

# תרגיל 1: היכרות עם Squeak

## הוראות כלליות

1. לפני הרצת קוד, שמרו את השינויים שעשיתם בקובץ נפרד על-ידי fileout.
2. במידה וחלקי קוד נמחקים לכם מה-workspace (כולל מחלקות או מתודות שנמחקות), אתם יכולים לשחזר את הקוד מקובץ log. שנמצא בתיקייה: `\Contents\Resources`
3. מטרת התרגיל: היכרות קצרה ובסיסית עם התכנות ב-Squeak. תרגיל זה אמור להקל בצורה משמעותית על ההתמודדות עם התרגילים הבאים.
4. בכדי להימנע מטעויות, אנא עיינו ברשימת ה-FAQ המתפרסמת באתר באופן שוטף.
5. את תרגיל זה עליכם לממש ב-Squeak 5.1, אותו ניתן להוריד מהאתר הרשמי.
6. אחראי על התרגיל: אריק. שאלות על התרגיל יש לשלוח למייל [eric.oop.course@gmail.com](mailto:eric.oop.course@gmail.com) עם הנושא: "236703 HW1".
7. מועד הגשה: 14/11/2018 בשעה 23:59.
8. הגשה בזוגות בלבד.

## הקדמה

בתרגיל בית זה נעסוק במימוש של פולינומים ב-Squeak, ונבחן שני סוגים של מימושים. הפולינומים אותם נבחן יהיו פולינומים במשתנה אחד, אינסופיים ולא מוגבלים בגודלם.

המקדמים בפולינום יהיו מספרים שלמים בלבד (Integer), והמעריכים בפולינום יהיו מספרים שלמים בלבד (Integer) הגדולים שווים לאפס. דרישה זו תקפה לשני החלקים!

## שימו לב –

1. על המחלקות בתרגיל בית זה להשתייך לקטגוריה חדשה בשם OOP1.
2. בתרגיל בית זה, זריקת חריגות מתבצעת על ידי: `.self error: 'invalid input'`.

## חלק א' – המחלקות Monom ו-Polynomial

בחלק זה נגדיר שתי מחלקות: האחת Monom, המייצגת פולינום בעל איבר בודד (לדוגמה:  $42x^2, -x, -8x^{14}$ ), והשנייה Polynomial המייצגת פולינום בעל מספר בלתי-מוגבל של מונומים (לדוגמה:  $6x^2 + 5x - 9x^3 - 2$ ).

## המחלקה Monom

instance variables:

1. `exp` מחזיק את המעריך של המונום ( $x^{\text{exp}}$ ). מאותחל ל-0.
2. `coef` מחזיק את המקדם של המונום ( $\text{coef} \cdot x$ ). מאותחל ל-0.

## אין להגדיר שדות נוספים מעבר לשדות שהוגדרו לעיל!

### :instance methods

1. `exp`  
מחזירה את ערך המעריך.
2. `exp: anInteger`  
מציבה את המספר `anInteger` כמעריך החדש.  
במידה והערך לא מספר שלם או שהערך שלילי, תזרק המחרוזת "invalid input" על ידי שימוש במתודה `error`: (המוגדרת ב-Object).
3. `coef`  
מחזירה את ערך המקדם.
4. `coef: anInteger`  
מציבה את המספר `anInteger` כמקדם החדש.  
במידה והערך לא מספר שלם, תזרק חריגה.
5. `derivative`  
מחזירה מונם חדש שיהיה הנגזרת של מקבל ההודעה.

## המחלקה **Polynom**

מחלקה זו מייצגת פולינום. פולינום חדש מאותחל להיות פולינום האפס  $P(x) = 0$ .

### :instance variables

1. `monoms`  
אוסף (collection) של **אובייקטים** מטיפוס `Monom`. עליכם לבחור את האוסף המתאים ביותר מבין האוספים שלמדנו.  
שמורות שחייבות להתקיים במהלך החיים של מופע של פולינום:
  - אין לשמור שני מונמים שונים באוסף בעלי אותו המעריך.
  - אין לשמור מונם באוסף שערך המקדם שלו הוא אפס.

