Bài 20: ITERATION VÀ MỘT SỐ HÀM HỖ TRỢ CHO ITERATION OBJECT TRONG PYTHON

Xem bài học trên website để ủng hộ Kteam: Iteration và một số hàm hỗ trợ cho iteration object trong Python

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage How Kteam nhé!

Dẫn nhập

Trong bài trước, Kteam đã giới thiệu đến bạn cách XỬ LÝ FILE trong Python

Ở bài này Kteam sẽ giới thiệu với các bạn **MỘT SỐ HÀM HỖ TRỢ CHO ITERABLE OBJECT** trong Python. Một trong những điều thiết yếu mà bất cứ ngôn ngữ lập trình nào bạn cũng đều phải tìm hiểu.

Nội dung

Để đọc hiểu bài này tốt nhất bạn cần:

- Cài đặt sẵn MÔI TRƯỜNG PHÁT TRIỂN CỦA PYTHON.

- Xem qua bài CÁCH CHẠY CHƯƠNG TRÌNH PYTHON.
- Nắm CÁCH GHI CHÚ và BIẾN TRONG PYTHON.
- KIỂU DỮ LIỆU LIST, KIỂU DỮ LIỆU TUPLE, KIỂU DỮ LIỆU SET, KIỂU DỮ LIỆU DICT trong Python.

Bạn và Kteam sẽ cùng tìm hiểu những nội dung sau đây

- Khái niệm iteration trong Python
- Giới thiệu iterable object trong Python
- Giới thiệu iterator object trong Python
- Một số hàm hỗ trợ cho iterable object trong Python

Khái niệm iteration trong Python

Iteration là một khái niệm chung cho việc lấy từng phần tử một của một đối tượng nào đó, bất cứ khi nào bạn sử dụng vòng lặp hay kĩ thuật nào đó để có được giá trị một nhóm phần tử thì đó chính là Iteration.

Ví dụ: như bạn ăn một snack, bạn sẽ lấy từng miếng trong bọc snack ra ăn cho tới khi hết thì thôi. Bạn có thể coi việc lấy bánh là một vòng lặp. Đương nhiên bạn cũng có thể chọn không lấy hết số bánh ra.

Giới thiệu iterable object trong Python

Iterable object là một object có phương thức __iter__ trả về một iterator, hoặc là một object có phương thức __getitem__ cho phép bạn lấy bất cứ phần tử nào của nó bằng indexing ví dụ như Chuỗi, List, Tuple.

Giới thiệu iterator object trong Python

Iterator object đơn giản chỉ là một đối tượng mà cho phép ta lấy từng giá trị một của nó. Có nghĩa là bạn không thể lấy bất kì giá trị nào như ta hay làm với List hay Chuỗi.

Iterator không có khả năng tái sử dụng trừ một số iterator có phương thức hỗ trợ như file object sẽ có phương thức seek.

Iterator sử dụng hàm next để lấy từng giá trị một. Và sẽ có lỗi **StopIteration** khi bạn sử dụng hàm next lên đối tượng đó trong khi nó hết giá trị đưa ra cho bạn.

Các **iterable object** chưa phải là iterator. Khi sử dụng hàm iter sẽ trả về một iterator. Đây cũng chính là cách các vòng lặp hoạt động.

Ví dụ minh họa:

```
>>> [x for x in range(3)] # thuộc lòng 3 giá trị của comprehension này
[0, 1, 2]
>>> itor = (x for x in range(3)) # sử dụng () cho ra một generator expression – một iterator
>>> itor
<generator object <genexpr> at 0x03374CC0>
>>> next(itor)
0
>>> next(itor)
1
>>> next(itor)
2
>>> next(itor) # chỉ có 3 giá trị, và ta đã lấy hết
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

File object cũng là một iterator. Bạn cũng có thể sử dụng cách này để đọc file.

