חשיבה מחשובית ותכנות בשפת פייתון
תרגיל בית 8

הנחיות כלליות:

- קראו **היטב** את השאלות והקפידו שהתכניות שלכם פועלות בהתאם לנדרש.
- את התרגיל יש לפתור לבד!
- הקפידו על הוראות ההגשה המפורסמות במודל. בפרט, יש להגיש את כל השאלות יחד בקובץ ex#_012345678.py המצורף לתרגיל, לאחר החלפת ה# במספר התרגיל והחלפת הספרות 012345678 במספר ת.ז שלכם, כל 9 הספרות כולל ספרת הביקורת. למשל, אם מספר תעודת הזהות שלי הינו 112233445 ואני מגיש את תרגיל מספר 1, שם הקובץ שאגיש יהיה ex1_112233445.py.
- מועד אחרון להגשה: כמפורסם באתר.
- בדיקה עצמית: כדי לוודא את נכונותן ואת עמידותן של התוכניות לקלטים שגויים, בכל שאלה, הריצו את תוכניתכם עם מגוון קלטים שונים, אלה שהופיעו כדוגמאות בתרגיל וקלטים נוספים עליהם חשבתם (וודאו כי הפלט נכון וכי התוכנית אינה קורסת).
- היות ובדיקת התרגילים עשויה להיות אוטומטית, **יש להקפיד על פלטים מדויקים על פי הדוגמאות (כן כן, עד לרמת הרווח).**
- אופן ביצוע התרגיל: בתרגיל זה עליכם להשלים את הקוד בקובץ המצורף.
- **יש לעבוד עם המשתנים שמופיעים בשלד התרגיל ואין לשנות את שמם.** על קטע הקוד של כל שאלה לעבוד ולספק את התוצאה הדרושה עבור קלט שיוזן במשתנים שמופיעים בשלד (המשתנים שלידם סימני שאלה ומחכים לקלט כפי שמופיע בדוגמאות בתרגול). יחד עם זאת, אתם רשאים להוסיף משתנים נוספים כראותם עינכם.
- מומלץ להתעדכן בפורום לגבי שאלות של סטודנטים אחרים וכמובן, במידה ועדיין משהו לא ברור, לשאול בעצמכם.

<u>שימו לבים! (תקף החל מתרגיל בית מספר 5 ואילך)</u>

- ✓ שימו לב 1, החתימה של כל אחת מהפונקציות שעליכם לממש מופיעה כבר בשלד התרגיל ואין לשנותה. עליכם לממש את <u>גוף הפונקציה בלבד</u>. יש למחוק את הפקודה pass (היא נמצאת שם רק כדי שתוכלו להריץ חלקי קוד בלי לשים בהערה את המימושים החסרים) ולהחליף אותה במימוש המתאים.
- ✓ שימו לב 2, בשאלה בה נתבקשתם לממש פונקציה, אין לעשות שום דבר מעבר לכך. ניתן ואף רצוי לקרוא לפונקציה שכתבתם עם מגוון קלטים על מנת לוודא את תקינותה אך אין להשאיר את הקריאות הללו בהגשת התרגיל.
- ✓ שימו לב 3, אף פונקציה לא מדפיסה דבר אלא רק מחזירה ערך כלשהו.

<u>שאלה 1</u> – ממשו את הפונקציה write_text_to_file(txt, file_path) המקבלת מחרוזת אחת המכילה טקסט כלשהו ומחרוזת שניה המכילה נתיב חוקי לקובץ כלשהו (למשל נתיב מלא: "C:/CompThink/HW8/hi.txt" או נתיב יחסי "hi.txt"). הפונקציה תכתוב את הטקסט לתוך הקובץ (ולא תחזיר כלום). למשל, עבור הקריאה write_text_to_file('Hi\nI love Python','hi.txt') הפונקציה תייצר קובץ חדש (באותה תיקיה, כי הועבר נתיב יחסי) שנקרא hi.txt ומכיל תוכן זהה לקובץ hi.txt שסופק עם תרגיל זה. שימו לב לא לשכוח לסגור את הקובץ (לא כאן ולא באף שאלה אחרת בה תעבדו עם קבצים).

