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 ואינכם 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 במודולים, 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 שישנם יותר איברים משותפים, כך שתי הקבוצות דומות יותר זו לזו, וערכו של אינדקס ג'קארד עולה.

ניתן ליישם את אינדקס ג'קארד גם על מילים, ע"י התייחסות למילים כאל קבוצות 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 ישנם 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 אותיות 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 וללא חשיבות לסדר). ניתן לעבור על אברי set באמצעות לולאת for, בדומה לרשימה.

```
def jacc_similarity(str1,str2):
```

סעיף ב (8 נקודות)

ממשו את הפונקציה 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 ההגדרות בסעיף הקודם.
- ☒ מחרוזת לא תופיע ברשימה יותר מפעם אחת.

<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.        ]]
```

כתבו את תשובתכם במסגרת שבעמוד הבא

```
def build_jacc_matrix(word_array):
```

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

- ☒ מותר להשתמש בפונקציות מהסעיפים הקודמים, גם אם לא פתרתם אותם.
- ☒ במידה ולמחרוזות אין אף מילה שעומדת בסף, הערך המתאים למפתח יהא 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 והוא קיים, אחרת זורקת שגיאה.

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

<u>דוגמאת ריצה:</u>

עבור הקלט מהסעיף הקודם: word_array = ["abbc" , "cad" , "b" , "abcd"], וסף thresh = 0.5

יתקבל הפלט:

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

לנוחיותכם מודפסת מטריצת הדמיון המתאימה לקלט הדוגמא:

```
[[ 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):
```

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 שלו.

סעיף א' (9 נקודות)

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

- set_new_performance_func(self,a,b,c), אשר מעדכנת את השדה performance_func באמצעות **פונקציה** המכילה את המקדמים a,b,c. **חובה** להשתמש בביטוי lambda.
- get_performance (self, x), אשר מקבלת את רמת המוטיבציה הנוכחית של השחקן, ומחזירה את רמת הביצועים שלו, המחושבת באמצעות הפונקציה שנשמרה בשדה performance_func.

<u>דוגמאת הרצה:</u>

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
```

סעיף ב' (8 נקודות)

קבוצת פוטבול מורכבת משחקני הגנה ומשחקני התקפה. ממשו את המחלקות Defense_Player ו-Offense_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.

- Offense_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 Defense_Player:

ממשו את המחלקה Offense_Player:

סעיף ג (20 נקודות)

קבוצת פוטבול מעסיקה יותר שחקנים מאשר ניתן לרשום למשחק בודד. ליגת הפוטבול הגדירה תקרה על השכר המצטבר של שחקני הקבוצה אשר רשומים למשחק כלשהו. קבוצה מעוניינת לבחור את השחקנים אשר סך רמת הביצועים שלהם (שמחושבת באמצעותהשיטה get_performance) 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 ומיוצגת בקריאה לפונקציה באמצעות ערך יחיד. תזכורת- ביצועי השחקן תלויים במוטיבציה שלו, וניתן להשיג אותם ע"י שימוש בשיטה get_performance.
- num_def, מספר שחקני ההגנה שיש לרשום למשחק.
- num_off, מספר שחקני ההתקפה 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.

לדוגמא:
עבור קבוצה הכוללת שחקנים עם התכונות הבאות (לטובת בהירות הדוגמא כל תכונה מוצגת באמצעות רשימה, במימוש עצמו כל המידע נמצא ברשימת האובייקטים 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.
- ☒ הפונקציה מקבלת רשימה שכל איבר בה הוא Football_Player, ולא את הרשימות שבדוגמא.
- ☒ ניתן להניח כי קיימת חלוקה כלשהי העומדת בתנאים.
- ☒ תזכורת: הפונקציה instanceof(x,y) מחזירה true אם המשתנה x מטיפוס y.

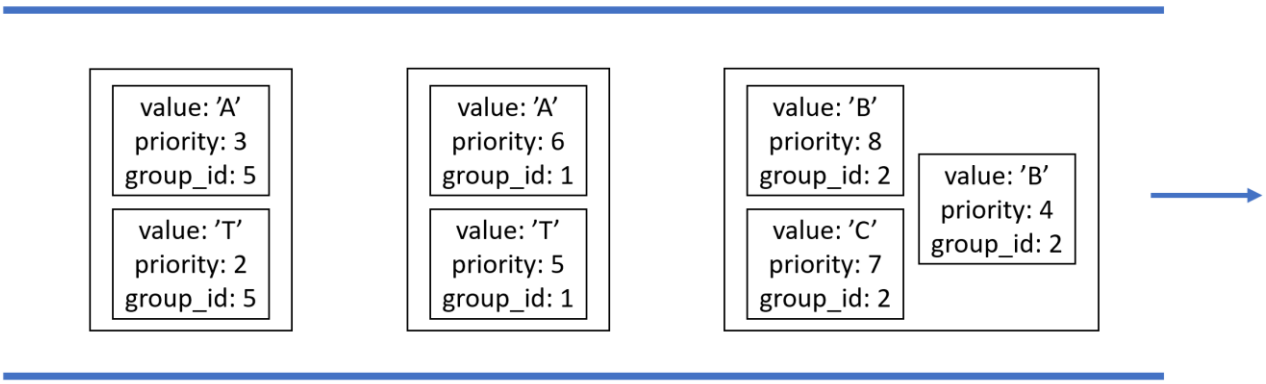
```
def choose_players(players, team_motivation, num_def,
num_off, max_salary):
```

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.

תור ישראלי לדוגמה:



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 מאפס) 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.

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 חברי הקבוצה שסך העדיפויות של חבריה הינו הגבוה ביותר מבין כל הקבוצות בתור. החזרה של האיברים 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.

הדרכה ורמזים: אין צורך לממש את מבנה הנתונים כך שיחזיק את הקבוצות לפי הסדר. אינכם נדרשים פה לפתרון יעיל ומומלץ לממש את מבנה הנתונים באופן הפשוט ביותר שיעמוד בתנאי השאלה, כך שהשיטות ידמו את ההתנהגות של IsraeliQueue – גם אם סיבוכיות זמן הריצה של השיטות תהיה O(n) - מעבר על כל איברי התור בפעולת הכנסת \ הוצאת איבר או קבוצה. ניתן לכתוב פונקציות עזר לפי הצורך.

דוגמת הרצה (הניחו כי השיטה __repr__ מומשה עבור IsraeliQueue):

```
IQ = IsraeliQueue()
IQ.enqueue('A', 3 , 1)
IQ.enqueue('B', 4 , 2)
IQ.enqueue('B', 7 , 2)
IQ.enqueue('B', 8 , 2)
IQ.enqueue('C', 5 , 1)
print(IQ)
```

```
Group 1 items:
1. Value:A, Priority:3, GroupID:1.
2. Value:C, Priority:5, GroupID:1.
Group 2 items:
1. Value:B, Priority:4, GroupID:2.
2. Value:B, Priority:7, GroupID:2.
3. Value:B, Priority:8, GroupID:2.
```

```
IQ.dequeue_item()
```

```
Value:B, Priority:8, GroupID:2.
```

```
IQ.dequeue_group()
```

```
[Value:B, Priority:4, GroupID:2., Value:B, Priority:7, GroupID:2.]
```

```
print(IQ)
```

```
Group 1 items:
1. Value:A, Priority:3, GroupID:1.
2. Value:C, Priority:5, GroupID:1.
```

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.