מס׳ מחברת : מס׳ ת.ז. :

סמסטר א׳ תשפ״ד
מועד ב׳, 24.5.2024

בחינה בקורס תכנות בפייתון למהנדסים 0509-1820

אביה עשהאל, חגי לוי, שחר צרפתי, אורן קציר, רותם פלח, ספיר שמש, רון סעד, עמר שפירא, ענבל חדד, איתי תשובה, שגיא שקד

הנחיות כלליות

- משך הבחינה שלוש שעות.
- חומר עזר מותר: דף A4 כתוב בשני הצדדים.
- במבחן שלוש שאלות הכוללות סעיפי משנה. במבחן 105 נקודות, כל ציון מעל 100 יעוגל ל-100. ניקוד כל סעיף מצוין לידו. אין בהכרח קשר בין ניקוד הסעיף ובין רמת הקושי שלו.
- את הפתרונות יש לכתוב במסגרות המסומנות לכל שאלה בטופס הבחינה.
- במחברת הטיוטא יש להשתמש כטיוטא בלבד. בבדיקת המבחן, מחברות הטיוטא לא ייבדקו.
- מומלץ לכתוב פתרון מלא במחברת הטיוטא ורק לאחר מכן להעתיק את הפתרון לטופס הבחינה.
- בכל סעיף ניתן להשתמש בקוד שהתבקשתם לכתוב בסעיפים קודמים, גם אם לא כתבתם אותו.
- ניתן להוסיף פונקציות עזר אם לא צוין אחרת בשאלה.
- ניתן להניח שהקלט לקוד שתכתבו תקין, אלא אם נכתב אחרת בשאלה.
- על כל סעיף ניתן לענות ״איני יודע/ת את התשובה״; על סעיף זה יינתנו 20% מהנקודות. במקרה זה אין להוסיף הסבר.
- אין להשתמש בחבילות או מודולים, אלא אם נאמר במפורש. כאשר אתם מתבקשים להשתמש בחבילה אין צורך לבצע import.
- יש לרשום את מספר תעודת הזהות ואת מספר הטופס בכל עמוד למקרה שחלק מהדפים יתלשו.
- מומלץ לקרוא כל שאלה עד סופה, על כל סעיפיה, לפני תחילת הפתרון.
- שימו לב כי מגבלת שורות במימוש מתייחסת לשורות כפי שנכתבות בעורך קוד

בהצלחה!

לשימוש הבודקים:

תעודת זהות	
מספר טופס	
שאלה 1	
שאלה 2	
שאלה 3	
ציון	

מס' מחברת : מס' ת.ז. :

שאלה 1 - (35 נקודות)

סעיף א' (4 נקודות). פתרון הסעיף חייב להיות רקורסיבי, אך ניתן להשתמש בלולאות. יש לממש את הסעיף ב10 שורות לכל היותר ממשו את הפונקציה get_n_z(lst) המקבלת רשימה של רשימות מקוננות. הרשימות מכילות גם מספרים מטיפוס int וגם float. הפונקציה תחזיר את מספר המספרים **השלמים** המופיעים ברשימה.
שימו לב: גם המספר 1 וגם המספר 1.0 נחשבים מספרים שלמים.
לדוגמא:

```
>>> get_n_z([1, [2, 3, 3.5, [4.0]]])
4
```

```
def get_n_z(lst):
```

סעיף ב' (6 נקודות). יש לפתור סעיף זה ללא שימוש ברקורסיה וללא שימוש בתנאים (if-else).יש לממש את הסעיף ב-3 שורות לכל היותר
נגדיר זוגות מספרים לא-צמודים ברשימה כזוג מספרים אשר האינדקסים שלהם ברשימה אינם עוקבים.
לדוגמא, ברשימה [1,52,4], קיים זוג אחד לא צמוד: 1 ו4.
ממשו את הפונקציה get_all_doublets המקבלת את הרשימה lst המכילה מספרים מטיפוס int ואת out - רשימה ריקה. הפונקציה **תעדכן** את out כך שתכיל את כל זוגות המספרים הלא-צמודים-האפשריים מתוך רשימת הקלט **עם חשיבות לסדר** (ראו דוגמא בהמשך). הפלט של הפונקציה יהיה רשימה של tuples, כאשר כל tuple מכיל את אחת מהאפשרויות לבחירת זוג איברים מרשימת הקלט.
הנחיות:

