תכנות הנדסי בשפת פייתון – תרגיל בית 4
מתרגל אחראי: אופיר יעיש

הנחיות כלליות:

- מועד אחרון להגשה: כמפורסם בתיבת ההגשה ב-Moodle.
- מטרת התרגיל הינה לתרגל כתיבה, ולרכוש מיומנות בכלים שנלמדו עד כה בכיתה.
- קראו את העבודה מתחילתה ועד סופה לפני שאתם מתחילים לפתור אותה. ודאו שאתם מבינים את כל המשימות.
- רמת הקושי של המשימות אינה אחידה.
- את התרגיל יש לפתור לבד!
- בתיבת ההגשה במערכת ה-VPL ישנו קובץ שלד לכתיבת הקודם שלכם. עבור כל משימה כתבו הפתרון שלכם במקום המתאים למשימה.
- אין למחוק שום קטע קוד הנמצא בשלד. עליכם רק להוסיף את הפתרון שלכם בתוכו.
- אין לשנות את שם/שמות ה-Requested Files.
- השאלות יבדקו באופן אוטומטי. הפלט שעליכם להחזיר בכל תרגיל צריך להיות בדיוק כפי שנדרש. כמו כן, באופן אקראי יבדקו גם עבודות באופן ידני.
- כאשר תבוצע בדיקה ידנית, תתבצע גם בדיקת Readability - שימו לב שאתם משתמשים בשמות משתנים אינפורמטיביים וכותבים הערות בכל סעיף.
- בדיקה עצמית: כדי לוודא את נכונותן ואת עמידותן של הפונקציות לקלטים שונים, בכל שאלה הריצו אותן עם מגוון קלטים: אלה שמופיעים בדוגמאות וקלטים נוספים עליהם חשבתם. וודאו כי הפלט נכון. הבדיקה תתבצע על מגוון דוגמאות ולא בהכרח אלה שיינתנו פה.
- ניתן להשתמש בחומר הנלמד עד לפרסום העבודה ורק בחומר הזה.
- אין להשתמש בחבילות או מודולים חיצוניים (כדוגמת math) למעט מקרים שבהם צוין אחרת במפורש.
- במידה ולא צוין אחרת, יש להניח את נכונות הקלט על פי תיאור המשימה.
- משקל כל שאלה הוא זהה.
- במידה ולא עניתם על שאלה מסוימת, נא מלאו את הפונקציה בכל מקרה על מנת שהקוד שלכם יצליח לעבוד.

הוראות חשובות למשימה זו:

- היצמדו להוראות של כתיבת פונקציית עזר רקורסיבית אם התבקשתם לכך.
- ניתן (ואף מומלץ במקרים מסוימים) להשתמש בפונקציות עזר רקורסיביות גם בשאלות שלא התבקשתם לכך.

<u>שאלות:</u>

שאלות 1 ו-2 יעסקו במילונים מקוננים (nested).

הגדרה – מילון מקונן (nested) הינו מילון שקיימים בו ערכים מסוג מילון.

ניתן להגדיר את עומק הקינון במילונים באופן הבא:

עומק 0 – למילון אין ערכים מסוג מילון (מילון לא מקונן). לדוגמה:
{1:"a",2:"b"}
עומק 1 – למילון קיימים ערכים מסוג מילון כאשר העומק המקסימלי שלהם הוא 0. לדוגמה:
{1:{1:"a",2:"b"},2:"b"}
עומק 2 – למילון קיימים ערכים מסוג מילון כאשר העומק המקסימלי שלהם הוא 1. לדוגמה:
{1:{1:"a",2:"b"},2:{1:{1:"a",2:"b"},2:"b"}}
וכן הלאה...

<u>שאלה 1:</u>

ממשו את הפונקציה dict_depth(d) המקבלת מילון d ומחזירה את עומק הקינון שלו כ-int. בנוסף, אין להניח שום הנחה על סוג הקלט: במידה ו-d אינו מסוג מילון יש לזרוק שגיאה מסוג TypeError עם מחרוזת הודעה לבחירתכם.

רמז – מומלץ לבנות פונקציית עזר רקורסיבית לפתרון.