## אין להגדיר שדות נוספים מעבר לשדה `monoms` שהוגדר לעיל!

### :instance methods

1. `addMonom: aMonom`  
מתודה זו **משנה** את מקבל ההודעה ומחברת את הפולינום הנוכחי עם מונם.  
אם `aMonom` אינו מופע של המחלקה `Monom`, יש לזרוק חריגה.  
הפולינום לא אמור להיות מושפע משינויים עתידיים של `aMonom` לאחר ההוספה.

2. `multiplyByMonom: aMonom`

מתודה זו **משנה** את מקבל ההודעה ומכפילה את הפולינום הנוכחי במונום.  
אם `aMonom` אינו מופע של המחלקה `Monom`, יש **לזרוק חריגה**.  
הפולינום לא אמור להיות מושפע משינויים עתידיים של `aMonom` לאחר ההוספה.

3. `asDictionary`

מתודה זו מחזירה את הפולינום מיוצג בתור מילון (כלומר, חוזר מילון), כאשר המפתחות הם המעריכים של המונומים והערכים הם המקדמים שלהם בהתאמה. לדוגמא: עבור  $P(x) = x^4 + 2$ , יוחזר המילון:  $\{0 \rightarrow 2, 4 \rightarrow 1\}$ .  
**עבור פולינום ה-0, יש להחזיר מילון ריק.**

4. `derivative`

מתודה זו מחזירה **פולינום חדש** שיהיה הנגזרת של מקבל ההודעה.

5. `eval: anInteger`

מתודה זו מחזירה את השערוך של הפולינום בנקודה `anInteger`.  
במידה ו-`anInteger` אינו מספר שלם, יש **לזרוק חריגה**.

6. `add: aPolynom`

מתודה זו מחזירה **פולינום חדש** השווה לסכום של הפולינום הנוכחי ופולינום הקלט.  
במידה ו-`aPolynom` לא מופע של המחלקה `Polynom`, יש **לזרוק חריגה**.  
יש לשים לב ולגרום לכך שפולינום הסכום לא יושפע משינויים עתידיים של `aPolynom` או של מקבל ההודעה.

## חלק ב' – המחלקה `PolyStream`

מחלקה זו תייצג **פולינום** שניתן לבצע עליו מספר פעולות מתמטיות במהירות, בניגוד לפולינום שהוגדר על ידי המחלקה הקודמת, אך השערוך שלו יהיה מורכב יותר.  
פולינום חדש מאותחל להיות פולינום האפס  $P(x) = 0$ .  
מחלקה זו **לא תשתמש** במחלקות `Monom` ו-`Polynom`.

ייתכן שבחלק זה תצטרכו להעתיק `Collection` בצורה עמוקה (`Deep Copy`) – כלומר המבנה יועתק וגם הנתונים בו יועתקו. כדי לעשות זאת ב-`Squeak` משתמשים בהודעה `deepCopy`.  
דוגמת שימוש: `myCollection := aCollection deepCopy`. נושא זה יועבר בהמשך באופן מסודר בתרגול המתקדם של `Squeak`.

## המחלקה `PolyStream`

### `:instance variables`

1. `block`

שדה המחזיק בלוק כלשהו (**טיפוסו יהיה `BlockClosure`**). תוכנו נתון לבחירתכם, קבעו אותו בחוכמה.  
באמצעות שדה זה עליכם לייצג את הפולינום.  
**רמז:** אפשר להשתמש בבלוק המקבל מספר ארגומנטים שונים (עד 3 ארגומנטים). ניתן לעשות זאת בצורה הבאה:

```
b := [ :first :second :third | "code goes here" ].  
b value:1 value:2 value:42.
```

