תרגיל בית 9

הנחיות כלליות:

- קראו בעיון את השאלות והקפידו שהתכניות שלכם פועלות בהתאם לנדרש.
- את התרגיל יש לפתור לבד!
- הקפידו על כללי ההגשה המפורסמים באתר. בפרט, יש להגיש את כל הפתרונות לשאלות יחד בקובץ ex9_012345678.py המצורף לתרגיל, לאחר החלפת הספרות 012345678 במספר ת.ז. שלכם, כל 9 הספרות כולל ספרת ביקורת.
- מועד אחרון להגשה: כמפורסם באתר.
- בדיקה עצמית: כדי לוודא את נכונותן ואת עמידותן של התוכניות לקלטים שגויים, בכל שאלה הריצו את תוכניתכם עם מגוון קלטים שונים, אלה שהופיעו כדוגמאות בתרגיל וקלטים נוספים עליהם חשבתם (וודאו כי הפלט נכון).
- אין לשנות את שמות הפונקציות והמשתנים שכבר מופיעים בקובץ השלד של התרגיל.
- היות ובדיקת התרגילים עשויה להיות אוטומטית, **יש להקפיד על פלטים מדויקים על פי הדוגמאות (עד לרמת הרווח).**
- אופן ביצוע התרגיל: שימו לב, בתרגיל זה עליכם להשלים את הקוד בקובץ המצורף.

שאלה 1

הערה: בתרגיל זה נממש מחלקה חדשה המייצגת מטריצות, בשם Matrix. את האינדקסים של המטריצות נסמן לפי "האינדוקס הפייתוני", שמתחיל את הספירה ב-0. עבור המטריצה הבאה $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ המספר 1 נמצא באינדקס (0,0), המספר 2 נמצא באינדקס (0,1) המספר 3 נמצא באינדקס (1,0) והמספר 4 נמצא באינדקס (1,1).

ממשו את המתודות הבאות לפי ההנחיות המצורפות.

מתודה	תיאור
__init__(self, lst)	מתודה זו תקבל כארגומנט רשימה של רשימות, lst, אשר מייצגת מטריצה. האיבר הראשון ברשימה ייצג את השורה הראשונה במטריצה, האיבר השני את השורה השנייה וכן הלאה. כל אובייקט מטיפוס מטריצה יחזיק שני attributes (שדות). הראשון (בשם lst) ישמור <u>העתק</u> של הרשימה שהתקבלה כקלט. כלומר, אם בבניית המטריצה M נמסרה הרשימה L (M=Matrix(L)), אזי M.lst ו-L יפנו אל אובייקטים שונים בזכרון. השדה השני, בשם dim, ישמור את מימדי המטריצה כ-tuple. שימו לב שאת מימדי המטריצה יש לחשב לפי הרשימה שהתקבלה. האיבר הראשון ב-tuple שמוחזק ב-dim יציין את מספר השורות במטריצה, והאיבר השני את מספר העמודות. ניתן להניח שהקלט (lst) המתקבל הוא אכן רשימה, המכילה לפחות רשימה אחת. כלומר, ניתן להניח שמספר השורות הוא לפחות 1, ומספר העמודות הוא לפחות 1). ניתן להניח שהקלט (lst) הוא רשימה של רשימות, כאשר איברי אותן תת רשימות הם מספרים (int או float). <u>יש לבדוק</u> שאורכי כל תת הרשימות זהים (כלומר שהשורות במטריצה בעלות אותו אורך). אם תנאי זה לא מתקיים, יש לזרוק (raise) שגיאה מסוג ValueError עם ההודעה: 'not all lines are of same length'
__repr__(self)	המתודה תחזיר ייצוג מחרוזתי של הרשימה המוחזקת בשדה lst. ראו דוגמא בהמשך.
get(self, i, j)	מתודה זו תקבל כארגומנטים שני מספרים (i,j) ותחזיר את הערך שנמצא במטריצה, באינדקס המתאים לאותם i,j (כלומר בשורה באינדקס i ובעמודה באינדקס j). המתודה תבדוק קודם כל האם זוג האינדקסים מייצג אינדקסים חוקיים במטריצה (ע"פ מימדי המטריצה). אם זוג האינדקסים אינו חוקי יש לזרוק שגיאה מסוג IndexError, עם ההערה: 'matrix index out of range' אחרת, המתודה תחזיר את הערך שנמצא במיקום המבוקש. ניתן להניח ש-i,j שמתקבלים כקלט הם שלמים אי-שליליים.
הערה: בכתיבת הקוד עבור המתודות הבאות, על-מנת לקבל ערך מתוך אובייקט מסוג Matrix עבור זוג אינדקסים, השתמשו במתודה get. אין להשתמש בשדה lst ישירות.	

