אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2022

תרגיל בית מספר 1 - להגשה עד 06/03/2022 בשעה 23:55

קיראו בעיון את הנחיות העבודה וההגשה המופיעות באתר הקורס, תחת התיקייה assignments. חריגה מההנחיות תגרור ירידת ציון / פסילת התרגיל.

הגשה:

- תשובותיכם יוגשו בקובץ pdf ובקובץ py בהתאם להנחיות בכל שאלה.
- השתמשו בקובץ השלד skeleton1.py כבסיס לקובץ ה py אותו אתם מגישים.
לא לשכוח לשנות את שם הקובץ למספר ת"ז שלכם לפני ההגשה, עם סיומת py.

- בסה"כ מגישים שני קבצים בלבד. עבור סטודנטית שמספר ת"ז שלה הוא 012345678 הקבצים שיש להגיש הם hw1_012345678.pdf ו- hw1_012345678.py.
- בנוסף, את הפונקציה שבשאלה 6 מגישים בקובץ נפרד דרך רכיב **משוב עמיתים** במודל. יש לממש את הפונקציה בקובץ `max_even_seq.py` שבאתר. לפני ההגשה, יש לשנות את שם הקובץ לחמש הספרות האחרונות בת"ז שלכם (בדוגמא לעיל: 45678.py).

- הקפידו לענות על כל מה שנשאלתם.
- לפני ההגשה ודאו כי הרצתם את הפונקציה `test()` שבקובץ השלד אך זכרו כי היא מבצעת בדיקות בסיסיות בלבד וכי בתהליך הבדיקה הקוד ייבדק על פני מקרים מגוונים ומורכבים יותר

- בכל שאלה, אלא אם מצוין אחרת באופן מפורש, ניתן להניח כי הקלט תקין

- תשובות מילוליות והסברים צריכים להיות תמציתיים, קולעים וברורים.
להנחיה זו מטרה כפולה:
  1. על מנת שנוכל לבדוק את התרגילים שלכם בזמן סביר.
  2. כדי להרגיל אתכם להבעת טיעונים באופן מתומצת ויעיל, ללא פרטים חסרים מצד אחד אך ללא עודף בלתי הכרחי מצד שני. זוהי פרקטיקה חשובה במדעי המחשב.

דוגמה לפונקציה

בחלק מהשאלות בתרגיל זה הנכם מתבקשים להגיש תוכניות בפייתון. את התוכניות יהיה עליכם להגיש כפונקציות, נושא שילמד בהרחבה בשבוע השני של הסמסטר. אולם פתרון כל השאלות לא מחייב הבנה של נושא זה, ולכן אפשר וכדאי להתחיל לעבוד על התרגיל כבר עכשיו. כדי להקל עליכם, להלן דוגמה של פונקציה פשוטה שמקבלת מספר בודד כקלט ומחזירה כפלט באמצעות הפקודה return את ערכו של המספר כפול 2 .

נשים לב למספר דרישות בכתיבת פונקציה:

1. הגדרת הפונקציה תתחיל במילה def ולאחריה שם הפונקציה
2. לאחר שם הפונקציה יפורטו הקלטים אותם היא מקבלת, מופרדים ע"י פסיק.
3. יש להקפיד על הזחה: קוד גוף הפונקציה ייכתב Tab אחד פנימה ביחס לשורת def.

הפונקציה תחזיר פלט ע"י כתיבת המילה **return (שימו לב: לא המילה print אשר רק מדפיסה למסך)** ולאחריה הערך שיוחזר כאשר תופעל הפונקציה.

```
def double_my_num(x):
     return 2*x
```

דוגמאות להפעלת הפונקציה הנ"ל:

```
>>> z = double_my_num(5)  #won't work with print…
>>> z
10
>>> double_my_num(10)
20
>>> a = 30
>>> double_my_num(a)
60
```

דוגמה נוספת לפונקציה שמקבלת שני קלטים מספריים x,y ומחזירה כפלט באמצעות הפקודה return את מכפלתם:

```
def mult_nums(x, y):
     return x*y
```

דוגמאות להפעלת הפונקציה הנ"ל:

```
>>> y = mult_nums(5, 10)
>>> y
50
>>> mult_nums(10, 3)
30
>>> a = 2
>>> b = 6
>>> mult_nums(a, b)
12
```

שאלה 1 (שאלת חימום – לא להגשה)

כפי שראיתם בהרצאה, ישנן בפייתון פונקציות שמשויכות למחלקה מסויימת, למשל למחלקת המחרוזות (str).

