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

תאריך המבחן:	22.01.2019
שמות המרצים:	ד"ר אסף זריצקי
שמות המתרגלים:	מר שגיא תובל, מר ניר פרידמן
שם הקורס:	מבוא לתכנות
מספר הקורס:	37211111
שנה: 2019	מועד א'
משך הבחינה:	שלוש שעות
חומר עזר:	דף A4 כתוב בכתב יד בשני הצדדים

אנא קראו היטב את ההוראות שלהלן:

- במבחן **שלוש** שאלות הכוללות סעיפי משנה. במבחן **106** 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 (32 נקודות)

אינדקס ג'קארד (Jaccard index) הוא מדד לדמיון בין שתי קבוצות A,B המוגדר כיחס בין האיחוד לחיתוך הקבוצות, $J(A,B)=\frac{|A\cap B|}{|A\cup B|}$. אינדקס ג'קארד בין קבוצות זהות יהיה 1, אינדקס ג'קארד לקבוצות ללא איברים משותפים יהיה 0. ככל שיש יותר איברים משותפים, כך שתי הקבוצות דומות יותר ואינדקס ג'קארד עולה. ניתן ליישם את מדד הדמיון הזה גם על מילים, ע"י ייחוס של כל מילה כאל קבוצה של תווים. בחישוב אינדקס ג'קארד בין מילים אין משמעות לסדר ולמספר החזרות של התווים במילה. לדוגמא, $J(\text{"abbc"},\text{"cad"})=\frac{2}{4}$, מאחר ו-a,c הם התווים המשותפים, וסה"כ יש 4 תווים שונים (a,b,c,d).

סעיף א (7 נקודות)

ממשו פונקציה jacc_similarity(str1,str2) אשר מקבלת שתי מחרוזות, ומחזירה את אינדקס ג'קארד ביניהן.

- ☒ הניחו כי הקלט תקין: המחרוזות אינן ריקות וכל התווים הם lower case.
- ☒ שימו לב ששתי המחרוזות עשויות להיות באורך שונה.
- ☒ תזכורת: מבנה הנתונים set של פייתון מקבל רשימה או מחרוזת בבנאי, ומייצג את איבריה כקבוצה (בלי חזרות וחשיבות לסדר).

```
def jacc_similarity(str1,str2):
    uni = len( set(str1+str2) )

    str1_set = set(str1)
    str2_set = set(str2)

    inter = 0
    for char1 in str1_set:
        for char2 in str2_set:
            if(char1 == char2):
                inter += 1
                break

    return inter/uni
```

סעיף ב (8 נקודות)

ממשו פונקציה build_jacc_matrix(word_array) אשר מקבלת כקלט מערך מטיפוס *numpy.array* המכיל מחרוזות, ומחזירה מטריצת דמיון מטיפוס *numpy.array*. תא i,j במטריצת הדמיון יכיל את אינדקס ג'קארד בין המילה ה-i למילה ה-j במערך הקלט. סדר העמודות והשורות יהיה זהה לסדר במערך הקלט word_array.

- ☒ הניחו כי המערך אינו ריק וכי כל האיברים בו הם מחרוזות לא ריקות תקינות לפי ההגדרה של הסעיף הקודם.
- ☒ מחרוזת לא תופיע פעמיים ברשימה.

<u>דוגמאת ריצה:</u>

עבור הקלט: word_array = ["abbc" , "cad" , "b" , "abcd"]

תוחזר המטריצה:

```
[[ 1.          0.5         0.33333333  0.75      ]
 [ 0.5         1.          0.          0.75      ]
 [ 0.33333333  0.          1.          0.25      ]
 [ 0.75        0.75        0.25        1.        ]]
```

```
def build_jacc_matrix(word_array):
    n = len(word_lst)
    dist_mat = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            curr_dist = jacc_similarity(word_lst[i] ,
word_lst[j])
            dist_mat[i,j] = curr_dist
            dist_mat[j,i] = curr_dist

    return dist_mat
```

סעיף ג (17 נקודות)

כעת תממשו מבנה נתונים שיאפשר גישה יעילה למחרוזות במערך על פי דמיון ג'קארד ממחרוזת קלט מאותו מערך. ממשו את הפונקציה closest_words(word_array, thresh) אשר מקבלת כקלט את:

- word_array, מערך מחרוזות הקלט מטיפוס *numpy.array*.
- thresh, סף לדמיון בין מחרוזות (מספר בין 0 ל-1).

