אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

תאריך המבחן:	----------
שמות המרצים:	ד"ר אסף זריצקי
שמות המתרגלים:	מר שגיא תובל, מר ניר פרידמן
שם הקורס:	מבוא לתכנות
מספר הקורס:	37211111
שנה: 2019	מבחן לדוגמא 1
משך הבחינה:	שלוש שעות
חומר עזר:	דף A4 כתוב בכתב יד בשני הצדדים

אנא קראו היטב את ההוראות שלהלן:

- במבחן **ארבע** שאלות הכוללות סעיפי משנה. במבחן **103** נקודות. כדי לקבל את מלוא הניקוד יש לענות נכון על כל השאלות. ניקוד כל סעיף מצוין לידו. אין בהכרח קשר בין ניקוד הסעיף ובין רמת הקושי שלו.
- מומלץ לקרוא כל שאלה עד סופה, על כל סעיפיה, לפני תחילת הפתרון.
- את הפתרונות יש לכתוב במסגרות המסומנות לכל שאלה בטופס הבחינה. המחברת שקיבלתם היא מחברת טיוטה, והיא לא תימסר כלל לבדיקה. **בסיום הבחינה נאסוף אך ורק את דף התשובות**. כל שאר החומר יועבר לגריסה.
- בכל סעיף ניתן להשתמש בקוד שהתבקשתם לכתוב בסעיפים הקודמים, **גם אם לא פתרתם אותם**.
- ניתן להניח שהקלט תקין, אלא אם נכתב אחרת בשאלה.
- במידה שאינכם יודעים את התשובה לסעיף שלם כלשהו, רשמו "לא יודע/ת" (במקום תשובה) ותזכו ב-20% מניקוד הסעיף. אם רשום "לא יודע/ת", ההתייחסות היא לכל הסעיף
- אין להשתמש בחבילות או מודולים, אלא אם נאמר במפורש. כאשר אתם מתבקשים להשתמש בחבילה אין צורך לבצע import.

בהצלחה!

שאלה 1 (16 נקודות)

סעיף א (8 נקודות)

ועדת "סל התרופות" במשרד הבריאות התכנסה כדי לבחור את התרופות שיכללו בסל. התרופות שמועמדות להיכלל בסל מיוצגות ברשימה של איברים מטיפוס:tuple

1. האיבר הראשון מציין את **רמת עדיפות** הכנסת התרופה לסל – מספר שלם בין 1 ל-10.
2. האיבר השני הוא מספר שלם וחיובי המציין את **עלות** הכנסת התרופה לסל (במיליוני שקלים).

דוגמא: ה-tuple (10,20) מייצג תרופה שזכתה לרמת עדיפות של 10 על ידי ועדת המומחים, ועלות הכנסתה לסל התרופות היא 20 מיליון שקלים.

בסעיף זה נסייע לחברי ועדת "סל התרופות" בעבודתם על ידי דירוג התרופות השונות על פי היחס בין עדיפותן לעלותן. כלומר התרופה (10,20) תדורג ע"פ היחס 10/20 = 0.5.

השלימו את מימוש הפונקציה sort_meds(meds , names) המקבלת שתי רשימות:

- הרשימה meds היא רשימת תרופות כמוגדר למעלה.
- הרשימה names מכילה את שמות התרופות בהתאם לסדר ברשימה meds.

הפונקציה מחזירה רשימה חדשה המכילה את שמות התרופות ממוינות בסדר יורד על פי היחס בין עדיפות התרופה לבין עלות הכנסתה לסל התרופות. אין חשיבות לסידור הפנימי ברשימה המוחזרת בין שמות תרופות להן אותו יחס.

דוגמת הרצה:

```
meds = [(5, 10), (4, 20), (9, 30), (4, 40), (9, 1)]
names = ['Lipitor', 'Herceptin', 'Copaxone', 'Adderall', 'Amoxicillin']
print sort_meds(meds, names)
```

פלט:

```
['Amoxicillin', 'Lipitor', 'Copaxone', 'Herceptin', 'Adderall']
```

הסבר: שם התרופה Amoxicillin מופיע ראשון ברשימה המוחזרת היות והיחס בין עדיפות התרופה (9) לבין עלות הכנסה לסל התרופות (1) הוא הגדול ביותר (9/1=9). שם התרופה Adderall מופיע אחרון ברשימה המוחזרת היות והיחס בין עדיפות התרופה (4) לבין עלות הכנסתה לסל התרופות (40) הוא הקטן ביותר (4/40=0.1).

```
def sort_meds(meds , names):
    combo_lst = []
    for i in range(len(meds)):
        combo_lst.append((meds[i], names[i]))

    sorted_combo_lst = sorted(combo_lst, key=lambda x:
    x[0][0] / x[0][1],reverse=True)




    result = [x[1] for x in sorted_combo_lst]
    return result
```