## אין להגדיר שדות נוספים מעבר לשדה block שהוגדר לעיל!

### instance methods:

1. **addCoef: coef withExp: exp**  
מתודה זו מוסיפה לפולינום הנוכחי מונם חדש -  $coef * x^{exp}$ .  
אם coef או exp לא מספרים שלמים, או ש exp שלילי, יש לזרוק חריגה.  
**סיבוכיות הזמן הנדרשת מפעולה זו היא  $O(1)$ .**
2. **block**  
מתודה זו תחזיר את השדה block.
3. **add: aPolyStream**  
מתודה זו **משנה** את הפולינום הקיים לפולינום השווה לסכום של הפולינום הנוכחי ופולינום הקלט.  
במידה ו-aPolyStream לא מופע של המחלקה PolyStream, יש לזרוק חריגה.  
**ניתן להניח ש-aPolyStream לא ישתנה לאחר המתודה הזו.**  
**סיבוכיות הזמן הנדרשת מפעולה זו היא  $O(1)$ .**
4. **substitute: anInteger**  
מתודה זו **משנה** את הפולינום הקיים לפולינום בו מציבים  $x * anInteger$  במקום  $x$ . למשל הקריאה `substitute: 4` תשנה את הפולינום ל- $f(4x)$ .  
במידה ו-anInteger אינו מספר שלם או שווה ל-0 יש לזרוק חריגה.  
**סיבוכיות הזמן הנדרשת מפעולה זו היא  $O(1)$ .**
5. **multiplyBy: anInteger**  
מתודה זו **משנה** את הפולינום הקיים לפולינום השווה למכפלת הפולינום הנוכחי ב-anInteger.  
במידה ו-anInteger אינו מספר שלם יש לזרוק חריגה.  
**סיבוכיות הזמן הנדרשת מפעולה זו היא  $O(1)$ .**
6. **filter: aSet**  
מתודה זו מקבלת **סט של מספרים שלמים** ותשנה את הפולינום הקיים **(בעת הקריאה בלבד!)** לפולינום בו אין חזקות המופיעות ב-aSet.  
המתודה תשנה את הפולינום הנוכחי **בלבד** ולא תשפיע על שינויים עתידיים (לאחר הקריאה ניתן להוסיף מונומים עם מעריכים שנמחקו).  
ניתן להניח שיועבר סט ושאיבריו יהיו מספרים שלמים (אין צורך לבדוק).  
כמו כן, ניתן להניח ש-aSet מגודל קבוע וידוע (כלומר גודלו הוא  $O(1)$ ).  
יש לוודא שהפולינום אינו מושפע משינויים עתידיים של aSet לאחר הקריאה.  
**סיבוכיות הזמן הנדרשת מפעולה זו היא  $O(1)$ .**
7. **intersectionWith: aPolyStream withinRange: anInteger willUpdate: aBoolean**  
מתודה זו מקבלת מופע של PolyStream וטווח, ומחזירה **בלוק** "שינחש" בכל שערך שלו את קואורדינטת ה-x של נקודת החיתוך בין מקבל ההודעה והארגומנט.  
אם הניחוש יחרוג **ממש** מהטווח הנתון בערך מוחלט (כלומר אם הניחוש  $x$  מקיים  $|x| >$  **יש לזרוק חריגה.** (anInteger)

אם הניחוש נכון יש להחזיר את הניחוש ואם לא יש להחזיר את המחרוזת "wait for it..." בלבד, ללא ירידות שורה. הניחוש ההתחלתי יהיה  $x = 0$  ובכל שערך יש לקדם את הניחוש (יש לעבור על כל הניחושים השלמים, החיוביים והשליליים בטווח, למשל עבור טווח 2 יש לעבור על  $-2, -1, 0, 1$ ). הערך המוחזר צריך להיות הערך קרוב ביותר לאפס. אם יש שני ערכים מתאימים (למשל 1 ו-1-) יש להחזיר את אחד הערכים באופן שרירותי. ניתן להניח ש-anInteger יהיה מספר חיובי.

אם הערך של aBoolean הוא true, על הבלוק המוחזר להתחשב בשינויים לפולינום המתרחשים לאחר הקריאה לפונקציה. אם הוא false, יש להתעלם משינויים עתידיים.

ניתן להניח שהבלוק לא ישוערך לאחר שנמצא ערך ה-x שבו יש חיתוך.