באינטרפרטר IDLE, אם תכתבו "str." ותלחצו על המקש tab, תיפתח חלונית עם מגוון פונקציות המשויכות למחלקת המחרוזות. כמובן, אפשר למצוא תיעוד רב על פונקציות אלו ואחרות ברשת. כמו כן אפשר להשתמש בפונקציה help של פייתון. למשל הפקודה help(str.title) תציג הסבר קצר על הפונקציה str.title שראיתם בהרצאה.

הערה כללית:

פונקציות של מחלקות ניתן להפעיל בשני אופנים שקולים. אם נסמן ב- C את שם המחלקה (למשל המחלקה str), וב- c_obj אובייקט קונקרטי מהמחלקה C (למשל מחרוזת "abc"), אז שתי הדרכים הן:

- C.func(c_obj,…), כלומר הפרמטר הראשון הוא c_obj ואחריו יתר פרמטרים, אם דרושים.
- c_obj.func(…), כלומר האובייקט c_obj לא מופיע בתוך הסוגריים אלא לפני שם הפונקציה.

להלן הדגמה על המחלקה str:

```
>>> course_name = "introduction to computer science"
>>> str.title(course_name)
'Introduction To Computer Science'
>>> course_name.title()
'Introduction To Computer Science'
```

מצאו שלוש פונקציות הקיימות במחלקה str שאינן קיימות במחלקה list, ובדקו מהו הפלט שלהן על המחרוזת "xyzw".

כעת, מצאו שלוש פונקציות הקיימות במחלקה list שאינן קיימות במחלקה str. בדקו אותן באופן דומה על הרשימה ['x', 'y', 'z', 'w'].

הערה: המושגים "מחלקה" ו"אובייקט" יוסברו יותר לעומק בהמשך הקורס

שאלה 2

בכיתה ראיתם קוד בפייתון לחישוב ספרת ביקורת בתעודת זהות.

האלגוריתם לחישוב ספרת ביקורת בת״ז ישראלית מתואר בקישור הזה. להלן הקוד שראיתם:

```
def control_digit(ID):
    """ compute the check digit in an Israeli ID number,
        given as a string of 8 digits
    """

    assert isinstance(ID, str)
    assert len(ID) == 8

    total = 0
    for i in range(8):
        val = int(ID[i]) # converts char to int
        if i%2 == 0:     # even index (0,2,4,6)
            total += val
        else:            # odd index (1,3,5,7)
            if val < 5:
                total += 2*val
            else:
                total += ((2*val)%10) + 1 # sum of digits in 2*val
                                          # 'tens' digit must be 1

    total = total%10             # 'ones' (rightmost) digit
    check_digit = (10-total)%10  # the complement modulo 10 of total
                                 # for example 42->8, 30->0

    return str(check_digit)
```

א. הריצו את הפונקציה על קלט לא תקין משני סוגים: 1) טיפוס שאינו מחרוזת, 2) מחרוזת באורך שונה מ- 8. צפו בהודעת השגיאה המתקבלת. הסבירו בקובץ ה-pdf בקצרה את מהות השגיאה.

ב. הוסיפו לקובץ ה pdf שתי טבלאות מעקב אחר המשתנים בתוכנית המופיעה מעלה, טבלה עבור כל אחד משני הקלטים הבאים:

a. "87654321" (כלומר הרצת הפקודה control_digit("87654321")).

b. מספר תעודת הזהות האישי שלכם.

הטבלה תיראה כך (בעמוד הבא):

iteration	i	ID[i]	val	total
1				
2				
…				
8				

שימו לב: בכל שורה יש לרשום את ערכי המשתנים בסוף האיטרציה הרלוונטית. למשל בשורה הראשונה (iteration 1) יש לרשום את ערכי המשתנים ברגע סיום האיטרציה הראשונה של לולאת ה- for. לפיכך בשורה 8 יופיעו ערכי המשתנים בסיום הלולאה ("רגע לפני" ביצוע הפקודה שמופיעה אחרי הלולאה).

