[Bài đọc] Iterator

Giới thiệu

Iterator là một đối tượng được sử dụng để lặp qua các đối tượng chứa nhiều giá trị như **list, tuple, dict và set**

Nội dung bài đọc sẽ có các phần chính sau:

- Iterator và Iterable
- Phương thức Iterator

Iterable và Iterator

Iterable được hiểu đơn giản là một đối tượng bất kì trong Python, có phương thức __iter__ trả về một iterator, hoặc một phương thức xác định __getitem__ có thể nhận các phần tử có vị trí bắt đầu từ 0. Vì vậy iterable là một object mà bạn có thể lấy ra một iterator.

Itearable bao gồm **list, tuple, dict và set**, cho phép truy cập qua các phần tử bằng chỉ mục hay còn gọi là index.

Còn **Iterator** là đối tượng với một công việc duy nhất là trả về phần tử tiếp theo trong các đối tượng iterable có phương thức __next__

Iterable cho phép nó được lặp lại trong vòng lặp for, và có độ dài vô hạn

Ví dụ:

```
mytuple = ("apple", "banana", "cherry")
for x in mytuple:
    print(x)
```

Kết quả:

```
apple
banana
cherry
```

Khi chúng ta đã biết iterable và iterator là gì và cách sử dụng chúng, thì chúng ta có thể xác định một hàm lặp qua một đối tượng iterable mà không cần sử dụng vòng lặp for

- Tạo một danh sách có thể lặp qua
- Dùng phương thức **iter** để lấy ra iterator

- Vòng lặp sẽ bắt đầu khi ta dùng phương thức **next**

- Vòng lặp sẽ lặp qua tất cả các iterator, và sẽ dừng lại khi không còn phần tử nào để lặp qua, sẽ trả về **StopIteration**

Ví dụ:

```
my_list = [4, 7, 0, 3]
my_iter = iter(my_list)
print(my_iter.__next__())
print(my_iter.__next__())
print(my_iter.__next__())
print(my_iter.__next__())
next(my_iter)
```

Kết quả:

```
4
7
0
3
---------------------------------------------------------------------------
-
StopIteration                             Traceback (most recent call
last)
<ipython-input-3-42971c095224> in <module>
      5 print(my_iter.__next__())
      6 print(my_iter.__next__())
----> 7 next(my_iter)

StopIteration:
```

Các phương thức của đối tượng Iterator

Trong Python, Iterator là một đối tượng thực hiện giao thức vòng lặp, sử dụng các phương thức **__iter__()** và **__next__()**

Trong đó

- Phương thức **__iter__()** dùng để khởi tạo , và luôn phải trả về chính đối tượng của iterator
- Phương thức **__next__()** dùng để thực hiện thao tác lặp lại, và phải trả về phần tử kế tiếp trong danh sách

Lưu ý: Đối tượng Iterator

- Có thể được truyền cho phương thức next() để trả về phần tử tiếp theo trong nhóm hoặc ngoại lệ StopIteration nếu không còn phần tử nào
- Trả về chính nó khi được truyền cho phương thức iter()

Ví dụ: Tạo 1 vòng lặp trả về các số, bắt đầu bằng 1 và mỗi chuỗi sẽ tăng thêm 1

```
class MyNumbers:
    def __iter__(self):
        self.a = 1
        return self

    def __next__(self):
        x = self.a
        self.a += 1
        return x

myclass = MyNumbers()
myiter = iter(myclass)

print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

Kết quả:

```
1
2
3
4
```

StopIteration

Để ngăn việc lặp đi lặp lại của vòng lặp, chúng ta thêm câu lệnh **StopIteration** để thoát khỏi chương trình lặp

Ví dụ:

```
class MyNumbers:
    def __iter__(self):
        self.a = 2
        return self

    def __next__(self):
        x = self.a
        self.a += 2
        if (self.a < 11):
            return x
        else:
            StopIteration

mynumber = MyNumbers()
inumber = iter(mynumber)

print(next(inumber))
print(next(inumber))
```

```
print(next(inumber))
print(next(inumber))
print(next(inumber))
```

Kết quả: Lệnh print cuối cùng nó trả về None vì kết quả đã vượt khỏi số 10.

```
2
4
6
8
None
```

Tổng kết

Qua bài viết chúng ta đã tìm hiểu:

- Khái niệm Iterator, Iterable
- Phương thức tạo Iterator, câu lệnh StopIterator