1. .

2. .

   a. Both errors present an "AssertionError" - that happens when an "assert" command fails to ensure the condition it's provided. When providing an input that is not of a string instance, or a string that is not 8 chars in length, one of the "assert" lines fails and we get this error.

   b. "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

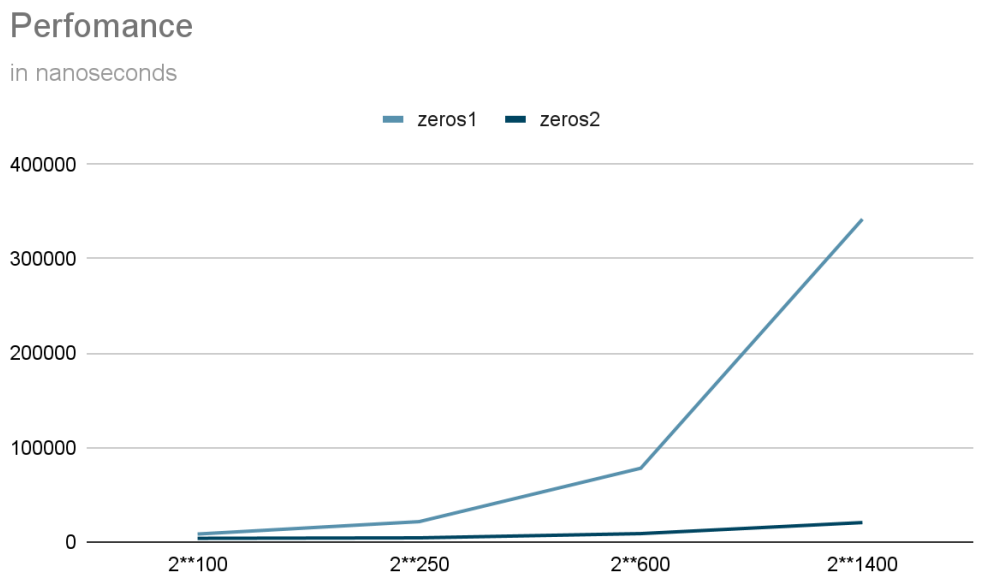
   My ID ("32286885")

iteration	i	ID[i]	val	total
1	0	'3'	3	3
2	1	'2'	2	7
3	2	'2'	2	9
4	3	'8'	8	16
5	4	'6'	6	22
6	5	'8'	8	29
7	6	'8'	8	37
8	7	'5'	5	38

3. For better understanding of the tiny differences between the functions, the values are hereby presented in nanoseconds.

   a.

	zeros1	zeros2
2**100	8442	3907
2**250	21577	4420
2**600	78172	8908
2**1400	341924	20610



We can clearly see that while for `2**100`, zeros0 performance time was about twice of that of zeros1, as we progressed into bigger numbers the difference grew. While zeros1 performance time grew quite slowly (considering the differences between the lengths of the numbers), zeros0 time grew rapidly and in an almost exponential manner.

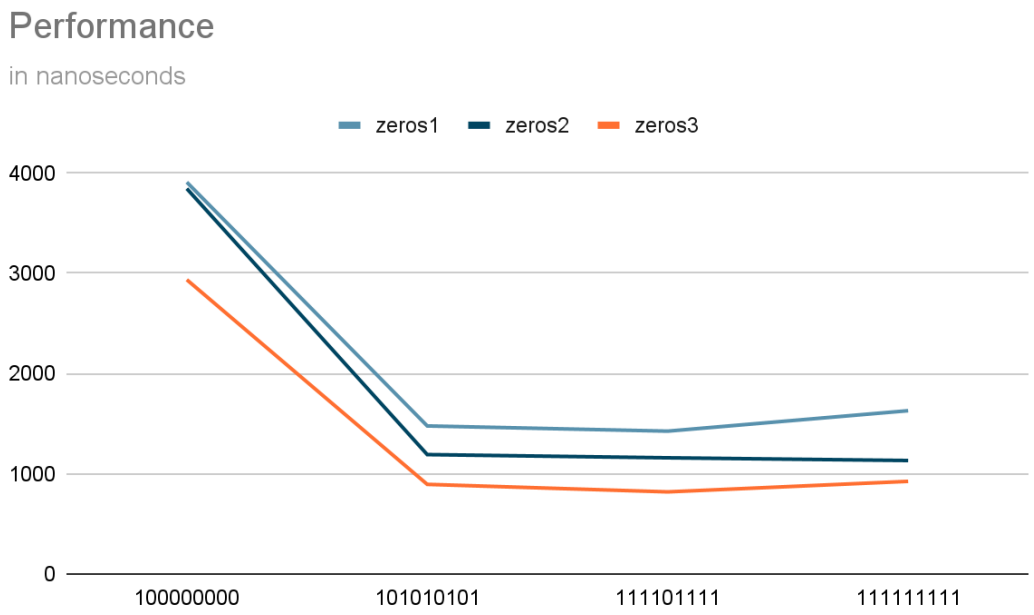
   b.

	zeros3
2**100	4242
2**250	2130

2**600	2129
2**1400	4790

Python's own str.count() function is indeed more efficient than zeros1, and more efficient than zeros2 in all but the first number (although, even there, the difference is miniscule). Something interesting happens here: the function actually gets **more** performant as we input bigger numbers into it, at least with the second and third number.

c. Below are the outputs:



	zeros1	zeros2	zeros3
100000000	3909	3845	2936
101010101	1477	1191	894
111101111	1425	1159	819
111111111	1629	1132	924

I've chosen four numbers, each containing nine digits. Across all functions, there is a stark difference between the input with 8 zeros and the inputs with less.

d. I assume the loop will take a very long time. The main difference between

this loop and zeros2 is that while zeros2 only wrote data if the digit was zero, this loop writes data (changes "cnt") on **every** iteration, and my assumption is that this action, when performed many times, is taking a big load on performance.

4. .

a. .

b. .

c. I'd add "0123456789." to the chars variable, and return 37 - len(chars), so that the initial length of charts is 37, and on every alphabet, number or dot the function would act the same way, and return essentially how many different alphabets+numbers+dot there were in the input.