תרגיל 4 חלק ב: רקורסיות מתקדם

תאריך פרסום:	18/11
תאריך הגשה:	2/12 בשעה 23:59
מתרגל אחראי:	אוריאל פרידמן
משקל תרגיל:	3 נקודות
מטרות העבודה:	רקורסיות מרובות קריאות.

הנחיות כלליות

קראו בעיון את ההנחיות והעבודה. לא יתקבלו ערעורים על טעויות שחרגו מההנחיות.

1. העבודה תבוצע ותוגש ביחידים.
2. מומלץ לקרוא את העבודה כולה לפני שאתם ניגשים לפתרון.
3. עליכם להוריד את קבצי הקוד שמסופק עם התרגיל ולהשלים את הקוד החסר. יש לממש את הפתרון לתרגיל אך ורק באזורים שהוגדרו לשם כך בקובץ!
4. כתיבת קוד קריא:
   - 4.1. השתמשו בשמות משתנים משמעותיים. שימוש בשמות לא משמעותיים עשוי לגרור לפגיעה בציון.
   - 4.2. כתבו תיעוד (הערות) שמסביר את הקוד שלכם. יש לכתוב תיעוד docstring (הערות) בכל פונקציה כפי שנלמד בכיתה.
   - 4.3. אין לכתוב הערות בעברית! עבודה שתכיל טקסט בשפה שאינה אנגלית (או פייתון) תקבל ציון אפס ללא אפשרות ערעור.
5. אין להשתמש בחבילות או במודולים חיצוניים מלבד מה שהוגדר בתרגיל! אם יש ספק ניתן לשאול בפורום המתאים (ראו סעיף 10).
6. יש לכתוב קוד אך ורק באזורים שהוגדרו לשם כך!
7. הנחות על הקלט:
   - 7.1. בכל שאלה יוגדר מה הקלט שהקוד מקבל וניתן להניח כי הקלט שנבדוק מקיים את התנאים הללו. אין להניח הנחות נוספות על הקלט מעבר למה שהוגדר.
   - 7.2. בכל שאלה סיפקנו עבורכם דוגמאות לקלט והפלט הרצוי עבורו. עליכם לערוך בדיקות נוספות לקוד שמימשתם ולא להסתמך על דוגמאות אלו בלבד.
8. בדיקת העבודה:
   - 8.1. העבודה תיבדק באופן אוטומטי ולכן על הפלטים להיות זהים לפלטים שמוגדר בתרגיל.
   - 8.2. טרם ההגשה יש לעבור על המסמך assignments checklist שנמצא במודל.
   - 8.3. מערכת הבדיקות קוראת לפונקציות שהוגדרו בתרגיל בצורה אוטומטית. אין לשנות את חתימות הפונקציות. חריגה מההנחיות תגרור ציון אפס.
9. העתקות:
   - **9.1. אל תעתיקו!**
   - 9.2. העתקת קוד (משנים קודמות, מחברים או מהאינטרנט) אסורה בהחלט. בפרט אין להעביר קוד בין סטודנטים. צוות הקורס ישתמש בכלים אוטומטיים וידניים כדי לזהות העתקות. תלמיד שייתפס בהעתקה יועמד בפני ועדת משמעת (העונש המינימלי לפי תקנון האוניברסיטה הוא כישלון בקורס).
   - 9.3. אנא קראו בעיון את המסמך שהכנו בנושא: https://moodle2.bgu.ac.il/moodle/mod/resource/view.php?id=1922556