- ישנם לפחות שני מספרים ברשימה lst.
- אין שני מספרים זהים ברשימה lst.
- יש חשיבות לסדר זוגות המספרים בתוך כל tuple, אך אין חשיבות לסדר הtuples ברשימה. לדוגמא:
  - עבור הקלט lst=[1,52], הערך של out לאחר ריצת הפונקציה יהיה: []
  - עבור הקלט lst=[1,52,4] , ערך אפשרי של out לאחר ריצת הפונקציה יהיה: [(1,4),(4,1)]
  - עבור הקלט lst=[1,5,2,4] ערך אפשרי של out יהיה: [(1, 2), (1, 4), (5, 4), (2, 1), (4, 1), (4, 5)]

  לשם עדכון הרשימה out, מותר לכם להשתמש רק במתודה append **פעם אחת בלבד**. כלומר, מותר שהמילה append תופיע בקוד פעם אחת בלבד.
- אין להשתמש בשרשור רשימות, ובמתודה extend ובכל דרך אחרת לשנות את הרשימה out מלבד append

רמז: השתמשו בשרשור של range שונים באותה לולאה ע"י הפיכתם לרשימה:
לדוגמא,
list(range(3))+list(range(2)) → [0,1,2,0,1]

```
def get_all_doublets(lst, out):
```

סעיף ג'. בעקבות הצלחת הקורס "תכנות למהנדסים", החליט ועד הנדסה לקנות לצוות הקורס מתנות. הוחלט להקצות לעניין סכום כסף שעומד בין x ל–y שקלים (x ≤ y). בשאלה זו תבחרו מתנות תחת עמידה במגבלות שונות. נתייחס לכל מתנה כ-tuple בעל 2 איברים: האיבר הראשון מייצג את מחיר המתנה (בשקלים) והאיבר השני את משקלה (בקילוגרמים).
לדוגמא, המתנה (4,1.2) עולה ארבעה שקלים ושוקלת 1.2 קילוגרמים.

1. (6 נקודות). פתרון הסעיף צריך להיות רקורסיבי.
ממשו את הפונקציה select_presents(presents, min_budget, max_budget) המקבלת שלושה פרמטרים:
- presents – רשימת מתנות באורך n. למשל: [(10, 2.5), (50, 1.1), (20, 3), (100, 5.2)]
- min_budget –הסכום המינימלי המוקצה לקניית מתנות.
- max_budget –הסכום המקסימלי המוקצה לקניית מתנות.

על הפונקציה להחזיר את מספר האפשרויות לבחור מתנות מתוך הרשימה, כך שיעמדו בטווח התקציב, כלומר את מספר אפשרויות הבחירה (תתי-קבוצות של מתנות) בהן סכום מחירי המתנות לא נמוך מהסכום המינימלי ולא גבוה מהסכום המקסימלי.
שימו לב:
- ניתן לבחור כל מתנה פעם אחת.
- ניתן לבחור גם תת-קבוצה ריקה של מתנות.
- זכרו כי ניתן להשתמש בפונקציות עזר.

דוגמאות:
- עבור רשימת המתנות [(10, 2), (50, 1), (20, 3), (100, 5)]
  - לא ניתן להגיע לסכום מתנות בטווח 81-99:

```
>>> select_presents ( [(10,2),(50,1),(20,3),(100,5)], 81, 99 )
0
```

  - ניתן להגיע לסכום מתנות בתחום 20-35, עם האפשרות [(10, 2), (20, 3)], ועם האפשרות [(20, 3)].

```
>>> select_presents ([(10,2),(50,1),(20,3),(100,5)], 20, 35 )
2
```

```
def select_presents(presents, min_budget, max_budget):
```

2. (7 נקודות).
נגדיר כי שווי מתנה x קטן משוויה של מתנה y אם המשקל של x חלקי המחיר של x קטן מהמשקל של y חלקי המחיר של y ($\frac{weight\ of\ x}{cost\ of\ x} < \frac{weight\ of\ y}{cost\ of\ y}$)
ממשו את הפונקציה merge_lists_of_gifts(left, right) המקבלת שתי רשימות של מתנות, הממוינות בסדר שווי עולה. הפונקציה תחזיר רשימה חדשה, שהיא איחוד ממוין של שתי רשימות הקלט. כלומר, רשימה **ממוינת** לפי שווי שמכילה את כל האיברים שבשתי רשימות הקלט, ואורכה כסכום האורכים שלהן.
הנחיות:

