



מטה מל"מ
המרכז הישראלי לחינוך מדעי טכנולוגי
על שם עמוס דה-שליט



משרד החינוך
המזכירות הפדגוגית
האגף לתכנון ולפיתוח תכניות לימודים



מגוון – מחקר ופיתוח בהוראת מדעי-המחשב
המחלקה להוראת הטכנולוגיה והמדעים
הטכניון – מכון טכנולוגי לישראל
ומוסד הטכניון למחקר ופיתוח בע"מ

תכנות פונקציונלי

פרדיגמה תכנותית נוספת

ספר שני

פרק שי"ט: פיתוח תכנות מורכבות

**החומרים ניתנים להורדה ושימוש בחינם במסגרת בית הספר.
חל איסור על הפצה מסחרית של החומרים.**

פיתוח היחידה:

פרופ' אורי לירון - ראש הפרויקט

תמי לפידות - ראש צוות הפיתוח

דלית לוי

תמר פז

יעוץ אקדמי: ד"ר שאול מרקוביץ

פרק שישי: פיתוח תכניות מורכבות

58.....	משחק הניחושים
61.....	משחק התליין
65.....	משחק איקס-עיגול
69.....	דוקטור

דוגמה ראשונה: משחק הניחושים

נכתוב תכנית שבה המחשב מגריל מספר אקראי בתחום 1-10, המשתמש מנחש מהו המספר, המחשב בודק את הניחוש ומגיב בהתאם.

א. נתחיל מהפונקציה `check` המקבלת שני מספרים: את `num` שהוא המספר האקראי, ואת `guess` שהוא הניחוש של המשתמש. הפונקציה תבדוק האם המספרים זהים ותחזיר הודעה מתאימה. כתבו את הפונקציה `check`.

ב. כתבו פונקציה בשם `rand-num` המחזירה מספר אקראי שלם בתחום 1-10. תזכורת: הפונקציה (`random n`) מחזירה מספר אקראי שלם בתחום 1-n.

ג. בדקו את הפונקציה הבאה:

```
(define (example)
  (print '( this is the first line ))
  (print (+ 1 2 3 ))
  (print 'shalom ))
```

הסבירו מה מבצעת הפונקציה `print`.

הסבירו מדוע יש צורך בפונקציה `print` במשחק הניחושים.

ד. כיצד יבקש המחשב את המספר מהמשתמש? לשם כך נכיר פונקציה נוספת בשם `read`. כאשר המחשב פוגש בפונקציה `read` הוא עוצר ומחכה לקלט מהמשתמש. לאחר שהמשתמש מקיש `enter` הקלט הזה מוחזר כפלט של הפונקציה `read` הגדירו את הפונקציות הבאות במחשב ובדקו מה מבצעת כל אחת מהן.

```
(define (sod1)
  (print '( please enter a number ))
  (+ (read) 5))
```

```
(define (sod2)
  (print '( what is your name ? ))
  (sod3 (read)))
```

```
(define (sod3 who)
  (print '( nice to meet you ))
  (print who)
  (print '( have a nice day )))
```

```
(define (sod4)
  (print '( enter a number 1-10 ))
  (sod5 (read)))
```

```
(define (sod5 x)
  (cond
    [( > x 5) 'big ]
    [( = x 5) 'five ]
    [else 'small ]))
```

כדי לכתוב תכניות אינטראקטיביות (כמו בדוגמה של משחק הניחושים) נעזר בפונקציות `print` ו-`read`. הן יאפשרו לנו להדפיס על המסך הודעות שונות ולקלוט את הנתונים של השחקן (המשתמש).

ה. כתבו פונקציה בשם `game` המקבלת פרמטר אחד - `num` - שהוא המספר אותו צריך לנחש. הפונקציה תדפיס הודעה למשתמש (נחש מספר בין 1 ל-10), תקבל ממנו את המספר (בעזרת `read`) ותקרא לפונקציה `check` שכתבתם בסעיף א'.

ו. כדי להשלים את התכנית, כתבו פונקציה בשם `top (main)` שמגרילה מספר אקראי בתחום 1-10 וקוראת לפונקציה `game` שכתבתם בסעיף הקודם. (ניתן להעזר בפונקציה `rand-num` שכתבתם בסעיף ב')

ז. נסו לשפר את התכנית ולקבל מהמשתמש את התחום ממנו יוגרל המספר (במקום 1-10).

