אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2023-24

תרגיל בית מספר 1 - להגשה עד 13/06/2024 בשעה 23:59

קיראו בעיון את הנחיות העבודה וההגשה המופיעות באתר הקורס במסמך הנהלים. חריגה מההנחיות תגרור ירידת ציון / פסילת התרגיל.

הגשה:

- תשובותיכם יוגשו בקובץ pdf ובקובץ py בהתאם להנחיות בכל שאלה.
- השתמשו בקובץ השלד skeleton1.py כבסיס לקובץ ה-py אותו אתם מגישים.
לא לשכוח לשנות את שם הקובץ למספר ת"ז שלכם לפני ההגשה, עם סיומת py.

- בסה"כ מגישים שני קבצים בלבד. עבור סטודנטית שמספר ת"ז שלה הוא 012345678 הקבצים שיש להגיש הם hw1_012345678.pdf ו- hw1_012345678.py.

- הקפידו לענות על כל מה שנשאלתם.
- מותר להוסיף פונקציות עזר משלכם לקובץ השלד.
- לפני ההגשה ודאו כי הרצתם את הפונקציה `test()` שבקובץ השלד אך זכרו כי היא מבצעת בדיקות בסיסיות בלבד וכי בתהליך הבדיקה הקוד ייבדק על פני מקרים מגוונים ומורכבים יותר.

- בכל שאלה, אלא אם מצוין אחרת באופן מפורש, ניתן להניח כי הקלט תקין.

- תשובות מילוליות והסברים צריכים להיות תמציתיים, קולעים וברורים.
להנחיה זו מטרה כפולה:
1. על מנת שנוכל לבדוק את התרגילים שלכם בזמן סביר.
2. כדי להרגיל אתכם להבעת טיעונים באופן מתומצת ויעיל, ללא פרטים חסרים מצד אחד אך ללא עודף בלתי הכרחי מצד שני. זוהי פרקטיקה חשובה במדעי המחשב.

הבהרות כללית לתרגילי הבית

פתרון באמצעות כלים שלא נלמדו

בפתרון שאלות בתרגילי הבית ובמבחנים, **אין** לבצע שימוש בספריות חיצוניות (באמצעות פקודת import, למי שמכירים) אלא אם כן נאמר מפורשות אחרת.

כל עוד לא **אסרנו שימוש** של כלי מסוים באופן מפורש, ניתן לפתור כל תרגיל **בכל דרך שעומדת בהנחיות**.

בשאלות שיש בהן מספר סעיפים, מותר להיעזר בפונקציות שמימשתם בסעיפים קודמים כאשר אתם פותרים את הסעיפים הבאים (אלא אם כן צוין אחרת).

עם זאת, יש לשים לב כי ניתן ורצוי לפתור כל תרגיל בית באמצעות הכלים שלמדנו עד למועד תרגיל זה (לפניו או עד למועד ההגשה שלו).

אנחנו ממליצים בחום לפתור את התרגילים באמצעות הכלים שלמדנו בכיתה עד למועד הרלוונטי ממספר סיבות. פייתון (ושפות תכנות רבות אחרות) מכילות אינספור ספריות שנכתבו לאורך השנים שמסוגלות לבצע **פעולות מורכבות** בהינף יד. המטרה שלנו בקורס היא לא ללמוד איך **להשתמש** בהן, אלא ללמוד איך **לחשוב כמו מדעני מחשב.**

בנוסף, בעוד מספר שבועות נלמד איך לנתח **יעילות של קוד**, ונתחיל להתעניין בפתרונות שלא רק נותנים את **התשובה הנכונה**, אלא גם עושים זאת **במהירות**. אנחנו נראה כי ישנם מקרים לא מעטים בהם **פתרון מובנה יכול להיות פחות יעיל לפתרון הבעיה** שלנו. על מנת לפתור את הבעיות הללו ביעילות, נצטרך את **ארגז הכלים שאנחנו בונים כעת**, וכדי לבנות את ארגז הכלים הזה אנחנו חושבים **שחייבים לתרגל אותו** - ובדיוק לשם כך יש את תרגילי הבית.