<u>דוגמאות:</u>

עבור הקריאה dict_depth({1:"a",2:"b"})
תוחזר הספרה 0

עבור הקריאה dict_depth({1: {1:"a",2:"b"},2:"b"})
תוחזר הספרה 1

עבור הקריאה dict_depth({1: {1:"a",2:"b"},2: {1:{1:"a",2:"b"},2:"b"}})
תוחזר הספרה 2

עבור הקריאה dict_depth(1)
תיזרק שגיאה מסוג TypeError עם מחרוזת הודעה לבחירתכם. לדוגמה:
TypeError("d is not a dict")

שאלה 2:

ממשו את הפונקציה nested_get(d, key) המקבלת מילון, d, ומספר שלם key ומחזירה רשימה של כל הערכים שהמפתח שלהם הוא key **ושאינם מסוג מילון (dict)**.

שימו לב – המילון d יכול להיות מקונן בעומק מסוים, או לא מקונן בכלל (עומק 0). לכן יכולים להיות כמה מפתחות ששווים ל-key. על הפתרון שלכם להיות כללי כך שיוכל להתאים למילונים בעומקים שונים.

ניתן להניח שכל הערכים במילון יהיו או מחרוזות או מילונים נוספים (על המפתחות במילונים אין הנחות מקדימות). בנוסף אין חשיבות לסדר הופעת הערכים ברשימה המוחזרת (אתם יכולים להחזיר אותם באיזה סדר שנוח לכם).

רמז – מומלץ להשתמש ברקורסיה לפתרון.

דוגמאות:

עבור הקריאה nested_get({1:"a",2:"b"}, 2) תוחזר הרשימה:

['b']

עבור הקריאה nested_get({1:{1:"a",2:"b"},2:"b"}, 3) תוחזר הרשימה הריקה:

[]

עבור הקריאה nested_get({1:{1:"a",2:"b"},2:{1:{1:"c",2:"b"},2:"b"}}, 1) תוחזר הרשימה (לא בהכרח בסדר הזה):

['a', 'c']

שאלה 3:

הגדרה – קובץ integer csv תקין הינו קובץ אשר מקיים את התנאים הבאים:

1. בכל שורה ישנם מספר שווה של ערכים המופרדים ביניהם בסימן פסיק (,).
2. כל הערכים בקובץ הינם מספרים שלמים.

ממשו את הפונקציה is_valid_integer_csv(file_name) המקבלת שם של קובץ ומחזירה את הערך הבוליאני True אם הקובץ הוא integer csv תקין. במידה ולא, יוחזר הערך הבוליאני False. במידה והקובץ לא נמצא יוחזר הערך הבוליאני False.

הנחות:

- הקובץ אינו ריק.
- הקובץ אינו מכיל שורות ריקות.

דוגמאות:

יחד עם קובץ ההוראות צירפנו מספר קבצי דוגמה לבדיקה. הדוגמאות הנ"ל מתייחסות לקבצים אלו. *שימו לב שעל מנת להשתמש בקבצים הללו עליכם להעביר אותם לתיקיה בה הקוד שלכם נמצא או לחלופין לשנות את הכתובת (address) שאתם קוראים לפונקציות עם הדוגמאות.

עבור הקריאה is_valid_integer_csv("a.csv") יוחזר הערך הבוליאני True

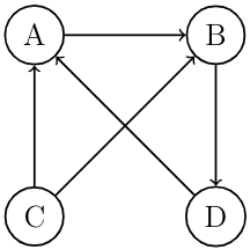
עבור הקריאה is_valid_integer_csv("b.csv") יוחזר הערך הבוליאני False

עבור הקריאה is_valid_integer_csv("c.csv") יוחזר הערך הבוליאני False

אם לא קיים קובץ בשם "not_found.csv", אז עבור הקריאה is_valid_integer_csv("not_found.csv") יוחזר הערך הבוליאני False

שאלה 4:

במסגרת השאלה הזאת נעסוק בגרפים מכוונים. גרף מכוון מורכב מצמתים וקשתות מכוונות. לדוגמה, באיור הבא מתואר גרף עם 4 צמתים: A,B,C,D ו-5 קשתות מכוונות. שימו לב שישנה משמעות לכיוון הקשתות – באיור לדוגמה מתואר שניתן להגיע מ-A ל-D (במסלול שמורכב מהקשת מ-A ל-B והקשת מ-B ל-D) אך לא ניתן להגיע מ-A ל-C.



בשאלה ייעשה שימוש בתיאור של גרף מכוון באמצעות מילון בו לכל צומת יש מפתח שהערך שלו הוא רשימה של הצמתים אליהן יש לצומת קשת. לדוגמה, הגרף באיור לעיל יתורגם למילון הבא:

{"A":["B"],"B":["D"],"C":["A","B"],"D":["A"]}

ממשו את הפונקציה connected_nodes(graph, node) המקבלת מילון המייצג גרף, graph, ומחרוזת node המייצגת צומת. הפונקציה תחזיר רשימה של הצמתים שאליהם ניתן להגיע מצומת node (כולל node עצמו). אין חשיבות לסדר הופעת הצמתים ברשימה המוחזרת.

