תרגיל בית מספר 1 – להגשה עד 26/10/2021 – אביתר שמש 322623182

תרגיל מספר 1

3 פונקציות שקיימות במחלקה str ואינן קיימות במחלקה list:

1. capitalize() – הופכת את האות הראשונה ב str לאות "גדולה"

```
>>> str1 = "xyzw"
>>> lst1 = ["x","y","z","w"]
>>> str1.capitalize()
'Xyzw'
>>> str.capitalize(str1)
'Xyzw'
>>> lst1.capitalize()
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    lst1.capitalize()
AttributeError: 'list' object has no attribute 'capitalize'
>>> list.capitalize(lst1)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    list.capitalize(lst1)
AttributeError: type object 'list' has no attribute 'capitalize'
```

2. lower() - הופכת כל אות ב str לאות "קטנה"

```
>>> str1 = "xyzw"
>>> lst1 = ["x","y","z","w"]
>>> str1.lower()
'xyzw'
>>> str.lower(str1)
'xyzw'
>>> lst1.lower()
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    lst1.lower()
AttributeError: 'list' object has no attribute 'lower'
>>> list.lower(lst1)
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    list.lower(lst1)
AttributeError: type object 'list' has no attribute 'lower'
```

3. upper() – הופכת כל אות ב str לאות "גדולה"

```
>>> str1 = "xyzw"
>>> lst1 = ["x","y","z","w"]
>>> str1.upper()
'XYZW'
>>> str.upper(str1)
'XYZW'
>>> lst1.upper()
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    lst1.upper()
AttributeError: 'list' object has no attribute 'upper'
>>> list.upper(lst1)
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    list.upper(lst1)
AttributeError: type object 'list' has no attribute 'upper'
```

3 פונקציות שקיימות במחלקה list ואינן קיימות במחלקה str:

1. pop([i]) – מוציאה מהרשימה את האיבר במיקום ה-i

```
>>> str1 = "xyzw"
>>> lst1 = ["x","y","z","w"]
>>> lst1.pop(0)
'x'
>>> list.pop(lst1,0)
'y'
>>> str1.pop(0)
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    str1.pop(0)
AttributeError: 'str' object has no attribute 'pop'
>>> str.pop(str1)
Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    str.pop(str1)
AttributeError: type object 'str' has no attribute 'pop'
```

2. append(x) – מכניסה את האובייקט object כאיבר במיקום האחרון ברשימה

```
>>> str1 = "xyzw"
>>> lst1 = ["x","y","z","w"]
>>> lst1.append(1)
>>> lst1
['x', 'y', 'z', 'w', 1]
>>> list.append(lst1,2)
>>> lst1
['x', 'y', 'z', 'w', 1, 2]
>>> str1.append(1)
Traceback (most recent call last):
  File "<pyshell#47>", line 1, in <module>
    str1.append(1)
AttributeError: 'str' object has no attribute 'append'
>>> str.append(str1,2)
Traceback (most recent call last):
  File "<pyshell#48>", line 1, in <module>
    str.append(str1,2)
AttributeError: type object 'str' has no attribute 'append'
```

3. insert(i,x) – מכניסה את האובייקט object כאיבר במיקום ה – i ברשימה

```
>>> str1 = "xyzw"
>>> lst1 = ["x","y","z","w"]
>>> lst1.insert(2,"new")
>>> lst1
['x', 'y', 'new', 'z', 'w']
>>> list.insert(lst1,3,"old")
>>> lst1
['x', 'y', 'new', 'old', 'z', 'w']
>>> str1.insert(2,"new")
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    str1.insert(2,"new")
AttributeError: 'str' object has no attribute 'insert'
>>> str.insert(str1,3,"old")
Traceback (most recent call last):
  File "<pyshell#60>", line 1, in <module>
    str.insert(str1,3,"old")
AttributeError: type object 'str' has no attribute 'insert'
```

תרגיל מספר 2

א.

- הרצה עם קלט מסוג int, לדוגמא 12345678.

```
>>> control_digit(12345678)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    control_digit(12345678)
  File "C:/Users/eviap/OneDrive - mail.tau.ac.il/מבוא מורחב למדעי המחשב/control_digit.py", line 3, in control_digit
    assert isinstance(ID, str)
AssertionError
>>>
```

מהות השגיאה הינה שהקלט שהכנסנו אינו תואם לבדיקת הקלט הראשונה, בה נבדק ש - ID הינו מסוג str, אך הקלט שהכנסנו הינו מסוג int.

- הרצת str באורך שונה מ – 8,לדוגמא str "12" (באורך 2)

```
>>> control_digit("12")
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    control_digit("12")
  File "C:/Users/eviap/OneDrive - mail.tau.ac.il/מבוא מורחב למדעי המחשב/control_digit.py", line 4, in control_digit
    assert len(ID) == 8
AssertionError
```

מהות השגיאה הינה שהקלט שהכנסנו אינו תואם לבדיקת הקלט השנייה, ש ID צריך להיות באורך 8, אך בקלט שהכנסנו אורך הקלט הינו 2.

ב.