תקינות קלט

לאורך כל הקורס – בהרצאות, בתרגולים, בתרגילי הבית ובמבחנים, ניתן להניח תמיד כי הקלטים הניתנים לבעיות השונות הם תקינים. לדוגמא, אם אתם מתבקשים לממש פונקציה המקבלת כקלט מספר שלם חיובי מטיפוס int יש להניח כי הפונקציה תיקרא רק על קלט מסוג זה ואין צורך להתמודד עם קלטים שלא עומדים בתנאים הללו.

הבהרות אלו תקפות לאורך כל הקורס.

דוגמה לפונקציה

בחלק מהשאלות בתרגיל זה הנכם מתבקשים להגיש תוכניות בפייתון. את התוכניות יהיה עליכם להגיש כפונקציות, נושא שילמד בהרחבה בשבוע השני של הסמסטר. אולם פתרון כל השאלות לא מחייב הבנה של נושא זה, ולכן אפשר וכדאי להתחיל לעבוד על התרגיל כבר עכשיו. כדי להקל עליכם, להלן דוגמה של פונקציה פשוטה שמקבלת מספר בודד כקלט ומחזירה כפלט באמצעות הפקודה return את ערכו של המספר כפול 2.

נשים לב למספר דרישות בכתיבת פונקציה:

1. הגדרת הפונקציה תתחיל במילה השמורה `def`, לאחריה רווח, ולאחריו שם הפונקציה.
2. לאחר שם הפונקציה יפורטו בתוך סוגריים הקלטים (אפס או יותר) אותם היא מקבלת, מופרדים ע"י פסיק. לאחר הסוגריים יופיע סימן נקודותיים (:) ולאחריו ירידת שורה.
3. יש להקפיד על הזחה: קוד גוף הפונקציה ייכתב Tab אחד (או 4 רווחים) פנימה ביחס לשורת ה- def.
הפונקציה תחזיר פלט ע"י כתיבת המילה **return (שימו לב: <u>לא</u> המילה print אשר רק מדפיסה למסך)** ולאחריה הערך שיוחזר כאשר תופעל הפונקציה.

```
def double_my_num(x):
     return 2*x
```

דוגמאות להפעלת הפונקציה הנ"ל:

```
>>> z = double_my_num(5) #won't work with print…
>>> z
10
>>> double_my_num(10)
20
>>> a = 30
>>> double_my_num(a)
60
```

דוגמה נוספת לפונקציה שמקבלת <u>שני</u> קלטים מספריים `x,y` ומחזירה כפלט באמצעות הפקודה `return` את מכפלתם:

```
def mult_nums(x, y):
     return x*y
```

דוגמאות להפעלת הפונקציה הנ"ל:

```
>>> y = mult_nums(5, 10)
>>> y
50
>>> mult_nums(10, 3)
30
>>> a = 2
>>> b = 6
>>> mult_nums(a, b)
12
```

שאלה 1 (שאלת חימום – לא להגשה)

כפי שראיתם בהרצאה, ישנן בפייתון פונקציות שמשויכות למחלקה מסוימת, למשל למחלקת המחרוזות (`str`).

באינטרפרטר IDLE, אם תכתבו "`str.`" ותלחצו על המקש tab, תיפתח חלונית עם מגוון פונקציות המשויכות למחלקת המחרוזות. כמובן, אפשר למצוא תיעוד רב על פונקציות אלו ואחרות ברשת. כמו כן אפשר להשתמש בפונקציה `help` של פייתון. למשל הפקודה `help(str.title)` תציג הסבר קצר על הפונקציה `str.title` שראיתם בהרצאה.

הערה כללית:

פונקציות של מחלקות ניתן להפעיל בשני אופנים שקולים. אם נסמן ב- `C` את שם המחלקה (למשל המחלקה `str`), וב- `c_obj` אובייקט קונקרטי מהמחלקה `C` (למשל מחרוזת "`abc`"), אז שתי הדרכים הן:

- `C.func(c_obj,…)`, כלומר הפרמטר הראשון הוא `c_obj` ואחריו יתר פרמטרים, אם דרושים.
- `c_obj.func(…)`, כלומר האובייקט `c_obj` לא מופיע בתוך הסוגריים אלא לפני שם הפונקציה.

להלן הדגמה על המחלקה `str`:

```
>>> course_name = "introduction to computer science"
>>> str.title(course_name)
'Introduction To Computer Science'
>>> course_name.title()
'Introduction To Computer Science'
```

מצאו שלוש פונקציות הקיימות במחלקה `str` שאינן קיימות במחלקה `list`, ובדקו מהו הפלט שלהן על המחרוזת "`xyzw`".

