Die `zip()`-Funktion von Python — Syntax und Semantik

Syntax

`zip(*iterables, strict=False)`

Die Funktion `zip()` iteriert über mehrere iterierbare Objekte und produziert Tupel (unveränderliche Listen) die von jedem Objekt jeweils ein Element enthalten.

`zip()` ist „faul“

`zip` ist *lazy* in dem Sinne, dass ein `zip`-Objekt erst dann ausgewertet wird, wenn z. B. mit einer Schleife darüber iteriert wird oder mit `list()` eine explizite Umwandlung in eine Liste erfolgt.

Beispiel 1

```
A = range(0, 3) # erzeugt das iterierbare Objekt (0, 1, 2)
B = ['Hi', 'Hej', 'Hola']
print(zip(A, B))

# Ausgabe:
# <zip object at 0x7f96...>
```

Beispiel 2

```
A = range(0, 3)
B = ['Hi', 'Hej', 'Hola']
for item in zip(A, B):
    print(item)

# Ausgabe:
# (0, 'Hi')
# (1, 'Hej')
# (2, 'Hola')
```

Beispiel 3

```
A = range(0, 3)
B = ['Hi', 'Hej', 'Hola']
print(list(zip(A, B)))

# Ausgabe:
# [(0, 'Hi'), (1, 'Hej'), (2, 'Hola')]
```

Iterierbare Objekte unterschiedlicher Länge

Haben die iterierbaren Objekte unterschiedliche Längen, so stoppt `zip()`, wenn das erste dieser Objekte keine Elemente mehr liefern kann. Mit dem Parameter `strict=True` [seit Version 3.10] wird eine Fehlermeldung (`ValueError`) ausgegeben, wenn nicht alle Objekte gleich viele Elemente haben.

Beispiel 4

```
A = [4, 7, 9, 8, 3]
B = ['a', 'x']
C = [True, False, True]
print(list(zip(A, B, C)))

# Ausgabe:
# [(4, 'a', True), (7, 'x', False)]
```

Beispiel 5

```
A = [4, 7, 9, 8, 3]
B = ['a', 'x']
C = [True, False, True]
print(list(zip(A, B, C, strict=True)))

# Ausgabe:
# ...
# ValueError: zip() argument 2 is shorter than argument 1
```

Der *-Operator

Eine weitere Fähigkeit des *-Operators besteht darin, die Elemente eines iterierbaren Objekts der Reihe nach „auszupacken“.

Beispiel 6

```
def add(a, b, c):
    return a + b + c

L = [3, 5, 2]

print(add(*L)) # '*' übergibt 'L' elementweise an 'add()'
# Ausgabe: 10
```

Beispiel 7

Ein nützliches Beispiel mit einer Prise „Magie“ in zwei Varianten:

```
L = ['a', 'b', 'c', 'd']
print(*L)
# Ausgabe: 'a b c d'
```

```
L = ['a', 'b', 'c', 'd']
print(*L, sep='+')
# Ausgabe: 'a+b+c+d'
```

Beispiel 8

Mit dem *-Operator können mehrere iterierbare Objekte, die z. B. in einer Liste zusammengefasst sind, vor der Verarbeitung mit `zip()` „entpackt“ werden, so dass es als Umkehrfunktion von `zip()` verwendet werden kann.

```
A = [1, 2, 3, 4]
B = [5, 6, 7, 8]
print(list(zip(*zip(A, B))))

# Ausgabe:
#[(1, 2, 3, 4), (5, 6, 7, 8)]
```

Beispiel 8

Mit der `zip()`-Funktion können Dictionaries mit wenig Code aus den Listen der Schlüssel und der Werte erzeugt werden.

```
keys = ['red', 'green', 'blue']
values = ['rot', 'grün', 'blau']

# herkömmliche Methode
D = dict()
for i in range(0, len(keys)):
    D[keys[i]] = values[i]

# mit zip():
D = dict(zip(keys, values))

print(D)
# Ausgabe: {'red': 'rot', 'green': 'grün', 'blue': 'blau'}
```

Übungen

Notiere die Ausgabe(n) der folgenden Codefragmente.

Aufgabe 1

```
A = [1, 2]
B = [5, 7]
C = [3, 8]
print(list(zip(A, B, C)))
```

Aufgabe 2

```
A = [4, 2]
B = [9, 7, 6, 1]
print(list(zip(A, B)))
```

Aufgabe 3

```
a = 'dig'
b = 'asu'
c = 'stt'
print(list(zip(a, b, c)))
```

Aufgabe 4

```
M = [[1,2], [3,4], [5,6]]
print(list(zip(*M)))
```

Aufgabe 5

```
A, B = zip(*zip((3,4,5), (2,1,7)))
print(A)
print(B)
```

Aufgabe 6

```
K = ['a', 'b', 'c']
V = [4, 7, 3]
D = dict(zip(K, V))
for key, value in D.items():
    print(key, value)
```