דוגמה שנייה: משחק התליין (Hangman)**תיאור המשחק**

המחשב בוחר באופן אקראי מילה מתוך רשימת מילים ומטרת המשתמש היא לגלות את המילה שנבחרה.

מהלך המשחק:

1. המחשב בוחר מילה ומציג על המסך קוים כמספר האותיות שלה.
2. המשתמש מנחש אות ומקליד אותה בתיבת הדו-שיח.
3. המחשב בודק את האות ומגיב בהתאם.
4. אם המשתמש עוד לא גילה את כל האותיות של המילה חוזרים לשלב 2.

דוגמת הרצה

> (hangman)

(the word is)

(_ _ _ _ _)

(enter new guess) t

(the word is)

(_ _ _ _ _)

(enter new guess) c

(the word is)

(c _ _ _ _)

(enter new guess) s

(the word is)

(c _ _ s s)

(enter new guess) a

(the word is)

(c _ a s s)

(enter new guess) p

(the word is)

(c _ a s s)

(enter new guess) l

(the word is)

(c l a s s)

success

תהליך הפיתוח של תכנית מורכבת מתנהל בשלבים.

א. תחילה נבחר את מבני הנתונים בהם נשתמש בתכנית.

במשחק התליין עלינו לייצג את הנתונים הבאים:

- אוסף של מילים מהן נבחרת המילה שהשחקן צריך לנחש
- מילה הנבחרת באקראי ואותה צריך לנחש
- אות שאותה מבקש השחקן לבדוק
- מילה המכילה את כל האותיות שהשחקן כבר ניחש ויתר האותיות מסומנות על-ידי קו _

בחרו מבני נתונים מתאימים.

לאחר שהחלטנו על מבני הנתונים ניתן לגשת לכתיבת הפונקציות של התכנית.

ב. כתבו פונקציה בשם `pick` שבוחרת באקראי איבר אחד מתוך רשימה של מספר אברים. שימו לב: אורך הרשימה אינו ידוע מראש.

ג. כתבו פונקציה בשם `build-places` שמקבלת רשימה באורך `n` ומחזירה רשימה המכילה `n` קוים. למשל, בעקבות ההוראה `(build-places ' (c l a s s))` נקבל את הרשימה `(_ _ _ _)`

ד. כתבו פונקציה רקורסיבית בשם `update` המקבלת שלושה פרמטרים: `target` (המילה שאותה צריך לנחש), `guess` (המילה שמכילה את כל האותיות שהשחקן הצליח לנחש עד כה) `letter` (האות שבודקים בשלב הנוכחי). הפונקציה תעבור על האותיות של `target` ותבדוק כל אות האם היא שווה לניחוש הנוכחי `letter`. אם כן, היא תעדכן את `guess` ותכניס בה במקום המתאם את הניחוש הנוכחי. למשל, בתגובה להוראה `' (c _ _ _) 's (update ' (c l a s s))` נקבל את הרשימה `(c _ _ s s)`

ה. כתבו פונקציה רקורסיבית בשם main המבצעת את עיקר המשימות בתכנית. היא מקבלת שני פרמטרים - target (המילה אותה צריך לנחש) ו-guess (המילה המכילה את האותיות שנמצאו עד כה). הפונקציה תבדוק האם target שווה ל-guess. אם כן, המשחק מסתיים ומוחזרת ההודעה success. אחרת, מודפסת בקשה למשתמש לנחש אות נוספת ומתבצעת קריאה חוזרת לפונקציה main עם הפרמטרים המתאימים. שימו לב ש-target לא משתנה ואילו guess צריך להתעדכן בעזרת הפונקציה update (אליה מועברים guess, target, והניחוש החדש באמצעות הפעלת הפונקציה read).

ו. השלימו את התכנית.

דוגמה שלישית: משחק איקס-עיגול

במשחק זה יש לוח 3×3 ובו 9 משבצות. יש שני שחקנים ולכל אחד מהם יש סימן קבוע. למשל, לשחקן אחד (המשתמש) הסימן x ולשני (המחשב) הסימן o. מטרת המשחקים היא להשלים שלישיה מאותו סימן. השלישיה יכולה להיות שורה, עמודה או אלכסון.

	X	
O		

א. תחילה נבחר את מבני הנתונים בהם נשתמש בתכנית.