כעת, מצאו שלוש פונקציות הקיימות במחלקה `list` שאינן קיימות במחלקה `str`. בדקו אותן באופן דומה על הרשימה ['`x', 'y', 'z', 'w`'].

הערה: המושגים "מחלקה" ו"אובייקט" יוסברו יותר לעומק בהמשך הקורס.

שאלה 2

בכיתה ראיתם קוד בפייתון לחישוב ספרת ביקורת בתעודת זהות.

האלגוריתם לחישוב ספרת ביקורת בת״ז ישראלית מתואר בקישור הזה. להלן הקוד שראיתם:

```
def control_digit(id_num):
    """ compute the check digit in an Israeli ID number,
        given as a string of 8 digits
    """

    assert isinstance(id_num, str)
    assert len(id_num) == 8

    total = 0
    for i in range(8):
        val = int(id_num[i]) # converts char to int
        if i%2 == 0:     # even index (0,2,4,6)
            total += val
        else:            # odd index (1,3,5,7)
            if val < 5:
                total += 2*val
            else:
                total += ((2*val)%10) + 1 # sum of digits in 2*val
                                          # 'tens' digit must be 1

    total = total%10             # 'ones' (rightmost) digit
    check_digit = (10-total)%10  # the complement modulo 10 of total
                                 # for example 42->8, 30->0

    return str(check_digit)
```

א. הריצו את הפונקציה על קלט לא תקין משלושה סוגים:

a. טיפוס שאינו מחרוזת

b. מחרוזת באורך קצר מ-8

c. מחרוזת באורך 8 המכילה תווים שאינם ספרות

צפו בהודעות השגיאה המתקבלות והסבירו בקובץ ה-pdf בקצרה את מהות השגיאות.

ב. כעת, הסירו את שתי הפקודות המתחילות במילה `assert` וחזרו על ההרצות מהסעיף הקודם. הסבירו בקצרה מה השתנה ומדוע.

ג. הוסיפו לקובץ ה-pdf שתי טבלאות מעקב אחר המשתנים בתוכנית המופיעה מעלה, טבלה עבור כל אחד משני הקלטים הבאים:

a. `"87654321"` (כלומר הרצת הפקודה `control_digit("87654321")`).

b. מספר תעודת הזהות האישי שלכם.

הטבלה תיראה כך (בעמוד הבא):

iteration	i	id_num[i]	val	total
1				
2				
…				
8				

שימו לב: בכל שורה יש לרשום את ערכי המשתנים בסוף האיטרציה הרלוונטית. למשל בשורה הראשונה (iteration 1) יש לרשום את ערכי המשתנים ברגע סיום האיטרציה הראשונה של לולאת ה-for. לפיכך בשורה 8 יופיעו ערכי המשתנים בסיום הלולאה ("רגע לפני" ביצוע הפקודה שמופיעה אחרי הלולאה).

בדוגמא הבאה מוצגת פונקציה לחישוב מספר האפסים במספר טבעי, וטבלת מעקב מתאימה.

```
orig_num = 1203
num = orig_num
cnt = 0

if num == 0:
     cnt = 1
else:
     while num > 0:
          if num % 10 == 0:
               cnt += 1
          num //= 10

print(orig_num, "has", cnt , "0-s")
```

	orig_num	*num*	*cnt*	**Program output**
Before the loop	1203	1203	0	
After iteration # 1	1203	120	0	
After iteration # 2	1203	12	1	
After iteration # 3	1203	1	1	
After iteration # 4	1203	0	1	
After the loop	1203	0	1	"1203 has 1 0-s"

שאלה 3

נדון בבעיה החישובית הבאה: בהינתן מספר שלם חיובי num, נרצה לדעת כמה פעמים מופיעה בו הספרה 0. למשל עבור הקלט 10030 הפלט המתאים הוא 3.

