Python zip

Summary: in this tutorial, you'll learn how to use the Python `zip()` function to perform parallel iterations on multiple iterables.

Introduction to the Python zip() function

Suppose you have two tuples: `names` and `ages` .

- The `names` tuple stores a list of names.
- The `age` s tuple stores a list of ages.

To map names and ages from these tuples one-by-one in sequence, you may use the `enumerate()` function. For example:

```
names = ('John', 'Jane', 'Alice')
ages = (20, 22, 25)

for index, name in enumerate(names):
    print((name, ages[index]))
```

Output:

```
('John', 20)
('Jane', 22)
('Alice', 25)
```

So John is 20, Jane is 22, and Alice is 25

However, It's getting more complicated if the sizes of the `names` and `ages` tuples are different. That's why the `zip()` function comes to play.

The following shows the syntax of the `zip()` function:

```
zip(*iterables, strict=False)
```

The `zip()` function iterates multiple iterables *in parallel* and returns the *tuples* that contain elements from each iterable.

In other words, the `zip()` function returns an iterator of tuples where i-th tuple contains the i-th element from each input iterable.

The following example shows how to use the `zip()` function to iterate over the `names` and `ages` tuples:

```
names = ('John', 'Jane', 'Alice')
ages = (20, 22, 25)

for employee in zip(names, ages):
    print(employee)
```

Output:

```
('John', 20)
('Jane', 22)
('Alice', 25)
```

In this example, the `zip()` returns a tuple in each iteration and assigns it to the `employee` variable. The tuple contains the i-th elements the `names` and `ages` tuples.

The `zip()` function returns a zip object which is an iterator:

```
names = ('John', 'Jane', 'Alice')
ages = (20, 22, 25)

employees = zip(names, ages)

print(type(employees))  # 👉 <class 'zip'>
```

The `zip()` is lazy. It means that Python won't process the elements until you iterate the iterable. To iterate the iterable, you can use:

- Using a `for` loop
- Calling the `next()` function
- Wrapping in a `list()`

For example:

```
names = ('John', 'Jane', 'Alice')
ages = (20, 22, 25)

employees = zip(names, ages)
employee_list = list(employees)

print(employee_list)  # 👉 [('John', 20), ('Jane', 22), ('Alice', 25)]
```

Iterables with different sizes

The iterables passed to the `zip()` function may have different sizes. The `zip()` has three different strategies to deal with this issue.

By default, the `zip()` will stop when it completes iterating the shortest iterable. It'll ignore the remaining items in the longer iterables. For example:

```
names = ('John', 'Jane', 'Alice', 'Peter')
ages = (20, 22, 25)

for name, age in zip(names, ages):
    print(name, age)
```

Output:

```
John 20
Jane 22
Alice 25
```

In this example, the `zip()` function performs three iterations based on the shortest size of the names and ages.

If you want to ensure that the iterables must have the same sizes, you can use the `strict=True` option. In this case, if the sizes of the iterables are different, the `zip()` will raise a `ValueError` .

> Note that the `strict` argument has been available since Python 3.10

For example, the following will raise a `ValueError` exception because the sizes of the iterables are different:

```
names = ('John', 'Jane', 'Alice', 'Peter')
ages = (20, 22, 25)

for name, age in zip(names, ages, strict=True):  # ValueError 🛑
    print(name, age)
```

Output:

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

If the iterables are of uneven size, and you want to fill missing values with a `fillvalue`, you can use the `zip_longest()` function from the `itertools` module:

```
itertools.zip_longest(*iterables, fillvalue=None)
```

By default, the `fillvalue` is None. For example:

```
from itertools import zip_longest

names = ('John', 'Jane', 'Alice', 'Peter')
ages = (20, 22, 25)

for name, age in zip_longest(names, ages):
    print(name, age)
```

Output:

```
John 20
Jane 22
Alice 25
Peter None
```

Unzip an iterable using the Python zip() function

Unzipping reverses the zipping by converting the zipped values back to individual values. To unzip, you use the zip() function with the unpacking operator (`*`). For example:

```
employees = (('John', 20), ('Jane', 22), ('Alice', 25))
names, ages = zip(*employees)

print(names)
print(ages)
```

Output:

```
('John', 'Jane', 'Alice')
(20, 22, 25)
```

In this example, we have a tuple where each element contains a name and age. The `zip()` function unpacks the tuple to create two different tuples (names and ages).

Summary

- Use the `zip()` function to iterate iterables in parallel.
- Use the `zip()` function with the unpacking operator (`*`) to unzip values.
- Use `itertool.zip_longest()` function to zip iterables of different sizes.