transpose(self)	המתודה תחזיר אובייקט חדש מסוג Matrix (מבלי לשנות את האובייקט המקורי). המטריצה החדשה תהיה הגרסה המשוחלפת (transpose) של המטריצה המקורית.
__add__(self, other)	ממשו מתודה שתיישם את אופרטור החיבור (+) בין שני אובייקטים מסוג Matrix. המתודה תחזיר אובייקט חדש מסוג Matrix, שיהווה את הסכום של שתי המטריצות. המתודה תעשה זאת מבלי לשנות את המטריצות המקוריות. אם מימדי המטריצות שונים – יש לזרוק שגיאה מסוג ValueError, עם ההודעה: 'dims do not match' ניתן להניח שהאובייקט שמתקבל כקלט הוא מסוג Matrix.
__mul__(self, other)	ממשו מתודה שתיישם את אופרטור הכפל (*) בין שני אובייקטים מסוג Matrix. המתודה תחזיר אובייקט חדש מסוג Matrix, שיהווה את מכפלת שתי המטריצות, כפל איבר-איבר. כלומר, עבור שתי מטריצות A, B, אם נגדיר C=A*B, אזי C[i,j]=A[i,j]*B[i,j] לכל i,j בטווחים המתאימים. המתודה תעשה זאת מבלי לשנות את המטריצות המקוריות. שימו לב שהמתודה אינה מבצעת מכפלת מטריצות רגילה. אם מימדי המטריצות שונים – יש לזרוק שגיאה מסוג ValueError, עם ההודעה: 'dims do not match' ניתן להניח שהאובייקט שמתקבל כקלט הוא מסוג Matrix.
dot(self,other)	ממשו מתודה שתיישם מכפלת מטריצות **רגילה**. המתודה תחזיר אובייקט חדש מסוג Matrix, שיהווה את מכפלת שתי המטריצות (כפל מטריציוני). המתודה תעשה זאת מבלי לשנות את המטריצות המקוריות. יש לבדוק שמימדי המטריצות מתאימים לכפל מטריציוני (עבור המכפלה A.dot(B) יש לבדוק שמספר העמודות של A שווה למספר השורות של B). אם המימדים אינם מתאימים - יש לזרוק שגיאה מסוג ValueError, עם ההודעה: 'dims do not match' ניתן להניח שהאובייקט שמתקבל כקלט הוא מסוג Matrix. **שימו לב** שבתרגיל 3 כתבתם קוד אשר מיישם כפל מטריצות שכזה. השתמשו בקוד זה בכתיבת המתודה.

דוגמאות הרצה:

הרשימה [[1,0,2],[3,1,1],[4,5,2],[0,2,8]] למעשה מייצגת את המטריצה הבאה: $\begin{pmatrix} 1 & 0 & 2 \\ 3 & 1 & 1 \\ 4 & 5 & 2 \\ 0 & 2 & 8 \end{pmatrix}$