מטרתנו בשאלה היא להשוות את זמני הריצה של שלושה פתרונות אפשריים לבעיה זו. לפניכם שלוש פונקציות הפותרות את הבעיה:

פתרון ראשון:

```
def zeros(num): # 1st solution
    m = num
    cnt = 0
    while m > 0:
        if m % 10 == 0:
            cnt = cnt + 1
        m = m // 10
    return cnt
```

פתרון שני:

```
def zeros2(num): # 2nd solution
    cnt = 0
    snum = str(num)  # num as a string
    for digit in snum:
        if digit == "0":
            cnt = cnt + 1
    return cnt
```

פתרון שלישי:

```
def zeros3(num): # 3rd solution
    cnt = str.count(str(num), "0")
    return cnt
```

שלושת הפונקציות מחזירות את התשובה כפלט ולכן נוכל להדפיס את הפתרון (למשל, של הפונקציה הראשונה) ע"י הפקודות:

```
num = 2**127
result = zeros(num)
print(num, "has", result, "zeros")
```

כדי למדוד זמן ריצה של פקודה או סדרת פקודות, נשתמש במעין "סטופר":

- נוסיף בראש התוכנית שלנו את הפקודה `import time`
- נוסיף מייד לפני קטע הקוד שאת זמן הריצה שלו ברצוננו למדוד את הפקודה:

```
t0 = time.perf_counter()
```

- נוסיף מייד לאחר קטע הקוד הנ"ל את הפקודה: `t1 = time.perf_counter()`
- זמן הריצה של קטע הקוד הוא ההפרש `t1-t0`. נוח להציגו למשל כך:

```
print("Running time: ", t1-t0, "sec")
```

הסבר קצר: `time` היא מחלקה של פייתון המאפשרת ביצוע פקודות שונות הקשורות לזמנים. הפקודה `import` הכרחית על מנת להשתמש במחלקה (היא "מייבאת" אותה. ניתקל במהלך הקורס בדוגמאות רבות ל"ייבוא" של מחלקות). למידע נוסף על המחלקה `time`: https://docs.python.org/3/library/time.html

א. מדדו את זמן הריצה של 2 הפתרונות הראשונים עבור המספרים: 2**100, 2**250, 2**600, 2**1400.

(תזכורת: האופרטור ** הוא אופרטור החזקה, כלומר x**y משוערך ל-x בחזקת y). ציינו מה היו זמני הריצה בטבלה שבה תהיה עמודה לכל אחד מהקלטים הנ"ל, וכן שורה עבור כל פתרון. הסבירו בקצרה את התוצאות (ובפרט התייחסו לקצב הגידול כתלות בגודל הקלט). ניתן, אם רוצים, להציג את התוצאות בגרף על מנת להקל על ההסבר.

שימו לב: על המדידה למדוד את זמן הריצה של הקריאה לפונקציה בלבד, ובפרט אין למדוד את הזמן של פקודות נוספות כגון הדפסת הפלט.

ב. פונקציות מובנות של פייתון, כמו למשל `str.count`, ממומשות בד"כ באופן יעיל למדיי, לעיתים אף באמצעות אלגוריתמים מסובכים יחסית. חיזרו על סעיף א' עבור הפתרון השלישי. מבלי להיכנס לפרטי המימוש של `str.count`, האם היא אכן יעילה יותר מבחינת זמן ריצה, בהשוואה לשני הפתרונות הראשונים?

ג. עבור קלטים בעלי מספר ספרות דומה, האם יש לפלט עצמו, כלומר למספר האפסים בקלט, השפעה כלשהי על זמן הריצה של כל אחד מהפתרונות? ביחרו קלטים מתאימים לבדיקת הסוגייה, ציינו מהם הקלטים בהם השתמשתם, הראו את תוצאות המדידות, והסבירו מה היא מסקנתכם.

ד. להלן לולאה פשוטה:

```
num = 2**1000
cnt = 0
for i in range(num):
     cnt = cnt + 1
```

תנו הערכה גסה לזמן שיקח ללולאה להסתיים. ציינו כל הנחה עליה התבססתם בהערכתכם. איך אתם מסבירים זאת, לאור העובדה שבסעיף א' לולאת ה- `for` של הפתרון השני רצה בזמן קצר באופן משמעותי?

שאלה 4

בשאלה זו נעבוד על ניתוח בסיסי של מחרוזות. בשאלה מספר סעיפים, ובכל סעיף יש לממש פונקציה אחת. בכל הסעיפים, אם לא נאמר אחרת, ניתן להניח כי המחרוזות שבקלט מכילות ספרות (0 עד 9), אותיות קטנות באנגלית (a, b, c וכו') רווחים, וסימני הפיסוק ! . , ? .