- אין להשתמש בפונקציות sort, sorted וכו'
- ניתן להשתמש בלולאה אחת לכל היותר

שימו לב: עליכם להחזיר את **רשימת מתנות** ולא רשימה של שווי המתנות.
דוגמת הרצה:

```
>>> print(merge_lists_of_gifts([(50,1), (10,2)],[(100,5), (20,3)]))
 [(50, 1), (100, 5), (20, 3), (10, 2)]
```

```
def merge_lists_of_gifts(left, right):
```

3. (7 נקודות). פתרון הסעיף צריך להיות רקורסיבי. ממשו בפונקציה merge_sort את אלגוריתם המיון merge sort עבור רשימת מתנות.
המימוש ייעשה לפי ההנחיות הבאות
בהינתן רשימה באורך 2 איברים או יותר:

1. חלקו את האיברים ברשימה לשתי תתי-רשימות באורך שווה (או כמעט שווה – חשבו מתי)
2. מיינו כל אחת מתתי-הרשימות בעזרת merge_sort (באופן רקורסיבי)
3. אחדו את תתי-הרשימות הממויינות לרשימה הממויינת הסופית

רמז: העזרו בתת סעיף קודם

מס' מחברת : מס' ת.ז. :

```
def merge_sort(lst):
```

4. (5 נקודות). מה תדפיס שורת ההרצה הבאה?

```
>>> l1= [(1,3),(2,2),(2,1)]
>>> l2= merge_sort(l1)
>>> print(sum([l1[a][1]-l2[-a][0] for a in range(len(l1))]))
```

הקיפו את התשובה הנכונה

1. -2
2. -1
3. 0
4. 1
5. 2
6. Error

מס' מחברת : מס' ת.ז. :

שאלה 2 (35 נקודות)

סעיף א' (6 נקודות):
ממשו את מחלקה **Player** המייצגת שחקן בענף ספורט קבוצתי כלשהו. המחלקה תכלול את המתודות הבאות:
1. __init__(self, name, salary, history). המתודה תקבל את הקלט הבא:
   - *name*: מחרוזת המייצגת את שם השחקן.
   - *salary: מספר מטיפוס float* המייצג את משכורת השחקן.
   - *history*: מילון **או** רשימה של tuples המייצגות את הזכיות השונות של השחקן בתארים.
     - במידה והועבר מילון, המפתח ייצג את סוג התואר והערך ייצג את מספר הזכיות בסוג תואר זה. לדוגמא: `{'championship':2, 'cup':3}`.
     - במידה והועברה רשימה של tuples, האיבר הראשון בtuple הוא סוג התואר, והאיבר השני הוא **שנת הזכיה**. ניתן להניח שאיבר לא מופיע ברשימה יותר מפעם אחת. לדוגמא:

```
[("cup", 2019), ("championship", 2020), ("championship", 2021),
 ("Cup", 2013), ("CUP", 2024)]
```

בבנאי עליכם ליצור שדות המכילים את שם השחקן ואת משכורתו לפי הערכים שהתקבלו בקלט (ללא עיבוד נוסף).
בנוסף, עליכם ליצור את השדה *titles* ולשמור בו **מילון** הממפה שמות תארים **למספר** הזכיות בתואר. **עליכם להפוך את שמות כל התארים לlowercase.** בדוגמאות של המילון והרשימה שהובאו לעיל יישמר בשדה titles מילון זהה:

```
{'championship':2, 'cup':3}
```

<u>שימו לב</u>:
- עבור רשימת זכיות ריקה יתקבל מילון ריק
- אין לשמור שדות נוספים מעבר לאלו שצוינו לעיל

2. __repr__(self) **יש לממש מתודה זו בשורה אחת בלבד**
המתודה תחזיר מחרוזת המייצגת את האובייקט בפורמט הבא (ראו דוגמא למטה):
Name: <שם השחקן>, Number of titles: <מספר הזכיות הכולל של השחקן>

```
>>> print(Player("messi", 3, [("championship", 2020), ("championship", 2022),
("best_player", 2024)]))
Name: messi, Number of titles: 3
>>> print(Player("ronaldo", 4, {'championship':2, "cup":3}))
Name: ronaldo, Number of titles: 5
```

```
class Player:
    def __init__(self, name, salary, history):




   def __repr__(self):
```