במשחק איקס-עיגול עלינו לייצג את הנתונים הבאים:

- לוח המשחק (בהתחלה הלוח ריק ובמשך המשחק הוא מתעדכן)
- מקום אותו מבקש השחקן לסמן
- כמו כן עלינו להחליט איך נסמן כל משבצת בלוח המשחק (כדי שהשחקנים יוכלו לבחור במשבצת כרצונם).

בחרו מבני נתונים מתאימים.

ב. כדי לאפשר למחשב (המשתתף במשחק עם הסימן o) לבחור משבצות בצורה הגיונית עלינו ללמד אותו את האסטרטגיה לנצחון. כלומר, כאשר צריך לבחור משבצת פנויה, מהי הבחירה הטובה ביותר. נסחו במילים שלכם את האסטרטגיה לנצחון במשחק איקס-עיגול.

ג. כתבו אלגוריתם ראשוני לתכנית.

ד. כתבו פונקציה בשם `get` שמקבלת לוח במצב מסוים ומשבצת מבוקשת ותחזיר את הסימן שיש במשבצת הזו כרגע. (שימו לב שהפונקציה יכולה להחזיר אחד מבין 3 ערכים: `x`, `o`, או מקום ריק).

ה. כתבו פונקציה בוליאנית בשם `free?` שמקבלת לוח במצב מסוים ומשבצת. הפונקציה תבדוק האם מותר לבחור במשבצת הזו. כלומר, האם היא פנויה (יש בה סימן `_`) ותחזיר `#t` אם כן, ו-`#f` אם לא.

ו. כתבו פונקציה בוליאנית בשם `full?` שמקבלת לוח ובודקת האם הוא מלא ולא נשארו בו משבצות פנויות (מצב כזה במשחק, שבו אין לאף אחד שלישייה ואין יותר מקום - נקרא תיקו).

ז. כתבו פונקציה בשם `print-board` שמקבלת לוח במצב מסוים ומדפיסה אותו על המסך בצורה מסודרת.

ח. כתבו פונקציה בוליאנית בשם `legal?` שמקבלת לוח ומשבצת מסוימת שאותה מבקש השחקן לסמן ובודקת האם המהלך הזה חוקי (למשל, האם המשבצת פנויה). אם כן, הפונקציה תחזיר `#t`, אחרת יוחזר `#f`.

ט. כתבו פונקציה בשם `update` שמקבלת לוח במצב מסוים, משבצת וסימן (`x` או `o`). הפונקציה תחזיר לוח חדש שבו מוכנס הסימן המבוקש במשבצת המתאימה. (בהנחה שמותר לשים שם את הסימן הזה).

י. כתבו פונקציה בוליאנית בשם win? שמקבלת לוח וסימן. הפונקציה תבדוק האם יש בלוח הזה שלישייה של הסימן המבוקש. אם כן (זהו נצחון לבעל הסימן), הפונקציה תחזיר #t, אחרת יוחזר #f.

יא. העזרו באלגוריתם שכתבתם ובפונקציות שהכנתם בסעיפים הקודמים כדי להשלים את התכנית.

דוגמה רביעית: דוקטור (ELIZA)

"דוקטור" היא תכנה שמשוחחת עם המשתמש ומגיבה למשפטים שלו בצורה שמחקה תגובה אנושית. התוכנה קולטת משפט מהמשתמש, מזהה את סוג המשפט (לפי מילות מפתח) ומגיבה בהתאם.

דוגמת הרצה של התוכנה

Hi, how are you ?

➤ I feel fine.

Go on.

➤ Yesterday I had a problem.

Yesterday you had a problem?

➤ Yes.

Earlier you said that you feel fine.

➤ Can you help me?

I ask the questions here!

➤ I need help.

Please continue.

➤ My mother is angree with me.

Do you love your family?

➤ Yes, but we had a big fight yesterday.

You think that you had a big fight yesterday?

➤ She said that I'm not working hard enough.

Please continue.

➤ I don't have time now. I have to go and study. Goodbye.

Goodbye.

Done.

שלב 1 - משפט בודד

בשלב ראשון נטפל במשפט בודד. נפתח בשלבים תכנית קצרה שמקבלת משפט אחד ומגיבה עליו. אם המשפט מכיל את המילה 'אמא', המחשב יגיב במשפט 'ספר לי עוד על המשפחה שלך'. בכל מקרה אחר, המחשב יגיב במשפט נייטראלי כמו 'ספר לי עוד משהו', 'מעניין מאד', 'אני נהנה לשוחח איתך'.

