Analytics Vidhya

Home

Beginner

Everything You Should Know about Iterables and Iterators in Python ...

Everything You Should Know about Iterables and Iterators in Python as a Data Scientist!

CHIRAG GOYAL

CHIRAG GOYAL

04 Aug, 2022 • 8 min read

This article was published as a part of the Data Science Blogathon

Introduction

Python is a beautiful Programming Language. Because of its flexibility and incredible functionalities, most people loved this language. While working with Python, I have come across a few functionalities whose usage is not commensurate to the number of complexities they simplify. Sometimes, these are also known as “Hidden Gems” in Python. These hidden things are not known by a lot of people but they become super useful when anyone wants to work in the domain of Analytics and Data Science.

In this category, Python Iterators and Iterables can be fitted. Their potential is immense!

Dare with Python: An experiment for all (intro) - Think Big iteratorImage Source:

One of the most important concepts in Python is Iteration. The term Iteration not comes alone but comes up with two other terms that pertain to iteration are iterators and iterables. Also, the understanding of iterables and iterators in Python is very important. But these python concepts are a little bit tricky to understand. In this article, I will try to give a comprehensive explanation of iterables and iterators. We will understand these things starting from what they are, how they work, and also see how to use them properly.

So let’s deep dive into the article and explore the world of Python Iterators and Iterables.

To follow this article properly, I assume you are familiar with the basics of Python. If not, I recommend the below popular course given by Analytics Vidhya to get started with the Python Basics:

Python for Data Science

Table of Contents

What is Iteration?

What are Iterables?

What are Python Iterators?

Why should you use iterators?

What is the relationship between Iterators and Iterables?

What is the difference between Iterators and Iterables?

What are the Limits of Iterator?

What is Iteration?

One of the most essential principles of software development is that Don’t Repeat Yourself, which is elaborated in “The Pragmatic Programmer” Book in the following way:

“Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.” — Andy Hunt and Dave Thomas

One specific application of the above-mentioned principle in modern programming is the use of Iteration that involves going over a list of items, on which we performed the defined operation. One of the most basic forms of iteration is a for loop.

Although many other languages such as and use three-expression for-loop, while Python programming language uses a more concise syntax of for-loop.

To understand what exactly iteration means, you have to understand the following points:

Iteration means ‘repeating steps’ in layman’s language.

In Programming, Iteration is defined as a repetition of a block of code a specified number of times.

To achieve the iterations, we can use loops such as for loop, etc.

What are Iterables?

Iterables are objects that can be iterated in iterations.

To understand what exactly iterables means, you have to understand the following points:

Iterable is an object which can be looped over or iterated over with the help of a for loop.

Objects like lists, tuples, sets, dictionaries, strings, etc. are called iterables. In short and simpler terms, iterable is anything that you can loop over.

In simpler words, iterable is a container that has data or values and we perform an iteration over it to get elements one by one. (Can traverse through all the given values one by one)

Iterable has an in-built dunder method \_\_iter\_\_.

How we can check whether an object is Iterable or not?

An object is called an iterable if you can get an iterator out of it.

A simpler way to determine whether an object is iterable is to check if it supports \_\_iter\_\_. How? Here we use the function named dir( ), which returns the list of attributes and methods supported by an object, and by seeing all attributes and methods, we can find all and select required methods from them.

Calling iter() function on an iterable gives us an iterator.

Calling the next() function on iterator gives us the next element.

If the iterator is exhausted(if it has no more elements), calling next() raises the StopIteration exception.

iterator image

Image Source:

For Example,

Python Code:

What are Python Iterators?

An Iterator is an object representing a stream of data that produces a data value at a time using the \_\_next\_\_() method.

To understand what exactly iterators mean, you have to understand the following points:

In Python, an iterator is an object which implements the iterator protocol, which means it consists of the methods such as \_\_iter\_\_() and \_\_next\_\_().

An iterator is an iterable object with a state so it remembers where it is during iteration. For Example, Generator

These iterators give or return the data one element at a time.

It performs the iteration to access the elements of the iterable one by one. As it maintains the internal state of elements, the iterator knows how to get the next value.

Which in-built methods does iterator have?

Iterator supports in-built dunder methods such as \_\_iter\_\_ and \_\_next\_\_

Iterators can only move forward using \_\_next\_\_. But remember that, iterators cannot go back or cannot be reset.

To learn more about the dunder methods in Python, refer to the link

Let’s learn more about the \_\_next\_\_ function:

The \_\_next\_\_() method takes no arguments and always returns the next element of the stream.

If there are no more elements in the stream, \_\_next\_\_() must raise the StopIteration exception.

Iterators don’t have to be finite. It’s perfectly reasonable to write an iterator that produces an infinite stream of data.

For Example,

number\_iterator = iter([1, 2, 3, 4, 5])

print(type(number\_iterator))

print(next(number\_iterator))

print(next(number\_iterator))

print(next(number\_iterator))

print(next(number\_iterator))

print(next(number\_iterator))

# Once the iterator is exhausted, next() function raise StopIteration.

print(next(number\_iterator))

Output