10. <u>שאלות על העבודה</u>:
    - 10.1. שאלות בנוגע לעבודה ישאלו בפורום שאלות לתרגיל במודל או בשעות הקבלה של המתרגל האחראי **<u>בלבד</u>** .
    - 10.2. אין לפנות במייל לבודקת התרגילים או למתרגלים אחרים בנוגע לעבודות הגשה. מיילים בנושאים אלו לא יקבלו מענה.
    - **10.3. לפני ששואלים שאלה בפורום יש לוודא שהשאלה לא נשאלה קודם!** שאלה שחוזרת על שאלה קיימת לא תענה.
    - 10.4. אנו מעודדים סטודנטים לענות על שאלות של סטודנטים אחרים. המתרגל האחראי יאשר שתשובה כזו נכונה.
11. <u>הגשת העבודה</u>:
    - 11.1. עליכם להוריד את הקבצים מתיקיית "תרגיל בית 4" מהמודל. התיקייה תכיל תיקייה נוספת ובה קבצי העבודה וקובץ הוראות. עליכם למלא את הפתרון במקום המתאים ובהתאם להוראות התרגיל.
    - 11.2. שימו לב: בנוסף לקבצי העבודה מצורף קובץ בשם get_id.py. עליכם למלא במקום המתאים בקובץ את תעודת הזהות שלכם. הגשה שלא תכיל את הקובץ הנ"ל עם תעודת הזהות הנכונה לא תיבדק ותקבל ציון אפס!
    - 11.3. את העבודה יש להגיש באמצעות תיבת ההגשה הייעודית במודל.
    - 11.4. פורמט ההגשה הוא להלן:
        - 11.4.1. את התיקייה בשם hw4b שהורדתם מתיקיית העבודה במודל, ובה נמצאים קבצים הקוד שלכם, יש לכווץ לפורמט zip (קבצים אחרים, כגון קבצי rar, יקבלו ציון 0).
        - 11.4.2. השם של התיקייה המכווצת יהיה תעודת הזהות שלכם.
        - 11.4.3. העלו את התיקייה המכווצת לתיבת ההגשה של העבודה.

הנחיות ספציפיות לעבודה:

בכל מקום בו לא נאמר אחרת – ניתן לממש פונקציות מעטפת ו\או פונקציות פנימיות. אין לממש פונקציות הכוללות פרמטרי ברירת מחדל. **<u>בעבודה זו חל איסור להשתמש בלולאות</u>**, אלא אם צוין אחרת במפורש. שימוש בלולאה במקום אסור יפסול את הפתרון.

שאלה 1

בעיית Sub-set sum שהוצגה בכיתה תיארה מצב בו בהינתן קבוצת מספרים וסכום קבוע ניתן יהיה לקבוע האם ניתן להגיע לסכום מחיבור תת קבוצה של אברים מהקבוצה ללא חזרה על אברים.

סעיף א'

עליכם לממש את הפונקציה:

`subset_sum_count(ls,sm)`

המקבלת **רשימה ls** המכילה מספרים שלמים וחיוביים ומספר חיובי שלם **sm** – ומחזירה בכמה אפשרויות שונות ניתן להגיע ל sm בעזרת חיבור של אברים מ-ls כאשר כל אבר יכול להיות בשימוש לכל היותר פעם אחת.

<u>קלט:</u> הפונקציה מקבלת:

- **`ls[List[int]]`** – רשימה המכילה מספרים שלמים וחיוביים בלבד, לא None.
- **`sm[int]`** – מספר שלם וחיובי – אינו None.

<u>פלט:</u> הפונקציה מחזירה בכמה אפשרויות שונות ניתן להגיע לסכום sm לפי הגדרות הסעיף.

<u>הנחות קלט:</u>

- ls מכילה מספרים שלמים וחיוביים בלבד (גדולים ממש מ 0), לא None.
- sm הוא מספר שלם וחיובי בלבד (גדול ממש מ 0).

<u>דגשים והערות:</u>

- כאשר ערך מסוים מופיע מספר פעמים ברשימה, ניתן להשתמש בכל מופע בפתרון באופן בלתי תלוי. לדוגמא עבור הרשימה [7,6,20,1,13,1] והסכום 14 עליכם לספור את הפתרון 7,6,1 פעמיים מכיוון שהערך 1 נמצא פעמיים ברשמית הקלט. כמובן שניתן להשתמש בשני המופעים, לדוגמא, אם הסכום היה 15 (לדוגמא בפתרון: 7,6,1,1).

<u>דוגמאות:</u>

1.

```
>> print(subset_sum_count([7,6,18,20],14))
0
```

2.

```
>> print(subset_sum_count([7,6,1,20],14))
1
```

3.

```
>> print(subset_sum_count([7,6,20,1,13,1],14))
4
```

סעיף ב'

עליכם לממש את הפונקציה:

`subset_sums(ls,sm)`

המקבלת **רשימה ls** המכילה מספרים שלמים בלבד ומספר שלם **sm** – ומחזירה רשימה של רשימות של כל הפתרונות האפשריים לבעיה מסעיף א' בסדר המקורי בו הופיעו ב ls.

