TASK

Iteration

Model-Answer Approach

Visit our website

Auto-graded task 1

The provided solution effectively calculates the average of numbers entered by the user until the user inputs `-1`. The process begins with initialising variables to store the sum of numbers and the count of valid inputs. These are set to zero at the start.

The code then enters a continuous loop, where it prompts the user to enter a number. If the user inputs -1, the loop breaks, ending the input process. The code checks each input to see if it consists of digits, allowing for negative numbers by stripping a leading `-` sign. This input validation prevents errors, such as "ValueError" or "ZeroDivisionError," that could occur if the input is non-numeric or blank.

Valid inputs are converted to floats and added to the total sum, while the count of valid inputs is incremented. After the loop ends, the code checks if any valid numbers were entered. If there are valid inputs, it calculates the average by dividing the total sum by the count and displays it. If no valid numbers are entered, it informs the user accordingly.

This method ensures robust input handling and accurate calculation of the average, providing a clear and efficient approach to the task.

Auto-graded task 2

This method effectively generates a pattern of asterisks based on the specified number of rows. First, the code validates the input to ensure `rows` is an even, positive integer, preventing execution with invalid input. This is done through a conditional statement that checks if `rows` is both even and positive by confirming that `rows` divided by 2, then multiplied back by 2, equals `rows`, and that `rows` is greater than zero.

Upon successful validation, the code initialises several variables: `count` to track the number of asterisks, `asterisk` as the character to be printed, and half to determine the midpoint. The pattern is generated using a for loop that iterates from 1 to rows - 1. In the first half of the loop, `count` is incremented to increase the number of asterisks. In the second half, `count` is decremented, creating a symmetric pattern.

Each iteration prints the current number of asterisks as determined by `count`. If the input validation fails, an error message is displayed, ensuring users provide the correct input. This concise approach combines input validation with a straightforward pattern generation process, making it easy to follow and efficient for producing the desired output.