<u>שאלה 2</u> – ממשו את הפונקציה get_file_length(file_path) המקבלת מחרוזת המכילה נתיב חוקי לקובץ כלשהו. הפונקציה תחזיר מספר שלם (int) את אורך הקובץ, כלומר מספר התווים *הכולל שיש בתוכו. להלן דוגמת הרצה:

```
>>> get_file_length('hi.txt')
16
```

*מה, צריך לספור גם רווחים? כן, גם רווחים. ומה עם \n? גם. וכו...

<u>שאלה 3</u> – ממשו את הפונקציה sum_file_nums(file_path) המקבלת מחרוזת המכילה נתיב חוקי לקובץ כלשהו המכיל מספרים. הפונקציה תחזיר מספר עשרוני

(float) המייצג את סכום המספרים שבקובץ. ניתן להניח שכל שורה מכילה מספר אחד שאחריו יש \n. כרגיל, אנחנו עובדים עם floatים ולכן אם המספר שהחזרתם רחוק בלא יותר מ0.00001 מהתשובה המדויקת, זה גם בסדר. להלן דוגמת הרצה:

```
>>> sum_file_nums('numbers.txt')
-1.5999999999999996
```

<u>שאלה 4</u> – הדבר הראשון שצריך לעשות כשרוצים לנתח נתונים כלשהם (ערכי מניות, ביקורות באמאזון, וכו) יש תחילה לטעון אותם לזיכרון של פייתון (שבתורו יפעיל עליהם את קסמיו) לתוך מבנה נתונים שיתאים לאסטרטגיה בה נרצה לנתח את הנתונים (רשימה/מטריצה/מילון וכו). הנתונים יגיעו ממאגר חיצוני (לא ציפיתם שהיוזר יזין אלפי ערכים, נכון?) כגון קובץ ו/או דף אינטרנט וכו. בדרך כלל, מקורות אלה יכילו נתונים בצורה unstructured (קרי, ברדק) ונצטרך לעבוד קשה מאוד על מנת לזקק אותם ולהביא אותם לצורה שתהיה לנו נוחה לעבודה. לפעמים מתמזל מזלנו והנתונים שלנו נמצאים בקובץ בפורמט מסודר (structured) או שאנחנו אלה ש"המזלנו" את עצמנו, כלומר, ניקינו את הנתונים מבעוד מועד ושמרנו אותם בקובץ מסודר. ישנם פורמטים מקובלים של קבצים שכאלה, למשל קובץ CSV (שזה קיצור לComma Separated Values) המכיל ערכים המופרדים ע"י פסיק (כמה מפתיע). באופן דומה קיים גם קובץ מסוג TSV (שזה קיצור של Tab Separated Values). נרצה לממש פונקציה גנרית (כללית) שתדע לטעון לזיכרון נתונים מכל מיני סוגי קבצים כאלה כל עוד נותנים לנו מראש את התו המפריד (delimiter). ממשו את הפונקציה parse_file_to_list(file_path, delimiter) המקבלת מחרוזת המכילה נתיב חוקי לקובץ כלשהו ומחרוזת נוספת במשתנה delimiter. הקובץ מכיל שורה אחת בלבד ובה מספרים המופרדים ע"י התו delimiter. הפונקציה תחזיר רשימה של מספרים עשרוניים (float) שהינם המספרים שהופיעו בקובץ. ניתן להניח שהdelimiter תואם את הקובץ (כלומר, לא תקבלו קובץ TSV עם delimiter=',' וכו). אין לבצע אף import לצורך מימוש הפונקציה. להלן דוגמת הרצה:

```
>>> parse_file_to_list('numbers.tsv', '\t')
[4.0, 5.6, -7.2]
```

