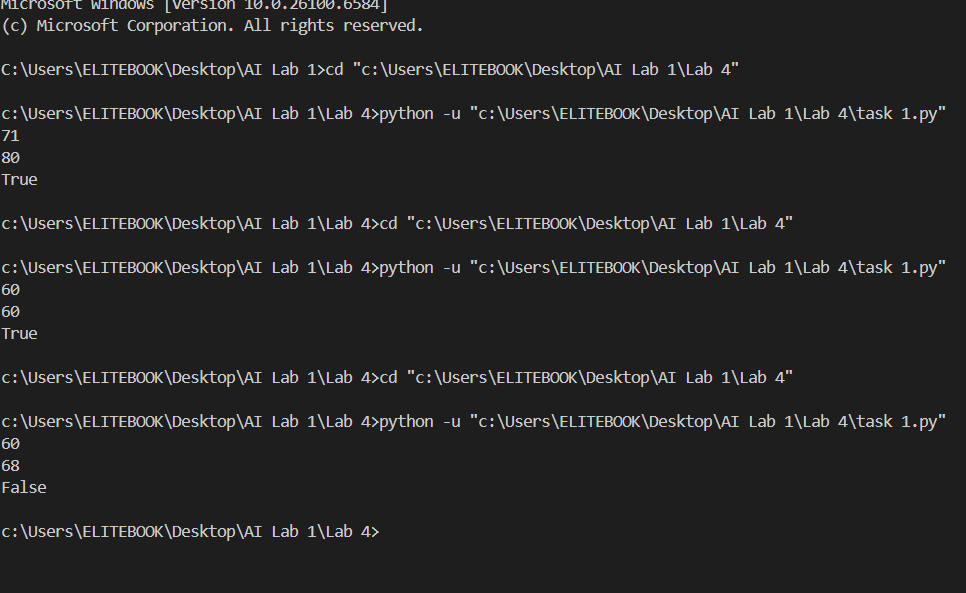
**Task 1**

**LUHN Algorithm:**The luhn\_check function is designed to verify if a 16-digit number (such as a credit or debit card number) is valid according to the **Luhn algorithm**, which is a simple checksum formula used to detect errors in identification numbers.

**How It Works**

1. **Remove spaces**  
   Any spaces in the input number are removed so the number can be processed as a continuous sequence of digits.
2. **Check the length**  
   The function makes sure the number contains exactly 16 digits. If it doesn’t, it is considered invalid and the function returns False.
3. **Separate the check digit**  
   The last digit of the number is saved as the *check digit*. This digit is used at the end of the calculation to confirm whether the number is valid.
4. **Reverse the remaining digits**  
   The first 15 digits (everything except the check digit) are reversed. This is done because the Luhn algorithm processes digits from right to left.
5. **Process every digit according to the Luhn algorithm**
   * Starting from the new leftmost digit (which was originally the second-to-last digit), every second digit is **doubled**.
   * If doubling a digit gives a number greater than 9, **9 is subtracted from it**.
   * All the resulting numbers are added together to get a running total.
6. **Add the check digit**  
   The saved check digit is added to the total from the previous step.
7. **Check if the total is valid**  
   If the final total is divisible by 10, the number is considered **valid** and the function returns True. Otherwise, it returns False.

**Output:**



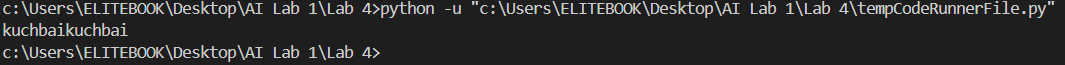
**Task 2:**

**Remove punctuations:** This program goes through each character in the given text and checks if it is **a letter or a number** (an alphanumeric character).

If the character is a letter or a number, it is printed on the screen **without any spaces or punctuation**.

As a result, all **special characters like spaces, exclamation marks, and question marks are removed**, and only the **letters and numbers** are shown together as a continuous string.

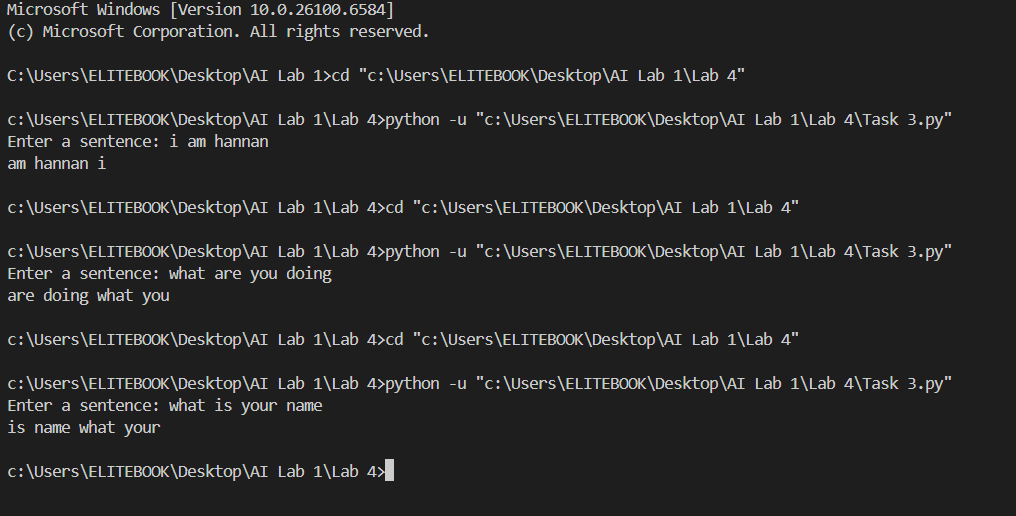
**Output:**

****

**Task 3:**

**Alphabetical order:** This program is designed to take a sentence entered by the user and rearrange all the words in that sentence so they appear in **alphabetical order**.

1. **Taking input from the user**  
   The program begins by asking the user to type a sentence. A sentence can be any collection of words separated by spaces.
2. **Breaking the sentence into words**  
   After receiving the sentence, the program divides it into individual words. This creates a list (or collection) of separate words, making it easier to work with each word one by one.
3. **Counting the number of words**  
   The program then counts how many words are present in the list. This number helps the program know how many times it should loop through the words to sort them properly.
4. **Sorting the words using pairwise comparison**  
   The program uses a sorting technique similar to **bubble sort**:
   * It repeatedly compares each word with the word next to it.
   * If the current word should come **after** the next word alphabetically, the two words are swapped (their positions are exchanged).
   * This process continues many times, gradually moving the larger words (in alphabetical order) towards the end and smaller words towards the beginning.
5. **Repeating the process until all words are sorted**  
   The program keeps looping and comparing words until all the words are arranged in correct alphabetical order from A to Z.
6. **Combining the sorted words into a sentence**  
   Once sorting is complete, the program joins the words back together, adding a space between each word, to form a complete sentence again.
7. **Displaying the result**  
   Finally, it displays the new sentence to the user, where all the words from the original sentence appear in **alphabetical order** regardless of their original position.

**Output:** ****