Advanced Python - Iterator Protocol

In this article I will introduce you to a Python Iterator Protocol.

You will learn how to implement the Iterator Protocol and when it should be used.

We'll commence with rudimentary ideas of implementation and how to include them in your projects.

What is a Python Iterator Protocol?

Python Iterator Protocol is nothing similar to a design pattern. It's a data container of collections or sequences with implemented two basic methods: "__iter__" and "__next__".

These two are a basic requirement of an Iterator Protocol. If you implement only "__iter__" method you will get an Iterable Object which is not an Iterator Protocol. Understanding the difference is important.

The method "__iter__" returns self iterator and includes declaration of internal Iterator fields. All internal Iterators fields should be declared in this method.

The method "__next__" returns next element of the iterator and includes implementation of the "StopIterator" Exception.

The "StopIterator" Exception is raised when the Iterator is exhausted and stops the iteration process.

What is a Python Iterator?

A Python Iterator is an Iterable Object created by the Iterator Protocol. The Iterator Protocol always includes methods: "__iter__" and "__next__".

What is Python Iterable Object?

An Iterable Object has an implemented method "__iter__".

Understanding the difference between Iterator and an Iterable object is very important. Remember, all Iterators are Iterable Objects but not all Iterable Objects are Iterators.

An example of Iterable Objects which are not Iterators: "list", "tuple", "set", "dictionary", "string".

How to implement Python Iterator Protocol?

In this part of the article I'll show you how to implement Iterator Protocol. We will implement an Iterator Protocol with an example of a list of friends.

https://gist.github.com/slawek87/06f18ad5561a053fac2787f01b963ab2
Once we have prepared the list of friends. Notice each element of the list includes: name, gender and age. Now we will implement the basic Iterator Protocol for the list.

https://gist.github.com/slawek87/e8c5e17778db05b1e4bba3762c9fbb3c

Our basic implementation contains methods including: "__init__", "__iter__", "__next__", "__len__". The "MyFriendsIterator" returned a basic Iterator with information about the friends.

For the moment moment there is nothing special to observe. (Is this sentence needed?)

Now we want to know which of the friends is older than 21. It would be ideal to get this information in the new boolean field "is_gte_21".

In this case we can add this field to Friend object but we don't want to modify basic list of our friends. We also use this list in another place where we don't want to have that field.

We know that new fields in each object of the list will use more memory.

Knowing that we will use the Iterator Protocol to add this field dynamically.

https://gist.github.com/slawek87/8379a43e897a00b17af60afb061a38c6

Now we have implemented the method "_add_fields". This is easy way to keep code clean when you need to add new fields dynamically.

Imagine that our friend Kate asked us to send her a list of girls invited to the party. In this case we will add female gender filter to the "MyFriendsIterator".

https://gist.github.com/slawek87/292d74d0ea4e4a96d9f7e8d1de37bcf5

The "MyFriendsIterator" returns only females, but our friend Kate has changed her mind and now wants a list of females who are older than 21.

Let's add another filter to our "MyFriendsIterator".

https://gist.github.com/slawek87/51dd61334f6966815b60c7470b3cde4f

The Iterator now returns only females older than 21. There are many methods on how to use the Iterator Protocol.

I hope that you already have found places in your code where you can use it.

When Python Iterator Protocol should be used?

I personally use the Iterator Protocol when I need to add extra fields to already created sequences which shouldn't be modified or when I need to filter those sequences.

You can also use Iterator Protocol when you work with collections, for example when you want to create your own implementation of an ordered dictionary.

The Iterator Protocol is also useful for ranges of data, for example when you need to create a range of dates in a given period of time.

You can could also use methods or functions to get the same results but, the Iterator Protocol is a nice pattern to keep python code simple and clean. Iterators also use less memory than normal Iterable Objects reducing the overall weight of your code.