הפונקציה תחזיר מילון D כאשר המפתחות שלו הם המילים מהמערך, והערך לכל מפתח הוא רשימה של כל המילים אשר הדמיון בין המפתח אליהן הוא לכל הפחות thresh, <u>ואינה כוללת את המילה עצמה</u>. על רשימת המילים (הערך במילון) להיות ממוינת כך שמילים בעלות אינדקס ג'קארד גבוה ביחס למילת המפתח יקדימו מילים פחות דומות.

דוגמא, בהינתן הקלט ["abbc" , "cad" , "b" , "abcd"] וסף של 0.5 נקבל מילון שמכיל, בין היתר, את המפתח "abc" ואת הערך המתאים D["abc"] = ["abcd","cad"].

- ☒ מותר להשתמש בפונקציות מהסעיפים הקודמים.
- ☒ במידה ולמחרוזות אין אף מילה שעומדת בסף, המפתח יצביע על רשימה ריקה.
- ☒ תזכורות:
  - השיטה array.argsort() מחזירה את האינדקסים לפי הסדר הממוין לכל שורה ב-array.
  - השיטה list.remove(val) מסירה את האיבר val מהרשימה במידה וקיים, אחרת זורקת שגיאה.

<u>שימו לב</u>: על הפתרון שלכם לכלול לא יותר מלולאה אחת! פתרון עם יותר מלולאה יחידי ב-8 נקודות לכל היותר.

<u>דוגמאת ריצה:</u>

עבור הקלט מהסעיף הקודם: word_array = ["abbc" , "cad" , "b" , "abcd"]

יתקבל הפלט:

```
D = { 'abbc': ['abcd', 'cad'],
      'cad': ['abcd', 'abbc'],
      'b': [],
      'abcd': ['cad', 'abbc']
    }
```

לנוחיותכם מודפסת מטריצת הדמיון המתאימה לקלט כדי להקל על בחינת הקלט:

```
[[ 1.          0.5         0.33333333  0.75        ]
 [ 0.5         1.          0.          0.75        ]
 [ 0.33333333  0.          1.          0.25        ]
 [ 0.75        0.75        0.25        1.          ]]
```

```
def closest_words(word_array, thresh):
    dist_mat = build_jacc_matrix(word_array)

    num_words = len(word_array)
    D = { }
    for w_num in range(num_words):
        curr_key = word_array[w_num]
        curr_row_rel_inds = dist_mat[w_num,:] >= thresh
        curr_row_rel_vals =
dist_mat[w_num,curr_row_rel_inds]
        curr_row_rel_words =
word_array[curr_row_rel_inds]

        curr_row_inds_sorted =
np.argsort(curr_row_rel_vals)[::-1]
        curr_row_rel_words_sorted =
list(curr_row_rel_words[curr_row_inds_sorted])

        curr_row_rel_words_sorted.remove(curr_key)
        D[curr_key] = curr_row_rel_words_sorted

    return D
```

שאלה 2 (37 נקודות)

בשאלה זו תממשו תכנה לניהול קבוצת פוטבול.

שחקן פוטבול (Football_Player) מוגדר ע"י השדות הבאים:

- name, שם מטיפוס מחרוזת.
- salary, עלות החוזה במיליוני דולרים מטיפוס int.
- performance_func, פונקציית ביצועים. פונקציית פולינומיאלית מהצורה $ax^2 + bx + c$ אשר מחשבת את הביצועים של השחקן על פי רמת המוטיבציה הנוכחית שלו, x (שהוא מספר חיובי). פונקציית המוטיבציה עשויה להיות בעלת מקדמים שונים לכל שחקן, ואף עשויה להשתנות במהלך הקריירה של שחקן.

סעיף א' (9 נקודות)

נתונה הגדרת המחלקה Football_Player והמימוש בנאי המחלקה. ממשו את השיטות הבאות:

- set_new_performance_func(self,a,b,c), אשר מעדכנת את השדה performance_func באמצעות המקדמים a,b,c.
- get_performance (self, x), אשר מקבלת את רמת המוטיבציה הנוכחית של השחקן, ומחזירה את רמת הביצועים שלו, המחושבת באמצעות השדה performance_func.

