**Lab Exercise 1 – Python Iterator**

In Python, an iterator is an object that represents a sequence of elements and allows you to traverse through those elements one at a time. It provides a way to access the elements of a collection or a custom iterable object without exposing the underlying details of the data structure. Iterators are extensively used in Python for loops, comprehensions, and other operations that involve going through a sequence of items.

To create and work with iterators in Python, you need to understand the following key concepts and methods:

**Iterable:**

* An iterable is an object that can be looped over or iterated.
* Common iterable objects include lists, tuples, dictionaries, strings, and custom objects that implement iteration methods.

**Iterator Protocol:**

Python uses an iterator protocol, which includes two essential methods:

* \_\_iter\_\_(): This method returns the iterator object itself. It is called when you use iter(iterable).
* \_\_next\_\_(): This method returns the next value from the iterator. It is called when you use next(iterator).

**Creating an Iterator:**

* You can create your own iterator by defining a class with the \_\_iter\_\_() and \_\_next\_\_() methods.
* The \_\_iter\_\_() method returns the iterator object (usually self), and \_\_next\_\_() defines how to get the next item from the sequence. When there are no more items, it raises the StopIteration exception.

**Iterating with a for Loop:**

You can iterate through an iterable using a for loop without explicitly calling iter() and next(). Python automatically handles these calls for you.

Here's an example of a custom iterator that generates even numbers up to a specified limit:

class EvenNumbers:

def \_\_init\_\_(self, limit):

self.limit = limit

self.current = 0

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.current <= self.limit:

result = self.current

self.current += 2

return result

else:

raise StopIteration

# Using the custom iterator

even\_iter = EvenNumbers(10)

for num in even\_iter:

print(num)

In this example, the EvenNumbers class is an iterable and an iterator. It generates even numbers up to the specified limit (10 in this case).

Python provides built-in iterators for common data types like lists, strings, dictionaries, and more. Additionally, you can create custom iterators to work with your own data structures or to implement specific iteration logic. Understanding iterators is essential for efficient and Pythonic coding when dealing with sequences of data.