סעיף ב (8 נקודות)

לאחר בחירת התרופות לסל, גדי הבחין כי התרופה שהוא צריך לא הוכנסה לסל השנה. תחילה חשב לארגן מחאה ע"י חסימת צומת השלום, אך אז אחיו דוד (מדען נתונים) הציע לו לקחת תרופה אחרת שנמצאת בסל התרופות, בעלת מרכיבים דומים ככל הניתן לתרופה המקורית. דוד חיבר טבלה אשר מסכמת אילו מרכיבים כל תרופה מהסל מכילה ואילו מרכיבים אינה מכילה. לדוגמא:

	מרכיב א'	מרכיב ב'	מרכיב ג'
תרופה א'	1	1	0
תרופה ב'	0	1	0

כאשר 1 מציין שהמרכיב קיים, ו-0 מציין שהמרכיב אינו קיים בתרופה.

בכדי להקל למצוא את התרופה הדומה ביותר לתרופה המקורית המיר דוד את טבלת מרכיבי התרופות בסל למטריצת numpy.array. עליכם לממש את הפונקציה similar_med(med_list, ing_mat, my_med_ing), אשר מקבלת את רשימת שמות התרופות שבסל (בהתאמה לסדר השורות), את מטריצת המרכיבים, ומערך numpy.array חד מימדי המכיל את מרכיבי התרופה שגדי צרך עד כה. הפונקציה תחזיר את שם התרופה שלה הכי הרבה <u>מרכיבים משותפים</u> עם התרופה המקורית.

- ☒ במידה וקיימות שתי תרופות עם מספר מרכיבים משותף, ניתן להחזיר כל אחת מהן.
- ☒ <u>השימוש בלולאות אסור לחלוטין</u> בסעיף זה. יש להשתמש בפונקציות של numpy בלבד.

☒ רמז- השתמשו בפונקציות numpy.matmul(a,b) אשר מבצעת כפל מטריציוני בין a ל-b, בפונקציה numpy.transpose(a) אשר מבצעת שחלוף למטריצה a, ובפונקציה numpy.argmax(a) אשר מחזירה את האינדקס של האיבר המקסימאלי ב-a.

דוגמאת ריצה:

עבור הקלט:

med_list = ['medicine 0', 'medicine 1', 'medicine 2', 'medicine 3', 'medicine 4']

ing_mat =

	0	1	2	3	4
0	1	1	0	0	0
1	1	1	0	0	1
2	0	1	1	1	0
3	0	0	0	0	1
4	1	1	1	0	0

my_med_ing =

	0	1	2	3	4
0	0	1	1	0	1

הפונקציה תחזיר "medicine 1", מאחר ולה יש 2 מרכיבים משותפים עם התרופה המקורית.

```
def similar_med(med_list, ing_mat, my_med_ing):
    num_common_all = np.matmul(my_med_ings ,
    np.transpose(ing_mat))
    max_comm_ind = np.argmax(num_common_all)
    return med_list[max_comm_ind]
```

שאלה 2 (35 נקודות)

בתכנית ריאליטי חדשה מקבלים המשתתפים רשימה של משימות וזמן נתון לביצוע משימות. כל משימה מיוצגת ע"י tuple המכיל זוג מספרים שלמים חיוביים (points, time):

- points, מספר הנקודות שזוכה משתתף שמשלים את המשימה.
- time, הזמן הדרוש לביצוע המשימה במלואה.

לדוגמא, השלמת המשימה (3,7) מזכה ב-3 נקודות ודורשת 7 דקות לביצועה. משתתף זוכה בנקודות רק על משימות שהשלים במלואן. מטרת המשתתף היא לצבור כמות מרבית של נקודות תוך זמן נתון. המשימות אינן תלויות זו בזו, ואין חשיבות לסדר בו הן יבוצעו.

סעיף א (20 נקודות)

השלימו את מימוש הפונקציה choose(missions, time) הפותרת את בעיית בחירת המשימות בזמן נתון, ומחזירה את מספר הנקודות המרבי שניתן לצבור עבור רשימת המשימות missions בזמן time. יש לממש את הפונקציה בעזרת Memoization. לשם כך, הפונקציה choose קוראת לפונקציית העזר choose_mem(missions, time, mem, i), ומעבירה למשתנה הקלט mem מילון ריק. במהלך ההרצה, mem יכיל את הפתרונות עבור תתי הבעיות שכבר נפתרו. שימו לב שההחלטה לגבי הגדרת המפתח נתונה לכם.

```
def choose_mem(missions, time, mem, i):
    key = (time, i)
    if key not in mem:
        if time == 0 or i < 0:
            mem[key] = 0
        else:
            p, t = missions[i]
            if t > time:
                mem[key] = choose_mem(missions,
time, mem, i-1)
            else:
                mem[key] = max(choose_mem(missions,
time, mem, i-1),
                                    p +
choose_mem(missions, time-t, mem, i-1))
    return mem[key]
```

```
def choose(missions, time):
    mem = {}
    return choose_mem(missions, time, mem,
len(missions)-1)
```