דוגמאת הרצה:

השורות הבאות ידפיסו למסך את הערך: 21

```
brady = Football_Player("Tom Brady" , 20)
brady.set_new_performance_func(2,5,3)
print(brady.get_performance(2))
```

```
class Football_Player:
    def __init__(self,name,salary,performance_func=None):
        self.name = name
        self.salary = salary
        self. performance_func = performance_func
```

```
    def set_new_performance_func(self,a,b,c):
        new_func = lambda x: a*(x**2) + b*x + c
        self.performance_func = new_func
```

```
    def get_performance (self, x):
        return self.performance_func(x)
```

סעיף ב' (8 נקודות)

קבוצת פוטבול מורכבת משחקני הגנה ומשחקני התקפה. ממשו את המחלקות Defense_Player ו-Offence_Player אשר יורשות מהמחלקה Football_Player. על המחלקות לרשת את מחלקת האב, בתוספת השדות והשיטות הבאים:

- Defense_Player מתמחה בהכשלת יריבים (tackle).
  - מעבר לאתחול שדות מחלקת האב, הבנאי מאתחל שדה חדש בשם total_tackles ל-0. שדות מחלקת האב מגיעים לפני השדה החדש בחתימת הבנאי.
  - שיטה בשם tackle(self) אשר מוסיפה אחד לשדה total_tackles.

- Offence_Player מתמחה בצבירת יארדים (מרחק שהשחקן רץ עם הכדור עד ששחקן הגנה הכשיל אותו).
  - מעבר לאתחול שדות מחלקת האב, הבנאי מאתחל שדה חדש בשם total_yards_gained ל-0. שדות מחלקת האב מגיעים לפני השדה החדש בחתימת הבנאי.
  - שיטה בשם run_yards(self, yards) אשר מקבלת את מספר היארדים ששחקן ההתקפה רץ עד שהוכשל, ומגדילה את השדה total_yards_gained בהתאם ל-yards. הניחו כי הקלט תקין.

דוגמאות הרצה:

הרצת השורות הללו:

```
moshe = Defense_Player("Moshe Cohen" , 15)
print(moshe.name)
print(moshe.salary)
moshe.tackle()
moshe.tackle()
print(moshe.total_tackles)

print(20*"*")

tzahi = Offense_Player("Tzahi Levy" , 17)
print(tzahi.name)
print(tzahi.salary)
tzahi.run_yards(3)
tzahi.run_yards(9)
print(tzahi.total_yards_gained)
```

תדפיס למסך את הפלט הבא:

```
Moshe Cohen
15
2
********************
Tzahi Levy
17
12
```

ממשו את המחלקה Defence_Player:

```
class Defense_Player(Football_Player):
    def __init__(self, name, cost, performance_func=None):
        Football_Player.__init__(self, name, cost, performance_func)
        self.total_tackles = 0



    def block(self):
        self.total_tackles += 1
```

ממשו את המחלקה Offence_Player:

```
class Offense_Player(Football_Player):
    def __init__(self, name, cost, performance_func=None):
        Football_Player.__init__(self, name, cost, performance_func)
        self.total_yards_gained  = 0


    def run_yards(self, yards):
        self.total_yards_gained  += yards
```

סעיף ג (20 נקודות)

קבוצת פוטבול מעסיקה יותר שחקנים מאשר ניתן לרשום למשחק. ליגת הפוטבול הגדירה תקרה על השכר המצטבר של שחקני הקבוצה אשר רשומים למשחק. קבוצה מעוניינת לבחור את השחקנים אשר סך רמת הביצועים (ע"פ המתודה get_performance) שלהם מקסימאלית.

ממשו את הפונקציה הרקורסיבית
choose_players(players, team_motivation, num_def, num_off, max_salary) אשר מקבלת כקלט:

- players, רשימה של Football_Player המועסקים בקבוצה.
- team_motivation, רמת המוטיבציה של כל שחקן בקבוצה. בסעיף זה נניח כי המוטיבציה אחידה לכל שחקן בקבוצה.
- num_def, מספר שחקני הגנה שיש לרשום למשחק.
- num_off, מספר שחקני התקפה שיש לרשום למשחק.
- max_salary, חסם עליון לשכר המצטבר של השחקנים הרשומים למשחק.

הפונקציה תחזיר את רמת הביצועים המצטברת המקסימלית מבין כל ההרכבים האפשריים תחת מגבלות הליגה.