ראו דוגמה בקובץ סיכום תרגול מספר 1 באתר הקורס. אין צורך להסביר כיצד הפונקציה פועלת.

שאלה 3

נדון בבעיה החישובית הבאה (שראינו בתרגול): בהינתן מספר שלם חיובי num, נרצה לדעת כמה פעמים מופיעה בו הספרה 0. למשל עבור הקלט 10030 הפלט המתאים הוא 3.

מטרתנו בשאלה היא להשוות את זמני הריצה של שלושה פתרונות אפשריים לבעייה זו (הערה: אנו נדון בבעייה הנ"ל ובשלושת הפתרונות הללו גם בתרגול הראשון/שני, אבל אפשר להתחיל לפתור את השאלה כבר לאחר התרגול הראשון). לפניכם שלוש פונקציות הפותרות את הבעיה:

פתרון ראשון:

```
def zeros(num):#1st solution
    m = num
    cnt = 0
    while m > 0:
        if m % 10 == 0:
            cnt = cnt + 1
        m = m // 10
    return cnt
```

פתרון שני:

```
def zeros2(num):#2nd solution
    cnt = 0
    snum = str(num)  #num as a string
    for digit in snum:
        if digit == "0":
            cnt = cnt + 1
    return cnt
```

פתרון שלישי:

```
def zeros3(num):#3rd solution
    cnt = str.count(str(num), "0")
    return cnt
```

שלושת הפונקציות מחזירות את התשובה כפלט ולכן נוכל להדפיס את הפתרון (למשל, של הפונקציה הראשונה) ע"י הפקודות:

```
num = 2**127
result = zeros(num)
print(num, "has", result, "zeros")
```

כדי למדוד זמן ריצה של פקודה או סדרת פקודות, נשתמש במעין "סטופר":

- נוסיף בראש התוכנית שלנו את הפקודה `import time`
- נוסיף מייד לפני קטע הקוד שאת זמן הריצה שלו ברצוננו למדוד את הפקודה:

```
t0 = time.perf_counter()
```

- נוסיף מייד לאחר קטע הקוד הנ"ל את הפקודה:

```
t1 = time.perf_counter()
```

- זמן הריצה של קטע הקוד הוא ההפרש `t1-t0`. נוח להציגו למשל כך:

```
print("Running time: ", t1-t0, "sec")
```

הסבר קצר: time היא מחלקה של פייתון המאפשרת ביצוע פקודות שונות הקשורות לזמנים. הפקודה import הכרחית על מנת להשתמש במחלקה (היא "מיבאת" אותה. ניתקל במהלך הקורס בדוגמאות רבות ל"ייבוא" של מחלקות).
למידע נוסף על המחלקה time: https://docs.python.org/3/library/time.html

א. מדדו את זמן הריצה של 2 הפתרונות הראשונים עבור המספרים: 2**100, 2**250, 2**600, 2**1400.
(תזכורת: האופרטור ** הוא אופרטור החזקה, כלומר x**y משוערך ל-x בחזקת y). ציינו מה היו זמני הריצה בטבלה שבה תהיה עמודה לכל אחד מהקלטים הנ"ל, וכן שורה עבור כל פתרון. הסבירו בקצרה את התוצאות (ובפרט התייחסו לקצב הגידול כתלות בגודל הקלט). ניתן, אם רוצים, להציג את התוצאות בגרף על מנת להקל על ההסבר.
שימו לב: על המדידה למדוד את זמן הריצה של הקריאה לפונקציה בלבד, ובפרט אין למדוד את הזמן של פקודות נוספות כגון הדפסת הפלט.

ב. פונקציות מובנות של פייתון, כמו למשל str.count, ממומשות בד"כ באופן יעיל למדיי, לעיתים אף באמצעות אלגוריתמים מסובכים יחסית. חיזרו על סעיף א' עבור הפתרון השלישי. מבלי להיכנס לפרטי המימוש של str.count, האם היא אכן יעילה יותר מבחינת זמן ריצה, בהשוואה לשני הפתרונות הראשונים?

ג. עבור קלטים בעלי מספר ספרות דומה, האם יש לפלט עצמו, כלומר למספר האפסים בקלט, השפעה כלשהי על זמן הריצה של כל אחד מהפתרונות? ביחרו קלטים מתאימים לבדיקת הסוגייה, ציינו מהם הקלטים בהם השתמשתם, הראו את תוצאות המדידות, והסבירו מה היא מסקנתכם.

