תרגיל 4 - רקורסיה

תאריך פרסום:	01/02
תאריך הגשה:	19/02 בשעה 23:59
מתרגלת אחראית:	רותם כהן
משקל תרגיל:	5 נקודות
מטרות העבודה:	רקורסיה בסיסית, עבודה עם פונקציות מעטפת, הרכבת אלגוריתם רקורסיבי.

הנחיות ספציפיות לתרגיל:

תרגיל זה עוסק בניתוח בעיות ובניית פתרונן .עליכם לממש את הפתרון באמצעות רקורסיה. אתם רשאים לממש פונקציות מעטפת. חלק גדול מהבעיות ניתן לממש בקלות באמצעות לולאה, ואת חלקן אף פתרנו כך בביתה, אך הדרישה מכם היא לממש באמצעות רקורסיה.
שימו לב – סעיף ג' בשאלה 4 הינו סעיף בונוס. פתרון נכון יעניק למטלה ניקוד נוסף בשווי 20% מהציון הסופי .

אלא אם נאמר אחרת:

- כל שימוש בפקודות השמורות for ו -while (בכל צורה שהיא – למשל list comprehension או min) יגרור את פסילת הפתרון.
- ניתן לממש פונקציות מעטפת ופונקציות עזר.
- שם פונקציית מעטפת צריך להיות תואם לשם הפונקציה שאתם מתבקשים לממש בסעיף. אין מגבלה על שם הפונקציה הרקורסיבית שנקראת מפונקציית המעטפת .
- אין להשתמש בפרמטרי ברירת מחדל בהגדרה של פונקציה.
- ניתן להניח שהקלט תקין.

שאלה 1

בהינתן רשימת תווים (רשימה של מחרוזות באורך 1) ומחרוזת קלט כלשהי – נרצה להכריע האם ניתן להרכיב את מחרוזת הקלט באמצעות התווים מרשימת הקלט תחת האילוצים הבאים:
(1) ניתן להשתמש בכל תו יותר מפעם אחת,
(2) אחרי שימוש בתו מסוים באינדקס i, אין להשתמש בתווים אשר באינדקסים < i.

למשל , עבור רשימ ת התווים [‘.’ , ’f’, ’d’, ‘ , ’a’, ’e’, ‘&’ , ’t’, ’b’ , ’a’] והמחרוז ת ’beef’, מחרוזת הקלט ניתנת להרכבה ע"י שימוש בתווים באינדקסים 1 (b), 4 (e) , 4 (e), 7 (f). עבור אותה רשימה והמחרוזת ’beat’ , מחרוזת הקלט אינה ניתנת להרכבה מאחר והתו האחרון במילה ’t’ ממוק ם באינדקס 2, ואילו התו ’a’, שנדרש להרכבת המילה לפני ’t’, נמצא באינדקס גבוה יותר (5).

ממשו את הפונקציה **word_reconstruction(***target_word, char_list***)**, המקבלת את שני הארגומנטים הבאים:

- target_word – מטיפוס מחרוזת, מילת הקלט שאותה נרצה להרכיב.
- char_list – מטיפוס רשימה, רשימת תווים לפי הגדרת השאלה.

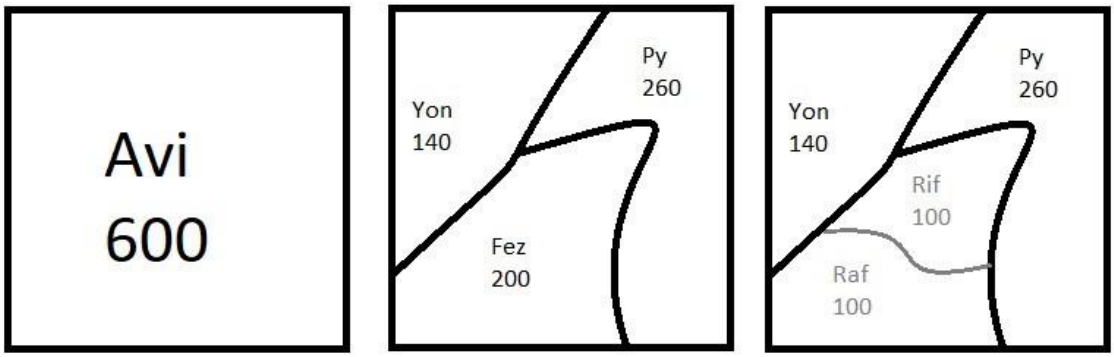
הפונקציה מחזירה פלט בוליאני (True / False) המייצג האם מחרוזת הקלט ניתנת להרכבה או לא.