סעיף ב' (6 נקודות):
כעת נרחיב את המחלקת Player. ממשו את המתודות הבאות:
1. *evaluate(self, weights)* **יש לממש מתודה זו ב3 שורות לכל היותר**
המתודה מחזירה את שוויו של השחקן אשר יוגדר באופן הבא: המתודה מקבלת מילון של משקלים בשם *weights*, אשר מציין עבור **כל** תואר אפשרי את משקלו בחישוב שווי השחקן. עבור כל תואר בו זכה השחקן, נכפיל את מספר הזכיות במשקל התואר. שווי השחקן מתקבל על ידי הסכימה של שורשי המכפלות חלקי משכורת השחקן:

$$playerValue = \frac{\sum_i\left(\sqrt{weights_i \cdot titleTimes_i}\right)}{salary}$$

- יש לעגל את התוצאה ל-2 ספרות אחרי הנקודה העשרונית.

לדוגמא: עבור אובייקט Player עם משכורת 4 וערך השדה history {'championship':3, 'cup':2}, ועבור מילון המשקלים הבא {'championship':5, 'cup':3} שווי השחקן הינו

$$\frac{\sqrt{5\cdot3}+\sqrt{3\cdot2}}{4} = \frac{(\sqrt{15}+\sqrt{6})}{4} = 1.58$$

<u>שימו לב</u>: קיים מילון דיפולטי לweight בחתימת/כותרת המתודה, וניתן להניח שכל התארים האפשריים מופיעים שם.
2. ממשו את המתודה __lt__(self, other) . **יש לממש מתודה זו בשורה אחת בלבד**
המתודה תחזיר True אם השחקן other בעל שווי (כפי שהוגדר בתת-הסעיף הקודם עם מילון המשקלים הדיפולטי) גדול משוויו של השחקן ממנו הופעלה המתודה. אם שוויים של השחקנים זהה, המתודה תחזיר True רק בתנאי ששם השחקן other גדול לקסיקוגרפית (כלומר, לפי ASCII) משם השחקן ממנו הופעלה המתודה. אחרת, המתודה תחזיר False.

```
>>> p1=Player("messi", 3, [("championship", 2020), ("championship", 2022),
("best_player", 2024)])
>>> print(p1.evaluate())
1.72
>>> p2=Player("ronaldo", 4, {'championship':2, "cup":3})
>>> print(p2.evaluate())
1.54
>>> print(p1<p2)
False
```

```
class Player:
…
    def evaluate(self, weights={'championship':5, 'best_player':4, 'cup':3}):



   def __lt__(self, other):
```

סעיף ג' (6 נקודות):
סטודנט מימש במחלקה Player יכולת לחבר שני שחקנים על ידי הסימן '+' כך שתחזיר שחקן חדש עם התכונות הבאות:

- שם השחקן החדש הינו שרשור שמות השחקנים המחוברים (על פי סדר החיבור) עם מקף בין השמות.
- משכורת השחקן החדש היא סכום המשכורת **אך לא מעל 10.** במידה והסכום גבוה מ-10, משכורת השחקן החדש תהיה 10.
- מילון התארים החדש מתקבל ע"י איחוד המילונים של שני השחקנים המחוברים:
  - כל תואר שהופיע באחד מהמילונים יופיע במילון השחקן החדש.
  - מספר הזכיות בתואר הינו סכום מספרי הזכיות של שני השחקנים המחוברים.

```
def __add__(self, other):
   new_name = self.name + "-" + other.name
   new_salary = max(10, self.salary+other.salary, key=lambda a: -a)
   new_titles = {}
   for cur_titles in [self.titles, other.titles]:
      for title in cur_titles:
         new_titles[title]= new_titles.get(title)+cur_titles[title]
   Player(new_name, new_salary, new_titles)
```

במימוש נפלו 2 טעויות. סמנו **במדויק** את הטעויות, וכתבו כיצד יש לתקנן. על כל טעות ותיקון להיות באורך 7 תווים לכל היותר.

_____

_____

_____

סעיף ד' (9 נקודות):
ממשו את מחלקה **Team** המייצגת קבוצה של שחקנים. המחלקה תכלול את המתודות הבאות:
1. __init__(self, name, players, salary_threshold)
name: מחרוזת המייצגת את שם הקבוצה.
players: רשימה של אובייקטים מסוג Player
salary_threshold: מספר מטיפוס float המייצג הפרדה בין שחקנים "יקרים" ל"זולים". שחקנים שמשכורתם לפחות salary_threshold ייחשבו כ"יקרים", אחרת, ייחשבו השחקנים "זולים".
על הבנאי לאתחל את ארבעת השדות הבאים בלבד:

- שם הקבוצה בשדה בעל שם תואם. **יש לוודא כי שם הקבוצה שהתקבל כקלט מכיל אותיות בלבד. אחרת, שם הקבוצה יוגדר כמחרוזת "default".**
- **2 שדות** cheap_players וexpensive_players המכילים רשימות של אובייקטי שחקנים "זולים" ו"יקרים", בהתאמה, לפי ההגדרה לעיל.
- total_value המייצג את סכום השווי עם מילון המשקלים הדיפולטי (כפי שהוגדר בסעיף ב') של **כל** שחקני הקבוצה.

