LU04l - Weitere Keywords

Python bietet eine Reihe von eingebauten Funktionen, die den Umgang mit Sequenzen und iterierbaren Objekten erleichtern. Diese Funktionen können oft in Kombination mit Generatoren, List Comprehensions und anderen fortgeschrittenen Features verwendet werden.

any

Die Funktion `any` überprüft, ob mindestens ein Element in einem iterierbaren Objekt als wahr bewertet wird.

Syntax

```
any(iterable)
```

Beispiel

```
if any(x > 0 for x in [-1, 0, 5]):
    print("Es gibt eine positive Zahl.")
# Output: Es gibt eine positive Zahl.
```

all

Die Funktion `all` überprüft, ob alle Elemente in einem iterierbaren Objekt als wahr bewertet werden.

Syntax

```
all(iterable)
```

Beispiel

```
if all(x > 0 for x in [1, 5, 8]):
    print("Alle Zahlen sind positiv.")
# Output: Alle Zahlen sind positiv.
```

sum

Die Funktion `sum` berechnet die Summe aller Elemente in einem iterierbaren Objekt.

Syntax

```
sum(iterable, start=0)
```

Beispiel

```
total = sum(x for x in [1, 2, 3])
print(total)
# Output: 6
```

min und max

Die Funktionen `min` und `max` finden das kleinste bzw. größte Element in einem iterierbaren Objekt.

Syntax

```
min(iterable)
max(iterable)
```

Beispiel

```
min_val = min(x for x in [3, 1, 4])
max_val = max(x for x in [3, 1, 4])
print(min_val, max_val)
# Output: 1 4
```

len

Beschreibung

Die Funktion `len` gibt die Länge eines iterierbaren Objekts zurück.

Syntax

```
len(s)
```

Beispiel

```
length = len([1, 2, 3])
```

```
print(length)
# Output: 3
```

enumerate

Die Funktion `enumerate` gibt einen Iterator zurück, der sowohl den Index als auch den Wert jedes Elements in einem iterierbaren Objekt liefert.

Syntax

```
enumerate(iterable, start=0)
```

Beispiel

```
for i, val in enumerate(['a', 'b', 'c']):
    print(i, val)
# Output: 0 a, 1 b, 2 c
```

zip

Die Funktion `zip` nimmt zwei oder mehr iterierbare Objekte und gibt einen Iterator zurück, der die Elemente paarweise zusammenfügt.

Syntax

```
zip(*iterables)
```

Beispiel

```
for a, b in zip([1, 2, 3], ['a', 'b', 'c']):
    print(a, b)
# Output: 1 a, 2 b, 3 c
```

iterator

Ein Iterator ist ein Objekt, das eine Sequenz von Elementen darstellt und die Möglichkeit bietet, durch diese Elemente zu iterieren. In Python wird ein Iterator durch die Implementierung der Methoden `__iter__()` und `__next__()` zu einem Objekt gemacht.

Syntax

```
iter(iterable)
```

Beispiel

```
my_list = [1, 2, 3]
my_iterator = iter(my_list)
print(next(my_iterator))
# Output: 1 (das erste Element der Liste)
```

next

Die Funktion `next` wird verwendet, um das nächste Element aus einem Iterator zu holen. Wenn es keine weiteren Elemente gibt, wird ein `StopIteration`-Fehler ausgelöst.

Syntax

```
next(iterator, default)
```

Hierbei ist `default` der Wert, der zurückgegeben wird, wenn der Iterator erschöpft ist. Dieser Parameter ist optional.

Beispiel

```
my_list = [1, 2, 3]
my_iterator = iter(my_list)
print(next(my_iterator, 'Ende')) # Output: 1
print(next(my_iterator, 'Ende')) # Output: 2
print(next(my_iterator, 'Ende')) # Output: 3
print(next(my_iterator, 'Ende')) # Output: Ende
```

```
# Vorlage
getraenke = [
    {'name': 'Mojito', 'zutaten': ['Rum', 'Minze', 'Zucker', 'Limette',
'Soda'], 'alkohol': True},
    {'name': 'Saft', 'zutaten': ['Orangensaft'], 'alkohol': False},
    {'name': 'Gin Tonic', 'zutaten': ['Gin', 'Tonic', 'Limette'], 'alkohol':
True},
    {'name': 'Kaffee', 'zutaten': ['Kaffee', 'Wasser'], 'alkohol': False}
]

new_list = list(map(lambda x: x['name'], filter(lambda x: x['alkohol'] ==
True and len(x['zutaten']) > 3, getraenke)))
```

M323-LU04