ד. להלן לולאה פשוטה:

```
num = 2**1000
cnt = 0
for i in range(num):
     cnt = cnt + 1
```

תנו הערכה גסה לזמן שיקח ללולאה להסתיים. ציינו כל הנחה עליה התבססתם בהערכתכם. איך אתם מסבירים זאת, לאור העובדה שבסעיף א' לולאת ה- for של הפתרון השני רצה בזמן קצר באופן משמעותי?

שאלה 4

בשאלה זו נעבוד על ניתוח בסיסי של מחרוזות. בשאלה חמישה סעיפים, ובכל סעיף יש לממש פונקציה אחת. בכל הסעיפים הקלט לפונקציה מכיל את המחרוזת `text` ולעתים קלטים נוספים.

אם לא נאמר אחרת, ניתן להניח כי משתנה הקלט text מכילה ספרות (0 עד 9), אותיות קטנות באנגלית (a, b, c וכו') רווחים ונקודות בלבד.

שימו לב – בכל אחד מהסעיפים יתכן כי `text` היא המחרוזת הריקה. ודאו כי הפתרון שלכם מטפל גם במקרה זה.

סעיף א'

ממשו את הפונקציה `replace(text, alphabet, new_alphabet)`. הפונקציה תקבל כקלט שלוש מחרוזות כאשר נתון כי המחרוזת `alphabet` לא מכילה חזרות וכן כי אורך המחרוזות `alphabet` ו-`new_alphabet` זהה. הפונקציה תחזיר מחרוזת חדשה באופן הבא: לכל תו במחרוזת `text`, אם התו מופיע ב-`alphabet` באינדקס `i`, הוא יוחלף בתו שמופיע במחרוזת `new_alphabet` באינדקס `i`. אם התו לא מופיע ב-`alphabet`, הוא ישאר כמו שהוא.

הנחיה: אין להשתמש במתודה `replace` של המחלקה `str`.

דוגמאות הרצה:

```
>>> replace("hello world", "abcde fghijkl", "1234567890xyz")
'95zzo6worz4'
>>> replace("python", "abc", "xyz")
'python'
>>> replace("string", "stg", "xxx")
'xxrinx'
```

סעיף ב'

פלינדרום היא מחרוזת הזהה להיפוך שלה. לדוגמא, המילה radar היא פלינדרום. נגדיר **פלינדרום מורחב** להיות מחרוזת הזהה להיפוך שלה, ללא התחשבות ברווחים וסימני פיסוק (במקרה שלנו – נקודות בלבד). לדוגמא, המחרוזת race car איננה פלינדרום (במובן שהגדרנו בתחילת הסעיף), אך היא כן פלינדרום מורחב.

ממשו את הפונקציה `is_pal(text)` המקבלת מחרוזת `text` כקלט. הפונקציה תחזיר `True` אם `text` היא פלינדרום מורחב ו-`False` אחרת. שימו לב כי יש להחזיר ערך מטיפוס `bool` (ולא למשל מחרוזת).

דוגמאות הרצה:

```
>>> is_pal("go hang a salami. im a lasagna hog")
True
>>> is_pal("radar")
True
>>> is_pal("hello lle")
False
```

סעיף ג'

ממשו את הפונקציה `num_different_letters(text)`. הפונקציה תחזיר כפלט את מספר האותיות (a, b, c וכו') השונות אשר מופיעות במחרוזת. שימו לב לא לספור גם ספרות (0 עד 9) או סימני פיסוק שעשויים להופיע.

רמז: חשבו איך להשתמש במשתנה `chars` אשר ניתן לכם בקובץ השלד.

דוגמאות הרצה:

```
>>> num_different_letters("aa bb cccc dd ee fghijklmnopqrstuvwxyz")
26
>>> num_different_letters("aaa98765432100000000")
1
```

סעיף ג' (המשך)

הסבירו בקובץ ה-pdf כיצד הייתם משנים את הפתרון שלכם כדי לבדוק כמה אותיות (a, b, c וכו') ספרות (0 עד 9) וסימני פיסוק שונים מופיעים במחרוזת.

