורה כי אפשר בכלל לשנות פונקציות. זה יש

```
(define funcA (lambda (ls)
                (length ls)))
```

★

```
(define myprog
  '(define funcA (lambda (ls)
                   (* 2 (length ls)))))
```

*

★ מאידך פונקציה funcA. ערך * לא עושה פעם.
בכלל שניתן (eval my-prog) את המוח מתבלבל
הגדרה של funcA.

עוד - יש שני דברים שאפשר להסתבך עליהם -
עילות מן ועילות מקום.
למשל חישוב עצרת

```
(define fact (lambda (n)
               (if (= n 0) 1
                   (* n (fact (- n 1)))))))
```

סימנויות בצמן הזה עיקריות ואפשר גם לבן אשתו מחינת
ביכרון. אך שנוא לבן מציין אך $n=0$ הזה לבן יכול
לבצע את הביטוי אך הזה גם בצמן הזה מתיק ביטוי
פתות בביכרון.

אף פיאה לחוסר יעילות משות בצמן גם רושם פיבונצ'י.
מתישהם הרשו ע"י הקשר הרקורסיבי $f(n)=f(n-1)+f(n-2)$
יש אפשר אקספוננציאלי של קריאות.

מחוסר יעילות הזה ניתן לטפל ע"י תיבדות פונקציות עזר
שניתות.

רקורסיית זנב tail recursion

tail position - נקודה שבה הפונקציה מחליטה שהיא
מחזירה ערך של פונקציה בלשהי. הסמנטיקה של
Scheme אומרת שהקומפיילר חייב לממש את
גם מיטלות- וכך המחסנית לבן מתפוצצת. ברגש
שמחזירים חלק של ה- ...b return הזה פשוט שולח
לפונקציה הבאה במחסנית ומכניס רק את מה
שרוצים ל- b.

שוויון כאן שבבה רוצים = לבן עובד עבור משתנים
מחיצניים. הפונקציות eq? equal? ו- eqv?
הן פונקציות- הן ניתנות תשובה לפי טיב משתנה.
הפונקציות הזאת הן ותמי שקילות וזה הן מהפלות שני משתנים
מסוגים שונים הן מחזירות false.

eqv? עובדת על bool, symbol, number, char,
null, empty, string, pair, list ו- function.
אפילו על רשימות הנכונים הבא מחזיר true רק אם הם
"שמה ממש מאותו מקום בביכרון". equal? עובדת כפיפי על כל אלה.

השיש טיפוסים קלת מורכבים יותר ברור שעדיין הוא מהבחר
שעדיין שם ערכים. ויהיו לו היות שפעמים אין להשוות כפונקציות.
שמשם פונקציות אי אפשר להשוות לפני ערכים. מה פשוט
מתי ואפשרי. הדרך הנכונה להשוות פונקציות הינו לפי
פונקציות.

ב - scheme כשמדברים identifiers לא מתייחסים אות
הטיפוס שלו והוא יכול להקבל כל ערך. הדרך היחידה
לבדוק טיפוס היא בצמן ריצה.
מה שכן נמצא זה מספר הפרמטרים - מוצאים שמספר
הפרמטרים בדה לא תמיד של הפונקציה.

בשפה שבה לכל פונקציה יש תמיד קומפילציה אפשר למצוא
שרשימת היא הומוגנית - כלומר שכל האיברים מסוגים מאותו
סוג. ב - scheme לה מתי אפשרי. את הטיפוס הזה
מתאם list ולכן (list(int) למשל.
את אפשר מאוד לחתוך הרבה. לה יתרון יפהפה. אולם החיסרון הזה
שאין שום מפיקה.

⑥ 27.3.07
Plab

עושה ותתן לראות את ההבדלים ביתר שפות
פונקציונליות.
אך עושין הפונקציות קיבלו מידע פנימיות והתנהגו מיוחד.
כאינו יכם פונקציות שמקבלות ומתנהגות פונקציות אומץ ערוב לה
גוה שלה אכ לפונקציות שאנחנות מיוחד.

יש אלגוריתמים שמקבלים פונקציות כפרמטר. למשל sort
מקשרי מלכתי ופונקציית השוואה. אבל sort הינ מספר יבוה.
הינו לכאורה קשורה למשל data בסביבה.

מה מאפשר לזיהוי אלגוריתמה של פונקציות. וכיצ אפשר לחתאים
בגלל גוריתמים כלשהו שהוה ארכסת ענו איך פונקציות בסביבות
ממומשת.

למשל אם יש פונקציה שמרכיבה פונקציות ופונקציה שמתציבה
נתכרת איך ישב אותנו יכולים לחשב נגזרת מכל סדר.

Streams

פונקציות קבוע מנסות לשנות את ה אוטומטים שהן וכל
לפעול. אבישבניה אם פונקציה מנסה לקחת את סבמה.

בינוי תסר פרמטרים נקרא promise - (2 () lambda)
ושמשלכים אותו מקבלים 2.

stream בת בכלל של המטחות.
והמטחה בה משהו שלא תפסו כיפון. לה בזמה לכו ק"מ אך שלא
מבחשת אותה. כיצ stream וכול להיות אינסופי.

5.6.07
PLLAB

הודעה: תרגיל 5 נדחה בכמה ימים. בהצלחה!

* * * * *

היום נדבר על תכנות לא פונקציונלי. מסתבר שאי אפשר לעשות הכל בתכנות פונקציונלי אז יש ב-scheme אופציה לתכנת גם לא פונקציונלית.

אנחנו נדבר על פקודות שיש להם תופעות לוואי, כלומר הן משפיעות על סביבת העבודה.
הדוגמה הראשית למשהו כזה היא פקודות קלט/פלט. למשל, יש שתי פונקציות שפולטות למסך – write ו-display. סה"כ הן עושות את אותו הדבר אבל יש ביניהן כמה הבדלים קטנים. יש גם פונקציות שקולטות מהמשתמש תווים בודדים או מחרוזות שלמות.
Begin מאפשר לנו להריץ רצף של פקודות אחת אחרי השנייה. זה נוח בשביל דיבאג אן נניח רוצים להדפיס פרמטרים שמועברים לפונקציה.
אפשר לעשות השמה. הסינטקס הוא (set! Identifier expression). ההבדל בין set! ל-define הוא ש-define מקצה תא בשיכרון ושם בו ערך ואילו set! מחפש את התא בזיכרון שמוקצה ל-identifier ושם בו ערך. לכן אי אפשר לעשות השמה לתוך משתנה שלא עשו לו define קודם. יש פקודות דומות עבור זוגות – set-car! ו-set-cdr!

Scheme יודעת לזעות תאים שאין אליהם מצביע ולמחוק אותם. אז אנחנו לא צריכים לדאוג לדליפות זיכרון.

החלק האחרון של הקורס – גרפיקה. יש חבילת גרפיקה שמאפשרת לעשות כל מיני דברים. באתר יש דוגמאות קוד. מומלץ להסתכל על העץ הפרקטלי.

סוף