2. __repr__(self) **יש לממש מתודה זו בשורה אחת בלבד.**
המתודה תחזיר מחרוזת המייצגת את האובייקט בפורמט הבא (ראו דוגמא למטה):
Name: <שם הקבוצה>, Is cheap team: <yes/no>
שימו לב: המחרוזת המייצגת את האובייקט תכיל "Is cheap team: yes" אם יותר ממחצית משחקניה זולים.
אחרת, המחרוזת תכיל "Is cheap team: no"
3. __contains__(self, player) **יש לממש את מתודה ב4 שורות לכל היותר.**
המתודה תחזיר True אם שם השחקן של האובייקט player זהה לשם אחד מהשחקנים המשויכים לקבוצה. אחרת, המתודה תחזיר False.
דוגמת הרצה:

```
>>> p1=Player("messi", 4, [("championship", 2010), ("championship", 2011),
("best_player", 2013)])
>>> p2=Player("ronaldo", 3, {'championship':2, "cup":3})
>>> p3=Player("zehavi", 2, [("championship", 2011)])
>>> p4=Player("peretz", 1, [("best_player", 2013)])
>>> t1=Team("galacticos", [p1,p4,p3,p2], 3)
>>> print(t1)
Name: galacticos, Is cheap team: no
>>> t2=Team("macabbi", [p3,p4], 3)
>>> print(t2)
Name: macabbi, Is cheap team: yes
>>> print(p1 in t2)
False
>>> print(p3 in t2)
True
```

```
class Team:
  def __init__(self, name, players, salary_threshold):









  def __repr__(self):


  def __contains__(self, player):


```

סעיף ה' (8 נקודות)

כעת נרחיב את המחלקה Team. ממשו את המתודה הבאה:

move_player(self, other, name)

המתודה תעביר את השחקן בעל השם name מהקבוצה ממנה הופעלה המתודה, אל הקבוצה other.
ניתן להניח כי לא קיימים 2 שחקנים בעלי שם זהה בתוך ובין הקבוצות.
אם לא קיים שחקן בשם זה, על המתודה להדפיס "Name was not found" בלבד.
אם קיים שחקן בשם name, המתודה תמחק את המופע שלו מהשדות באובייקט ממנו הופעלה המתודה, ותוסיף אותו לאובייקט other לפי הכלל הבא:
אם המשכורת של השחקן בשם name גבוהה מהמשכורת המקסימלית של השחקנים הזולים בקבוצה other, השחקן יתווסף לרשימת השחקנים היקרים בother. אחרת, הוא יתווסף לרשימת השחקנים הזולים.
שימו לב: יש לעדכן את כל השדות באובייקטים מסוג Team בהתאם לשינויים שנעשו ברשימות השחקנים שלהן כך שהם יעמדו בתנאים שהוגדרו בסעיף הקודם (חישבו מה צריך לעדכן וכיצד).

```
class Team:

    def move_player(self, other, name):
```

שאלה 3 (35 נקודות)

Numpy

הניחו כי לפני הסעיפים בחלק זה רצה השורה הבאה :

```
import numpy as np
```

1. (4 נקודות) יש לממש שאלה זו ב-8 שורות לכל היותר. אין להשתמש בלולאות ובתנאים. ממשו את הפונקציה convert_to_binary המקבלת מטריצה המייצגת תמונה בגווני אפור, ומחזירה **מטריצה חדשה** המייצגת תמונה בינארית (שחור-לבן ללא ערכי אפור, כאשר שחור מיוצג על ידי 0 ולבן על ידי 255). התמונה הבינארית תיוצר באופן הבא:

- תחילה מצאו את ערך הפיקסל הנפוץ ביותר (שחוזר הכי הרבה פעמים בתמונה). ניתן להניח כי קיים ערך יחיד כזה.
- כעת, שימו פיקסלים לבנים בתמונה הבינארית בכל מקום בו הערך המקורי של הפיקסל היה גדול או שווה לערך של הפיקסל הנפוץ ביותר. ובשאר המקומות שימו פיקסלים שחורים.