דוגמאות הרצה (עבור רשימת תווים זהה):

```
Target word is :  sad
Character list is :  ['f', 'z', 'i', 'e', 'x', '5', 'a', '@', 'd', '~']
word_reconstruction output = False
Target word is :  feed
Character list is :  ['f', 'z', 'i', 'e', 'x', '5', 'a', '@', 'd', '~']
word_reconstruction output = True
Target word is :  fax
Character list is :  ['f', 'z', 'i', 'e', 'x', '5', 'a', '@', 'd', '~']
word_reconstruction output = False
```

שאלה 2

נחלה משפחתית היא חלקת אדמה שניתן להעביר בירושה. בעל נחלה יכול לחלק את הנחלה ליורשיו, כאשר החלקים נמדדים ביחידות שלמות של מטר רבוע. למשל, כפי שמתואר בשרטוטים, אבי הוא בעל נחלה ששטחה 600 מ"ר שהחליט לחלק לצאצאיו, יון, פז ופיי, נחלות בשטחים של 140, 200, ו-260 מ"ר בהתאמה. פז בוחר לחלק את נחלתו באופן שווה בין חבריו הטובים , ריף ורף.



נגדיר חלוקה להיות *"הומוגנית"*, אם כל חלוקה של נחלה ותתי-הנחלות שלה היו שוויוניות. בדוגמא, אבי לא חילק את נחלתו בצורה שוויונית, אולם פז חילק את נחלתו בצורה שוויונית – ולכן (רק) החלוקה של פז מוגדרת כהומוגנית.

אם ריף יחליט לחלק את נחלתו לשתי נחלות בשטחים שאינם שווים, אז החלוקה של פז לא תוגדר כהומוגנית, מכיוון שקיימת בנחלתו תת-חלוקה שאינה שוויונית.
אם ריף יחלק את נחלתו באופן שווה לשתי נחלות ורף יחלק את נחלתו באופן שווה לשלוש נחלות, אז החלוקה תהיה הומוגנית, למרות שליורשים של ריף נחלות בשטח שונה מהיורשים של רף.

נייצג חלוקה של נחלות באמצעות רשימה מקוננת כאשר מספר מייצג שטח של נחלה, ורשימה מייצגת חלוקה של נחלה. למשל, השרטוט השמאלי ייוצג ע"י הרשימה [600], השרטוט האמצעי ייוצג ע"י הרשימה [260, 200, 140] (או סדר אחר של תתי-הנחלות), השרטוט הימני ייוצג ע"י הרשימה [260, [100, 100], 140] (כאשר הסדר הפנימי של תתי הנחלות אינו חשוב).

ממשו את הפונקציה: **estate_homogeneity(***divisions, total_area***)**

שמקבלת את הארגומנטים הבאים:

- divisions – משתנה מטיפוס רשימה, מייצג חלוקה של נחלות.
- total_area – משתנה מטיפוס שלם, מייצג את שטח הנחלה הכולל במ"ר.

ומחזירה ערך מטיפוס בוליאני המייצג האם החלוקה הומוגנית.

<u>הנחיה</u> - בשאלה זו אין להשתמש ב slicing.

דוגמאות הרצה :

```
Total Area : 600
Divisions List: [300, [100, [50, 50], [25, 25, 25, 25]]]
estate_homogeneity Output = True
Total Area : 200
Divisions List: [[50, 50], 100]
estate_homogeneity Output = True
Total Area : 300
Divisions List: [[50, 50], 100, [50, [10, 20, 20]]]
estate_homogeneity Output = False
```

שאלה 3

נגדיר רשימת מספרים lst להיות "זיג-זג" אם כל איבר במקום ה- i ברשימה מקיים אחד מן התנאים הבאים:

1. קטן ממש ממחצית ערכם של כל אחד מהאיברים במיקומים i-1, i+1 ברשימה.
2. גדול ממש מפעמיים ערכם של כל אחד מהאיברים במיקומים i-1, i+1 ברשימה.

