HyperionDev

The String and Numerical Data Types

Model-answer approach

Visit our website

Auto-graded task 1

The provided solution demonstrates a clear approach to manipulating a string by removing specific characters, converting it to uppercase, and reversing the order of the characters. The task is structured in a step-by-step manner, making the logic easy to follow and implement.

First, the code begins with a string containing exclamation marks between each word. It utilises the `replace()` method to remove all occurrences of exclamation marks, replacing them with spaces. This ensures that the sentence retains its readability and spacing between words.

Next, the program converts the sentence to uppercase using the `upper()` method, which simplifies the process of transforming all lowercase letters to uppercase. This is a simple, effective method for altering the case of a string in Python.

Afterward, the code uses slicing (`[::-1]`) to reverse the sentence. This technique is commonly used for reversing strings and works by stepping through the string backwards. The resulting reversed sentence is printed, completing the transformation.

Overall, this solution employs basic Python string operations, including replacing characters, converting to uppercase, and reversing a string, which are foundational string manipulation skills. The structure is easy to understand, with each step building on the previous one, resulting in a well-formed and functional program. This approach effectively demonstrates the useful string methods and how to apply them in a sequence to achieve the desired output.

Potential improvements could include handling cases where additional punctuation or whitespace could affect the output.

Auto-graded task 2

The solution provided demonstrates a clear approach to string manipulation in Python. It begins by prompting the user to input a sentence, followed by calculating and printing the length of the string using the `len()` function.

Next, it identifies the last character of the string and replaces all occurrences of that character with the '@' symbol, making use of the `replace()` method. The code also reverses the last three characters of the string using slicing and prints them, showcasing Python's ability to manipulate strings concisely.

Finally, a new five-letter word is created by combining the first three and last two characters of the string. This demonstrates a simple but effective way to extract and combine specific parts of a string. Overall, the solution exemplifies practical string manipulation techniques that are straightforward and efficient.

The solution is efficient, but there are a few potential pitfalls to consider. First, the code assumes that the user will enter a sentence with more than three characters; if the input is too short, it may lead to unexpected results or errors when slicing. Additionally, the `replace()` function will replace every occurrence of the last letter, including the last instance itself, which might not always be desired. Lastly, the solution does not account for any trailing whitespace or case sensitivity in string comparisons, which could affect the accuracy of the replacements and manipulations.

Auto-graded task 3

The provided solution takes three user inputs as integers and performs basic arithmetic operations, including summation, subtraction, multiplication, and division. The program begins by prompting the user to enter three numbers, storing each in the variables `num1`, `num2`, and `num3`.

Once the inputs are collected, the code proceeds to perform a series of calculations. It first computes the total sum of the three numbers and stores the result in `total_sum`. Then, it calculates the difference between the first and second numbers, storing the result in `difference`. Following that, the product of the third and first numbers is computed and saved as `product`. Lastly, the program divides the total sum by the third number, storing the result in `division`.

The final step involves printing each of these results—sum, difference, product, and division – using separate `print()` statements, providing clear output for each calculation.

While the program works effectively, the code assumes that the user will input valid integers, so an error could occur if non-integer values are entered, leading to a `ValueError`. Additionally, there is a potential risk of a `ZeroDivisionError` if the user enters 0 as the third number, as division by zero is undefined. Including error handling for these scenarios would make the program more robust and user-friendly.

Challenge 1

The provided solution uses Heron’s formula to calculate the area of a triangle based on the lengths of its three sides. It starts by asking the user to input the lengths of all three sides of the triangle, which are stored in the variables `side1`, `side2`, and `side3`.

After collecting these inputs, the semi-perimeter `s` is calculated by summing the three sides and dividing by 2. Heron’s formula is then applied to compute the area of the triangle, using the square root function from Python's `math` module. Finally, the program outputs the calculated area to the user in a clear and readable format.

This code showcases excellent usage of mathematical functions and formulas, however, the program assumes that the inputs form a valid triangle. If the side lengths do not satisfy the triangle inequality theorem, the square root function could attempt to calculate the square root of a negative number, resulting in an error. Additionally, the program does not handle non-numeric inputs, which could raise exceptions. Adding validation for both the triangle's validity and input data would improve the solution's reliability.

Challenge 2

The provided solution enables users to input their favourite restaurant and favourite number, which are stored in the variables `string_fav` and `int_fav`, respectively. The program starts by prompting the user to enter the name of their favourite restaurant, capturing the input as a string. Subsequently, it asks for the user's favourite number, which is converted to an integer using the `int()` function.

After storing these values, the code prints the user's favourite restaurant and number in a formatted output, ensuring clarity in communication. A potential area of concern is the attempt to cast `string_fav` to an integer. The comment notes that if `string_fav` contains any non-numeric characters, a `ValueError` will be raised, indicating that the conversion cannot take place. This highlights a crucial point regarding type conversion in Python: only strings that represent valid numerical values can be successfully cast to integers.

It should be considered that attempting to convert non-numeric strings to integers will lead to an error, disrupting the flow of the program. Implementing input validation to ensure that users enter appropriate values, thereby preventing runtime exceptions could enhance the overall robustness of the code. Printed output might contain unexpected results if the input is not handled properly.