שימו לב – בכל אחד מהסעיפים יתכן כי מחרוזת הקלט (או אחת מהן אם יש יותר מאחת) היא המחרוזת הריקה. ודאו כי הפתרון שלכם מטפל גם במקרה זה.

סעיף א'

ממשו את הפונקציה `union_strings(st1, st2)`. הפונקציה מקבלת כקלט שתי מחרוזות `st1, st2`.

הפונקציה תחזיר מחרוזת המכילה את התווים המופיעים לפחות באחת משתי המחרוזות. כלומר, התו `x` יופיע במחרוזת הפלט אם ורק אם `x` מופיע ב-`st1` <u>או</u> ב-`st2` או בשתיהן.

<u>הנחיות:</u> כל תו יופיע במחרוזת הפלט לכל היותר פעם <u>אחת</u> (אף על פי שבקלט התווים יכולים לחזור על עצמם, כפי שרואים בדוגמא בהמשך). <u>אין</u> חשיבות לסדר התווים במחרוזת הפלט.

<u>דוגמת הרצה:</u>

```
>>> union_strings("aabcccdde", "bccay")
'cbayed'
```

סעיף ב'

בשאלה זו נממש פונקציה שתשנה מחרוזת לפי תבנית מסוימת; כלומר, נממש גרסא מצומצמת של ה formatting שפייתון מאפשר (<u>לינק</u>). הפונקציה `format_str` תקבל כקלט מחרוזת בשם `text_to_format` וכן `st_to_insert`. על הפונקציה להכניס את `st_to_insert` למקומות המסומנים ב-`text_to_format` ולהחזיר את המחרוזת המתוקנת. בפרט, במחרוזת `text_to_format` הפונקציה תחליף כל תו '?' במחרוזת הנתונה ב-`st_to_insert`, אשר מכילה רק תווים באנגלית.

למשל עבור הקלטים `text_to_format` ו-`st_to_insert` (משמאל לימין):

```
'? is among the most popular programming languages, thus making ?
important to learn.', 'Python'
```

הפונקציה תחזיר:

```
'Python is among the most popular programming languages, thus making
Python important to learn.'
```

<u>הנחיות:</u> **אין** להשתמש בפונקציה המובנית `replace` או בפונקציות מובנות דומות אחרות.

<u>דוגמאות הרצה:</u>

```
>>> format_str('I2?','CS')
"I2CS"
>>> format_str('???', 'W')
"WWW"
>>> format_str('ABBC', 'z')
"ABBC"
```

סעיף ג'

פלינדרום (Palindrome) היא מחרוזת הזהה להיפוך שלה. לדוגמא, המחרוזת *radar* היא פלינדרום (שימו לב שע"פ הגדרה זו המחרוזת *race car* למשל איננה פלינדרום עקב הריווח שלה). למספר שלם אי-שלילי k נגדיר **k-כמעט פלינדרום** להיות מחרוזת הזהה להיפוך שלה, עד כדי לכל היותר k אי-התאמות. כלומר, מחרוזת היא k-כמעט פלינדרום אם המספר המינימלי של תווים שיש **לשנות** (בשונה מלמחוק או להוסיף) כדי שתהפוך לפלינדרום הוא k. לדוגמא, *race car* היא 1-כמעט פלינדרום כי אם למשל נשנה את הרווח ל-e היא תהיה פלינדרום. שימו לב כי 0-כמעט פלינדרום היא הגדרה זהה לפלינדרום.

ממשו את הפונקציה `least_pal(text)` המקבלת כקלט מחרוזת `text`. הפונקציה תחזיר את המספר השלם (אי שלילי) הנמוך ביותר k כך ש-`text` היא k-כמעט פלינדרום.

דוגמאות הרצה:

```
>>> least_pal('abcdefgh')
4
>>> least_pal('race car')
1
```

סעיף ד'

ממשו את הפונקציה `least_frequent(text)`. הפונקציה תחזיר את התו שמספר המופעים שלו בה הוא **הקטן** ביותר מתוך אלו **שמופיעים** במחרוזת. ניתן להניח כי אין שני תווים במחרוזת שמספר המופעים שלהם הוא מינימלי וכן כי `text` איננה מחרוזת ריקה.