עבור 0 < i < len(lst)-1 .

למשל, הרשימה [3 ,7 ,2 ,5] מהווה רשימת זיג זג מאחר והאיבר במקום השני (7) מקיים את התנאי השני, והאיבר במקום השלישי (2) מקיים את התנאי הראשון.

בהינתן שתי רשימות מספרים, נגדיר רשימת "זיג-זג בריבוע" על שתי הרשימות האלו לפי התנאים הבאים:

- רשימת זיג-זג.
- כל איבר מאחת מרשימות הקלט יכול להופיע לכל היותר פעם אחת ברשימת הזיג-זג בריבוע.
- כל איבר באינדקס i מרשימת קלט שמופיע ברשימת הזיג-זג בריבוע, יהיה באינדקס נמוך **ברשימת הזיג-זג בריבוע** מאשר כל האיברים שהאינדקסים שלהם גדולים מ-i ברשימת הקלט שממנה הגיע.
  כלומר אין החלפות בסדר הופעת האיברים ביחס לרשימות הקלט.

למשל – בעזרת שתי רשימות הקלט הבאות (מסומנות בצבעים שונים):

[10, 11 , 2] , [12 , 6, 9]

נוכל להרכיב את רשימות הזיג זג החוקיות הבאות:

[]
[12]
[10, 6]
[2, 9]
[12, 2, 9]
[11, 2, 6]

בתרגיל זה, עליכם להרכיב רשימת זיג-זג בריבוע באורך מקסימלי.
ממשו את הפונקציה **merge_zigzag(***list1, list2***)**, המקבלת את הקלטים הבאים:

- list1, list2 – משתנים מטיפוס רשימה, רשימות קלט המכילות מספרים אי-שליליים .

על הפונקציה להחזיר רשימת זיג-זג בריבוע באורך מקסימלי בעזרת רשימות הקלט.

דוגמאות הרצה

```
list1 :  [1, 3, 5, 23, 11]
list2 :  [2, 0, 6, 17, 18]
Longest Zig-Zag list = [2, 0, 6, 1, 17, 3, 18, 5, 23, 11] , Length 10
list1 :  [15, 10, 5, 3]
list2 :  [2, 11, 13, 14]
Longest Zig-Zag list = [10, 2, 11, 5, 13, 3, 14] , Length 7
list1 :  [10, 11, 2]
list2 :  [12, 6, 9]
Longest Zig-Zag list = [11, 2, 9] , Length 3
```

שימו לב

עשויות להיות מספר רשימות זיג-זג בריבוע באורך מקסימלי, כל אחת מהן תתקבל כפתרון תקין.

שאלה 4

בשאלה זו תממשו אלגוריתם הפרד ומשול אשר מקבל מטריצה ומחזיר אותה מסובבת על צירה. ראשית, תממשו פונקציות עזר, ולאחר מכן תשתמשו בהן כדי לממש את האלגוריתם.

נגדיר את המושגים האלגברים הבאים וכיצד נייצג אותם בפייתון:

<u>וקטור ממימד n</u> - ייוצג ע"י רשימה באורך n .

למשל – הווקטור (9, 1, 0) שאורכו 3 ייוצג ע"י הרשימה [9, 1, 0] שאורכה 3.

<u>מטריצה מסדר $m \times n$</u>– תיוצג ע"י רשימה שמכילה m רשימות, באורך n כל אחת.

למשל – המטריצה $\begin{pmatrix} 0 & 1 & 2 \\ 9 & 10 & 11 \end{pmatrix}$ מסדר 3 × 2 תיוצג ע"י הרשימה [[0, 1, 2], [9, 10, 11]]

נגדיר אופרטור חדש בשם colcat, שמשרשר עמודות ,ונסמן אותו $+^{CC}$. אופרטור זה מקבל כקלט שתי מטריצות, כאשר כל אחת מהן בעלת m שורות, ומחזיר מטריצה בעלת m שורות שכל שורה בה מהווה שרשור של שורות מטריצות הקלט.