שמימדיה הם (4,3).

```
>>> a = Matrix([[1,0,2],[3,1,1],[4,5,2],[0,2]])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    raise ValueError('not all lines are of same length')
ValueError: not all lines are of same length
>>> A = Matrix([[1,0,2],[3,1,1],[4,5,2],[0,2,8]])
>>> print A
[[1, 0, 2], [3, 1, 1], [4, 5, 2], [0, 2, 8]]
>>> A.dim
(4, 3)
>>> A.lst
[[1, 0, 2], [3, 1, 1], [4, 5, 2], [0, 2, 8]]

>>> A.get(2,1)
5
>>> A.get(2,3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    raise IndexError('matrix index out of range')
IndexError: matrix index out of range

>>> B = A.transpose()
>>> print B
[[1, 3, 4, 0], [0, 1, 5, 2], [2, 1, 2, 8]]
>>> B.dim
(3, 4)

>>> C = Matrix([[1,3,4],[0,1,5],[2,1,2],[4,6,8]])
>>> D = A+C
>>> print D
[[2, 3, 6], [3, 2, 6], [6, 6, 4], [4, 8, 16]]
>>> D.dim
(4, 3)

>>> print A * C
[[1, 0, 8], [0, 1, 5], [8, 5, 4], [0, 12, 64]]
>>> A * B
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    raise ValueError('dims do not match')
ValueError: dims do not match


>>> A.dot(C)
Traceback (most recent call last):
```

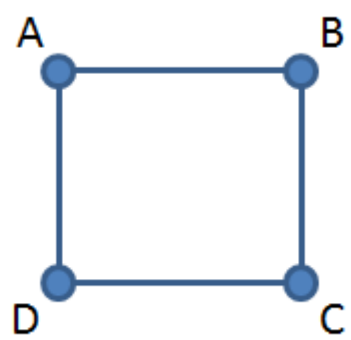
```
  File "<stdin>", line 1, in <module>
    raise ValueError('dims do not match')
ValueError: dims do not match
>>> E = A.dot(B)
>>> print E
[[5, 5, 8, 16], [5, 11, 19, 10], [8, 19, 45, 26], [16, 10, 26, 68]]
```

שאלה 2

בשאלה זו נכתוב מחלקות של מצולעים: מצולע כללי (Polygon), ריבוע (Square) ומשולש (Triangle).

מצולע יוגדר לפי אוסף קודקודים, כאשר כל קודקוד הוא נקודה במישור (Point). בשאלה זו ניתן להניח שהקודקודים המרכיבים כל מצולע ניתנים על-פי סדר הופעתם. אם הקלט של פוליגון מסוים הוא אוסף הנקודות (X,Y,Z,W,Q). הכוונה היא לפוליגון שקודקודיו הם X, Y, Z, W, Q, ושהצלעות המרכיבות אותו הן: XY, YZ, ZW, WQ, QX. כמו כן, ניתן להניח שמתוך הנקודות שמתקבלות אין 3 נקודות היושבות על ישר אחד.

למשל, עבור הריבוע באיור הנ"ל, סדר מתאים של הקודקודים שייתקבלו כקלט יכול להיות (A,B,C,D) או למשל (C,D,A,B) או אפילו (B,A,D,C). אבל לעומת זאת הסדר הבא אינו תקין (A,C,D,B).



לצורך הגדרת המחלקות הבאות **יש לעשות שימוש במחלקת Point שמוגדרת כבר בשלד הקוד, מבלי לערוך אותה** (מחלקה זו דומה לדוגמא שראינו בתירגול).

את הקוד יש לכתוב באופן שבו המחלקות Square ו-Triangle יורשות מהמחלקה Polygon. יש להקפיד על כתיבה יעילה כמה שאפשר – אין לכתוב פעמיים את אותו מימוש, אלא יש להסתמך על מנגנון הירושה. ייתכן ויש צורך לכתוב מתודה יותר מפעם אחת כדי להתאים אותה לכל אובייקט, אך זה אומר שהמתודה לא תכתב בדיוק אותו דבר.

