Python Generator Expressions

Summary: in this tutorial, you'll learn about the Python generator expression to create a generator object.

Introduction to generator expressions

A generator expression is an expression that returns a generator object.

Basically, a generator function is a function that contains a `yield` statement and returns a generator object.

For example, the following defines a generator function:

```
def squares(length):
    for n in range(length):
        yield n ** 2
```

The `squares` generator function returns a generator object that produces square numbers of integers from `0` to `length - 1` .

Because a generator object is an iterator, you can use a for loop to iterate over its elements:

```
for square in squares(5):
    print(square)
```

Output:

```
0
1
4
9
16
```

A generator expression provides you with a more simple way to return a generator object.

The following example defines a generator expression that returns square numbers of integers from 0 to 4:

```
squares = (n** 2 for n in range(5))
```

Since the `squares` is a generator object, you can iterate over its elements like this:

```
for square in squares:
    print(square)
```

As you can see, instead of using a function to define a generator function, you can use a generator expression.

A generator expression is like a list comprehension in terms of syntax. For example, a generator expression also supports complex syntaxes including:

- if statements
- Multiple nested loops
- Nested comprehensions

However, a generator expression uses the parentheses `()` instead of square brackets `[]` .

Generator expressions vs list comprehensions

The following shows how to use the list comprehension to generate square numbers from 0 to 4:

```
square_list = [n** 2 for n in range(5)]
```

And this defines a square number generator:

```
square_generator = (n** 2 for n in range(5))
```

1) Syntax

In terms of syntax, a generator expression uses parentheses `()` while a list comprehension uses the square brackets `[]` .

2) Memory utilization

A list comprehension returns a list while a generator expression returns a generator object.

It means that a list comprehension returns a complete list of elements upfront. However, a generator expression returns a list of elements, one at a time, based on request.

A list comprehension is eager while a generator expression is lazy.

In other words, a list comprehension creates all elements right away and loads all of them into the memory.

Conversely, a generator expression creates a single element based on request. It loads only one single element to the memory.

3) Iterable vs iterator

A list comprehension returns an iterable. It means that you can iterate over the result of a list comprehension again and again.

However, a generator expression returns an iterator, specifically a lazy iterator. It becomes exhausting when you complete iterating over it.

Summary

- Use a Python generator expression to return a generator.