תרגיל בית ראשון בשפות תכנות. סמסטר 2018 ב.

הגשה בזוגות

יש להשתמש ב- python 3 (לא python 2).

יתכן שהתרגיל ייבדק בבדיקה אוטומטית. לכן יש להיצמד למה שמתואר בשאלות. למשל

אין לשנות את שם הפונקציות או את החתימה שלהן.

יש לכתוב גם tests לפתרונות שלכם (לפחות שניים לכל פתרון -- לפחות אחד שיבדוק מקרה קצה).

ה- tests יעשו שימוש ב- assert כך שבמקרה שמתגלות טעויות יזרק exception.

שאלה 1

כתבו פונקציה (matrix) upper\_half.

**גוף הפונקציה צריך לכלול שורה בודדת** (list comprehension).

הפרמטר matrix זו מטריצה. המטריצה מיוצגת ע"י רשימה (list) של רשימות כאשר כל רשימה פנימית מייצגת שורה אחת של המטריצה. מספר השורות לא בהכרח שווה למספר העמודות של המטריצה.

הפונקציה מחזירה את matrix עם השינויים הבאים: השורה הראשונה נותרת ללא שינוי. מהשורה השניה

נמחק האיבר הראשון. מהשורה השלישית נמחקו שני האיברים הראשונים וכן הלאה.

לדוגמא אם הארגומנט הוא

[[1, 2, 3, 4, 5],

[6, 7, 8, 9, "spam"],

[11, 12, 13, 14, 15],

[16, "stam", 18, 19, 20]]

אז הפונקציה תחזיר:

[[1, 2, 3, 4, 5],

[ 7, 8, 9, "spam"],

[ 13, 14, 15],

[ 19, 20]]

אם matrix היא הרשימה הריקה אז יוחזר [].

שאלה 2

כתבו פונקציה עם החתימה encrypt (string, key=1) שמקבלת מחרוזת (המכילה אותיות קטנות בא"ב האנגלי) ו-"מפתח הצפנה" כארגומנטים ומחזירה את המחרוזת המוצפנת. בשיטת הצפנה זו כל אות מוחלפת באות אחרת – זו שמופיעה בסדר הרגיל של אותיות הא"ב האנגלי k מקומות אחרי האות המוחלפת. k הוא מפתח ההצפנה. למשל אם k הוא 1 אז האות a תוחלף ב- b, b תוחלף ב- c וכן הלאה. (האות z תוחלף ב- a). encrypt("spam") אמורה להחזיר "tqbn". דוגמא נוספת: אם מפתח ההצפנה הוא 3 אז האות a תוחלף ב- d, האות b תוחלף ב- e וכן הלאה. (האות w תוחלף ב- z, x תוחלף ב- a, y תוחלף ב- b וכן הלאה). במקרה זה encrypt("spam", 3) אמורה להחזיר "vsdp".

**הדרכה**:

ניתן להשתמש בפונקציות chr ן- ord כדי להמיר תו (מחרוזת עם תו אחד – אין ב- python טיפוס char) למספר שלם ולהיפך. (הן מוסברות ב- <https://docs.python.org/3/library/functions.html>).

למשל chr(ord('0') + 3) תחזיר את המחרוזת '3'.

שאלה 3

כתבו שתי גרסאות של הפונקציה sum\_digits(n) המחזירה את סכום הספרות של המספר השלם n (כאשר n נכתב בכתיב עשרוני).

למשל sum\_digits(1492) יחזיר 16 כי 1 + 4 + 9 + 2 = 16

**סעיף א**. כתבו את הגרסה הראשונה וקראו לה sum\_digits1(n)

**הדרכה:**

יש לחלץ את הספרות של המספר בעזרת מניפולציות אריתמטיות פשוטות.

n % 10 זו הספרה הימנית ביותר של n (כשכותבים את n בבסיס 10).

חלוקה של n ב- 10 "תמחק" את הספרה הימנית ביותר שלו. אפשר להשתמש באופרטור

// (floor division) שמעגל את התוצאה כלפי מטה למספר שלם. (ב- python 3

חלוקה של מספרים שלמים עם האופרטור / (לא //) מניבה ערך מסוג (float.

**סעיף ב**

כתבו את הגרסה השנייה וקראו לה sum\_digits2(n)

**הדרכה**:

יש להמיר את המספר למחרוזת ואז ניתן לחלץ את הספרות בעזרת איטרציה על המחרוזת

נזכיר שאיטרציה על מחרוזת מניבה את התווים המרכיבים את המחרוזת כמחרוזות בנות תו אחד.

כדי לסכם את הספרות יהיה צורך להמירן בחזרה למספרים שלמים (int).

אפשר להשתמש בפונקציות str() ו- int(). למשל str(1492) = '1492'

int ("17") = 17

ניתן לכתוב את הפונקציה כך שתכיל שורה בודדת (תוך שימוש בפונקציה sum וב- list comprehension).

