LU02.L05 - Listen

Schritt 1: Verwenden von List Comprehension oder "map"-Funktion

Wir können diese Aufgabe mit einer Schleife, einer List Comprehension oder der map-Funktion lösen. Hier sind Beispiele für beide Ansätze:

Mit Schleife:

```
def increment_numbers(numbers):
    new_numbers = [] # create an empty list to store the new numbers
    for number in numbers:
        new_numbers.append(number + 1) # append the incremented value to the
new list
    return new_numbers
```

Diese Techniken werden wir im Verlauf des Kurses genauer kennenlernen.

Mit List Comprehension:

```
def increment_numbers(numbers):
    return [number + 1 for number in numbers]
```

Mit map-Funktion:

```
def increment_numbers(numbers):
    return list(map(lambda x: x + 1,
numbers))
```

Schritt 2: Rückgabe der neuen Liste

Die Funktion gibt die neue Liste zurück, wie in den vorherigen Schritten gezeigt.

Schritt 3: Testen der Funktion

```
if __name__ == '__main__':
    new_numbers = increment_numbers(numbers)
    print('Original numbers:', numbers)
    print('Incremented numbers:', new_numbers)
```

Schritt 4: Testen der Funktion

Lassen Sie die Pytests laufen.

Ausgabe

```
Original numbers: [1, 2, 3, 4, 5]
Incremented numbers: [2, 3, 4, 5, 6]
```

Zusammenfassung

Die Musterlösung zeigt zwei verschiedene Möglichkeiten, wie eine Funktion eine neue Liste erstellen kann, in der alle Elemente der Eingabeliste um eins erhöht wurden, ohne die ursprüngliche Liste zu verändern. Dies ist ein wichtiger Aspekt der funktionalen Programmierung und hilft dabei, den Code besser verständlich und vorhersagbar zu machen.

Warum ist die Lösung pure und immutable?

Die Lösung zur Aufgabe achtet auf die Prinzipien einer **pure function** und **immutable values** aus folgenden Gründen:

- **Pure Function:**
  - Die Funktion `increment_numbers` ist eine reine Funktion, da sie keine Seiteneffekte hat. Sie verändert keine globalen Variablen, druckt nichts auf die Konsole aus und interagiert nicht mit externen Systemen.
  - Die Funktion hängt ausschließlich von ihren Eingabewerten ab und liefert immer das gleiche Ergebnis für die gleichen Eingaben. Dies macht sie vorhersehbar und testbar.

- **Immutable Values:**
- Die ursprüngliche Liste, die an die Funktion übergeben wird, bleibt unverändert. Dies stellt sicher, dass die Daten, mit denen die Funktion arbeitet, unveränderlich (immutable) bleiben.
- Die Funktion erstellt eine neue Liste, in der die Werte um +1 erhöht werden, ohne die ursprüngliche Liste zu verändern. Dies ist ein zentraler Aspekt der Unveränderlichkeit und stellt sicher, dass die ursprünglichen Daten intakt bleiben.

- **Verwendung von append und Unveränderlichkeit:**
- Obwohl in der Funktion die Methode `append` verwendet wird, stellt dies kein Problem dar, da `append` auf eine neue, innerhalb der Funktion erstellte Liste angewendet wird.
- Diese neue Liste `new_numbers` ist nicht die ursprüngliche Liste, die als Argument übergeben wurde, sondern eine separate, leere Liste, die in der Funktion erzeugt wird.

- Somit wird die ursprüngliche Liste `numbers` nicht verändert, und das Prinzip der Unveränderlichkeit bleibt gewahrt. Die Verwendung von `append` führt lediglich dazu, dass die neuen Werte zur neuen Liste hinzugefügt werden, ohne die Eingabeliste zu beeinflussen.

Zusammengefasst:

- Die Funktion `increment_numbers` erfüllt die Anforderungen einer pure function, da sie keine Seiteneffekte aufweist und sich ausschließlich auf ihre Eingabewerte stützt.
- Die Werte in der ursprünglichen Liste bleiben unverändert, was den Grundsatz der Unveränderlichkeit unterstützt und eine klare, verständliche und vorhersagbare Funktionsweise ermöglicht.
- Die Verwendung von `append` ist in diesem Kontext unproblematisch, da sie nur auf eine neu erstellte Liste angewendet wird und die ursprüngliche Liste unverändert bleibt.