<u>פירוט דרישות</u>

Polygon

מתודה	תיאור
__init__(self, points)	מתודה זו תקבל כארגומנט tuple המכיל אובייקטים מסוג Point, שמהווים את קודקודי הפוליגון. כל אובייקט מטיפוס Polygon יחזיק attribute (שדה) אחד בשם vertices, שישמור את אותו tuple שיתקבל כקלט. ניתן להניח שהקלט points הוא אכן tuple של איברים מסוג Point. כאמור לעיל ניתן להניח שהקודקודים מתקבלים בסדר תקין (שיוצר פוליגון) וכן שלא קיימות 3 נקודות שיושבות על אותו ישר. יש לבדוק שהתקבלו יותר משני קודקודים. אם התקבלו פחות מ-3 נקודות, יש לזרוק שגיאה מסוג ValueError עם ההערה הבאה: 'not enough vertices'
__repr__(self)	המתודה תחזיר את ייצוג המחרוזת של ה-tuple המחזיק את קודקודי המצולע (ראו דוגמה בהמשך). יש להסתמך על העובדה שגם ל-tuple וגם לאובייקט מסוג Point יש כבר מימוש לאופרטור __repr__.
shift(self,dx,dy)	מתודה זו תקבל כארגומנטים שני מספרים: dx, dy, והיא תזיז את הפוליגון כולו בערך dx לאורך ציר x, ובערך dy לאורך ציר y. כדי להזיז את הפוליגון יש להוסיף לקואורדינאטת x של כל אחד מהקודקודים את הערך dx, ולהוסיף לקואורדינאטת y של כל קודקוד את הערך dy. במימוש של מתודה זו השתמשו במתודה shift של המחלקה Point. ניתן להניח שהערכים dx, dy הם מסוג int או float.
circumference(self)	המתודה תחזיר את ההיקף של הפוליגון. כדי לחשב את ההיקף יש למעשה לסכום את המרחקים בין כל 2 קודקודים סמוכים. כלומר, עבור הפוליגון P=Polygon((A,B,C,D,E,F)), כדי לחשב את היקפו יש לסכום את המרחקים בין הקודקודים A ו-B, B ו-C, C ו-D, D ו-E, E ו-F, F ו-A. בכתיבת הקוד למתודה זו השתמשו במתודה distance המחשבת מרחק בין שתי נקודות כפי שהוגדרה במחלקה Point.

Square

מתודה	תיאור
__init__(self, points)	מתודה זו תקבל כארגומנט tuple המכיל אובייקטים מסוג Point, שמהווים את

	קודקודי הריבוע. בדומה, ל-Polygon, כל אובייקט מטיפוס Square יחזיק attribute בשם vertices, שישמור את אותו tuple שייתקבל כקלט. אך בנוסף, כל אובייקט מטיפוס Square יחזיק attribute בשם edge שישמור את אורך הצלע של הריבוע (בריבוע אורך כל הצלעות זהה. שימו לב שאורך הצלע לא מתקבל כקלט אלא צריך להיות מחושב לפי הקודקודים שניתנו). יש לבדוק שאכן מדובר בריבוע, כלומר: 1. התקבלו 4 נקודות בלבד. 2. אורכי כל הצלעות שווים. 3. אורכי שני אלכסוני הריבוע שווים. אם אחד מהסעיפים הנ"ל לא מתקיים, יש לזרוק שגיאה מסוג ValueError עם ההודעה: "the given vertices don't form a square" ניתן להניח שהקלט points הוא אכן tuple של איברים מסוג Point.
__repr__(self)	המתודה תחזיר מחרוזת הדומה למחרוזת שמחזירה __repr__ של המחלקה Polygon, אך בנוסף, יופיע בתחילת המחרוזת הקטע הבא: 'Square - '. כחלק מכתיבת הקוד של מתודה זו, יש להשתמש במתודה __repr__ של מחלקת Polygon.
shift(self,dx,dy)	מתודה זו תקבל כארגומנטים שני מספרים: dx, dy, והיא תזיז את הריבוע כולו בערך dx לאורך ציר x, ובערך dy לאורך ציר y. כדי להזיז את הריבוע יש להוסיף לקורדינטת x של כל אחד מהקודקודים את הערך dx, ולהוסיף לקורדינטת y של כל קודקוד את הערך dy. ניתן להניח שהערכים dx, dy הם מסוג int או float.
circumference(self)	המתודה תחזיר את ההיקף של הריבוע. בשונה מפוליגון כללי, את ההיקף של ריבוע קל יותר לחשב. את היקפו של הריבוע יש לחשב ע"י הכפלת אורך צלע הריבוע ב-4.
area(self)	המתודה תחזיר את השטח של הריבוע. שטח של ריבוע קל לחשב לפי אורך צלעו – השטח הוא ריבוע אורך הצלע.

Triangle

מתודה	תיאור
__init__(self, points)	מתודה זו תקבל כארגומנט tuple המכיל אובייקטים מסוג Point, שמהווים את קודקודי המשולש. בדומה, ל-Polygon, כל אובייקט מטיפוס Triangle יחזיק attribute בשם vertices, שישמור את אותו tuple שייתקבל כקלט.

