Python → Iterators and generators → Itertools module

Theory: Itertools module

29 minutes 6 / 7 problems solved

Start practicing

1312 users solved this topic. Latest completion was about 4 hours ago.

You are already familiar with iterators and know how to create an iterator from a list or other iterable objects. In this topic, you will learn how to create iterators from multiple collections (e.g., two lists) with the help of the methods implemented in the `itertools` module.

The `itertools` module contains some useful iterator building blocks. To use its functionality, you will need to import the module first:

```
import itertools
```

Current topic:

✓ Itertools module

Topic depends on:

✓ Load module Stage 1 5

✓ Iterators

Table of contents:

§1. itertools.chain()

`itertools.chain(iterable1, iterable2, ...)` is handy when you need to treat a number of consecutive sequences as a single sequence. The code below prints out the names of all the students taking different subjects:

```
students_maths = ['Ann', 'Kate', 'Tom']
students_english = ['Tim', 'Carl', 'Dean']
students_history = ['Jane', 'Mike']


for student in itertools.chain(students_maths, students_english, students_history):
    print(student)

# Ann
# Kate
# Tom
# Tim
# Carl
# Dean
# Jane
# Mike
```

So, the `itertools.chain` takes a number of lists (or any other iterables) as input and returns an iterator that returns the elements from the first list one by one until the list is exhausted, and then proceeds to the second one and so on until all the lists are exhausted.

Note that this approach is different from concatenating all the lists first and then looping over the resulting list because `itertools.chain` doesn't actually create this intermediate concatenated list and therefore saves up memory.

The `itertools` module implements other useful combinatorial functions, such as `product()` and `combinations()`.

§2. itertools.product()

Another useful tool is the `itertools.product(iterable1, iterable2, ...)`, which takes several iterables and returns the elements of their Cartesian product one by one. Cartesian product of several iterables is an iterator of all possible tuples such that the first element is coming from the first argument, the second element is coming from the second argument, and so on. Here is an example:

```
first_list = ['Hi', 'Bye', 'How are you']
second_list = ['Jane', 'Anton']

for first, second in itertools.product(first_list, second_list):
    print(first, second)

# Hi Jane
# Hi Anton
# Bye Jane
# Bye Anton
# How are you Jane
# How are you Anton
```

Again, note that these combinations are not stored in memory but produced on-the-fly, only when the `for` loop asks for a new one. This is especially important when you work with a lot of data. Compare:

```
# Trying to create a list containing 10^12 elements will result in a memory error:
too_long_list = list(itertools.product(range(1000000), range(1000000)))

# However, works with iterators:
my_iterator = itertools.product(range(1000000), range(1000000))
for i in range(5):
    print(next(my_iterator))

# (0, 0)
# (0, 1)
# (0, 2)
# (0, 3)
# (0, 4)
```

§3. itertools.combinations()

Imagine that you need to obtain all possible combinations of `r` items from an iterable containing `n` elements.

For example, let's consider all possible combinations of any two numbers between 1 and 1000000. There are so many of them it's practically impossible to fit in memory. How to deal with this problem? Use iterators!

`itertools.combinations(iterable, r)` does exactly what we want. Take a look at the example:

```
my_iter = itertools.combinations(range(1, 1000000), 2)

for i in range(5):
    print(next(my_iter))

# (1, 2)
# (1, 3)
# (1, 4)
# (1, 5)
# (1, 6)
```

§4. Conclusions

- The `itertools` module implements useful iterators.
- Iterators don't work as finite sets but rather generate elements one-by-one.
- Using an iterator helps to save memory.

Report a typo

117 users liked this theory. 4 didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing

Comments (1) Hints (0) Useful links (6) Show discussion