במידה ו-aPolyStream אינו מופע של המחלקה PolyStream או ש-anInteger אינו מספר שלם יש לזרוק חריגה.

דוגמאות: בהינתן הפולינומים  $P_1(x) = -x^2 + 6$ ,  $P_2(x) = -x$

Workspace	Transcript
<pre>  b p1 p2    b := p1 intersectionWith: p2 withinRange: 2. Transcript show: (b value); cr. Transcript show: (b value); cr. : Transcript show: (b value); cr.  b := p1 intersectionWith: p2 withinRange: 1 Transcript show: (b value); cr. : Transcript show: (b value); cr.</pre>	<pre>wait for it... wait for it... : -2  wait for it... : (empty. throws "invalid input")</pre>

רמז לפתרון:

האלגוריתם לניחוש הבא לא עובד: ננחש 0, אח"כ  $-1, -2$  וכך הלאה. נניח שנקודות החיתוך הן  $x = 1$  ו- $x = -5$ , הבלוק יחזיר את  $-5$  כשהוא צריך להחזיר את 1.

סיבוכיות הזמן הנדרשת מפעולה זו היא  $O(1)$ . שימו לב שמדובר בסיבוכיות הזמן של

המתודה ולא של מציאת הפתרון ע"י שערך הבלוק.

8. eval: anInteger

מתודה זו תחזיר את השערך של הפולינום בנקודה anInteger.

במידה ו-anInteger אינו מספר שלם, יש לזרוק חריגה.

מפעולה זו אין דרישה על סיבוכיות הזמן.

בתרגיל זה אין להוסיף שדות, מלבד אלו שצוינו במפורש לאף אחת מהמחלקות! באופן כללי, אין כמובן סיבה לחסוך בשימוש בשדות בסקוויק.

## מאיפה מתחילים?

וודאו שעברתם על [המדריך לסביבת Squeak](#) ואתם מבינים כיצד להשתמש בסביבת העבודה. חזרו על שני התרגולים הראשונים. מומלץ מאוד לממש את התרגיל לפי סדר המחלקות שהוצגו. לאחר שמימשתם את חלק א', וודאו שאתם מצליחים להגדיר ולהשתמש בפולינומים. לאחר מכן, קראו את כל הדרישות בכל אחת מהמתודות בחלק ב', תכננו היטב את המימוש וכתבו אותו.

## טיפים, מגבלות וסייגים

- "פולינום בלתי מוגבל" – הכוונה לפולינום שאינו מוגבל ע"י התכנית שכתבתם. היות ומדובר בתכנית שרצה על המחשב (ולא באלגוריתם תיאורטי) ברור כי באופן מעשי קיימת מגבלה המוכתבת ע"י שפת התכנות, כמות הזיכרון וכו'.
- מהירות ביצוע אינה נושא מרכזי בתרגילי הבית בקורס תכנות מונחה עצמים. לכן בכל מקרה של התלבטות בין פשטות לבין ביצועים, העדיפו את המימוש הפשוט. לא תתקבלנה טענות בסגנון: "העדפתי מהירות ולכן הקוד יצא קצת פחות ברור".
- Squeak היא שפה שבה כל מתודה לא צריכה לקחת יותר מכמה שורות. הימנעו משכפול קוד והשתמשו בקוד שכבר כתבתם כדי לחסוך לכם עבודה. תכנון נכון של הקוד והוספת מתודות-עזר במקרה הצורך יקטינו משמעותית את כמות הקוד שתצטרכו לכתוב ולבדוק.
- הקפידו לתעד את הקוד שלכם. כל חלק לא טריוויאלי בקוד יש לתעד.
- מומלץ לעבור על שני התרגולים הראשונים בסקוויק ועל [המעבדה](#) לפני תחילת העבודה.
- כדי לבדוק שערך מסוים הוא מטיפוס היורש ממחלקה אחרת (או טיפוס של מחלקה מסוימת), יש להשתמש בהודעה `isKindOf: class`.
- מותר לכם להגדיר מתודות כאוות נפשכם. אין להוסיף שדות מעבר לאלה שהוגדרו בתרגיל.
- **אם התוכנית שלכם נתקעה לחצו על 'alt+' בכדי ליצור פסיקת משתמש אשר תפסיק את ביצוע התכנית ותאפשר לכם לראות היכן היא נתקעה.**
- **בלוקים ב-Squeak הם closures ויש להם תכונות מיוחדות. קראו שוב את התרגולים כדי לראות דוגמאות שימוש ספציפיות.**
- אופרטור השוואה (=) הדיפולטי של סקוויק מבצע השוואות by reference. נלמד עוד על האופרטור בהמשך הקורס, אך אם רוצים להשתמש בתרגיל ב-collections כגון Dictionary או Set המבצעים השוואות על האובייקטים, יש צורך לממש את אופרטור = וגם את המתודה hash עבור האובייקט שיעבור השוואה (אם טרם מומש). דוגמאות מימוש לשתי הפונקציות עבור המחלקה point2D המכילה שדות x ו-y:

```
= anObject  
  ^ (anObject class = Point2D) and: [(x = anObject getX) and: [y = anObject getY]]
```

```
hash  
  ^ (x + y) hash
```