סעיף ב (15 נקודות)

בשלב השני, מחולקים המשתתפים לזוגות. כל זוג מקבל אוסף של משימות. זוג זוכה בנקודות רק עבור משימות שהושלמו במלואן. כל משימה מיוצגת ע"י tuple המכיל שלושה מספרים שלמים חיוביים (points, time1, time2):

- points, מספר הנקודות ששווה המשימה.
- time1, הזמן הדרוש לבן הזוג הראשון לביצוע חלקו במשימה.
- time2, הזמן הדרוש לבן הזוג השני לביצוע חלקו במשימה.

לדוגמא, השלמת המשימה (3,7,5) מזכה ב-3 נקודות, לבן הזוג הראשון נדרשות 7 דקות ולבן הזוג השני נדרשות 5 דקות לביצוע חלקו במשימה.

שימו לב: בני הזוג מבצעים את המשימות בנפרד, כאשר אחד מהם מסיים את חלקו במשימה הוא עובר למשימה אחרת ואינו מחכה לבן זוגו. משימה מושלמת רק כאשר שני בני הזוג השלימו את חלקם.

ממשו את הפונקציה choose_two(missions, time) המקבלת את רשימת המשימות missions ואת הזמן המוקצב time, ומחזירה את מספר הנקודות המרבי הניתן לצבירה. הפונקציה קוראת לפונקציית עזר רקורסיבית rec_two_choose המקבלת את רשימת המשימות ומשתני קלט נוספים לפי הצורך.

רמז: בכל שלב של הרקורסיה, התייחסו לזמן הנותר לכל אחד מבני הזוג בנפרד. שימו לב כי לכל אחד מבני הזוג נדרש פרק זמן שונה לביצוע המשימות.

```
def choose_two(missions, time):
    return choose_two_rec(missions, time, time,
len(missions)-1)




def choose_two_rec(missions, time1, time2, i):
    if i < 0 or time1 == 0 or time2 == 0: # stop
        return 0

    p, t1, t2 = missions[i]
    if t1 > time1 or t2 > time2: # one participant
cannot complete the mission
        return choose_two_rec(missions, time1,
time2, i-1)
    return max(choose_two_rec(missions, time1,
time2, i-1),
               p + choose_two_rec(missions, time1-
t1, time2-t2, i-1))
```

שאלה 3 (32 נקודות)

בשאלה זו תממשו אלגוריתם לדחיסת תמונות. נעסוק בתמונות בינאריות (שחור/לבן), כלומר לכל פיקסל יש אחד משני ערכים (255/0). נשתמש באבחנה כי כל שורה בתמונה מורכבת מרצפי פיקסלים בצבעים שונים: תחילה ברצף של פיקסלים לבנים (באורך 0 או יותר), ואחריו רצף של פיקסלים שחורים (באורך 1 או יותר), וכן הלאה.

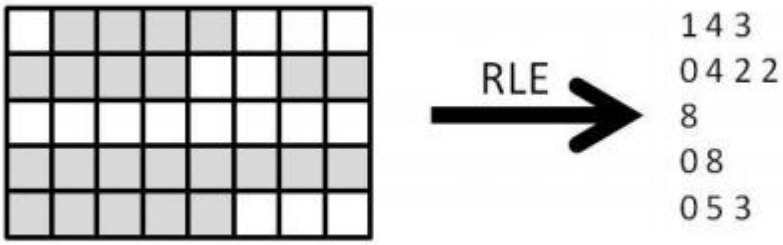
עליכם לממש "קידוד אורך חזרה" (או RLE, Run Length Encoding) המבוסס על הרעיון הבא:

קידוד ה-RLE של תמונה יהיה רשימה מקוננת.

- כל שורת פיקסלים בתמונה תיוצג ע"י רשימה:
  - האיבר הראשון יהיה אורך רצף הפיקסלים הלבנים בתחילת השורה.
  - האיבר השני יהיה אורך רצף הפיקסלים השחורים בהמשך.
  - וכך הלאה.
- סדר הרשימות הפנימיות הוא כסדר השורות בתמונה מלמעלה למטה.

ייצוג RLE של תמונה יהיה יעיל (ביחס לייצוג כל פיקסל באופן ישיר) אם בתמונה רצפים ארוכים של פיקסלים באותו צבע.

להלן דוגמא:



כלומר התמונה תקודד ע"י הרשימה: [[1,4,3],[0,4,2,2],[8],[0,8],[0,5,3]].

