שאלה 2

א. השגיאה שקיבלנו היא "AssertionError". הפונקציה control_digit מצפה לקבל מחרוזת של 8 ספרות. מהות השגיאה היא להתריע בפני המשתמש שהוא הכניס קלט שחורג מדרישת הפונקציה. אילולא הייתה עושה זאת, היה עלול להיווצר מצב שבו המשתמש מקבל פלט שגוי או במקרה אחר התוכנית אפילו עלולה לקרוס.

ב.

a.

87654321

iteration	i	ID[i]	val	total
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

b.

31887960

iteration	i	ID[i]	val	total
1	0	‘3’	3	3
2	1	‘1’	1	5
3	2	‘8’	8	13
4	3	‘8’	8	20
5	4	‘7’	7	27
6	5	‘9’	9	36
7	6	‘6’	6	42
8	7	‘0’	0	42

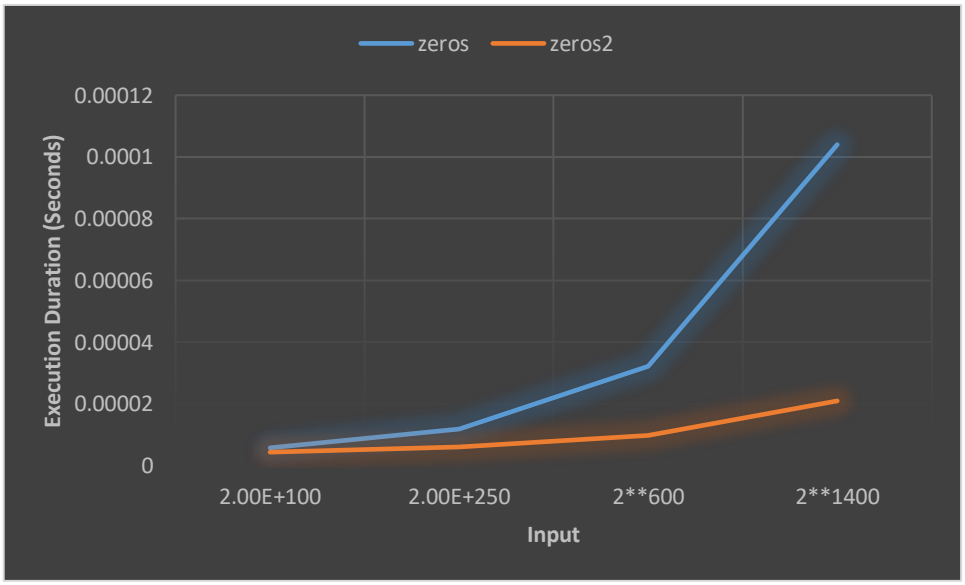
שאלה 3

א. להלן טבלת זמני הריצה של שתי הפונקציות zeros, zeros2 עם הקלטים הנדרשים:

קלט:	2**100	2**250	2**600	2**1400
zeros (בשניות)	0.0000058	0.0000118	0.0000322	0.0001040
zeros2 (בשניות)	0.0000044	0.0000061	0.0000099	0.0000210

ניתן לראות שככל שהקלט גדול יותר כך גם זמן הריצה גדול יותר, כלומר ככל שהקלט גדול יותר לוקח יותר זמן לשתי הפונקציות לחשב את כמות האפסים של המספר.

בגרף, החוקיות באה לידי ביטוי בשיפוע הגרף- עולה ממש (בשתי הפונקציות).



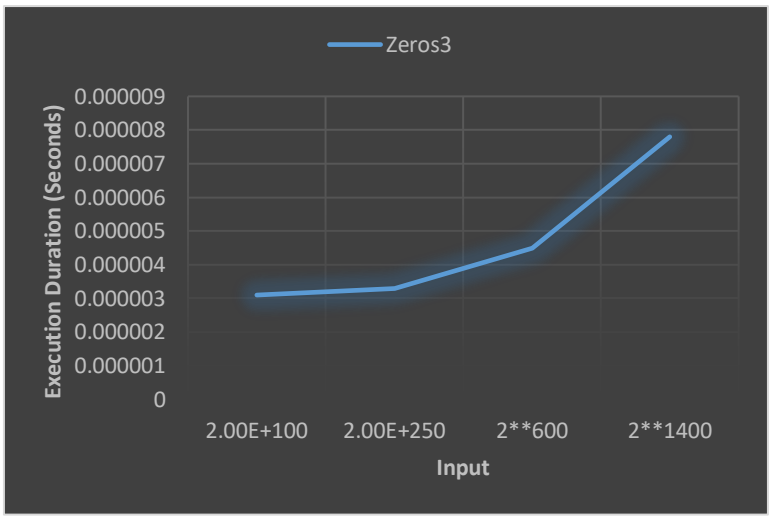
ב. להלן טבלת זמני הריצה של הפונקציה zeros3 עם הקלטים הנדרשים:

קלט:	2**100	2**250	2**600	2**1400
Zeros3 (בשניות)	0.0000031	0.0000033	0.0000045	0.0000078

ניתן לראות שעבור כל אחד מהקלטים זמן הריצה של zeros3 קטן מזמני הריצה של שתי הפונקציות zeros1, zeros2.