לדוגמא - הפעלת האופרטור על שתי מטריצות בעלות 2 שורות

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}_{2\times 2} +^{CC} \begin{pmatrix} 8 \\ 9 \end{pmatrix}_{2\times 1} = \begin{pmatrix} 0 & 1 & 8 \\ 2 & 3 & 9 \end{pmatrix}_{2\times 3}$$

שימו לב שפעולת האופרטור אינה <u>קומוטטיבית</u>!

<u>סעיף א'</u>

פייתון תומך בשרשור עמודות כפי שהוגדר לעיל רק עבור וקטורים, לדוגמא אם נרצה לשרשר שני וקטורים (1, 2), (-9, 6) בפייתון נשתמש באופרטור '+' :

```
>>> [1,2] + [-9,6]
[1, 2, -9, 6]
```

עליכם להרחיב את האופרטור שיתמוך גם במטריצות:

```
>>> colcat([[1, 2]], [[-9, 6]])
[[1, 2, -9, 6]]
```

לשם כך ממשו את הפונקציה **colcat(***mat_a, mat_b***)**, שמקבלת את הארגומנטים:

- mat_a – מטריצת קלט.
- mat_b – מטריצת קלט. מספר השורות זהה לזו של mat_a.

ומחזירה את המטריצה שתתקבל ע"י הפעלת $mat\_a +^{CC} mat\_b$

הנחיות

- האופרטור מוגדר רק עבור $m, n \geq 1$, הניחו שהקלט תקין.

דוגמאות הרצה

```
>>> mat_1 = [ [ 0 , 1 ] ,[ 2 , 3] ]
... mat_2 = [ [ 6 ] , [ 7 ] ]
... print(colcat(mat_2 ,mat_1 ))
[[6, 0, 1], [7, 2, 3]]
>>> print(colcat(mat_1 ,mat_1 ))
[[0, 1, 0, 1], [2, 3, 2, 3]]
```

סעיף ב'

ניתן להתייחס לכל מטריצה O מסדר $m \times n$ כאל תוצאת ההפעלה של אופרטור colcat בין שתי מטריצות A ו-B , אשר הסדרים שלהם $m \times a$ ו- $m \times b$ בהתאמה, ומתקיים $a + b = n$, באופן הבא:

$O_{m\times n} = A_{m\times a} +^{CC} B_{m\times b}$.

נגדיר אופרטור חדש בשם vertical_split, שמקבל מטריצה מסדר $m \times n$ ומחזיר את שתי המטריצות A ו -B, עבור המקרה הפרטי בו $a = n//2$.

לדוגמא – הפעלת האופרטור vertical_split על המטריצה הבאה $\begin{pmatrix} 0 & 1 & 8 & 4 & 5 \\ 2 & 3 & 9 & 4 & 6 \end{pmatrix}$ תחזיר לנו את המטריצות הבאות:

$$A = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}, B = \begin{pmatrix} 8 & 4 & 5 \\ 9 & 4 & 6 \end{pmatrix}$$

ממשו את הפונקציה **vertical_split(***input_mat***)** המקבלת:

- input_mat – רשימה באורך m של רשימות באורך n המייצגת מטריצה מסדר $m \times n$.

ומחזירה *tuple* המכיל את הייצוגים של מטריצות $A_{m\times a}, B_{m\times b}$ המתאימות לפי הפעלת האופרטור *vertical_split* על מטריצת הקלט.

דוגמאות הרצה

```
>>> in_mat = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
...     [13, 14, 15, 16]
... ]
>>> output = vertical_split(in_mat)
... print(type(output))
<class 'tuple'>
>>> print('mat A: \n' ,output[0])
mat A:
 [[1, 2], [5, 6], [9, 10], [13, 14]]
>>> print('mat B: \n' ,output[1])
mat B:
 [[3, 4], [7, 8], [11, 12], [15, 16]]
```

<u>אלגוריתם לסיבוב מטריצה בכיוון השעון</u> **סעיף בונוס**

כעת נסקור תיאור של אלגוריתם הפרד ומשול, אשר מקבל מטריצת קלט סימטרית ומחזיר מטריצת פלט אשר מהווה סיבוב של מטריצת הקלט ב-°90 בכיוון השעון.

