HyperionDev

Data Structures – The List

Model Answer Approach

Visit our website

Auto-graded task

This approach effectively demonstrates the use of lists in Python by storing and retrieving data about friends' names and their corresponding ages. Initially, it defines a list called `friends_names`, which contains three strings representing the names of friends. The program then accesses the first and last names using their respective indices – `friends_names[0]` for the first friend and `friends_names[2]` for the last, showcasing how to retrieve elements from a list. Additionally, it calculates and prints the length of the list using the `len()` function, which provides insight into the number of elements present.

The program continues by defining a second list, `friends_ages`, which holds the ages of the friends in the same order as their names. It then prints each friend's name along with their age by matching the indices in both lists. For example, `friends_names[0]` is paired with `friends_ages[0]` to print the age of the first friend.

However, potential pitfalls include assuming that both lists will always have the same length. If one list is modified without a corresponding update to the other, accessing an index that doesn't exist in one of the lists could lead to an `IndexError`. Additionally, if the names or ages are stored or retrieved incorrectly, the program might produce misleading output. To mitigate these risks, it's essential to ensure that any modifications to the lists are carefully managed and that error handling is implemented to catch potential index-related errors. Overall, this program provides a clear example of list manipulation while highlighting the importance of maintaining consistency between related data structures.