# Iterables

This lesson explains iterables in Python.

# Iterables

Iteration is defined as *the repetition of a process or utterance.*

## Iterable

In Python an iterable is an object which can be looped over its member elements using a for loop. An iterable is capable of returning its members one by one. The most common type of iterables in Python are sequences which include lists, strings and tuples. They support efficient access via indexing. The **`__getitem()__`** can be invoked to return a member at the specified index. They also return their lengths via the **`__length()__`** method. But not every type in Python is a sequence. Dictionaries, sets, file objects and generators *aren't indexable but are iterable*. Python also allows us to create iterables that are infinite called generators that we'll learn in later sections.

In order to qualify as an iterable, an object must define one of the two methods:

- __iter__()
- __getitem__()

Container objects need to define the `__iter__()` method to support iteration. The `__iter__()` method returns an *iterator* for the object. Using the iterator we can sequentially access the elements of an iterable object.

## Iterator

Iterator is an object which can be used to sequentially access the elements of an **Iterable** object. The iterator exposes `__next__()` method in Python 3 and `next()` in Python 2. Both the methods fetch the next element in sequence of the iterable object. An iterator must support the following methods

- __iter__()
- __next__()

The iterator object returns itself for the `__iter__()` method. And this allows us to use the iterator and the iterable in a for loop

When the end of an iterable object's iteration is reached, `next()` throws a `StopIteration` exception. **Put together, these rules are called the iterator protocol.** The `__iter__()` method for a container can also return something called a *generator*, which is also an iterator.

## Example

Study the code below which demonstrates iterable, iterator, and iteration in Python

iteration in Python.

```
if __name__ == "__main__":
    lstOfInts = list()
    lstOfInts.append(1)
    lstOfInts.append(2)
    lstOfInts.append(3)

    # get iterator of list using __iter__()
    it = lstOfInts.__iter__()
    print("iterator of list: " + str(it))

    # get member element of list using __getitem__()
    print("iterator of list: " + str(lstOfInts.__getitem__(2
)))

    # iterator returns itself when passed to the iter function
    print("it is iter(it) = " + str(it is iter(it)))

    # get another iterator for list using the built in iter
() method
    it_another = iter(lstOfInts)
    print("it_another = " + str(it_another))

    print("iteration using iterator in a for loop")
    # iterate using the iterator
    for element in it_another:
        print(element)

    print("iteration using iterable in a for loop")
    # iterate using the iterable
    for element in lstOfInts:
        print(element)
```

You can run the above code in the code widget below.

```
if __name__ == "__main__":
    lstOfInts = list()
    lstOfInts.append(1)
    lstOfInts.append(2)
    lstOfInts.append(3)

    # get iterator of list using __iter__()
    it = lstOfInts.__iter__()
    print("iterator of list: " + str(it))

    # get member element of list using __getitem__()
    print("iterator of list: " + str(lstOfInts.__getitem__(2)))

    # iterator returns itself when passed to the iter function
    print("it is iter(it) = " + str(it is iter(it)))

    # get another iterator for list using the built in iter() method
    it_another = iter(lstOfInts)
    print("it_another = " + str(it_another))

    print("iteration using iterator in a for loop")
    # iterate using the iterator
    for element in it_another:
        print(element)

    print("iteration using iterable in a for loop")
    # iterate using the iterable
    for element in lstOfInts:
        print(element)
```

Back
Introduction

Next
Yield

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/iterables__asyncio__python-concurrency-for-senior-engineering-interviews)