<u>קלט:</u> כמו בסעיף א'.

<u>פלט:</u> הפונקציה מחזירה את רשימת הפתרונות. אורכה מוגדר לפי תנאי סעיף א'. כל איבר מכיל פתרון חוקי (ללא חזרות) - רשימה של איברים מ-ls עם סכום שווה ל-sm.

<u>הנחות קלט:</u> ראו סעיף א'.

<u>דוגמאות:</u>

.1

```
>> print(subset_sums([7,8,7],14))
[[7, 7]]
```

.2

```
>> print(subset_sums([7,6,5,1,14,13],14))
[[7, 6, 1], [1, 13], [14]]
```

.3

```
>> print(subset_sums([7,8,17],14))
[]
```

סעיף ג'

בסעיף זה עליכם ליעל את ריצת הקוד הרקורסיבי באמצעות ממויזציה, עליכם לממש את הפונקציה:

`subset_sum_memo(ls,sm)`

המקבלת **רשימה `ls`** המכילה מספרים שלמים בלבד ומספר שלם **`sm`** – ומחזירה האם ניתן או לא ניתן להגיע ל sm בעזרת קומבינציה כלשהיא של אברים מ ls, בעזרת ממויזציה.

קלט: כמו בסעיף א'

פלט: הפונקציה מחזירה ערך בוליאני True באם ניתן להגיע לסכום sm בעזרת איזו שהיא קומבינציה של ערכים הקיימים מ ls ללא חזרות או False אם אחרת.

הנחות קלט: ראו סעיף א'

- ls מכילה מספרים שלמים בלבד.
- sm הוא מספר שלם.

דגשים והערות:

- **מימוש ללא ממויזציה יפסל.**

דוגמאות:

1.
```
>> print(subset_sum_memo([7,6,18,20],14))
False
```

2.
```
>> print(subset_sum_memo([7,6,5,1,14,13],14))
True
```

3.
```
>> print(subset_sum_memo([1,1,1,1,1,1,1,1,1],10))
False
```

סעיף ד'

עליכם לממש את הפונקציה:

`subset_sum_with_repeats(ls,sm)`

המקבלת **רשימה `ls`** המכילה מספרים שלמים וחיוביים בלבד, **ממוינים בסדר עולה** ומספר שלם **sm** ומחזירה רשימה של רשימות של כל הקומבינציות האפשריות מהרשימה ls שסכומן הוא sm **עם חזרות**.
חזרות - כאשר ניתן להשתמש בפתרון יותר מפעם אחת בכל איבר מרשימת הקלט, לדוגמא אם ברשימה יש אבר בערך 1 (הוא חייב להיות הראשון כי היא ממוינת) ניתן להרכיב בעזרתו כל מספר.

- קלט: ראו סעיף א'

פלט: כמו בסעיף ב', רק שמספרים יכולים לחזור על עצמם ברשימה.

הנחות קלט: ראו סעיף א'.

דגשים והערות:

- **בסעיף זה ניתן להשתמש בלולאה אחת בלבד.**
- ניתן להניח כי כל ערך ברשימה מופיע פעם אחת בלבד.

דוגמאות:

1.

```
>> print(subset_sum_with_repeats([1,3,5],5))
[[1, 1, 1, 1, 1], [1, 1, 3], [1, 3, 1], [3, 1, 1], [5]]
```

2.

```
>> print(subset_sum_with_repeats ([1,2,3,4,5],5))
[[1, 1, 1, 1, 1], [1, 1, 1, 2], [1, 1, 2, 1], [1, 1, 3],
[1, 2, 1, 1], [1, 2, 2], [1, 3, 1], [1, 4], [2, 1, 1, 1],
[2, 1, 2], [2, 2, 1], [2, 3], [3, 1, 1], [3, 2], [4, 1],
[5]]
```

3.

```
>> print(subset_sum_with_repeats ([1,2,3],4))
[[1, 1, 1, 1], [1, 1, 2], [1, 2, 1], [1, 3], [2, 1, 1],
[2, 2], [3, 1]]
```

שאלה 2

בשאלה זו נעסוק ביצירת כל האפשרויות ליצירת מילה בגודל נתון (לאו דווקא קיימת) מכל האפשרויות הקיימות מאותיות נתונות.

רשימת המילים שיוחזרו יהיו ממוינות לקסיקוגרפית.

