LU04e - Die filter-Funktion in Python

Die `filter`-Funktion ist eine eingebaute Funktion in Python, die es ermöglicht, Elemente einer Liste oder eines anderen Iterables zu filtern. Diese Funktion nimmt eine Funktion und ein Iterable als Argumente und gibt ein neues Iterable zurück, das nur die Elemente enthält, für die die Funktion `True` zurückgibt.

Syntax

Die Syntax der `filter`-Funktion ist wie folgt:

```
filtered_result = filter(function, iterable)
```

- **function**: Die Funktion, die auf jedes Element des Iterables angewendet wird.
- **iterable**: Das Iterable, das gefiltert werden soll.

Beispiele

Liste mit Zahlen

Ein einfaches Beispiel für die Verwendung der `filter`-Funktion ist das Filtern aller geraden Zahlen aus einer Liste:

```
numbers = [1, 2, 3, 4, 5]
even_numbers = filter(lambda x: x % 2 == 0, numbers)
print(list(even_numbers))  # Output: [2, 4]
```

Während viele Beispiele die Verwendung von Lambda-Funktionen mit `filter` zeigen, ist es auch möglich, reguläre Funktionen zu verwenden. Die einzige Anforderung ist, dass die Funktion ein Argument akzeptiert und entweder `True` oder `False` zurückgibt.

```
def is_even(number):
    return number % 2 == 0

even_numbers = filter(is_even, [1, 2, 3, 4, 5])
print(list(even_numbers))  # Output: [2, 4]
```

Filtern von Dictionarys in einer Liste

Ein weiteres häufiges Szenario ist das Filtern von Dictionarys in einer Liste. Zum Beispiel könnten Sie alle Dictionarys filtern wollen, die einen bestimmten Schlüssel-Wert haben:

```
persons = [
```

```
    {'name': 'Anna', 'age': 25},
    {'name': 'Bob', 'age': 30},
    {'name': 'Charlie', 'age': 28}
]
adults = filter(lambda x: x['age'] >= 30, persons)
print(list(adults)) # Output: [{'name': 'Bob', 'age': 30}]
```

Filtern von Objekten in einer Liste

Das Filtern von Objekten funktioniert ähnlich wie das Filtern von Dictionarys. Hier ist ein Beispiel, bei dem eine Liste von Objekten gefiltert wird, um nur diejenigen zu erhalten, die eine bestimmte Methode erfüllen:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def is_adult(self):
        return self.age >= 30

persons = [Person('Anna', 25), Person('Bob', 30), Person('Charlie', 28)]
adults = filter(lambda x: x.is_adult(), persons)
print([person.name for person in adults]) # Output: ['Bob']
```

Vergleich mit List Comprehensions

Die `filter`-Funktion hat viele Gemeinsamkeiten mit List Comprehensions:

- **Gemeinsamkeiten**:
  1. Beide können verwendet werden, um Elemente eines Iterables zu filtern.
  2. Beide erzeugen ein neues Iterable mit den gefilterten Werten.
- **Unterschiede**:
  1. Die `filter`-Funktion gibt ein Filter-Objekt zurück, das in eine Liste konvertiert werden muss, während List Comprehensions direkt eine Liste zurückgeben.
  2. List Comprehensions können auch verwendet werden, um die Elemente gleichzeitig zu transformieren, während `filter` nur zum Filtern verwendet wird.

Beispiel mit List Comprehension

Das obige Beispiel mit der `filter`-Funktion könnte auch mit einer List Comprehension geschrieben werden:

```
numbers = [1, 2, 3, 4, 5]
even_numbers = [x for x in numbers if x % 2 == 0]
```

```
print(even_numbers)  # Output: [2, 4]
```

List Comprehensions bieten eine prägnantere und oft pythonischere Lösung zum Filtern und Transformieren von Listen.

M323-LU04, M323-CG4, M323-CF4, M323-CE4