שאלה 4

כתבו את הפונקציה flatten(alist) שמקבלת רשימה של רשימות

ומחזירה רשימה שהיא השרשור של רשימות אלו.

דוגמא:

flatten([[1, 2, 3], ['foo', 'bar'], [], [[1, 2], [3]]])

תחזיר את הרשימה [1, 2, 3,'foo','bar',[1, 2], [3]]

יש לכתוב את הפונקציה תוך שימוש בפונקציה reduce שנמצאת במודול functools.

(יש לכתוב import functools ואז אפשר לקרוא לפונקציה functools.reduce).

שאלה 5

כתבו פונקציה dijkstra(graph, weights, source)

הפרמטרים graph ו- weights מתארים גרף מכוון שבו לכל קשת יש משקל לא שלילי.

הפרמטר source הוא אחד הצמתים בגרף.

הפונקציה מחזירה מילון הממפה כל צומת בגרף למרחק מה- source אליה.

שני פרמטרים מתארים את הגרף: graph הוא מילון הממפה כל צומת לרשימה של השכנים שלו (הצמתים שיש

קשתות מהצומת אליהם). למשל אם הצומת 'a' ממופה לרשימה ['b', 'c'] זה אומר שיש בגרף קשתות

a->b ו- a->c.

weights הוא מילון הממפה כל קשת למשקל שלה. המפתחות של מילון זה הם tuples המייצגים קשתות: (x, y) מייצג את הקשת x -> y .

דוגמא. (כאן <<< זה ה- prompt של ה- interpreter)

>>> graph = {'a': ['b', 'c'],

'b': ['d'],

'c': [],

'd': ['c', 'e'],

'e': [],

'f': ['e'] }

>>> weights = { ('a', 'b'): 2,

('a', 'c'): 10,

('b', 'd'): 3,

('d', 'c'): 1,

('d', 'e'): 12,

('f', 'e'): 0

}

>>> dijkstra(graph, weights, 'a')

{'a': 0, 'b': 2, 'c': 6, 'd': 5, 'e': 17, 'f': 'infinity'}

(סדר הפריטים במילון המוחזר עשוי להיות שונה. שימו לב שהפונקציה אינה מדפיסה את המילון – היא רק מחזירה אותו ובדוגמא ה- interpreter (כרגיל) מדפיס את הערך המוחזר).

כפי שרואים בדוגמא, כאשר לא קיים מסלול מ-source לצומת מסוים, המרחק במילון המוחזר צריך להיות

'infinity' (זו מחרוזת).

הדרכה: הפונקציה צריכה להשתמש באלגוריתם הידוע של Dijkstra. אפשר למצוא אותו ב- Wikipedia.

אפשר להניח שכל המרחקים קטנים ממיליון. השתמשו בהגדרה הבאה:

INFINITY = 1000000

(בהתחלת האלגוריתם, המרחקים הטנטטיביים של הצמתים מאותחלים ל- INFINITY).

האלגוריתם של Dijkstra מוצא (בכל איטרציה) את הצומת שמרחקה (הטנטטיבי) מצומת המקור הוא הקטן ביותר. לצורך כך אפשר לשמור ברשימה ממוינת צמתים ביחד עם המרחקים הטנטטיביים למשל

[(3, 'y'), (10, 'c'), (50, 'b'), (1000000, 'z')]

ואז בכל פעם שמעדכנים מרחק של צומת אז יש להוציאו מהרשימה, להכניסו שוב עם המרחק המעודכן

ולמיין את הרשימה שוב. (כש- Python משווה בין שני tuples כדי למצוא את הקטן ביניהם אז היא משווה בין הרכיבים הראשונים. אם הם שווים אז היא משווה בין הרכיבים השניים וכך הלאה ).

פתרון יעיל יותר מחזיק את הזוגות הנ"ל במבנה נתונים שנקרא priority queue (או heap). מבנה הנתונים הזה מאפשר להוסיף איברים ולהוציא את האיבר המינימלי באופן יעיל. אפשר גם להוציא איבר כלשהו

(לא המינימלי) בצורה יעילה בעזרת טריק קטן המתואר בתיעוד של heapq (ראו בהמשך). במקרה שלנו האיברים יהיו tuples שכל אחד מהם כולל מרחק וצומת למשל (3, 'y'). האיבר המינימלי יהיה זה עם המרחק הקטן ביותר.

כדי להשתמש ב- priority queue (הממומש ע"י heap) השתמשו ב-module בספריה הסטנדרטית של

python שנקרא heapq. התיעוד של המודול הזה ב- python.org הוא טוב מאוד.

אל תשכחו לעשות import heapq בתחילת הקובץ שלכם ואז תוכלו לקרוא לפונקציות

heapq.heapify, heapq.heappush וכן הלאה.

(מינוס 5 נקודות לפתרון שלא כולל שימוש ב- heapq).