Python Iterators

Summary: in this tutorial, you'll learn about Python iterator and how to define a custom iterator using the iterator protocol.

What is a Python iterator

An iterator is an object that implements:

- `__iter__` method that returns the object itself.
- `__next__` method that returns the next item. If all the items have been returned, the method raises a `StopIteration` exception.

Note that these two methods are also known as the **iterator protocol**.

Python allows you to use iterators in `for` (https://www.pythontutorial.net/python-basics/python-for-loop-list/) loops, comprehensions, and other built-in functions including `map`, `filter`, `reduce`, and `zip`.

Python iterator example

The following example defines `Square` iterator class that returns the square numbers. Note that a square number is a product of an integer with itself.

```
class Square:
    def __init__(self, length):
        self.length = length
        self.current = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.current >= self.length:
            raise StopIteration

        self.current += 1
        return self.current ** 2
```

How it works.

First, initialize the `length` and `current` attributes in the `__init__` method.

The `length` attribute specifies the number of square numbers that the class should return. And the `current` attribute keeps track of the current integer..

Second, implement the `__iter__` method that returns the `self` object.

Third, implement the `__next__` method that returns the next square number. If the number of square numbers have been returned is greater than the length, the `__next__` method raises the `StopIteration` exception.

Using the iterator object

The following shows how to use the `Square` iterator in a `for` loop:

```
square = Square(5)
```

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

How it works:

- First, create a new instance of the `Square` class.
- Then, use the `for` loop to iterate over items of the square iterator.

Once you iterate over all the items, the iterator is exhausted. It means you need to create a new iterator to iterate over its items again.

If you attempt to use the iterator that is already exhausted, you'll get the StopIteration exception. For example:

```
next(square)
```

Error:

```
StopIteration
```

Also, an iterator cannot be restarted because it only has the `__next__` method that returns the next item from a collection.

Summary

- An iterator is an object that implements `__iter__` and `__next__` methods.
- An iterator cannot be reusable once all items have been returned.