<u>לדוגמא:</u>
עבור קבוצה הכוללת שחקנים עם התכונות הבאות (לטובת בהירות הדוגמא כל תכונה מוצגת באמצעות רשימה, במימוש עצמו כל המידע נמצא ברשימה players):

- ביצועים: [-1, 4, 13, 18, 27, 32, 41, 46, 55, 60]
- משכורות: [10, 9, 12, 7, 14, 5, 16, 3, 18, 80]
- תפקידים:

['Defense', 'Offense', 'Defense', 'Offense', 'Defense', 'Offense', 'Defense', 'Offense', 'Defense', 'Offense']

- num_def: 4 שחקני הגנה.
- num_off: 4 שחקני התקפה.
- max_salary: תקרת שכר של 100 מיליון דולר.

הפונקציה תחזיר 236, כל השחקנים למעט מהראשון (בגלל רמת ביצועים נמוכה) והאחרון (בגלל מגבלת תקרת השכר).

שימו לב:

- ☒ ניתן להגדיר פונקציות עזר.
- ☒ ניתן להניח כי קיימת חלוקה כלשהי שעומדת בתנאים.
- ☒ הפונקציה מקבלת רשימה של Football_Player, ולא את הרשימות שבדוגמא.

```
def choose_players(players, team_motivation, num_def,
num_off, max_salary):

    mot_level = 2
    return choose_players_helper(players, num_def,
num_off, max_salary , 0 , mot_level)




def choose_players_helper(players, num_def, num_off,
max_salary , sum_qual , mot_level):
    if len(players) == 0:
        if num_def == 0 and num_off == 0 and max_salary
>= 0 :
            return sum_qual
        else:
            return 0

    if num_def < 0 or num_off < 0 or max_salary < 0 :
        return 0
```

```
    dont_choose_curr =
choose_players_helper(players[1:], num_def, num_off,
max_salary , sum_qual , mot_level)



    new_sum_qual = sum_qual +
players[0].get_performance(mot_level)
    new_max_sal = max_salary - players[0].cost
    if isinstance(players[0] , Offense_Player ):

        choose_curr = choose_players_helper(players[1:],
    num_def, num_off - 1 , new_max_sal  ,
    new_sum_qual , mot_level)


    else:
        choose_curr =
choose_players_helper(players[1:], num_def - 1 ,
num_off, new_max_sal , new_sum_qual , mot_level)



    return max(choose_curr , dont_choose_curr)
```

שאלה 3 (37 נקודות)

בשאלה זו תממשו את המחלקה IsraeliQueue, מבנה נתונים מסוג תור בו האיברים מאוגדים בקבוצות. תור ישראלי מורכב מאובייקטים מטיפוס IsraeliQueueItem המכילים שלושה שדות: ערך (value) מכל טיפוס, חשיבות (priority) – מספר שלם חיובי ומספר קבוצה לה הוא שייך (group_id) – מספר שלם חיובי.

התור עצמו (IsraeliQueue) מכיל קבוצות חברים. קבוצה מורכבת מחברים מטיפוס IsraeliQueueItem עם שדה group_id זהה. בראש התור נמצאת הקבוצה שסכום העדיפויות (priority) של חבריה הוא הגבוה ביותר ובסוף התור הקבוצה עם סכום העדיפויות הנמוך ביותר. כאשר מתווסף חבר חדש (מטיפוס IsraeliQueueItem) הוא נכנס לקבוצה בתור לפי שדה ה-group_id או יוצר קבוצה חדשה בתור. למימוש IsraeliQueue השתמשו באחד ממבני הנתונים המובנים של פייתון לבחירתכם.

סעיף א' (7 נקודות)

צרו מחלקת IsraeliQueueItem וממשו את השיטות הבאות:

- __init__(self, value, priority, group_id), בנאי המחלקה מקבל כקלט את הערך, העדיפות ומספר הקבוצה. יש לוודא כי העדיפות (priority) ומספר הקבוצה (group_id) הינם מטיפוס int, אחרת יש לזרוק שגיאה מסוג TypeError עם ההודעה: "Invalid arguments types". אם העדיפות או מספר הקבוצה אינם מספר חיובי (גדול ממש מ-0) יש לזרוק שגיאה מסוג ValueError עם ההודעה: "Invalid arguments values".
- __repr__(self), ייצוג מחרוזתי של אובייקט. דוגמה - האובייקט עם הערך "Hello World" עדיפות 5 וקבוצה 6 ייוצג באמצעות המחרוזת הבאה: Value:Hello World, Priority:5, GroupID:6.

```
class IsraeliQueueNode:
    def __init__(self, value, priority, group_id):
        if not isinstance(priority, int) or not
isinstance(group_id, int):
            raise TypeError('Invalid arguments types')
        if priority < 0 or group_id < 0:
            raise ValueError('Invalid arguments
values')
        self.value = value
        self.priority = priority
        self.group_id = group_id


    def __repr__(self):
        return "Value:" + str(self.value) + ".
Priority:" + str(self.priority) + ". GroupID:" +
str(self.group_id) + "."
```

