Python Iterables

Summary: in this tutorial, you'll learn about the Python iterables and iterators.

Introduction to Python iterables

In Python, an iterable is an object that includes zero, one, or many elements. An iterable has the ability to return its elements one at a time.

Because of this feature, you can use a `for` loop (https://www.pythontutorial.net/python-basics/python-for-loop-list/) to iterate over an iterable.

In fact, the `range()` function is an iterable because you can iterate over its result:

```
for index in range(3):
    print(index)
```

Output:

```
0
1
2
```

Also, a string (https://www.pythontutorial.net/python-basics/python-string/) is an iterable because you can use a for loop (https://www.pythontutorial.net/python-basics/python-for-range/) to iterate over it:

```
str = 'Iterables'
for ch in str:
    print(ch)
```

Lists (https://www.pythontutorial.net/python-basics/python-list/) and tuples (https://www.pythontutorial.net/python-basics/python-tuples/) are also iterables because you can loop over them. For example:

```
ranks = ['high', 'medium', 'low']

for rank in ranks:
    print(rank)
```

The rule of thumb is that if you know if can loop over something, it's iterable.

What is an iterator

An iterable can be iterated over. And an iterator is the agent that performs the iteration.

To get an iterator from an iterable, you use the `iter()` function. For example:

```
colors = ['red', 'green', 'blue']
colors_iter = iter(colors)
```

Once you have the iterator, you can get the next element from the iterable using the `next()` function:

```
colors = ['red', 'green', 'blue']
colors_iter = iter(colors)
```

```
color = next(colors_iter)
print(color)
```

Output:

```
red
```

Every time, you call the `next()` function, it returns the next element in the iterable. For example:

```
colors = ['red', 'green', 'blue']
colors_iter = iter(colors)

color = next(colors_iter)
print(color)

color = next(colors_iter)
print(color)

color = next(colors_iter)
print(color)
```

Output:

```
red
green
blue
```

If there isn't any more element and you call the `next()` function, you'll get an exception.

```
colors = ['red', 'green', 'blue']
colors_iter = iter(colors)

color = next(colors_iter)
```

```
print(color)

color = next(colors_iter)
print(color)

color = next(colors_iter)
print(color)

# cause an excpetion
color = next(colors_iter)
print(color)
```

This example first shows three elements of the colors list and then issues an exception:

```
red
green
blue
Traceback (most recent call last):
  File "iterable.py", line 15, in <module>
    color = next(colors_iter)
StopIteration
```

The iterator is stateful. It means that once you consume an element from the iterator, it's gone.

In other words, once you complete looping over an iterator, the iterator becomes empty. If you iterate over it again, it'll return nothing.

Since you can iterate over an iterator, the iterator is also an iterable object. This is quite confusing. For example:

```
colors = ['red', 'green', 'blue']
iterator = iter(colors)
```

```
for color in iterator:
    print(color)
```

Output:

```
red
green
blue
```

If you call the `iter()` function and pass an iterator to it, it'll return the same iterator back.

Later, you'll learn how to create iterables.

Summary

- An iterable is an object that can be iterated over. An iterable has the ability to return one of its elements at a time.
- An iterator is an agent that performs iteration. It's stateful. And an iterator is also an iterable object.
- Use the `iter()` function to get an iterator from an iterable object and the next() function to get the next element from the iterable object.