Theory: Iterators

16 minutes 5 / 9 problems solved

Start practicing

1462 users solved this topic. Latest completion was about 5 hours ago.

§1. Iterables and iterators

In Python, we call any object we can loop over an **iterable**. Very common examples of iterable objects are lists, strings and dictionaries.

Iterables in Python implement the `__iter__()` method that returns an **iterator**, an object that traverses an iterable and returns its elements one by one. Iterators represent a stream of data. They implement the `__next__()` method, which returns the items of an iterable one by one.

You can create an iterator passing an iterable to the built-in `iter()` function.

```
# This is a list...
my_list = [1, 2, 3]

# ... and this is how we create an iterator from it
my_iterator = iter(my_list)
print(my_iterator)

# <list_iterator object at 0x000001F06D792B70>
```

Each time you want to get the actual values, you need to pass iterator to the `next()` function:

```
print(next(my_iterator))
# 1

print(next(my_iterator))
# 2

print(next(my_iterator))
# 3


print(next(my_iterator))

# StopIteration exception
```

Note that when we call `next()` for the fourth time, we get a `StopIteration` exception. It's because of our list contains just three elements, and iterator can only pass them once.

But do you always have to call `next()` manually? Not if you create and use iterators in `for` loop statements using the following syntax:

```
for item in iterable:
    ...
```

Python `for` loop will automatically create an iterator from a given iterable and get its elements one by one with the help of the `next` method until the iterable is exhausted. Thus, to print out the elements of `my_list` defined above, we can simply write the following:

```
for item in my_list:
    print(item)

# 1
# 2
# 3
```

§2. zip()

Now you know how to create an iterator from a single iterable. What if you need to look over the elements of not one but multiple lists at the same time? Well, then the built-in `zip()` comes in very handy.

Current topic:

Iterators

Topic depends on:

For loop 12

Topic is required for:

Itertools module

Sorting a list

Table of contents:

↑ Iterators

§1. Iterables and iterators

§2. zip()

§3. enumerate()

§4. Conclusions

Feedback & Comments

Suppose, for example, that you have two lists with the first and last names of the employees, and you need to print out the full names. With `zip()`, this can be easily done as follows:

```
first_names = ['John', 'Anna','Tom']
last_names = ['Smith', 'Williams', 'Davis']

for name, last_name in zip(first_names, last_names):
    print(name, last_name)

# John Smith
# Anna Williams
# Tom Davis
```

`zip()` takes several iterables and returns an iterator of tuples, where each tuple contains one element from each of the given iterables. Note that if `zip()` gets iterables of different lengths, iteration will stop as soon as the shortest iterable is exhausted:

```
short_list = [1, 2, 3]
long_list = [10, 20, 30, 40]

for a, b in zip(short_list, long_list):
    print(a, b)

# 1 10
# 2 20
# 3 30
```

§3. enumerate()

Another very useful tool is the built-in `enumerate()` function, which takes an iterable and returns its elements one by one along with their indexes. For instance, the code below prints out the names of the months (stored in a list) along with their numbers:

```
months_list = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
               'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

for n, month in enumerate(months_list):
    print(n + 1, month)

# 1 Jan
# 2 Feb
# 3 Mar

# 4 Apr

# 5 May

# 6 Jun

# etc.
```

Note that by default the counter starts at 0, but you can actually explicitly specify any starting point:

```
for n, month in enumerate(months_list, start=1):
    print(n, month)

# 1 Jan
# 2 Feb
# 3 Mar
# etc.
```

§4. Conclusions

- In Python, iterator objects traverse an iterable, e.g., a list.
- There are several built-in functions to create iterators, for example, `zip()` and `enumerate()`.
- `zip()` performs parallel iteration over several iterables.

- `enumerate()` returns elements of an iterable along with their indexes one by one.

Report a typo

141 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Comments (3) Hints (1) Useful links (0) Show discussion