סעיף ב' (20 נקודות)

צרו מחלקת IsraeliQueue וממשו את השיטות הבאות:

- __init__(self), בנאי המחלקה.
- __len__(self), מספר הקבוצות בתור.
- enqueue(self, val, priority, group_id), הכנסת איבר חדש מסוג IsraeliQueueItem עם הערך val, העדיפות priority ומספר הקבוצה group_id.
- dequeue_item(self) – הוצאה והחזרה של האיבר עם העדיפות הגבוהה ביותר. ניתן להניח שהתור אינו ריק וכי אין שני איברים עם אותה עדיפות.
- dequeue_group(self) – הוצאה של כל חברי הקבוצה שסך העדיפויות של חבריה הינו הגבוה ביותר מבין כל הקבוצות בתור. החזרה של איברי הקבוצה במבנה נתונים מסוג ברשימה.

<u>תזכורת</u>: הפקודה del מאפשרת למחוק איברים ממבני הנתונים המובנים של פייתון. לדוגמה del lst[1] תמחק את האיבר באינדקס 1 ברשימה lst ו-del dic['a'] תמחק את המפתח 'a' והערך המתאים מהמילון dic.

<u>שימו לב</u>: **<u>אינכם נדרשים פה לפתרון יעיל</u>**, ממשו באופן הפשוט ביותר שיעמוד בתנאי השאלה – זה בסדר לממש כך שסיבוכיות הזמן של שיטה תהיה O(n), כלומר מעבר על כל איברי התור. ניתן לכתוב פונקציות עזר לפי הצורך.

```
class IsraeliQueue:
    """ Class that represents an Israeli queue"""
    def __init__(self):
        self.content = []

    def __len__(self):
        return len(self.content)


    def enqueue(self, val, priority, group_id):
        for i in range(len(self)):
            curr_group_id = self.content[i][0].group_id
            if group_id == curr_group_id:

self.content[i].append(IsraeliQueueNode(val, priority,
group_id))
                return
        self.content.append([IsraeliQueueNode(val,
priority, group_id)])
```

```
def dequeue_item(self):
        max_priority = 0
        item_to_remove = (0,)
        for group_ind in range(len(self.content)):
            for item_ind in
range(len(self.content[group_ind])):
                item =
self.content[group_ind][item_ind]
                if item.priority > max_priority:
                    max_priority = item.priority
                    item_to_remove = (group_ind,
item_ind)
        res =
self.content[item_to_remove[0]][item_to_remove[1]]
        del
self.content[item_to_remove[0]][item_to_remove[1]]
        # Remove group if empty
        if len(self.content[item_to_remove[0]]) == 0:
            del self.content[item_to_remove[0]]
        return res



    def group_total_priority(self, group):
        res = 0
        for item in group:
            res += item.priority
        return res
```

```
    def dequeue_group(self):
            max_group_ind = 0
            max_group_priorities = 0
            for group_ind, group in
    enumerate(self.content):
                if self.group_total_priority(group) >
    max_group_priorities:
                    max_group_ind = group_ind
                    max_group_priorities =
    self.group_total_priority(group)
            res = self.content[max_group_ind]
            del self.content[max_group_ind]
            return res
```

סעיף ג' (10 נקודות)

ממשו את הפונקציה sort_queue_by_priorities(queue) המקבלת איבר מטיפוס IsraeliQueue ומחזירה רשימה של איברי התור ממוינים בסדר עולה. יחס הסדר נקבע תחילה לפי מספר הקבוצה לה הם שייכים ולאחר מכן לפי העדיפות שלהם. ניתן להשתמש בפונקציה המובנית sorted ובפונקציות עזר במידת הצורך וכן ניתן לגשת לשדות האובייקט queue, אך לא לשנות אותם.

```
def sort_queue_by_priorities(queue):
    all_items = []
    for group in queue.content:
        for item in group:
            all_items.append(item)


    def comp(item):
        return item.group_id, item.priority

    return sorted(all_items, key=comp)
```