סעיף ד'

ממשו את הפונקציה `most_frequent(text)`. הפונקציה תחזיר את התו שמספר המופעים שלו במחרוזת הוא הגדול ביותר. ניתן להניח כי אין שני תווים במחרוזת שמספר המופעים שלהם הוא מקסימלי וכן כי `text` איננה מחרוזת ריקה.

הנחיה: אין להשתמש במתודה `count` של המחלקה `str`.

```
>>> most_frequent('abcda')
'a'
>>> most_frequent('a b c d e')
' '
>>> most_frequent('abcdee')
'e'
```

סעיף ה'

ממשו את הפונקציה `kth_order(text, k)`. הפונקציה תקבל כקלט מחרוזת `text` ושלם חיובי `k`. הפונקציה תחזיר את התו ה-`k` בשכיחותו. כלומר, עבור `k=1` הפונקציה תחזיר את התו שמופיע הכי הרבה פעמים במחרוזת, עבור `k=2` היא תחזיר את התו השני בשכיחותו וכן הלאה). ניתן להניח כי מספר התווים השונים במחרוזת הוא לפחות `k` וכן כי אין שני תווים שמספר המופעים שלהם זהה.

הנחיה: אין להשתמש במתודה `count` של המחלקה `str`.

רמז: העזרו בפונקציה מהסעיף הקודם.

```
>>> kth_order('aaaabbbccd', 2)
'b'
>>> kth_order('abcdeabcdabcaba', 1)
'a'
```

שאלה 5

בשאלה זו נממש "מחשבון מחרוזות" בסיסי. הפונקציה `calc` תקבל כקלט **מחרוזת** `expression` המכילה ביטוי מהצורה הבאה:

$$a_0 \oplus_0 a_1 \oplus_1 a_2 \oplus_2 \cdots$$

כאשר $\oplus_i$ היא אחת מבין הפעולות: $+, -, *$ (כלומר: חיבור, חיסור או כפל) וכל a_j מתחיל ונגמר בתו ' (גרש בודד). כל איבר בביטוי a_j מקיים את התנאים הבאים:

- אם הוא מופיע אחרי סימן חיסור ($-$) אזי a_j מכיל תו בודד
- אם הוא מופיע אחרי סימן כפל ($*$) אזי a_j מכיל מספר שלם חיובי
- אחרת, הוא יכיל מספר חיובי של תווים (אחד או יותר)

שערוך הביטוי `expression` יהיה התוצאה של הפעלת הפעולות הבאות על תתי הביטויים לפי **סדר הופעתם משמאל לימין** באופן הבא (שימו לב כי סדר הפעולות הוא משמאל לימין ולא ע"פ סדר הפעולות המתמטי הסטנדרטי):

- נתחיל מהמחרוזת a_0
- אם הפעולה הבאה היא פעולת חיבור, נשרשר את המחרוזת הבאה בסוף המחרוזת שאנחנו מחזיקים
- אם הפעולה הבאה היא פעולת חיסור, נשמיט את כל המופעים של התו שמופיע אחרי הפעולה מהמחרוזת שאנחנו מחזיקים
- אם הפעולה הבאה היא פעולת כפל, נכפיל את המחרוזת שאנחנו מחזיקים כמספר שמופיע לאחר הפעולה
- בסוף, נחזיר את המחרוזת שקיבלנו

כמקרה קצה, אם הביטוי `expression` שקיבלנו כקלט ריק, נחזיר את המחרוזת הריקה.

לדוגמא, הביטוי "`'a'+'b'*'2'+'c'-'b'`" ישוערך למחרוזת "`aac`" לפי הלוגיקה הבאה: נתחיל מהמחרוזת "`a`", נשרשר לה את "`b`" ונקבל את "`ab`", נשכפל את התוצאה פעמיים ונקבל "`abab`", נוסיף לתוצאה "`c`" ונקבל "`ababc`" ולבסוף נסיר את כל המופעים של "`b`" ונקבל "`aac`".

שימו לב: יש בפייתון מספר דרכים (שקולות) ליצור מחרוזות. בתרגיל זה נשתמש במרכאות כפולות "`...`" כיוון שאלו יאפשרו לנו להשתמש במרכאות יחידות '`...`' בגוף המחרוזת ללא שימוש בתווים מיוחדים. אתם מוזמנים לקרוא כאן על דרכים נוספות ליצור מחרוזת ולהתנסות בהבדלים (הקטנים) ביניהם בעצמכם.