טבלת מעקב "87654321"

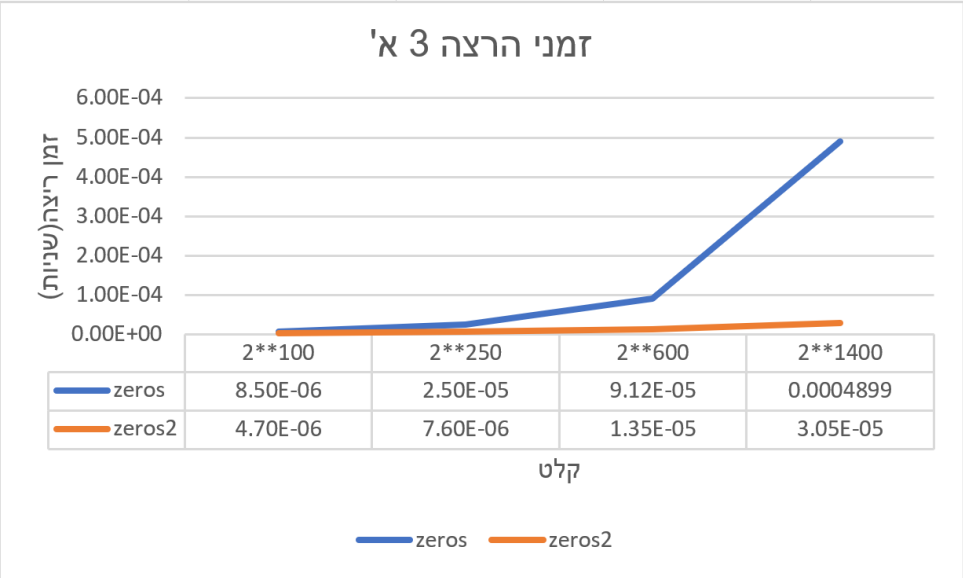
Iteration	i	ID[i]	val	total	Output
1	0	“8”	8	8	
2	1	“7”	7	13	
3	2	“6”	6	19	
4	3	“5”	5	20	
5	4	“4”	4	24	
6	5	“3”	3	30	
7	6	“2”	2	32	
8	7	“1”	1	34	“6”

טבלת מעקב ת.ז. שלי: "32262318"

Iteration	i	ID[i]	val	total	Output
1	0	“3”	3	3	
2	1	“2”	2	7	
3	2	“2”	2	9	
4	3	“6”	6	12	
5	4	“2”	2	14	
6	5	“3”	3	20	
7	6	“1”	1	21	
8	7	“8”	8	28	“2”

תרגיל מספר 3

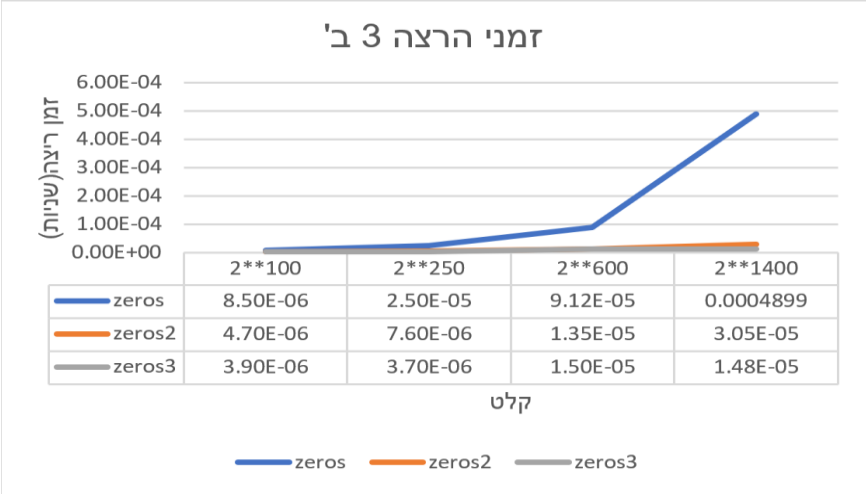
א.



באופן כללי, זמני הריצה גדלים ככל שהקלט גדל, כי כמות הפעמים שהלולאה בכל אחת מהפונקציות רצה, גדלה.

ניתן לראות, שככל שהקלט גדל, זמני הריצה של zeros גדלים באופן מובהק, לכן השפעת גודל הקלט משמעותית.
לעומתו, זמני הריצה של zeros2 גדלים באופן המתקרב לקבוע, לכן השפעת גודל הקלט היא מועטת.