העזרו בפונקציות הבאות:

- **המתודה** ()flatten של מחלקה המייצגת מטריצות בnumpy. המתודה מחזירה מטריצה חדשה, שמייצגת וקטור חד מימדי (מערך) המורכב משרשור השורות של המטריצה המקורית. לדוגמא:

```
>>> ar=np.array([[1,2],[3,4]])
>>> ar.flatten()
>>> array([1,2,3,4])
```

- הפונקציה np.bincount(x), המקבלת וקטור (מערך) של מספרים אי שליליים מטיפוס int מחזירה את מספר המופעים של כל מספר בטווח 0 עד max(x) (כולל), לפי הסדר הערכים. לדוגמא:

```
>>> ar=np.array([1,3,4,3])
>>> np.bincount(ar)
array([0,1,0,2,1])
```

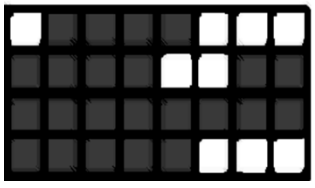
```
def convert_to_binary(im):
```

בסעיפים הבאים תממשו אלגוריתם לדחיסת תמונות (כלומר, הפיכת ייצוג תמונה לכזה הדורש פחות במקום בזיכרון). אנו נתמקד בדחיסת תמונות בינאריות (כלומר, שחור-לבן בלבד ללא גווני אפור). נשתמש באבחנה כי כל שורה בתמונה מורכבת מרצפי פיקסלים בצבעים שונים: תחילה מרצף של פיקסלים שחורים או לבנים, לאחריו רצף של פיקסלים בצבע ההפוך וכו'.

קידוד של תמונה (המיוצגת ע"י מטריצה) ימומש באמצעות וקטור באופן הבא:

- **עבור כל שורה במטריצה המקורית** נשים את אורך רצף הפיקסלים **הלבנים** שבתחילתה.
  - כאשר שורה מתחילה בפיקסל שחור – האיבר הראשון בקידוד יהיה 0 (מכיוון שיש 0 פיקסלים לבנים בתחילת השורה).
  - ניתן להניח כי לא קיימת שורת פיקסלים שכולה לבנה.
- לאחריו, נשים את אורך רצף הפיקסלים **השחורים**.
- לאחריו, נשים את אורך רצף הפיקסלים הלבנים הבא וכו'.
- כך נמשיך בצורה מחזורית עוד סוף השורה.
- **בין כל 2 שורות, נכניס את האיבר 1-.**

להלן דוגמא:



143-10422-108-1053

2. (**5 נקודות) בסעיף זה לא ניתן להשתמש בלולאות ובתנאים. יש לפתור סעיף זה ב-5 שורות לכל היותר.** ממשו את הפונקציה encode_row המקבלת כקלט שורה בודדת (וקטור/מערך) מתמונה בינארית. המתודה תחזיר וקטור (מספריית numpy) המייצג דחיסה של השורה כמתואר בתחילת השאלה.
לרשותכם עומדת הפונקציה encode_row_helper המקבלת שורה מתמונה בינארית ומחזירה וקטור המכיל את כל האינדקסים בהם הייתה החלפה (כלומר, ערך הפיקסל שונה מזה שלפניו).
דוגמאות:

- עבור השורה הראשונה בתרשים לעיל, הפונקציה encode_row_helper תחזיר את הוקטור [1,5].
- עבור השורה השניה בתרשים לעיל, הפונקציה encode_row_helper תחזיר את הוקטור [0,4,6].

שימו לב כי אם השורה מתחילה בפיקסל שחור, אזי לפי ההגדרה לעיל יחזור הערך 0 בתחילת וקטור הפלט.
רמז: זכרו לטפל במקרי קצה בקצוות השורה.

