LU01.A03 - Funktionaler Bubblesort - Erweiterte Aufgabenstellung

Ablauf

Schritt 1: Verständnis des Bubble-Sort-Algorithmus

Bubble-Sort ist ein einfacher Sortieralgorithmus, der benachbarte Elemente vergleicht und sie vertauscht, wenn sie in der falschen Reihenfolge sind. Der Prozess wird wiederholt, bis keine Vertauschungen mehr erforderlich sind.

Beispiel:

Unsortierte Liste: `[5, 2, 9, 1, 5, 6]`

Nach dem ersten Durchlauf: `[2, 5, 1, 5, 6, 9]` (größte Zahl ist an der letzten Stelle)

Schritt 2: Grundkonzept der Funktionalen Programmierung

In der funktionalen Programmierung sind Daten unveränderlich, und Operationen werden als Funktionen ohne Seiteneffekte dargestellt. Wir müssen den Bubble-Sort-Prozess als Rekursion mit Unveränderlichkeit darstellen.

Schritt 3: Erstellen einer Rekursiven Funktion

Die Herausforderung bei der Implementierung eines rekursiven Bubble-Sort-Algorithmus besteht darin, die typische iterative Struktur, die wir in einem Bubble-Sort sehen, in eine rekursive Struktur umzuwandeln. Das Ziel ist es, einen Durchlauf des Bubble-Sort-Prozesses als rekursive Funktion darzustellen, wobei die Unveränderlichkeit im Sinne der funktionalen Programmierung gewahrt bleibt.

Verständnis des rekursiven Durchlaufs

- **Basisfall:** Wenn die Liste ein Element oder leer ist, wird die Liste selbst zurückgegeben, da keine Sortierung erforderlich ist.
- **Rekursive Fälle:** Die rekursive Funktion wird aufgerufen, indem zwei benachbarte Elemente verglichen und ggf. vertauscht werden. Dann wird der Rest der Liste rekursiv durch die gleiche Funktion bearbeitet.

Beispiel:

Unsortierte Liste: `[5, 2, 9, 1, 5, 6]`

Wenn wir die Funktion mit dieser Liste aufrufen, überprüfen wir das erste und das zweite Element (5 und 2). Da 5 größer als 2 ist, vertauschen wir sie.

Dann rufen wir die Funktion rekursiv auf mit `[5] + restliche_liste` auf, wobei die restliche Liste durch die gleiche Funktion bearbeitet wird.

```
def bubble_pass(lst):
    if len(lst) <= 1:
        return lst
    if lst[0] > lst[1]:
        return [lst[1]] + bubble_pass([lst[0]] + lst[2:])
    return [lst[0]] + bubble_pass(lst[1:])
```

Funktionsweise:

- **Basisfall:** Wenn die Länge der Liste weniger oder gleich 1 ist, wird die Liste zurückgegeben.
- **Vergleichen und Vertauschen:** Wenn das erste Element größer als das zweite ist, werden sie vertauscht. Dann wird die Funktion rekursiv mit dem vertauschten Element und dem Rest der Liste aufgerufen.
- **Vergleichen ohne Vertauschen:** Wenn das erste Element nicht größer als das zweite ist, wird die Funktion rekursiv mit dem gleichen ersten Element und dem Rest der Liste aufgerufen.
- **Rückkehr der sortierten Liste:** Durch das wiederholte rekursive Aufrufen wird die sortierte Liste schrittweise aufgebaut und schließlich zurückgegeben.

Dieser rekursive Ansatz ermöglicht es, einen Durchlauf des Bubble-Sort-Prozesses funktional und unveränderlich darzustellen, wobei jeder rekursive Aufruf einen Teil der Liste sortiert. Der Algorithmus geht weiter, indem er den Prozess auf dem Rest der Liste wiederholt, bis die gesamte Liste sortiert ist.

Schritt 4: Implementierung des Gesamten Bubble-Sort

Nachdem wir im vorherigen Schritt eine Funktion für einen einzelnen Durchlauf von Bubble Sort erstellt haben, müssen wir nun eine weitere rekursive Funktion erstellen, die diesen Durchlauf so oft aufruft, bis die Liste vollständig sortiert ist.

Erstellung einer Rekursiven Funktion für den gesamten Bubble Sort:

- **Basisfall:** Wenn die Größe n der unsortierten Teilliste gleich 1 ist, wird die gesamte Liste zurückgegeben, da sie bereits sortiert ist.
- **Rekursive Fälle:** Wenn die Größe n der unsortierten Teilliste größer als 1 ist, wird die `bubble_pass` Funktion aufgerufen, um einen Durchlauf des Bubble-Sort-Prozesses auf der Liste auszuführen. Anschließend wird die Funktion `bubble_sort` rekursiv aufgerufen, wobei die Größe n um 1 verringert wird.

```
def bubble_pass(lst):
    # Code aus Schritt 3
```

```
def bubble_sort(lst, n=None):
    if n is None:
        n = len(lst)
    if n == 1:
        return lst
    lst = bubble_pass(lst)
    return bubble_sort(lst, n-1)
```

Funktionsweise:

Basisfall: Wenn die unsortierte Größe der Liste 1 ist, ist die Liste sortiert, und wir geben sie zurück. Durchlauf eines Bubble-Sort-Prozesses: Wir rufen die `bubble_pass` Funktion auf, um einen Durchlauf des Bubble-Sort-Prozesses durchzuführen, bei dem die benachbarten Elemente verglichen und vertauscht werden, falls erforderlich. Rekursiver Aufruf: Nachdem ein Durchlauf abgeschlossen ist, rufen wir die `bubble_sort` Funktion rekursiv auf und verringern die Größe `n` um 1, um die Sortierung fortzusetzen. Rückkehr der sortierten Liste: Die rekursiven Aufrufe setzen sich fort, bis die unsortierte Größe der Liste 1 erreicht. Zu diesem Zeitpunkt wird die vollständig sortierte Liste zurückgegeben. Diese rekursive Herangehensweise stellt sicher, dass die Liste schrittweise sortiert wird, wobei jeder rekursive Aufruf der `bubble_sort` Funktion einen weiteren Durchlauf des Sortierprozesses durchführt. Durch die Kombination der beiden rekursiven Funktionen `bubble_pass` und `bubble_sort` wird ein funktionaler und unveränderlicher Ansatz zur Implementierung des Bubble-Sort-Algorithmus erreicht.

Schritt 5: Testen der Implementierung

Abschließend sollten Sie Ihre Implementierung mit verschiedenen Listen testen, um sicherzustellen, dass sie korrekt funktioniert.

Beispiel:

```
if __name__ == '__main__':
    unsorted_list = [5, 2, 9, 1, 5, 6]
    sorted_list = bubble_sort(unsorted_list)
    print(sorted_list)  # Ausgabe: [1, 2, 5, 5, 6, 9]
```

Zusammenfassung:

Diese Aufgabe demonstriert, wie der Bubble-Sort-Algorithmus unter Verwendung der funktionalen Programmierprinzipien von Rekursion und Unveränderlichkeit implementiert werden kann. Es zeigt auch, wie funktionaler Code oft aus kleineren Funktionen zusammengesetzt wird, die jeweils eine spezifische Aufgabe erfüllen.