הנחות:

- המילון מייצג גרף באופן תקין: כל המפתחות במילון הן צמתים בגרף, כל הערכים במילון הן רשימות של הצמתים אליהן יש לצומת קשת. במידה ואין לצומת קשתות יוצאות הערך שלו יהיה רשימה ריקה.
- הצמתים יתוארו על ידי מחרוזת.
- node נמצא בגרף.

רמז – ניתן להשתמש בהכוונה הבאה (לא חובה):

1. על מנת להימנע ממעגלים אינסופיים, רצוי להשתמש ברשימה הזמנית שמכילה את הצמתים שכבר "ביקרנו" בהם (ז"א אלו שכבר זיהנו שאנחנו יכולים להגיע אליהם מ-node). כך תוכלו להימנע מביקור בצומת שכבר "ביקרתם" בה, ולכן להימנע מביקורים חוזרים ונשנים (ז"א ממעגל אין סופי).
2. רצוי לתחזק רשימה נוספת שתכיל את כל הצמתים שאנחנו עתידים "לבקר" בהם. במידה ו"ביקרנו" בצומת מסוימת מהרשימה, אזי נסיר אותה מהרשימה, ובמידה ואנחנו "מגלים" שישנם צמתים חדשים שלא "ביקרנו" בהם, אזי נכניס אותם לרשימה וכך נדע "לבקר" בהם בהמשך. לדוגמה, בתחילה רשימה זאת תכיל את node בלבד, ולאחר ש"נבקר" ב-node, נסיר את node מהרשימה, ונוסיף את הצמתים שיש מ-node אליהם קשת. כאשר אנחנו "מבקרים" בצומת מסוימת ומוסיפים לרשימה את הצמתים שאנחנו עתידים "לבקר" (נוסיף צמתים שאליהם יש קשת מהצומת המסוימת), אזי יש לוודא שאנחנו מוסיפים רק צמתים שלא "ביקרנו" בהם בעבר (כאמור, כדי להימנע מהמעגל האינסופי).

דוגמאות:

עבור הקריאה connected_nodes({"A":["B"],"B":["D"],"C":["A","B"],"D":["A"]},"A") תוחזר הרשימה (לא בהכרח בסדר הזה):

['A', 'B', 'D']

עבור הקריאה connected_nodes({"A":[]},"A") תוחזר הרשימה:

['A']

<u>שאלה 5:</u>

בשאלה זו נעסוק בבעיית התרמיל:

בהינתן קבוצה של חפצים בעלי משקל וערך ובהינתן תרמיל שיכול להכיל לכל היותר משקל כלשהו, מהו הערך המקסימלי של החפצים שאני יכול לשים בתרמיל.

לדוגמה: אם יש לנו 3 חפצים בעלי משקלים 10, 20, 30 עם הערכים 60, 100, 120 בהתאמה, ותרמיל שיכול להכיל חפצים עם משקל כולל של לכל היותר 50, אזי הערך המקסימלי של החפצים שאנו יכולים לשים בתיק הינו 220 (אם ניקח את החפץ השני והשלישי).

בקובץ השלד מופיע פתרון לבעיית התרמיל באמצעות הפונקציה knapsack(weights, values, max_weight) הקוראת לפונקציית עזר רקורסיבית.

עליכם לשנות את פונקציית העזר כך שתעשה שימוש ב-memoization כפי שלמדנו בתרגולים, זאת מבלי לשנות את החתימה של הפונקציות (השמות והארגומנטים שלהן). על הפונקציה knapsack(weights, values, max_weight) להישאר כפי שהיא בקובץ השלד.

ניתן להניח את תקינות הקלט - weights ו-values יהיו רשימות באותו גודל ויכילו רק מספרים שלמים אי שליליים.
max_weight יהיה מספר שלם אי שלילי.

<u>דוגמאות:</u>

לצורך בדיקה, אנא ודאו שההוספה של memoization לא משנה את ערך הפתרון:

עבור הקריאה knapsack([10, 20, 30], [60, 100, 120], 50)
יוחזר הערך 220

עבור הקריאה knapsack([10, 20, 30], [110, 100, 120], 50)
יוחזר הערך 230

עבור הקריאה knapsack([10, 20, 30], [60, 100, 120], 5)
יוחזר הערך 0