מטלת שיעורי בית 1

Question 1

a (א)

a. (non string input)
   result = control_digit(3)
   function fails at the **"assert isinstance(id_num, str)"** check.
   isinstatance check validity of type.
   since, the type of the input didn't match the required type the script stopped
b. (string with a length less than 8)
   result = control_digit("3")
   function fails at the **"assert len(id_num) === 9"** check.
   since the string isn't 8 character long, the statement returns false and the script is stopped
c. (non digit string input)
   result_1c = control_digit("helo thr")
   function fails at **"val = int(id_num[i])"**
   since id_num[i] is a letter, the conversion to an integer value fails and the script quits.

b (ב)

Removing the definite **"assert"** checks, causes all the problems to occur on **"val = int(id_num[i])"**. Only now it might be more difficult to decipher why.

- For a non-string input, an error will occur at "**val = int(id_num[i])"** because python cannot use the int() function to convert what's already an integer to an integer. And so, the program will quit.
- For an input that's shorter than 8 characters, the loop, which runs for a fixed number of iterations, will get to a point where "**id_num[i]"** points at a non-existing entity. (i is greater than the index of the final character in the string) and so, the program will quit.
- For a string input of non-digit character, the problem again will be the conversion to an integer. Since the character isn't a digit, the function "**int()"** won't know what to do and the program will quit.

(c is on the next page)

Tracking table for input “87654321”

iteration	i	id_num[i]	val	"total
1	0	"8"	8	8
2	1	"7"	7	15
3	2	"6"	6	21
4	3	"5"	5	26
5	4	"4"	4	30
6	5	"3"	3	33
7	6	"2"	2	35
8	7	"1"	1	36

Tracking table for input “32612687”

iteration	i	id_num[i]	val	total
1	0	"3"	3	3
2	1	"2"	2	5
3	2	"6"	6	11
4	3	"3"	1	12
5	4	"2"	2	14
6	5	"6"	6	20
7	6	"8"	8	28
8	7	"7"	7	35

(Question 3 on next page)

Question 3

After running each of the functions for different numbers, here are the results (includes results for the third function for later):

- **2**1400**
    - **Function 1:** 0.00013271000352688134 sec
    - **Function 2:** 0.00004248999999845182 sec
    - **Function 3:** 0.00003638000001160435 sec
- **2**600**
    - **Function 1:** 0.00005672900078934617 sec
    - **Function 2:** 0.00002656000287970528 sec
    - **Function 3:** 0.000013550998119171709 sec
- **2**250**
    - **Function 1:** 0.00004724999962490983 sec
    - **Function 2:** 0.000018619999536895193 sec
    - **Function 3:** 0.000013528999261325225 sec
- **2**100**
    - **Function 1:** 0.0000549500000488478 7 sec
    - **Function 2:** 0.000014829998690402135 sec
    - **Function 3:** 0.000010480001947144046 sec

a. In each example, It's plain to see that the runtime of each of the functions grows progressively longer the larger the input number is. This tracks since all of their's complexity is O(n) meaning it scales linearly with the size of the input

b. The third function, .count(). Indeed has a quicker runtime than both other functions in each case.

c. After comparing the runtime of the functions for the input 2**1400 and 10 ** 421+123456789. Both numbers have 422 digits but 2**1400 have 31 zeros whereas10 ** 421+123456789 has 412 zeros.

Runtime for 2*1400 for all three functions
- **Function 1:** 0.00013271000352688134 sec
- **Function 2:** 0.00004248999999845182 sec
- **Function 3:** 0.00003638000001160435 sec

Runtime for 2*100 for all three functions
- **Function 1:** 0.00013271000352688134 sec
- **Function 2:** 0.00004248999999845182 sec
- **Function 3:** 0.00003638000001160435 sec

From the measurements above, we can conclude that a difference in the number of zeros in the input number doesn't affect the final runtime. This makes sense because counting the occurrences of zero doesn't happens **while** running iterating over the number and the number of iterations is dictated only by the number of digits in the input number.

d. For the loops presented in the question, the runtime multiple orders of magnitude longer. Since instead of running for 422 iterations like in the previous question (once for each of the digits in the number), we're now running over all the whole numbers between 0 and 2*1000 which, needless to say, is much more than 422. I'd assume it could take hours or even days.

Question 4

a. Implemented in the python file
b. Implemented in the python file
c. Implemented in the python file
d. Implemented in the python file
e. Implemented in the python file
f. Implemented in the python file

Question 5

a. Implemented in the python file
b. The offered solution is invalid because it doesn't consider the possibility of a character that appears in st2 and not in st1. For example if **st1=ab** and **st2=abccccccc,** the code will go over each of the characters in Implemented in the python fileand make sure that the same number of that character appears in **st2** so it doesn't consider the c's in **st2** whatsoever.
c. Implemented in the python file

Question 6

a. Implemented in the python file
b. Implemented in the python file