<u>שלב א'</u> – עבור מטריצת קלט מסדר $m \times n$, המטריצה תחולק רקורסיבית לארבעה רביעים (תתי-מטריצות) מסדר דומה ככל האפשר (ההפרש בין מס' השורות או העמודות של הרביעים יהיה 1 לכל היותר).
החלוקה תיפסק כאשר m=n=1.

<u>שלב ב'</u> –כל חלוקה לרביעים שהתקבלה במהלך שלב א' תסובב ב- °90 בכיוון השעון. כלומר ,תוחזר מטריצה חדשה כאשר כל רביע יעבור למיקום הבא עם כיוון השעון ביחס למטריצת הקלט.
לדוגמא, עבור החלוקה לרביעים (שלב א') של $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$, תוחזר המטריצה $\begin{bmatrix} C & A \\ D & B \end{bmatrix}$.

למשל, עבור הקלט הבא (מטריצה בגודל 4x4).

בשלב א' תתבצע חלוקה לרביעים:

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix} => \begin{matrix} A = \begin{pmatrix} 0 & 1 \\ 4 & 5 \end{pmatrix} & B = \begin{pmatrix} 2 & 3 \\ 6 & 7 \end{pmatrix} \\ C = \begin{pmatrix} 8 & 9 \\ 12 & 13 \end{pmatrix} & D = \begin{pmatrix} 10 & 11 \\ 14 & 15 \end{pmatrix} \end{matrix}$$

כל רביע יחולק לרביעים באופן דומה, למשל עבור רביע A :

$$\begin{pmatrix} 0 & 1 \\ 4 & 5 \end{pmatrix} => \begin{matrix} A_2 = (0) & B_2 = (1) \\ C_2 = (4) & D_2 = (5) \end{matrix}$$

בשלב ב '– עבור כל חלוקה לרביעים יתבצע סיבוב עם כיוון השעון, ותוחזר המטריצה החדשה.
רביע A לאחר הסיבוב:

$$\begin{pmatrix} C_2 & A_2 \\ D_2 & B_2 \end{pmatrix} = \begin{pmatrix} 4 & 0 \\ 5 & 1 \end{pmatrix}$$

באופן דומה גם רביעים B , C ו-D יחולקו לרביעים ויסובבו:

$$\begin{matrix} A = \begin{pmatrix} 4 & 0 \\ 5 & 1 \end{pmatrix} & B = \begin{pmatrix} 6 & 2 \\ 7 & 3 \end{pmatrix} \\ C = \begin{pmatrix} 12 & 8 \\ 13 & 9 \end{pmatrix} & D = \begin{pmatrix} 14 & 10 \\ 15 & 11 \end{pmatrix} \end{matrix}$$

כל הרביעים המסובבים יאוחדו למטריצה חדשה:

$$\begin{pmatrix} C & A \\ D & B \end{pmatrix} => \begin{pmatrix} 12 & 8 & 4 & 0 \\ 13 & 9 & 5 & 1 \\ 14 & 10 & 6 & 2 \\ 15 & 11 & 7 & 3 \end{pmatrix}$$

שהינה מטריצה הזהה למטריצת הקלט לאחר סיבוב של 90° עם כיוון השעון.

<u>סעיף ג'</u> – סעיף בונוס

ממשו את הפונקציה **rotate_mat_rec** , המקבלת את הארגומנט input_mat: רשימה של רשימות המייצגת מטריצה,
ומחזירה רשימה של רשימות המייצגת את מטריצה הקלט, לאחר סיבוב של 90° בכיוון השעון .
לצורך כך, השתמשו בשתי הפונקציות שמימשתם – vertical_split ו-colcat.

חידוד – מתוקף הגדרת האופרטורים אותם תפעילו, ניתן להניח ש $m = n = 2^{\times}$

בהצלחה!