דוגמת הרצה:

```
>>> im_row =[255,0,255,255]
>>> print(encode_row(im_row))
[1, 1, 2]
```

```
def encode_row(im_row):
```

3. **(4 נקודות) יש לפתור סעיף זה ב6 שורות לכל היותר.** ממשו פונקציה בשם encode המקבלת כקלט את פלט סעיף 1 (המטריצה הבינארית). תחזיר וקטור (מספריית numpy) המייצג דחיסה של התמונה כמתואר בתחילת השאלה.

```
>>> im_bin=np.array([[0,0,255,255],
 [255,0,255,255],
 [0,0,255,255]])
>>> encode(im_bin)
[ 0.,  2.,  2., -1.,  1.,  1.,  2., -1.,  0.,  2.,  2.]
```

```
def encode(im):
```

4. (3 נקודות) יחס הקידוד יוגדר להיות היחס בין מספר הפיקסלים בתמונה הדחוסה למספר הפיקסלים בתמונה המקורית. בדוגמא לעיל, מכיוון שבדוגמא היו 32 פיקסלים שקודדו ל-15 ערכים בוקטור הדחוס
([1,4,3,**-1**,0,4,2,2,**-1**,0,8,**-1**,0,5,3]), יחס הקידוד 12/32 מכיוון ששלושת המופעים של הספרה -1 אינם נלקחים בחשבון.

סטודנט מימש את הפונקציה encoding_rate המקבלת מטריצה המייצגת תמונה בינארית (im) ותמונה מקודדת ומחזירה את יחס הקידוד שלה באופן הבא:

```
def calc_ratio(im,im_enc):
    return np.sum(im_enc[im_enc!=-1])/im.size
```

על מנת לבדוק את הפונקציה, הסטודנט הריץ את קטע הקוד הבא:

```
>>> im=np.array([[0,1,2,3],
[2,1,2,3],
[0,1,2,3]])
>>> im_bin=convert_to_binary(im)
>>> im_enc=encode(im_bin)
>>> print(calc_ratio(im_bin,im_enc))
```

מה תדפיס השורה האחרונה (בתצוגה עשרונית)?

1. 3/12
2. 9/12
3. Error
4. 1
5. 6/12
6. 0

Pandas

בחלק זה אין להשתמש בלולאות. הניחו כי לפני הסעיפים בחלק של Pandas רצה השורה הבאה :

```
import pandas as pd
```

בסעיפים הבאים נחשב ציונים בקורס פייתון למהנדסים.
נתון כי טבלת ציוני תרגילי הבית מוגדרת בפורמט הבא (כל העמודות מטיפוס int):

student_id	hw1	hw2	hw3	….	hw<n>
10	70	100	90	….	80
20	0	60	70	….	100
30	60	60	0	….	60

כאשר n הוא מספר התרגילים שניתן בסמסטר. לדוגמא, אם ניתנו 5 תרגילים, שם העמודה האחרונה הוא hw5.
העמודה הראשונה מייצגת את מזהה הסטודנט ושאר העמודות הם ציונים בתרגילי הבית.

5. **(2 נקודות) יש לממש סעיף זה ב5 שורות לכל היותר.** ממשו את הפונקציה read_grades_by_semester(semester_id, min_grade), המקבלת מחרוזת המייצגת מזהה סמסטר, וציון תרגילי בית מינימלי מטיפוס int. הפונקציה תטען קובץ בשם "<מזהה סמסטר>.csv". לדוגמא, עבור המזהה "2324a", שם הקובץ יהיה "2324a.csv". בנוסף, תחליף את ערכי מטלות שקיבלו 0 לmin_grade, ותחזיר את הטבלה המעודכנת. ניתן להניח שמספרי הזיהוי של הסטודנטים גדולים ממש מאפס.

```
def read_grades_by_semester(semester_id, min_grade):
```

6. **(4 נקודות) יש לממש סעיף זה ב4 שורות לכל היותר.**
נגדיר **טבלת איחורים בהגשות התרגילים** בפורמט הבא (כל העמודות מטיפוס int):

student_id	late1	late2	late3	….	late<n>
10	1	3	5	….	11
30	10	20	30	….	40

העמודה הראשונה מייצגת את מזהה הסטודנט ושאר העמודות מכילות את מספר ימי האיחור שהצטברו **מהתרגיל הראשון ועד לתרגיל הבית התואם לעמודה**. למשל, הסטודנט בעל מספר הזיהוי 30, איחר ב-10 ימים לתרגיל הראשון, וב-10 ימים לתרגיל השני, ולכן עבור התרגיל השני מספר ימי האיחור המצטבר שלו הוא 20.

