Python Iterators

Iterators are methods that iterate collections like lists, tuples, etc. Using an iterator method, we can loop through an object and return its elements.

Technically, a Python **iterator object** must implement two special methods, `__iter__()` and `__next__()`, collectively called the **iterator protocol**.

Iterating Through an Iterator

In Python, we can use the next() function to return the next item in the sequence.

Let's see an example,

```
# define a list
my_list = [4, 7, 0]

# create an iterator from the list
iterator = iter(my_list)

# get the first element of the iterator
print(next(iterator))  # prints 4

# get the second element of the iterator
print(next(iterator))  # prints 7

# get the third element of the iterator
print(next(iterator))  # prints 0
```

Output

```
4
7
0
```

Here, first we created an iterator from the list using the iter() method. And then used the `next()` function to retrieve the elements of the iterator in sequential order.

When we reach the end and there is no more data to be returned, we will get the `StopIteration` Exception.

Using for Loop

A more elegant way of automatically iterating is by using the for loop. For example,

```
# define a list
my_list = [4, 7, 0]

for element in my_list:
    print(element)
```

Output

```
4
7
0
```

Working of for loop for Iterators

The `for` loop in Python is used to iterate over a sequence of elements, such as a list, tuple, or string.

When we use the `for` loop with an iterator, the loop will automatically iterate over the elements of the iterator until it is exhausted.

Here's an example of how a `for` loop works with an iterator,

```
# create a list of integers
my_list = [1, 2, 3, 4, 5]

# create an iterator from the list
iterator = iter(my_list)

# iterate through the elements of the iterator
for element in iterator:

    # Print each element
    print(element)
```

In this example, the `for` loop iterates over the elements of the iterator object.

On each iteration, the loop assigns the value of the next element to the variable element, and then executes the indented code block.

This process continues until the iterator is exhausted, at which point the for loop terminates.

Building Custom Iterators

Building an iterator from scratch is easy in Python. We just have to implement the `__iter__()` and the `__next__()` methods,

- `__iter__()` returns the iterator object itself. If required, some initialization can be performed.
- `__next__()` must return the next item in the sequence. On reaching the end, and in subsequent calls, it must raise `StopIteration`.

Let's see an example that will give us the next power of **2** in each iteration. Power exponent starts from zero up to a user set number,

```
class PowTwo:
    """Class to implement an iterator
    of powers of two"""

    def __init__(self, max=0):
        self.max = max

    def __iter__(self):
        self.n = 0
        return self

    def __next__(self):
        if self.n <= self.max:
            result = 2 ** self.n
            self.n += 1
            return result
        else:
            raise StopIteration


# create an object
numbers = PowTwo(3)

# create an iterable from the object
i = iter(numbers)

# Using next to get to the next iterator element
print(next(i)) # prints 1
print(next(i)) # prints 2
print(next(i)) # prints 4
print(next(i)) # prints 8
print(next(i)) # raises StopIteration exception
```

Output

```
1
2
4
8
```

```
Traceback (most recent call last):
  File "<string>", line 32, in <module>
File "<string>", line 18, in __next__
StopIteration
```

We can also use a `for` loop to iterate over our iterator class.

```
for i in PowTwo(3):
    print(i)
```

Output

```
1
2
4
8
```

To learn more about object-oriented programming, visit Python OOP.

Python Infinite Iterators

An infinite iterator is an iterator that never ends, meaning that it will continue to produce elements indefinitely.

Here is an example of how to create an infinite iterator in Python using the `count()` function from the `itertools` module,

```
from itertools import count

# create an infinite iterator that starts at 1 and increments by 1 each time
infinite_iterator = count(1)

# print the first 5 elements of the infinite iterator
for i in range(5):
    print(next(infinite_iterator))
```

Output

```
1
2
3
4
5
```

Here, we have created an infinite iterator that starts at **1** and increments by **1** each time.

And then we printed the first **5** elements of the infinite iterator using the `for` loop and the `next()` method.

Table of Contents

- Introduction
- Iterating Through an Iterator
- Working of for loop for Iterators
- Building Custom Iterators
- Python Infinite Iterators