הנחיה: אין להשתמש במתודה `count` של המחלקה `str`.

```
>>> least_frequent('aabcc')
'b'
>>> least_frequent('aea.. e')
' '
>>> least_frequent('zzz')
'z'
```

סעיף ה'

ממשו את הפונקציה `longest_common_suffix(lst)`. הפונקציה מקבלת כקלט רשימת מחרוזות (לא ריקה) המורכבות מתווים קטנים באנגלית בלבד. הפונקציה תחזיר את תת-המחרוזת הארוכה ביותר שמהווה סיפא (suffix, התווים המופיעים בסוף המחרוזת) עבור **כל** המחרוזות. בהיעדר כזו, תוחזר המחרוזת הריקה ("").

דוגמאות הרצה:

```
>>> longest_common_suffix(["abccdba", "cba", "zaba"])
'ba'
>>> longest_common_suffix(["hello", "world"])
''
>>> longest_common_suffix(["intro", "maestro"])
'tro'
```

```
>>> longest_common_suffix(["intro"])
'intro'
```

סעיף ו'

ממשו את הפונקציה `is_int(text)` המקבלת כקלט מחרוזת המורכבת מתווים כלשהם (לאו דווקא ספרות, אותיות רווחים ונקודות). הפונקציה תחזיר `True` אם `text` היא מחרוזת המייצגת מספר שלם (חיובי, שלילי או אפס) באופן תקין ו-`False` אחרת. בייצוג תקין הכוונה היא כי המספר לא מכיל אפסים מובילים (מלבד המספר אפס).

הנחיה: בסעיף זה אין להשתמש בפונקציות המרה (לדוגמא הפונקציה המובנית `int`) או בפונקציות המבצעות את הבדיקה ישירות דוגמת `isdecimal`.

דוגמאות הרצה:

```
>>> is_int("123")
True
>>> is_int("0123")
False
>>> is_int("49x2")
False
>>> is_int("-0")
False
>>> is_int("0")
True
```

הערה: בדוגמא השלישית רואים כי `"-0"` איננו יצוג תקין של שלם מבחינתנו, וזאת על אף שמבחינת פייתון זהו ייצוג תקין ושקול למספר 0.

סעיף ז' (רשות)

ממשו את הפונקציה `merge(text1, text2)`. הפונקציה תקבל כקלט שתי מחרוזות `text1, text2` המכילות תווים קטנים באנגלית בלבד (כלומר, תווים מ-a עד z). מובטח לנו כי בכל מחרוזת קלט התווים מסודרים בסדר עולה (כלומר, לא יתכן למשל שמחרוזת תכיל את התו 'e' ובנקודה כלשהיא אחריו יופיע התו 'd'). הפונקציה תחזיר מחרוזת המשלבת את התווים שבשתי מחרוזות הקלט בסדר עולה.

טיפ: בהינתן שני תווים, ניתן להשתמש באופרטור > או באופרטור => כדי לבדוק את הסדר בין התווים, כמו בדוגמאות הבאות:

```
>>> 'd' < 'k'
True
>>> 'b' <= 'b'
True
>>> 'c' < 'a'
False
```

שימו לב שהשוואה זו לא בהכרח תעבוד באופן תקין עבור מצב בו נשווה תווים קטנים לגדולים (בהמשך הקורס נלמד מדוע).

הנחיה: בסעיף זה **אין להשתמש** בפונקציות מיון מובנות של פייתון.

דוגמאות הרצה:

```
>>> merge("abccc", "aabdd")
'aaabbcccdd'
>>> merge("aaddxx", "")
'aaddxx'
>>> merge("abc", "xyz")
'abcxyz'
```

שאלה 5

אנגרמה (לינק) ('טריפת אותיות') היא מחרוזת שנוצרה מסידור מחדש של תווי מחרוזת אחרת (ללא הוספה/מחיקה של אף תו). לדוגמא המחרוזת *silent* היא אנגרמה של המחרוזת *listen* (וההפך). בפרט, כל מחרוזת היא אנגרמה (טריוויאלית) של עצמה, וכל היפוך מחרוזת הוא אנגרמה של המחרוזת המקורית.