ממשו את הפונקציה fuse_late_days_per_hw(semester_id, df_late_days, min_grade) המקבלת את semester_id ו־min_grade, בדומה לסעיף הקודם. בנוסף, הפונקציה תקבל טבלה (מטיפוס DataFrame) של איחורים בהגשות התרגילים בפורמט שהוגדר לעיל. הפונקציה תקרא את קובץ הציונים כפי שתואר בסעיף הקודם, ותחבר בין עמודות טבלה זו לעמודות טבלת האיחורים בתרגילים (df_late_days) כך שעמודות הטבלה החדשה יהיו מזהה הסטודנט, ציוני תרגיליו ומספר ימי האיחור אותם צבר עד כל תרגיל **בסדר זה**.
שימו לב:

- סטודנט שלא איחר בשום תרגיל אינו מופיע בטבלת האיחורים
- לא קיים סטודנט שמופיע בטבלת האיחורים אך לא מופיע בטבלת הציונים
- יש להמנע משינוי סדר העמודות והכנסת עמודות מיותרות בטבלת הפלט.

```
def fuse_late_days_per_hw(semester_id, df_late_days, min_grade):
```

7. **(4 נקודות) יש לממש סעיף זה ב5 שורות לכל היותר.** ממשו את הפונקציה late_penalty(df_merged_late, n_hw, grace_days, min_grade) את הפרמטרים הבאים:
- df_merged_late: טבלת הפלט מהסעיף הקודם
- n_hw: מס' שיעורי הבית שפורסמו בסמסטר (int)
- grace_days: מספר "ימי החסד", כלומר, ימים בהם ניתן לאחר בכל התרגילים ביחד כפי שהוגדר בסמסטר זה (int)
- min_grade: ציון מינימלי לתרגיל (int)

הפונקציה תחליף את הציון בטבלה בכל תרגיל בו מס' ימי האיחור המצטבר גדול מה-grace_days לערך שהועבר ב-min_grade.
רמז: ניתן לבצע masking עם השמה בתחביר דומה לזה שנלמד בnumpy באמצעות הפעלת values על הmask.
לדוגמא, df1[(df1>0).values]=0

```
def late_penalty(df_merged_late, n_hw, grace_days, min_grade):
```

8. **(5 נקודות)** השלימו את המימוש של הפונקציה calc_final_grade(df_semester, n_hw, n_hw_for_final_grade) המחשבת את ציון התרגילים הסופי בסמסטר. הפונקציה תקבל את טבלת הפלט מהסעיף הקודם, מספר שיעורי הבית שהיו בסמסטר (n_hw) ומספר שיעורי הבית הנחשבים לציון הסופי (n_hw_for_final_grade, מטיפוס int). הפונקציה **תוסיף עמודה לטבלה** בה יחושב הממוצע הסופי, הנקבע ע"י ממוצע ה n_hw_for_final_grade הטובים ביותר.
שימו לב כי עליכם להשלים **בשורה אחת בלבד** גם את מימוש פונקצית העזר my_helper.

```
def my_helper(a, n_hw_for_final_grade):

    _____________________________________________

def calc_final_grade(df_semester, n_hw, n_hw_for_final_grade):
    df_semester['hw_final']=_________________.apply(lambda a: my_helper(a,
n_hw_for_final_grade), axis=1)
```

9. **(4 נקודות)** כדי לחשב את הציון המטלות הסופי, צוות הקורס הריץ את כל הפונקציות לעיל עבור טבלאות כל הסמסטרים ושמר את התוצאה הסופית בקבצים המקוריים.
מכיוון שישנם סטודנטים שחוזרים על הקורס, החליט הצוות להחשיב עבורם לציון המטלות הסופי את ממוצע המטלות הגבוה ביותר שקיבלו באחד מהסמסטרים בעלי המזהה 2324a, 2223a,2223b.
כדי לחשב ציון מטלה עבור כל סטודנט, הריץ הצוות את קטע הקוד הבא:

```
semester_files=['2324a.csv', '2223a.csv', '2223b.csv']
df_semesters=[]
for semester_file in semester_files:
    cur_df=pd.read_csv(semester_file)
    df_semesters.append(cur_df)
df_cat=pd.concat(df_semesters)
print(df_cat.loc[df_cat.groupby('student_id')['hw_final'].max(), ["student_id",
"hw_final"]])
```

בקוד שכתב הצוות נפלו שתי טעויות. סמנו **במדויק** את כל הטעויות וכתבו כיצד יש לתקנן

_____

_____

_____

_____

מס' מחברת : מס' ת.ז. :

מסגרת חירום 1

מס' מחברת : מס' ת.ז. :

מסגרת חירום 2