	יש לבדוק שאכן התקבלו 3 נקודות בלבד. אם התקבל מספר אחר של נקודות, יש לזרוק שגיאה מסוג ValueError עם ההערה: 'triangle must have 3 vertices' ניתן להניח שהקלט points הוא אכן tuple של איברים מסוג Point.
__repr__(self)	המתודה תחזיר מחרוזת הדומה למחרוזת שמחזירה __repr__ של המחלקה Polygon, אך בנוסף, יופיע בתחילת המחרוזת הקטע הבא: 'Triangle - '. כחלק מכתיבת הקוד של מתודה זו, יש להשתמש במתודה __repr__ של מחלקת Polygon.
shift(self,dx,dy)	מתודה זו תקבל כארגומנטים שני מספרים: dx, dy, והיא תזיז את המשולש כולו בערך dx לאורך ציר x, ובערך dy לאורך ציר y. כדי להזיז את המשולש יש להוסיף לקואורדינאטת x של כל אחד מהקודקודים את הערך dx, ולהוסיף לקואורדינאטת y של כל קודקוד את הערך dy. ניתן להניח שהערכים dx, dy הם מסוג int או float.
circumference(self)	המתודה תחזיר את היקף המשולש. כדי לחשב את ההיקף יש למעשה לסכום את המרחקים בין כל 2 קודקודים סמוכים. כלומר, עבור המשולש T=Triangle((A,B,C)), כדי לחשב את היקפו יש לסכום את המרחקים בין הקודקודים A ו-B, B ו-C, C ו-A.
area(self)	המתודה תחזיר את שטח המשולש. שטח משולש ניתן לחשב לפי נוסחת הרון, באופן הבא: אם נסמן את מחצית מהיקף המשולש באות s, ונסמן את אורך כל אחת מ-3 הצלעות באותיות a,b,c, אזי: $area = \sqrt{s(s-a)(s-b)(s-c)}$

דוגמאות הרצה:

```
>>> a = Point(0,0)
>>> b = Point(0,2)
>>> c = Point(2,2)
>>> d = Point(2,0)
>>> e = Point(1,-1)
>>> P = Polygon((a,b))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    raise ValueError('not enough vertices')
ValueError: not enough vertices
>>> P = Polygon((a,b,c,d,e))
>>> P
((0, 0), (0, 2), (2, 2), (2, 0), (1, -1))
>>> P.shift(1,0)
>>> P
```

```
((1, 0), (1, 2), (3, 2), (3, 0), (2, -1))
>>> P.shift(0.5,-1)
>>> P
((1.5, -1), (1.5, 1), (3.5, 1), (3.5, -1), (2.5, -2))
>>> P.circumference()
8.82842712474619

>>> a = Point(0,0)
>>> b = Point(0,2)
>>> c = Point(2,2)
>>> d = Point(2,0)
>>> e = Point(-1,-1)

>>> S = Square((a,b,c))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    raise ValueError("the given vertices don't form a square")
ValueError: the given vertices don't form a square
>>> S = Square((b,c,d,e))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    raise ValueError("the given vertices don't form a square")
ValueError: the given vertices don't form a square

>>> S = Square((a,b,c,d))
>>> S
Square - ((0, 0), (0, 2), (2, 2), (2, 0))
>>> S.shift(2,2)
>>> S
Square - ((2, 2), (2, 4), (4, 4), (4, 2))
>>> S.circumference()
8.0
>>> S.area()
4.0

>>> a = Point(0,0)
>>> b = Point(0,2)
>>> c = Point(2,2)
>>> T = Triangle((a,b,c,d))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
      raise ValueError('triangle must have 3 vertices')
ValueError: triangle must have 3 vertices
>>> T = Triangle((a,b,c))
>>> T
Triangle - ((0, 0), (0, 2), (2, 2))
```

```
>>> T.shift(-1,-1)
>>> T
Triangle - ((-1, -1), (-1, 1), (1, 1))
>>> T.circumference()
6.82842712474619
>>> T.area()
1.9999999999999993
```

בהצלחה!