סעיף א'

ממשו את הפונקציה `is_anagram(st1, st2)` המקבלת כקלט שתי מחרוזות `st1, st2` המורכבות מתווים קטנים באנגלית בלבד, הפונקציה תחזיר ערך `True` אם `st1` היא אנגרמה של `st2`, ואחרת `False`.

הנחיות: אין להשתמש בפונקציות ספירת תווים ובפונקציות מיון מובנות של פייתון.

דוגמאות הרצה:

```
>>> is_anagram("tommarvoloriddle", "iamlordvoldemort")
True
>>> is_anagram("abce", "abcd")
False
>>> is_anagram("listen", "silent")
True
```

סעיף ב'

בהמשך לסעיף א', כעת מותר בנוסף השימוש בפונקציה המובנית count (לינק) (ובה בלבד). פתרון הקוד הבא הוצע למימוש בעיית האנגרמה:

```
def is_anagram_v2(st1, st2):
    for ch in st1:
        if st1.count(ch) != st2.count(ch):
            return False
    return True
```

בפתרון המוצע יש שגיאה. הסבירו בקצרה מה לא נכון במימוש ומיצאו קלט עבורו התכנית **לא** תפעל כמצופה.

ממשו גירסה מתוקנת של is_anagram_v2.

סעיף ג'

כעת הותר השימוש בפונקציות המובנות של מיון (לינק) (ובהן בלבד). בסעיף זה נפתור את בעיית האנגרמה שצוינה בסעיף א', הפעם ע"י מימוש שונה. עליכם לממש את הפונקציה `is_anagram_v3`, הזהה בדרישותיה לסעיף

הקודם, **בשורה אחת**.

הסבירו בקצרה מה השוני במימוש בין סעיף א׳ לנוכחי; איזה מהם עשוי לרוץ מהר יותר (היזכרו בשאלה 3)? האם נכון יהיה לומר כי באופן כללי פתרון שלוקח פחות שורות לכתוב אותו יהיה יעיל ומהיר יותר? הסבירו מדוע.

שאלה 6

חלק 1

הגדרה: **מונום** הוא פונקציה מהצורה $m(x) = ax^b$ כאשר $a \in \mathbb{Z} \setminus \{0\}$ ו-$b \in \mathbb{N}$ (כלומר, a מספר שלם שאיננו אפס ו-b שלם שאיננו שלילי). ל-a נקרא **המקדם** של המונום, ול-b **הדרגה** של המונום. להלן דוגמאות למונומים:

$-5x^{10}, 4x^1, -2x^0$.

נייצג מונום בפייתון על ידי מחרוזת:

- המתחילה בתו "+" או "−" (בהתאם לסימן של a)
- אחריה יופיע המקדם a
- אחריו התו "x" ואחריו התו "^" המסמן חזקה
- לבסוף תופיע הדרגה b

למשל, את המונום $5x^3$ נייצג על ידי המחרוזת "+5x^3" ואת המונום $-40x^1$ על ידי המחרוזת "-40x^1".

הגדרה: **פולינום** הוא פונקציה המורכבת מסכום של מונום אחד או יותר מדרגות שונות.

לדוגמא: $-5x^4 + 3x^2 + 1x^{10}$.

נייצג פולינום בפייתון על ידי שרשור המחרוזות המייצגות את המונומים המרכיבים את הפולינום. למשל את הפולינום $2x^2 - 7x^5 + 1x^3$ נייצג על ידי המחרוזת "+2x^2-7x^5+1x^3".

שימו לב כי המחרוזות בייצוגים **לא** מכילות סוגריים אף פעם.

הנחיות נוספות:

- בשאלה זו אין להשתמש בספריות חיצוניות או בפקודות שיערוך מובנות (כמו eval).
- מותר להניח שהקלט לפונקציות שאתם נדרשים לממש הוא תקין בהתאם להגדרות הנ״ל ואין צורך לבדוק זאת. בפרט הקלט אינו מחרוזת ריקה.

סעיף א׳

ממשו את הפונקציה `eval_mon(monomial, val)` המקבלת כקלט מחרוזת בשם `monomial` ומשתנה מטיפוס `int` בשם `val`. הפונקציה תפרש את המחרוזת `monomial` כמונום `m(x)` ותחזיר את הערך של `m` עבור `x = val` (כלומר, את הערך `m(val)`). פעולה זו נקראת שערוך (evaluation).