א. מבני נתונים

כמו בדוגמאות הקודמות, גם הפעם נתחיל מהנתונים.

כדי שהמחשב יוכל להגיב בצורה אקראית למשפטים שלנו, עלינו לשמור רשימה של משפטים "ניטראליים". נגדיר משתנה בשם NS שמכיל משפטים כאלה.

(define NS '((tell me something else) (really?) (Oh, great) (please continue)))

ב. כתבו פונקציה בשם pick שבוחרת משפט אקראי מתוך הרשימה NS.

ג. כתבו פונקציה בשם random-answer שמדפיסה משפט אקראי מתוך הרשימה NS (כדאי להעזר בפונקציה pick).

ד. כתבו פונקציה בוליאנית בשם member? שמקבלת מילה ומשפט (רשימת מילים). הפונקציה תבדוק האם המילה מופיעה במשפט. אם כן - יוחזר #t, אחרת יוחזר #f.

ה. כתבו פונקציה בשם `response` שמקבלת משפט (רשימת מילים). הפונקציה תבדוק האם המילה `mother` מופיעה במשפט. אם כן - תודפס הודעה מתאימה (למשל, ספר לי עוד על המשפחה שלך). אחרת, תודפס תגובה אקראית (כדאי להעזר בפונקציה `random-answer`).

שלב 2 - הרחבה לטיפול ב"משפטי משפחה"

בשלב הקודם טיפלנו רק במילה בודדת 'אמא' וחיפשנו אם היא מופיעה במשפט. כעת נשפר את תגובת המחשב בכך שהוא יוכל לזהות מילת משפחה כלשהי ולא דוקא 'אמא'. כלומר, אם המשפט יכול את מילת משפחה (כמו 'אבא', 'סבתא', 'דודה' וכדומה), המחשב יגיב במשפט 'ספר לי עוד על המשפחה שלך'. בכל מקרה אחר, המחשב יגיב כמו קודם במשפט נייטרלי. כיצד נעשה זאת?

נגדיר משתנה חדש בשם `family` ונשמור בו רשימה של מילות משפחה.

(`define family (mother father sister brother uncle)`)

בהמשך, נעזר ברשימה הזו כדי לבדוק האם אחת מהמילים האלה מופיעה במשפט.

א. כתבו פונקציה בוליאנית בשם `contains?` שמקבלת שתי רשימות. הפונקציה תבדוק האם אחת מהמילים של הרשימה הראשונה מופיעה ברשימה השנייה. אם כן, יוחזר `#t` אחרת יוחזר `#f`.

ב. כתבו מחדש את הפונקציה `response` (מהשלב הקודם) כך שתזהה הפעם מילת משפחה כלשהי ותגיב בהתאם.

שלב 3 - הרחבה לטיפול בשאלות

עד כה זיהינו רק סוג אחד של משפט (משפט משפחה). הפעם נרחיב את תגובת המחשב לטיפול בשאלות. שאלה היא משפט שמסתיים בסימן שאלה.

א. כתבו פונקציה בוליאנית בשם `question-asked?` שמקבלת משפט (רשימת מילים) ומחזירה `#t` אם הוא מסתיים בסימן שאלה. בכל מקרה אחר, יחזר `#f`.

ב. כתבו מחדש את הפונקציה `response` כך שתזהה הפעם משפט שאלה ותגיב עליו במשפט כרצונכם.

שלב 4 - מעבר לדו-שיח חוזר

עד כה טיפלנו במשפט יחיד, אולם התכנית השלמה צריכה לטפל במשפטים רבים. המטרה שלנו היא ליצור לולאה של דו-שיח. כלומר, המחשב יקבל משפט מהמשתמש, יבדוק אותו ויגיב בצורה מתאימה. לאחר מכן, יחכה המחשב למשפט חדש מהמשתמש, יגיב עליו וחוזר חלילה.

קיימות שתי דרכים ליצירת לולאה בתכנית ונציג את שתיהן.

אפשרות א' ליצירת לולאה:

האפשרות הראשונה היא להוסיף קריאה רקורסיבית לפונקציה `converse` לאחר שסיימנו לטפל בכל אחת מהאפשרויות במשפט התנאי.