סעיף א'

עליכם לממש את הפונקציה:

`abc_words(num)`

המקבלת **מספר num** המכיל מספר שלם וחיובי ומחזירה רשימה של כל המחרוזות השונות האפשריות באורך num הבנויות מהאותיות a,b,c, (כולל חזרה של אותיות) ממוינות בסדר לקסיקוגרפי עולה.

<u>קלט:</u>

- **`num[int]`** – מספר שלם וחיובי.

<u>פלט:</u> הפונקציה מחזירה רשימת מחרוזות ייחודיות שכל אחת מהן באורך num. הרשימה תכיל את כל המחרוזות השונות האפשריות המורכבות מהאותיות a,b,c . המחרוזות ברשימה יהיו ממוינות בסדר לקסיקוגרפי.

<u>הנחות קלט:</u>

- num - מספר חיובי שלם (גדול מאפס).

<u>דגשים והערות:</u>

- **אין להשתמש במיון (כולל sorted), עליכם ליצור את הרשימה כך שהיא כבר ממוינת.**

<u>דוגמאות:</u>

1.

```
>> print(abc_words(1))
['a', 'b', 'c']
```

2.

```
>> print(abc_words(3))
['aaa', 'aab', 'aac', 'aba', 'abb', 'abc', 'aca', 'acb',
'acc', 'baa', 'bab', 'bac', 'bba', 'bbb', 'bbc', 'bca',
'bcb', 'bcc', 'caa', 'cab', 'cac', 'cba', 'cbb', 'cbc',
'cca', 'ccb', 'ccc']
```

סעיף ב'

עליכם לממש את הפונקציה:

`char_to_char_words(ch1,ch2,num)`

המקבלת שני תווים **ch1, ch2 - בין 'A' ל 'Z', כאשר ל-ch2 ערך ascii גבוה משל ch1, ומספר שלם וחיובי num**. הפונקציה מחזירה רשימה של כל המחרוזות השונות באורך num הבמורכבות מהאותיות בטווח ch1 עד ch2 ממוינות בסדר לקסיקוגרפי עולה. ניתן לחזור על תו מספר פעמים במחרוזת.

<u>קלט:</u>

- **ch1[str]** – מחרוזת עם תו אחד בלבד בין 'A' ל 'Z'.
- **ch2[str]** – מחרוזת עם תו אחד בלבד בין ch1 ל 'Z'.
- **`num[int]`** – מספר שלם וחיובי.

<u>פלט:</u> הפונקציה מחזירה רשימת מחרוזות ייחודיות באורך num ממוינות לקסיקוגרפית, על הרשימה להכיל את כל המחרוזות השונות האפשריות המורכבות מהאותיות בין ch1 ל ch2 כולל.

<u>הנחות קלט:</u>

- num - מספר חיובי שלם (גדול מאפס).

<u>דגשים והערות:</u>

- אין להשתמש בכל מיון שהוא (גם אם שלכם וגם אם רקורסיבי לחלוטין), יש ליצור את הרשימה כך שהיא כבר ממוינת.

<u>דוגמאות:</u>

1.

```
>> print(char_to_char_words('A','B',4))
['AAAA', 'AAAB', 'AABA', 'AABB', 'ABAA', 'ABAB', 'ABBA',
'ABBB', 'BAAA', 'BAAB', 'BABA', 'BABB', 'BBAA', 'BBAB',
'BBBA', 'BBBB']
```

2.

```
>> print(char_to_char_words('D','F',2))
['DD', 'DE', 'DF', 'ED', 'EE', 'EF', 'FD', 'FE', 'FF']
```

3.

```
>> print(char_to_char_words('Q','Q',5))
['QQQQQ']
```

שאלה 3

כפי שראיתם בתרגול, ניתן לפתור מבוך באמצעות אלגוריתם רקורסיבי שממפה את כל האפשרויות להתקדמות מכל נקודה ונקודה במבוך. בשאלה זו נייצג מבוך דו ממדי באמצעות מטריצה שאותה נממש באמצעות רשימה מקוננת של מספרים שלמים. תחת תנאים מסוימים (ראו בהמשך) ניתן להתקדם בין אברים "שכנים" המוגדרים באמצעות שכנות-4: מעל, מתחת או מהצד. נגדיר מסלול מונוטוני עולה ממש כמסלול בו כל אבר (למעט הראשון) גדול ממש מהאבר הקודם.