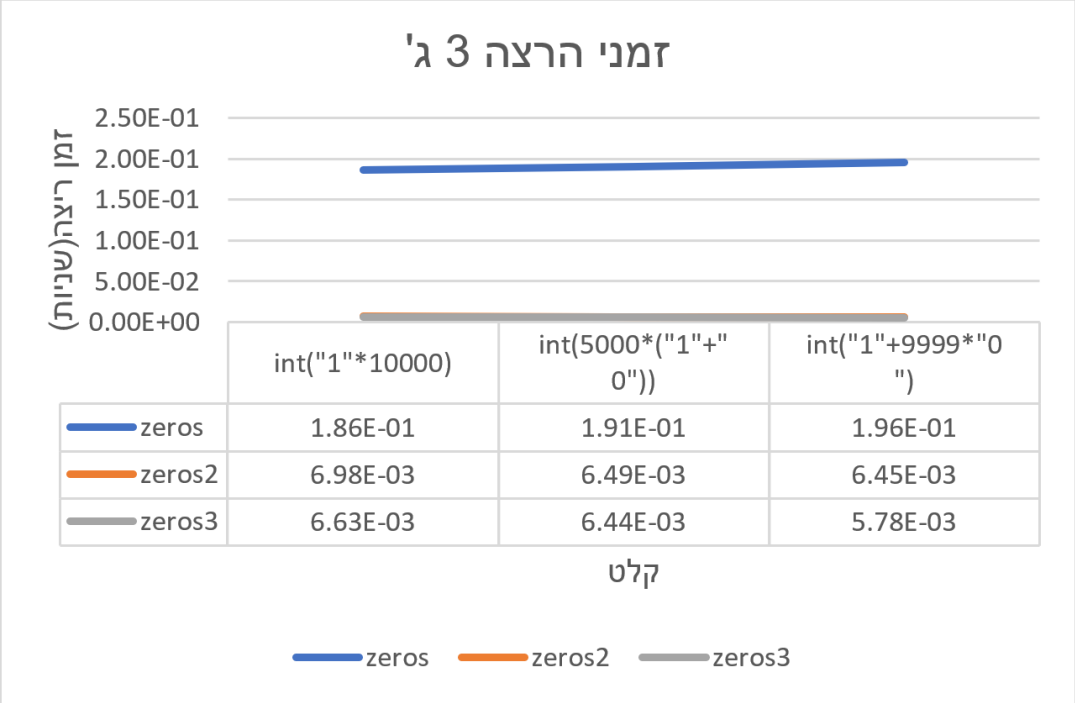
ב.



ניתן לראות ש zeros3 אכן יעילה יותר משני הפתרונות הראשונים כתלות בגודל הקלט, אך די דומה ליעילותה של zeros2.

יעילותן של zeros2 ו zeros3 נשמרת באופן יחסי ככל שהקלט גדל, בעוד יעילותה של zeros נפגמת באופן משמעותי ככל שהקלט גדל.

ג.



הקלטים שבחרתי הינם:

- רצף של 10000 פעמים 1
- רצף של 5000 פעמים 1 ואז 0
- 1 ולאחריו 9999 פעמים 0

מטרת הקלטים הללו היא עבור מספר מאוד גדול, להראות שיש הבדל קטן בין הפונקציות עבור כמות האפסים במספר, אך לא הבדל משמעותי שמשנה לחלוטין את מידת היעילות של כל אחת מהפונקציות אל מול השנייה.

zeros נשארת הכי פחות יעילה.

היעילות של zeros2 ו zeros3 נשארת מאוד קרובה אחת לשנייה.

ד. בלולאה זו מבוצעות 3 פעולות:

- בדיקה האם i לא גדול מ num
- העלאת המשתנה cnt ב 1
- הגדלת i ב 1

על המספר 2**1000, הלולאה תצטרך לרוץ 2**1000 פעמים, כלומר 3*(2**1000) יבוצעו.

הזמן שייקח ללולאה לרוץ תלוי במהירות המעבד, לצורך ההמחשה ניקח מעבד של מחשב על עם כוח חישוב עצום(ביחס למעבד של מחשב ביתי) של 10**15 פעולות לשנייה. ועדיין, ייקח בהערכה גסה 1.01e**294 שנים לסיום הרצת הפעולה.. רצוי שלא נחכה לקבל את מספר האפסים של מספר זה, ונשקיע את הזמן בכתיבת קוד יעיל יותר 😉.

החישוב מוסבר ומחושב באמצעות הקוד מטה.

```
>>> 3*(2**1000)/(10**15)*60*60*24*365
1.0137328630867837e+294
```

ניתן להסביר זאת לאור העובדה שבסעיף א' לולאת ה-for של הפתרון השני(zeros2) רצה בזמן קצר באופן משמעותי בכך שהיא לא רצה על כל מספר בין 0 ל 2**1000(כלומר 2**1000 פעמים), אלא רצה ככמות הספרות במספר זה, שהיא לפי הרצת הקוד הבא 302 פעמים.

```
>>> len(str(2**1000))
302
```

<u>תרגיל מספר 4</u>

א. אם היינו רוצים שגם הספרות ייבדקו ויחושבו בספירת התווים השונים, הייתי משנה את הרשימה lst1 שתכיל גם את כל הספרות(0-9) , וככה הבדיקה הייתה חלה גם על הספרות.

ב. ניתן להשתמש במתודה זו כדי למחוק תו מסוים ע"י השמת new=""". מה שיקרה זה שבכל פעם שב-text יופיע old, במקום להתווסף למחרוזת החדשה ערכים, לא יתווסף כלום כי new היא מחרוזת ריקה.