Python Tutorial: Itertools

Das Modul `itertools` in Python bietet eine Sammlung von Werkzeugen für effiziente Schleifen und Iterationen. Es ermöglicht die Erstellung von komplexen Iteratoren auf eine einfache und speichereffiziente Weise. `itertools` enthält Funktionen, die häufig in Datenverarbeitungs-Pipelines und in Situationen verwendet werden, in denen Sie mit großen Datensätzen oder komplexen Iterationsmustern arbeiten.

Einführung in Itertools

Das `itertools`-Modul ist Teil der Python-Standardbibliothek und bietet eine Vielzahl von Funktionen, die sich mit iterierbaren Datenstrukturen (wie Listen oder Generatoren) arbeiten lassen. Die Funktionen des Moduls können in drei Hauptkategorien unterteilt werden: unendliche Iteratoren, Iteratoren, die auf iterierbaren Eingaben arbeiten, und Kombinatorische Iteratoren.

Wichtige Funktionen von Itertools

Unendliche Iteratoren

- `count(start=0, step=1)`: Gibt sukzessive Werte mit dem angegebenen Startwert und Schritt zurück.
- `cycle(iterable)`: Wiederholt unendlich oft die Elemente des übergebenen iterierbaren Objekts.
- `repeat(object, [times])`: Wiederholt ein Objekt unendlich oft oder eine bestimmte Anzahl von Malen.

Iteratoren, die auf iterierbaren Eingaben arbeiten

- `chain(*iterables)`: Verkettet mehrere iterierbare Objekte zu einem einzigen Iterator.
- `compress(data, selectors)`: Filtert Elemente, indem nur diejenigen zurückgegeben werden, deren korrespondierender Selektor `True` ist.
- `dropwhile(predicate, iterable)`: Überspringt Elemente, solange das Prädikat `True` ist, und gibt dann den Rest zurück.
- `takewhile(predicate, iterable)`: Gibt Elemente aus dem iterierbaren Objekt zurück, solange das Prädikat `True` ist.

Kombinatorische Iteratoren

- `product(*iterables, repeat=1)`: Kartesisches Produkt der Eingabeiterablen.
- `permutations(iterable, r=None)`: Gibt sukzessive r-Längen-Permutationen aller Elemente des iterierbaren Objekts zurück.
- `combinations(iterable, r)`: Gibt die r-Längen-Untermengen der Elemente des iterierbaren Objekts zurück.
- `combinations_with_replacement(iterable, r)`: Wie `combinations`, erlaubt jedoch wiederholte Elemente.

Beispiele

Verwendung von `chain`

```
from itertools import chain

for i in chain([1, 2, 3], ['a', 'b', 'c']):
    print(i)
# Ausgabe: 1 2 3 a b c
```

Verwendung von `cycle`

```
from itertools import cycle, islice

# Begrenzen der Ausgabe mit islice, da cycle unendlich ist
for i in islice(cycle(['a', 'b', 'c']), 6):
    print(i)
# Ausgabe: a b c a b c
```

Verwendung von `combinations`

```
from itertools import combinations

for combo in combinations([1, 2, 3, 4], 2):
    print(combo)
# Ausgabe: (1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)
```