```
>>> lst = [6, 3, 7, 'kteam', 3.9, [0, 2, 3]]
>>> iter_list = iter(lst) # iter_list là một iterator tạo từ list
>>> iter_list
<list_iterator object at 0x03647730>
>>> iter_list[0] # đương nhiên, iterator không hỗ trợ indexing
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'list_iterator' object is not subscriptable
>>> next(iter_list)
6
>>> next(iter_list)
3
>>> next(iter_list)
7
>>> next(iter_list)
'kteam'
>>> next(iter_list)
3.9
>>> next(iter_list)[-2]
2
>>> next(iter_list)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Bạn cũng lưu ý, iterator này cũng dính một vấn đề như List, Dict đó chính là chỉnh một, thay đổi hai.

```
>>> it_1 = iter('kteam')
>>> it_1
<str_iterator object at 0x03647770>
>>> it_2 = it_1
>>> next(it_2)
'k'
>>> next(it_2)
't'
>>> next(it_2)
'e'
>>> next(it_1)
'a'
>>> next(it_1)
'm'
```

```
>>> next(it_2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>> next(it_1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Một số hàm hỗ trợ cho iterable object trong Python

Một điều lưu ý: Các hàm này buộc phải lấy các giá trị của iterable để xử lí, do đó nếu bạn đưa vào một iterator. Thì bạn sẽ không sử dụng iterator đó được nữa.

Hàm tính tổng – sum

Cú pháp:

```
sum(iterable, start=0)
```

Công dụng: Trả về tổng các giá trị của iterable và iterable này chỉ chứa các giá trị là số. Còn start chính là giá trị ban đầu. Có nghĩa là sẽ cộng từ start lên. Mặc định là **0**

Ví dụ:

```
>>> sum([1, 6, 3])
10
>>> sum([1, 6, 3], 10)
20
>>> sum(iter([6, 3, 9]))
```

```
18
>>> it = (x for x in range(3))
>>> sum(it)
3
>>> next(it)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Hàm tìm giá trị lớn nhất – max

Cú pháp:

> **max(iterable, *[, default=obj, key=func])**

Công dụng: Nhận vào một iterable.Tìm giá trị lớn nhất bằng key (mặc định là sử dụng operator >). Default là giá trị muốn nhận về trong trường hợp không lấy được bất kì giá trị nào trong iterable.

- Dấu * chính là kí hiệu yêu cầu **keyword-only argument**. Bạn sẽ hiểu thêm khi Kteam giới thiệu parameter trong function.

```
>>> max([1, 2, 3])
3
>>> max([1, 2, 3], default='default value')
3
>>> max([], default='default value')
'default value'
```

Hoặc

> **max(arg1, arg2, *args, *[, key=func])**

Trong đó:

- ***args** là packing arguments (bạn sẽ hiểu thêm khi Kteam giới thiệu với bạn packing arguments). Ở đây không có parameter **default**, vì khi theo cách này, bạn luôn luôn có ít nhất 2 giá trị so sánh

```
>>> max(1, 2, 3)
3
>>> max(1, 2)
2
```

Hàm tìm giá trị nhỏ nhất – min

Cú pháp:

```
min(iterable, *[, default=obj, key=func])
```

hoặc

```
min(arg1, arg2, *args, *[, key=func])
```

Ý nghĩa: giống như hàm max. Khác ở chỗ đây là tìm giá trị nhỏ nhất

```
>>> min([1, 2, 3])
1
>>> min([], default='kteam')
'kteam'
```

Hàm sắp xếp – sorted

Cú pháp:

```
sorted(iterable, /, *, key=None, reverse=False)
```

Công dụng: Giống với phương thức sort của List object.

Ví dụ:

```
>>> sorted([1, 6, 7, 2])
[1, 2, 6, 7]
>>> sorted([1, 6, 7, 2], reverse=True)
[7, 6, 2, 1]
```

Củng cố bài học

Đáp án bài trước

Bạn có thể tìm thấy câu hỏi của phần này tại CÂU HỎI CỦNG CỐ trong bài XỬ LÝ FILE TRONG PYTHON.

1. w+ tạo ra một file nếu file đó hiện chưa có.
2. Vì khi Tèo ghi xong, con trỏ file nằm ở cuối file > Tèo không đọc được gì. Trường hợp đó, ta sử dụng phương thức seek.

Kết luận

Qua bài viết này, Bạn đã hiểu hơn về ITERABLE OBJECT trong Python.

Ở bài viết sau. Kteam sẽ nói về NHẬP XUẤT TRONG PYTHON.

Cảm ơn bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên "**Luyện tập – Thử thách – Không ngại khó**".