LU04i - Generator Expressions

Generator Expressions bieten eine kompakte Möglichkeit, Generatoren zu erstellen. Sie ähneln List Comprehensions, verwenden jedoch runde Klammern `()` anstelle von eckigen Klammern `[]`. Der Hauptvorteil von Generator Expressions ist, dass sie „lazy“ sind, d.h. sie generieren Werte „on-the-fly“ und halten nicht die gesamte Sequenz im Speicher.

Hauptverwendungszweck von Generator Expressions

Generator Expressions werden in Python vorwiegend zur „lazy“ Transformation von Daten verwendet. Sie sind besonders nützlich, wenn man eine bestehende Liste (oder ein anderes iterierbares Objekt) nehmen und jedes ihrer Elemente auf eine bestimmte Weise transformieren möchte, ohne die gesamte Sequenz im Speicher zu speichern.

Die Syntax

```
gen_expr = (expression for item in iterable if condition == True)
```

Beispiel: "lazy" Verhalten sichtbar machen

Dieses Beispiel zeigt, dass eine Generator Expression Werte erst erzeugt, wenn sie wirklich gebraucht werden (z.B. mit `next()`) und dass ein Generator nach dem Konsumieren nicht automatisch „neu startet“.

```
# Beispiel: Generator Expression ist "lazy" (Werte entstehen erst beim
Iterieren)

numbers = [1, 2, 3, 4, 5]

squares_gen = (n**2 for n in numbers)   # Generator Expression
squares_list = [n**2 for n in numbers]  # List Comprehension (sofort fertig
berechnet)

print(squares_gen)   # <generator object ...>
print(squares_list)  # [1, 4, 9, 16, 25]

# Lazy sichtbar machen: Der Generator liefert Werte erst, wenn man ihn
"anzapft"
print(next(squares_gen))  # 1
print(next(squares_gen))  # 4

# Jetzt sind schon 2 Werte "verbraucht" – übrig bleiben nur noch die
restlichen:
```

```
print(list(squares_gen))  # [9, 16, 25]

# Wichtig: Ein Generator kann danach nicht nochmals von vorne iteriert werden:
print(list(squares_gen))  # []
```

Punkt	Was man sieht	Merksatz
Lazy-Effekt	`print(squares_gen)` zeigt nur ein Generator-Objekt	„Noch nichts berechnet.“
Verbrauch	`next()` holt Werte einzeln	„Werte entstehen beim Bedarf.“
Einmaligkeit	zweites `list(squares_gen)` ist leer	„Generatoren kann man nicht resetten.“

Teile einer Generator Expression

Condition

Ein optionaler Filter, um nur bestimmte Elemente einzuschließen.

```
gen_expr = (x for x in range(10) if x % 2 == 0)
```

Die Bedingung `if x % 2 == 0` liefert `True` für alle geraden Zahlen, so dass der Generator alle geraden Zahlen von 0 bis 9 enthält.

Iterable

Das Iterable kann ein beliebiges iterierbares Objekt sein, z. B. eine Liste, ein Tupel, ein Set usw.

1. **Beispiel mit einer Liste**:

   ```
   squared = (x**2 for x in [1, 2, 3, 4])
   # Ergebnis beim Iterieren: 1, 4, 9, 16
   ```

2. **Beispiel mit einem Tupel**:

   ```
   doubled = (x * 2 for x in (1, 2, 3, 4))
   # Ergebnis beim Iterieren: 2, 4, 6, 8
   ```

3. **Beispiel mit range()**:

   ```
   gen_expr = (x for x in range(10))
   # Ergebnis beim Iterieren: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
   ```

Expression

Der aktuelle Gegenstand der Iteration, aber auch das Ergebnis, das vor seinem Einzug in den neuen Generator manipuliert werden kann.

1. **Beispiel mathematische Operation**:

```
#Expression = x * 2
doubled = (x * 2 for x in range(5))
# Ergebnis beim Iterieren: 0, 2, 4, 6, 8
```

2. **Beispiel mit Funktionsaufruf**:

```
words = ["apple", "banana", "cherry"]
uppercased = (word.upper() for word in words)
```

Item

Das aktuelle Element, das in der Iteration verarbeitet wird. Der Name des Item wird im `for`-Konstrukt definiert: `for item in iterable`

Beispiel

```
words = ['apple', 'banana', 'cherry']
uppercased = (item.upper() for item in words)
```

Beispiele

Filtern und Verwenden von next mit Generator Expressions

Angenommen, Sie haben eine Liste von Dictionarys, die verschiedene Benutzer darstellen, und jeder Benutzer hat eine eindeutige ID. Sie möchten einen bestimmten Benutzer anhand seiner ID finden.

Beispiel-Daten:

```
users = [
    {'id': 1, 'name': 'Alice'},
    {'id': 2, 'name': 'Bob'},
    {'id': 3, 'name': 'Charlie'},
    {'id': 4, 'name': 'David'}
]
```

Um den Benutzer mit der `ID 3` zu finden, können Sie eine Generator Expression verwenden und `next()` aufrufen, um das erste (und in diesem Fall einzige) Ergebnis zu erhalten:

```
user_id_to_find = 3
```

```python
user = next((user for user in users if user['id'] == user_id_to_find), None)
print(user)  # Output: {'id': 3, 'name': 'Charlie'}
```

In diesem Beispiel wird die Generator Expression `(user for user in users if user['id'] == user_id_to_find)` verwendet, um durch die Liste der Benutzer zu iterieren und nur diejenigen zurückzugeben, deren ID der gesuchten ID entspricht. Die Funktion `next()` gibt dann den ersten Benutzer zurück, der die Bedingung erfüllt. Wenn kein Benutzer gefunden wird, gibt `next()` den Wert None zurück, da dies als Standardwert angegeben ist.

Dieser Ansatz ist besonders effizient, da er nicht die gesamte Liste durchlaufen muss, sondern bei der ersten Übereinstimmung stoppt.

Generator Expressions sind ein mächtiges Werkzeug in Python, das den Code sauberer und verständlicher machen kann. Sie ermöglichen die schnelle Transformation von Daten und unterstützen dabei, den Code näher an die menschliche Logik und mathematische Notation zu bringen, ohne den Speicher zu überlasten.

M323-LU04