דוגמאות הרצה:

```
>>> calc("'123321'*'2'")
"123321123321"
>>> calc("'Hi there '*'3'+'you2'")
"Hi there Hi there Hi there you2"
>>> calc("'abc'+'abc'*'2'-'c'")
'abababab'
>>> calc("'abc'+'def'*'2'-'x'")
'abcdefabcdef'
```

הנחיות:

- ניתן להניח כי אין רווחים בין הפעולות וה- a_i
- פלט הפונקציה צריך להיות מטיפוס `str`
- ניתן להניח כי המחרוזת `expression` תקינה (מקיימת את הפורמט שמוגדר בשאלה)
- אין להשתמש בספריות חיצוניות או בפקודות שיערוך מובנות (כמו `eval`)
- כדאי להשתמש בפונקציה `split` של המחלקה `str`. נסו להבין כיצד היא פועלת וכיצד היא יכולה לסייע לכם. ניתן לקרוא על הפונקציה ואופן השימוש בה על ידי חיפוש זריז בגוגל.

שאלה 6

שאלה זו תשמש אתכם לצורך תהליך של משוב עמיתים (peer review) שיתבצע במהלך הסמסטר.

מיומנויות הקשורות לתכנות לא מסתכמות רק בכתיבת קוד "שעובד", אלא כוללות גם קריאת קוד והבנתו, איתור שגיאות ותיקונן, בניית מערך טסטים מקיף, הערכת יעילות, תכנון מקדים של הפתרון, וכו'. **המוטיבציה למשוב עמיתים היא להפגיש אתכם עם מיומנויות אלו בשלב מוקדם של הלימודים, ולהיעזר לשם כך בחבריכם וחברותיכם לספסל הלימודים.**

בשלב זה של התהליך, אתם נדרשים לממש את הפונקציה שמפורטת להלן. כדי שתהליך משוב העמיתים יהיה מועיל ואף מהנה, אתם מתבקשים לעבוד על השאלה באופן עצמאי וללא עזרה מחברים וחברות לקורס או מסגל הקורס. על מנת לעודד זאת, **הניקוד להגשה של שאלה זו יינתן על עצם הגשת פתרון שנכתב עצמאית, אפילו אם אינו נכון לגמרי או אם יש בו בעיות מסוגים שונים**. במילים אחרות, אנחנו מבקשים שתכתבו ותגישו את הפתרון הטוב ביותר שביכולתכם בשלב זה על בסיס מה שלמדתם עד כה.

בשאלה זו נכתוב פונקציה שבהינתן מספר שלם $n \geq 0$ מחשבת מהו האורך המקסימלי של רצף ספרות זוגיות ב-n.
למשל, עבור $n = 23300247524689$ האורך המקסימלי של רצף ספרות זוגיות הוא 4 (ישנם שני רצפים שמתאימים לאורך זה: הרצף 0024 שמתחיל באינדקס 3 והרצף 2468 שמתחיל באינדקס 9).

הערות:

- במקרה שהמספר אינו מכיל ספרות זוגיות האורך המקסימלי הינו 0
- ניתן להניח כי הקלט תקין ואין צורך לבדוק זאת

דוגמת הרצה:

```
>>> max_even_seq(23300247524689)
4
```

ממשו את הפונקציה `max_even_seq(n)` שבקובץ השלד `max_even_seq.py` (ולא בקובץ השלד שבו נמצאות שאר שאלות הקוד) על פי ההנחיות לעיל. לאחר מכן, **שנו את שם הקובץ לחמש הספרות האחרונות בתעודת הזהות שלכם** והגישו אותו ברכיב משוב עמיתים במודל (ולא בתיבת ההגשה של תרגיל בית 1) כאשר כותרת ההגשה זהה לשם הקובץ. במהלך העבודה ניתן לערוך את הקובץ כרצונכם (למשל, כדי לבדוק את נכונות הקוד שלכם על קלטים שונים), אך על ההגשה הסופית לכלול אך ורק את המימוש של הפונקציה ללא קטעי קוד נוספים. כמו כן, כדי לאפשר אנונימיות בהמשך התהליך אין להוסיף להגשה הסופית פרטים מזהים מלבד אלה שהוזכרו לעיל.

סוף.