שימו לב שבקריאה החוזרת אנחנו נעזרים בפונקציה `read` כדי לקבל מהמשתמש משפט חדש. המשפט הזה יוכנס לפרמטר `text` והביצוע המחודש של הפונקציה יתייחס למשפט החדש הזה.

```
(define (converse text)
  (cond
    [(contains? family text) (reply-to-family) (converse (read))]
    [(question-asked? text) (reply-to-question) (converse (read))]
    [else (random-answer) (converse (read))]))
```

אפשרות ב' ליצירת לולאה:

האפשרות השנייה היא להוסיף את הקריאה החוזרת לפונקציה `converse` רק בפונקציות העזר, כמו למשל, בסיום הפונקציה `reply-to-question` (תגובה למשפט שאלה).

שימו לב שגם הפעם בקריאה לפונקציה `converse` אנחנו נעזרים בפונקציה `read`.

```
(define (converse text)
  (cond
    [(contains? family text) (reply-to-family)]
    [(question-asked? text) (reply-to-question)]
    [else (random-answer)]))

(define (reply-to-question)
  (print '( I ask the questions here! ))
  (converse (read)))
```

מכאן ואילך נמשיך את הפיתוח של התכנית עם הלולאה מהסוג השני (אפשרות ב').

שלב 5 - אתחול הלולאה וסיום התכנית

לאחר שיצרנו לולאה של דו-שיח חוזר, עלינו לדאוג לאתחול של הלולאה וכן לסיום מסודר של התכנית.

א. כתבו פונקציה בשם `doctor`. הפונקציה (ללא פרמטרים) תדפיס משפט פתיחה (למשל, 'שלום'. על מה תרצה לשוחח?) ותקרא לפונקציה `converse`.

ב. כדי שנוכל לסיים את התכנית בצורה מסודרת עלינו לזהות משפטי "סיום". משפט סיום יכול להיות 'להתראות', 'עצור', או כל משפט אחר כרצונכם. תחילה נגדיר משתנה שיכיל את מילות הסיום:

```
(( stop ) ( bye ) ( goodbye ))
```

כעת עלינו להוסיף לפונקציה `converse` תנאי שיבדוק האם `text` מכיל מילות סיום ולהגיב בהתאם. שנו את הפונקציה `converse` כך שתבצע זאת.

שלב 6 - שיפור התגובה האקראית של המחשב

עד כה, אם המחשב לא זיהה משפט מיוחד הוא הגיב במשפט נייטראלי מתוך `NS`. כעת נשפר את התגובה של המחשב ובכל פעם שהוא יצטרך להגיב בצורה אקראית הוא יבחר בין שתי צורות של תגובה - תגובה מסוג אחד תהיה כמו קודם לכן (משפט אקראי מתוך `NS`), ואילו התגובה השנייה תהיה התייחסות למשפט קודם של המשתמש. כלומר, מדי פעם המחשב יגיב למשל כך - 'קודם אמרת שאתה אוהב בעלי חיים'.

א. תחילה עלינו לדאוג לכך שהמשפטים הקודמים של המשתמש ישמרו. נעשה זאת בעזרת **פרמטר חדש** שיקרא בשם `history`. הוא יכיל רשימה של כל המשפטים הקודמים שסיימונו לטפל בהם (או חלק מהמשפטים האלה, כרצונכם).

עברו על כל הפונקציות שכתבנו עד כה וסמנו באילו פונקציות יש צורך להכיר את הפרמטר .history.

```
(define (doctor)
  (print '( Hi, what would you like to talk about ? ) )
  (converse (read)))
```

```
(define (converse text)
  (cond
    [(contains? family text) (reply-to-family)]
    [(question-asked? text) (reply-to-question)]
    [else (random-answer)]))
```

```
(define (reply-to-family)
  (print '( Please tell me more about your family ) )
  (converse (read)))
```

```
(define (reply-to-question)
  (print '( I ask the questions here! ) )
  (converse (read)))
```

```
(define (contains? L1 L2)
  ( ..... ))
```

```
(define (question-asked? L)
  ( ..... ))
```

לאחר שסימנתם את הפונקציות, הוסיפו את השינויים הדרושים לתכנית כך שנוכל לטפל במשפטים קודמים.

ב. כתבו מחדש את הפונקציה random-answer לפי ההנחיות הבאות:
המחשב יבחר באקראי בין שתי צורות של תגובה -
תגובה מסוג אחד תהיה כמו קודם לכן (משפט אקראי מתוך NS),
ואילו התגובה השנייה תהיה התייחסות למשפט קודם של המשתמש.