ממשו את הפונקציה:

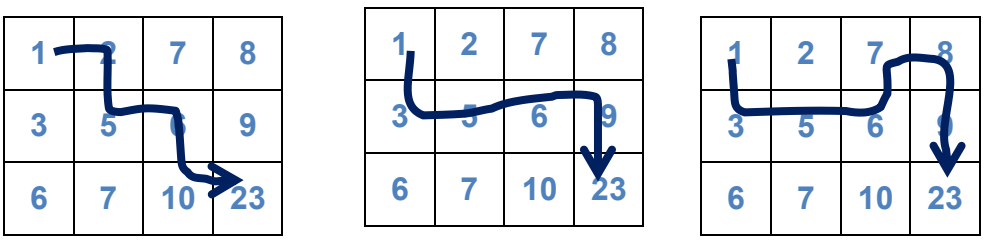
`solve_maze_monotonic(maze)`

המקבלת רשימה מקוננת בה כל הרשימות הפנימיות באותו האורך, כאשר כל רשימה פנימית מכילה מספרים שלמים בלבד, ומחזירה מסלול מונוטוני עולה מהפינה השמאלית העליונה (אינדקסים [0][0]) לימנית התחתונה (אינדקסים [len(maze)-1][len(maze[0])-1]) או רשימה ריקה במקרה שלא קיים מסלול כזה.

לדוגמא, המבוך באיור מתואר באמצעות הרשימה [[1,2,3,8],[3,5,6,9],[6,7,10,23]] המייצגת מבוך הנראה כך:

1	2	7	8
3	5	6	9
6	7	10	23

המסלולים הבאים (ולא רק הם) יהיו מסלולים מונוטוניים עולים ממש לפי הדרישה:



מסלול ייוצג כרשימה מקוננת של אנדקסים (שורה, טור) המתארים את המסלול. לדוגמא, המסלול השמאלי ייוצג כך: [[0,0],[0,1],[1,1],[1,2],[2,2],[2,3]]

<u>קלט:</u>

- rec [List[List[int]]] – רשימה מקוננת של רשימות באורך זהה המכילות מספרים שלמים בלבד.

<u>פלט:</u> הפונקציה מחזירה רשימה מקוננת של מסלול שנמצא ליציאה מהמלבן, כל רשימה פנימית ברשימה המקוננת תהיה רשימה באורך 2 המכילה באינדקס 0 את אינדקס השורה ובאינדקס 1 את אינדקס הטור של כל אבר במטריצה שהיה חלק מהמסלול כאשר כל שתי רשימות פנימיות צמודות יהיו של אברים שכנים.

הנחות קלט:

- ניתן להניח כי כל הרשימות הפנימיות הן באותו אורך.

דגשים והערות:

- יכולות להיות רשימות מקוננות ללא מסלול חוקי (במקרה זה תוחזר רשימה ריקה).

דוגמאות:

.1
```
>> print(solve_maze_monotonic([[1,2,3],[2,0,4],[3,4,5]]))
[[0, 0], [0, 1], [0, 2], [1, 2], [2, 2]]
```

.2
```
>> print(solve_maze_monotonic([[1,2,3],[2,0,5],[3,4,5]]))
[[0, 0], [1, 0], [2, 0], [2, 1], [2, 2]]
```

.3
```
>> print(solve_maze_monotonic([[1,2,3,4,5],
[12,11,10,4,6], [13,9,9,8,7], [14,9,9,8,7],
[15,16,17,18,19]]))
[[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [1, 4], [2, 4],
[2, 3], [2, 2], [1, 2], [1, 1], [1, 0], [2, 0], [3, 0],
[4, 0], [4, 1], [4, 2], [4, 3], [4, 4]]
```

.4
```
>> print(solve_maze_monotonic ([[1,2,3,4], [2,3,4,5],
[3,4,5,6],
[4,5,6,7]]))
[[0, 0], [0, 1], [0, 2], [0, 3], [1, 3], [2, 3], [3, 3]]
```

.5
```
>> print(solve_maze_monotonic([[1,2,3,4], [2,3,4,5],
[3,4,5,6], [4,5,6,6]]))
[]
```

.6
```
>> print(solve_maze_monotonic([[19]]))
[[0,0]]
```

בהצלחה !