דוגמאות הרצה:

```
>>> eval_mon("+5x^3", 4)
320
```

```
>>> eval_mon("-10x^0", 4)
-10
>>> eval_mon("+1x^10", 2)
1024
```

סעיף ב'

בסעיף זה נשתמש בפונקציה `eval_mon` לשערוך פולינום.

ממשו את הפונקציה `eval_pol(polynomial, val)` המקבלת כקלט מחרוזת בשם `polynomial` ומשתנה מטיפוס `int` בשם `val`. הפונקציה תפרש את המחרוזת `polynomial` כפולינום `p(x)` ותחזיר את הערך של `p` עבור `x = val` (כלומר את הערך `p(val)`).

הנחיות: יש לקרוא ל-`eval_mon`.

דוגמאות הרצה:

```
>>> eval_pol("+5x^3-17x^2", 1)
-12
>>> eval_pol("+11x^12", 2)
45056
>>> eval_pol("+5x^3-4x^2+7x^1-5x^0", 4)
279
>>> eval_pol("+5x^3-4x^2+7x^1-5x^0", 0)
-5
```

חלק 2

הקדמה – צ'אטבוט הקורס

מטרת חלק זה בשאלה היא להתנסות בשימוש בכלי בינה מלאכותית (AI), ובפרט במודלים של שפה (LLMs – Large Language Models) כדוגמת ChatGPT או Claude. אופן הפעולה של כלים אלו אינו כלול בחומר הקורס והוא נושא שנלמד בקורסי בחירה מתקדמים כמו עיבוד שפה טבעית. יחד עם זאת השימוש בהם פשוט ונגיש לכל, וחשוב להבין את היכולות ובמיוחד את המגבלות שלהם (נכון להיום).

בימים אלו אנחנו בונים צ'אטבוט עבור הקורס מבוא מורחב למדעי המחשב, שמטרתו לעזור לסטודנטים ולסטודנטיות בהתמודדות עם תרגילי הבית והבנתם לעומק, ע"י קבלת משוב מיידי על נסיונות לפתרון השאלה. הצ'אטבוט "מכיר" את חומרי הקורס וכן את השאלה הספציפית שבעמוד הבא, ויכול לסייע לכם באופן הבא: בהינתן קוד עם נסיון שלכם לפתור את השאלה, בין אם נכון או שגוי, הצ'אטבוט "יחווה דעתו" על הפתרון שלכם ויציע הצעות לתיקון שגיאות (אם מצא כאלו). זיכרו שכמו כל מודל שפה, גם הצ'אטבוט שלנו עלול לטעות ולהטעות. בהמשך, אנחנו נבקש לקבל מכם משוב על השימוש בכלי כדי לשפר אותו עבורכם בתרגילים הבאים הסמסטר ועבור הסמסטרים הבאים.

הערות על השימוש בצ'אטבוט:

- הצ'אטבוט אוסף מידע על האינטראקציה איתו, אבל המידע **אנונימי** ובכל מקרה אין לאינטראקציה עצמה השפעה על הציון שלכם בתרגיל או בקורס. יחד עם זאת, אל תשתפו פרטים אישיים (גם אין סיבה לעשות זאת).
- האינטראקציה היא באנגלית בלבד
- בכל שאלה או בעיה שעולות יש לפנות למייל: chatbot1001@outlook.com.
- לינק לצ'אטבוט יפורסם בעוד כמה ימים במודל עם הנחיות שימוש.

הנחיות לשימוש בצ'אטבוט עבור השאלה הנוכחית

נסו תחילה להתמודד בעצמכם עם שני הסעיפים שבחלק הראשון של שאלה זו. לאחר שנפרסם את הקישור לצ'אטבוט, עליכם להעתיק את ניסיון הפתרון שלכם: עבור סעיף א (eval_mon) תנו לו את המימוש שלכם לפונקציה זו בלבד.

עבור סעיף ב' (eval_pol) הזינו לצ'אטבוט את המימוש שלכם לשני הסעיפים (כי סעיף ב מסתמך על א).

בחנו את התשובה שקיבלתם – וציינו האם היא מועילה ע"י לחיצה על אחד הכפתורים שמתחת לתשובת הצ'אטבוט:

helpful | not helpful | partially helpful

כעת אתם יכולים לנסות פעם נוספת, עד 6 פעמים בכל יום.

סוף.