### לנוחיותכם מספר שגיאות נפוצות:

- אי-ציון המילה self כשאובייקט שולח הודעה לעצמו (בניגוד ל-C++ שם ניתן להשמיט את המילה this).
- בעת חישוב בוליאני (ע"י הודעות and: , or: ) הביטוי בצד ימין חייב להיות בלוק שמכיל ביטוי בוליאני. דוגמה: `[x > y] and: [x > z]`.
- לא ניתן להציב בתוך פרמטר של מתודה, בתוך גוף המימוש שלה. למשל, אם x הוא פרמטר של המתודה, אזי ההצבה `x:=x*2` לא תשנה את ערכו של x.

- אם ברצונכם להשתמש ברשימה מקושרת (LinkedList), עליכם להשתמש באובייקט מסוג Link. כלומר, כל אובייקט שאתם רוצים להכניס לרשימה חייב לרשת מ-Link.

## הוראות הגשה

- בקשות לדחייה, מכל סיבה שהיא, יש לשלוח למתרגל האחראי על הקורס (נתן) במייל בלבד תחת הכותרת "236703 HW1".
- שימו לב שבקורס יש מדיניות איחורים, כלומר ניתן להגיש באיחור גם בלי אישור דחייה (עבור קנס בניקוד). פרטים נוספים באתר הקורס תחת General info.
- הגשת התרגיל תתבצע אלקטרונית בלבד (יש לשמור את אישור השליחה!).
- יש להגיש קובץ בשם OOP1\_<ID1>\_<ID2>.zip המכיל:
  - קובץ בשם readme.txt המכיל שם, מספר זיהוי וכתובת דואר אלקטרוני עבור כל אחד מהמגישים בפורמט הבא:
 

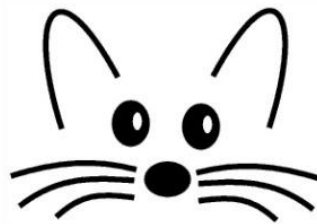
```
Name1 id1 email1
Name2 id2 email2
```
  - קובץ הקוד: OOP1.st. על הקובץ להכיל רק את מימוש המחלקות המוזכרות בתרגיל ומתודות לצורך פתרון התרגיל. אין להגיש קוד נוסף, למשל טסטים.
- נקודות יורדו למי שלא יעמוד בדרישות ההגשה (rar במקום zip, קבצים מיותרים נוספים, readme בעל שם לא נכון וכו').

## WHO WOULD WIN?

### A SIMPLE ALGEBRAIC CONCEPT

$5x^{12} - 2x^6 + x^5 - 198x + 1$	degree : 12
$x^4 - x^3 + x^2 - x + 1$	degree : 4
$56x^{23}$	degree : 23
$5x - 7$	degree : 1
$-8$	degree : 0

### ONE SQUEAKY BOY



בהצלחה!