ללא כל קשר, מי שירצה לאתגר את עצמו (ולהתכונן ל"חיים עצמם") מוזמן לנסות לממש פונקציה נוספת parse_file_to_matrix(file_path, delimiter) המקבלת קלט

דומה רק שהפעם הקובץ יכיל יותר משורה אחת. כל שורה צריכה להיטען כרשימה וכל הקובץ יחד ייטען כרשימה של רשימות. לא יינתן ניקוד בונוס על פתרון שאלה זו ולכן אין צורך להגיש את המימוש של parse_file_to_matrix.

<u>שאלת בונוס</u> (5 נק'): הפעם ניקח את יכולות המילון שלנו לקצה. בידינו מילון המכיל מיפוי בין שם קורס לרשימת הסטודנטים הרשומים אליו (כמו הפלט של שאלת הבונוס מתרגיל הבית הקודם). להלן דוגמא:

```
>>> course2students = {'Math': ['Yuval', 'Noam'], 'Computer Science':
['Yuval'], 'Statistics': ['Yuval', 'Gal', 'Noam'], 'Algebra': ['Gal'],
'Physics': ['Gal'], 'Programming': ['Noam']}
```

ברצוננו להבין את היחסים בין הקורסים השונים. לשם כך, נרצה לבדוק עבור כל זוג קורסים כמה סטודנטים נרשמו לשניהם. בדוגמא של מבנה הנתונים לעיל, עבור הקורסים Math ו- Statistics יש 2 סטודנטים משותפים (Noam ו- Yuval), ולקורסים Math ו- Physics ישנם 0 סטודנטים משותפים. ממשו את הפונקציה count_courses_intersection(course2students) המקבלת מילון בפורמט למבנה הנ"ל (המפתחות בו הם שמות של קורסים, ולכל קורס, הערך הוא רשימה של מחרוזות אשר מייצגות את שמות הסטודנטים הרשומים לקורס. הפונקציה תחזיר מילון שהמפתחות שלו הינם <u>זוגות</u> של קורסים (tuples), ולכל זוג קורסים, הערך יהיה מספר שלם המתאר את מספר הסטודנטים הרשומים לשני הקורסים הנ"ל. אין חשיבות לסדר הפנימי בתוך הtuple כל עוד כל זוג מופיע כמפתח בדיוק פעם אחת (למשל, רק אחד מבין שני הזוגות הבאים יופיע כמפתח במילון: או הזוג ('Math', 'Statistics') או הזוג ('Statistics', 'Math'), ולא שניהם). להלן דוגמת הרצה:

```
>>> intersection = count_courses_intersection(course2students)
>>> print(intersection)
{('Math', 'Computer Science'): 1, ('Math', 'Statistics'): 2, ('Math',
'Algebra'): 0, ('Math', 'Physics'): 0, ('Math', 'Programming'): 1,
('Computer Science', 'Statistics'): 1, ('Computer Science', 'Algebra'): 0,
('Computer Science', 'Physics'): 0, ('Computer Science',
'Programming'): 0, ('Statistics', 'Algebra'): 1, ('Statistics', 'Physics'): 1,
```

('Statistics', 'Programming'): 1, ('Algebra', 'Physics'): 1, ('Algebra', 'Programming'): 0, ('Physics', 'Programming'): 0}

הדרכה- כדאי לנסות לפרק את הבעיה המורכבת ולהתחיל ולטפל בזוג קורסים בודד. בהינתן פונקציה שכזו, הפתרון יהיה פחות מורכב (מדוע?). ממשו את פונקציית העזר:
intersection_of_course1_and_course2(course2students, course1, course2)
המקבלת מילון בפורמט שתואר לעיל וכן שני שמות של קורסים, ומחזירה את מספר הסטודנטים המשותפים.