שימו לב:

- אורך הקידוד כל שורת פיקסלים בתמונה עשוי להיות שונה משורה לשורה (התמונה עצמה תמיד ריבועית)..
- כאשר שורה מתחילה בפיקסל שחור – האיבר הראשון בקידוד השורה יהיה 0 (מספר הפיקסלים הלבנים בתחילת השורה).
- הניחו שצבעים שחור ולבן מיוצגים כ- 0 ו-255 בהתאמה.
- רמז: המתודה append של רשימה מוסיפה איבר לסוף הרשימה.

סעיף א (18 נקודות)

ממשו את הפונקציה encode_rle(image), המקבלת תמונה בשחור-לבן כרשימה מקוננת (אין צורך לבדוק את תקינות הקלט), ומחזירה את קידוד ה-RLE המתאים.

```
def encode_rle(image):
    rle = [[] for i in range(len(image))]

    for x in range(len(image)):
        # count is the number adjacent pixels
in the current color
        count, current_color = 0, 255
        for y in range(len(image[0])):
            if image[x][y] == current_color:
                # additional pixel in the
current color
                count += 1
            else:
                # a pixel in a different color
                rle[x].append(count)
                current_color = 255 -
current_color
                count = 1
        # the pixels of the last color in the
line
        # are accounted for here
        rle[x].append(count)
    return rle
```

סעיף ב (14 נקודות)

ממשו את הפונקציה decode_rle(rle_image) אשר מקבלת תמונה מקודדת בקידוד RLE כפי שנעשה בסעיף הקודם, מחזירה את התמונה המקורית (בשחור-לבן) כרשימה מקוננת. הניחו כי הקלט תקין.

```
def decode_rle(rle_image):
    image = []

    for y in range(len(rle_image)):
        color = 255
        curr_row = []
        for count in rle_image[y]:
            for i in range(count):
                curr_row.append(color)
            color = 255 - color
        image.append(curr_row)

    return image
```

שאלה 4 (20 נקודות)

פולינום הוא ביטוי מהצורה:

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0 x^0$$

שבו x הוא משתנה ו-a_i הם **מקדמי הפולינום** (coefficients).

ה-n הגבוה ביותר שעבורו $a_n \neq 0$ נקרא **המעלה של הפולינום** (degree).

לדוגמא, הביטוי $3x^3 - 5x + 4.2$ מתאר פולינום ממעלה 3, שמקדמיו הם: [4.2 , -5.0 , 0 , 3.0].

סעיף א (5 נקודות)

הגדירו מחלקה (class) בשם Polynom וממשו את מתודת הבנאי __init__ שתקבל כקלט את רשימת המקדמים של הפולינום, ותאתחל שדה (attribute) מתאים. שימו לב, האיבר ה-i ברשימת הקלט הוא המקדם של x^i.

```
class Polynom:
    def __init__(self, coeffs):
        self.coeffs = coeffs
```

סעיף ב (7 נקודות)

חיבור פולינומים מתבצע ע"י חיבור המקדמים המתאימים בשני הפולינומים.

לדוגמא, חיבור של הפולינומים $3x^3 - 5x + 4.2$ עם $x^4 + 3x$ ייצור את הפולינום החדש $x^4 + 3x^3 - 2x + 4.2$.

ממשו את האופרטור + המחבר בין שני פולינומים ומחזיר פולינום חדש המכיל את תוצאת החיבור. הניחו שהקלט תקין. אין לשנות את self או פולינום הקלט.

```
def __add__(self, other):

    l1 = len(self.coeffs)
    l2 = len(other.coeffs)

    res = Polynom([0]*max(l1, l2))
    for i in range(l1):
        res.coeffs[i] += self.coeffs[i]

    for j in range(l2):
        res.coeffs[j] += other.coeffs[j]

    return res
```

סעיף ג (8 נקודות)

הכפלת פולינומים מתבצעת על ידי הכפלת כל איבר בפולינום אחד בכל אחד מאיברי הפולינום השני. התוצאה של הכפלת $a_i x^i$ באיבר $b_j x^j$ היא איבר שהמקדם שלו הוא מכפלת המקדמים והחזקה שלו: $(a_i * b_j)x^{(i+j)}$.

ממשו את האופרטור * המכפיל שני פולינומים ומחזיר פולינום חדש המכיל את תוצאת המכפלה. אין לשנות את self או פולינום הקלט

```
def __mul__(self, other):

    l1 = len(self.coeffs)
      l2 = len(other.coeffs)

      res = Polynom([0]*(l1+l2))
      for i in range(l1):
        for j in range(l2):
                res.coeffs[i+j] +=
self.coeffs[i]*other.coeffs[j]

      return res
```