ולכן היא אכן יעילה יותר מבחינת זמן ריצה בהשוואה לשני הפתרונות הראשונים.

גרף עם זמני הריצה של zeros3:



ג. כדי לבחון את העניין, הכנסתי לכל פונקציה כקלט את שני הערכים הבאים:
   i. המספר שמורכב מ1000 ספרות שכולן 1: int(“1”*1000).
   ii. המספר שספרתו השמאלית ביותר 1 ומימינו 999 אפסים: int(“1”+”0”*999).

הפונקציות (סביר להניח שגם השלישית) מבדילות בין ספרות שהן 0 לבין אילו שלא, ולכן מספיק לבדוק מספר שמורכב רק מאחדים ואין צורך להציג תוצאות עם ספרות אחרות. בחרתי במקרה קיצון שבו כל ספרות המספר הראשון אינן אפס וכל ספרות המספר השני למעט השמאלית ביותר הן 0 (חשוב כי הפונקציה הראשונה קובעת שיש עוד ספרות שצריך לבדוק למספר אם הוא גדול מ-0), כדי שאוכל לראות בבירור אם קיים הבדל בזמן הריצה. כמו כן, נשים לב ששני מספרים אלו בעלי אותה כמות ספרות- 1000.

להלן זמני הריצה:

פונקציה/קלט	int(“1”*1000)	int(“1”+”0”*999)
zeros (seconds)	0.0004675	0.0006330
zeros2 (seconds)	0.0000514	0.0000753
zeros3 (seconds)	0.0000212	0.0000221

ניתן לראות שלכל אחת משלושת הפונקציות לוקח יותר זמן לתת תשובה עבור המספר עם האפסים (המספר הii), באופן שאינו זניח (הרצתי את הבדיקה יותר מפעם אחת וקיבלתי זמנים דומים ובכך הורדתי משמעותית את הסיכוי שמדובר בצירוף מקרים).

אומנם zeros3 מציגה זמנים קרובים יחסית, אבל היא אכן עקבית בכך שזמן הריצה של המספר עם כמות האפסים הגדולה יותר- גדול יותר.

המסקנה המתבקשת היא שלמספר האפסים בקלט יש השפעה על זמן הריצה של כל אחד מהפתרונות, כך שככל שיש יותר אפסים במספר, זמן הריצה גדול יותר.

ד. עבור num=10**7, לוקח ללולאה בערך 0.75 שניות. עבור num=1 לוקח ללולאה 0.0000039 שניות, כלומר לעומת מספר בגודל 10**7 הזמן שלוקח להריץ את השורה שבה אנו מצהירים על הלולאה הוא זניח בהחלט. ולכן, ניתן לומר שלוקח ללולאה בערך 0.75 שניות לכל 10**7 איטרציות. ואז כדי לתת הערכה גסה, נוכל לומר שעבור 2**1000 איטרציות יקח ללולאה 0.75*[(2**1000)/(10**7)] שניות, כלומר בערך 8*10**293 שניות, שאלו בערך 25,000,000 שנים.

ההבדל בין לולאה זו ללולאה בסעיף א הוא שבסעיף א הלולאה ביצעה איטרציות ככמות הספרות בקלט, ואילו בלולאה הנוכחית כמות האיטרציות היא המספר עצמו שהוכנס כקלט. כלומר עבור קלט של מספר בעל n ספרות הלולאה בסעיף א בצעה n איטרציות בעוד שהלולאה הנוכחית ביצעה לפחות 10**(n-1) איטרציות. ולכן עבור מספר גדול מאוד כמו 2**1000 ההבדל בין כמות האיטרציות של הלולאה בסעיף א לבין הלולאה הנוכחית הוא עצום, וכך גם ההבדל בזמן הריצה.

<u>שאלה 4</u>

ג. הייתי מוסיף את הספרות 0-9 ואת סימני הפיסוק השונים למחרוזת של הalphabet וכמובן מעדכן את שמות המשתנים מהתייחסות לאותיות להתייחסות לתווים שיש למנות.