Department of Computer Science and Engineering
PES University, Bangalore, India

Lecture Notes
Python for Computational Problem Solving
UE23CS151A

Lecture #106

Iterators – User defined

By,
Prof. Sindhu R Pai,
Anchor, PCPS - 2023
Assistant Professor
Dept. of CSE, PESU

Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Centers, Former Chairperson, CSE, PES University)
Prof. Chitra G M(Asst. Prof, Dept. of CSE, PCPS Anchor – 2022)

Introduction

Technically, Iterator is something which has some implementation of **__iter__ and __next__ functions.** These two together are known as **iteration protocol.**

Consider the scenario.

In an examination hall, there are around 60 students. Few are from 3rd semester and few are from 1st semester. If the hall has one invigilator, how should the invigilator distribute the booklets? If the booklets are same for all, the invigilator would like to distribute the booklets starting from first student to 60th student. He might also decide to distribute the booklets to all 3rd sem first and then all 1st semester students. If in case there are different subjects for which students are writing the exam, invigilator might decide to distribute the booklets based on the order of subjects.

Point to think-> Can we have more than one invigilator?

Examination Hall – It is a **container or a collection.** The **invigilator** has to visit each student. He is iterating through the students in the class. He is an **iterator.** It is possible to have a class with multiple invigilators. A container can have multiple iterators. Observe the fact that walking through a container depends on a container, but is not part of the container normally.

Create a list of integers and create a list of strings.

```
Li_ints = [23,55,11,77,99,55]
Li_strings = ["a", "aba", "python", "aaa", "pes"]
```

How do you walk through all the elements of these lists? Create the iterator using iter() and then use next() on it.

```
iter_ints = iter(Li_ints)
print(next(iter_ints));print(next(iter_ints))
print(next(iter_ints));print(next(iter_ints))
print(next(iter_ints));print(next(iter_ints))
print(next(iter_ints));print(next(iter_ints))
```

```
C:\Users\Dell>python classtest.py
23
55
11
77
99
55
Traceback (most recent call last):
  File "C:\Users\Dell\classtest.py", line 39, in <module>
    print(next(iter_ints));print(next(iter_ints))
          ^^^^^^^^^^^^^^^
StopIteration
```

```
iter_strings = iter(Li_strings)
print(next(iter_strings))
print(next(iter_strings))
print(next(iter_strings))
print(next(iter_strings))
print(next(iter_strings))
print(next(iter_strings))
```

```
C:\Users\Dell>python classtest.py
a
aba
python
aaa
pes
Traceback (most recent call last):
  File "C:\Users\Dell\classtest.py", line 41, in <module>
    print(next(iter_strings))
          ^^^^^^^^^^^^^^^^^^
StopIteration
```

Use for loop to handle stopIteration.

```
for val in iter_ints:
   print(val, end = " ")
print("\n----------------")
for val in iter_strings:
   print(val, end = " ")
```

```
C:\Users\Dell>python classtest.py
23 55 11 77 99 55 ----------------
a aba python aaa pes
C:\Users\Dell>python classtest.py
23 55 11 77 99 55
----------------
a aba python aaa pes
C:\Users\Dell>
```

Can we create iterators for list of user defined objects. Let us discuss with an example

Create a type called Student with name as its attribute.

```
class Student:
   def __init__(self, name):
      self.name = name
   def __str__(self):
      return self.name
```

```
#Create three objects and store it in a list
s1 = Student("sindhu")
s2 = Student("indhu")
s3 = Student("bindhu")
li_students = [s1, s2, s3]
lit = iter(li_students)
print(next(lit))
print(next(lit))
print(next(lit))
```

```
C:\Users\Dell>python classtest.py
sindhu
indhu
bindhu
Traceback (most recent call last):
  File "C:\Users\Dell\classtest.py", line 72, in <module>
    print(next(lit))
          ^^^^^^^^^
StopIteration
```

Think about creating our own iterator type which can be applied on any type.

Need of user defined Iterator type: When you want to walk through a type, you might want to iterate through only alternate elements of the iterable or there might be any constraint to walk through elements of the iterable.

```
class MyIterator:   #User defined iterator type as __iter_() and __next__() overridden.
    def __init__(self, li):
        self.li = li
    def __iter__(self):
        self.i = 0
        return self
    def __next__(self):
        self.i += 2
        if((self.i - 2) < len(self.li)):
            return self.li[self.i-2]
        else:
            raise StopIteration
```

On a list of integers, apply this iterator,

```
Li_ints = [23,55,11,77,99,55]
mit = MyIterator(Li_ints)
mit_it = iter(mit)
print(next(mit_it))
print(next(mit_it))
print(next(mit_it))
print(next(mit_it))
```

```
C:\Users\Dell>python classtest.py
23
11
99
Traceback (most recent call last):
  File "C:\Users\Dell\classtest.py", line 85, in <module>
    print(next(mit_it))
          ^^^^^^^^^^^^
  File "C:\Users\Dell\classtest.py", line 74, in __next__
    raise StopIteration
StopIteration
```

On a list of strings, apply this iterator,

```
Li_strings = ["a", "aba", "python", "aaa", "pes"]
mit = MyIterator(Li_strings)
mit_it = iter(mit)
print(next(mit_it))
print(next(mit_it))
print(next(mit_it))
print(next(mit_it))
```

```
C:\Users\Dell>python classtest.py
a
python
pes
Traceback (most recent call last):
  File "C:\Users\Dell\classtest.py", line 83, in <module>
    print(next(mit_it))
          ^^^^^^^^^^^^
  File "C:\Users\Dell\classtest.py", line 74, in __next__
    raise StopIteration
StopIteration
```

On a list of student objects, apply this iterator,

```
s1 = Student("sindhu")
s2 = Student("indhu")
s3 = Student("bindhu")
li_students = [s1, s2, s3]
li_iterator = MyIterator(li_students)
lit = iter(li_iterator)
print(next(lit))
print(next(lit))
print(next(lit))
```

```
C:\Users\Dell>python classtest.py
sindhu
bindhu
Traceback (most recent call last):
  File "C:\Users\Dell\classtest.py", line 96, in <module>
    print(next(lit))
          ^^^^^^^^^
  File "C:\Users\Dell\classtest.py", line 74, in __next__
    raise StopIteration
StopIteration
```

Use for loop to get away with the exceptions.

```
Li_ints = [23,55,11,77,99,55]
print("Actual list", Li_ints)
mit = MyIterator(Li_ints)
for i in mit:
   print(i, end = " ")
print("\n----------------------------------")
Li_strings = ["a", "aba", "python", "aaa", "pes"]
print("Actual list", Li_strings)
mit = MyIterator(Li_strings)
for i in mit:
   print(i, end = " ")
print("\n----------------------------------")
s1 = Student("sindhu")
s2 = Student("indhu")
s3 = Student("bindhu")
li_students = [s1, s2, s3]
print("Actual list", end = " ")
for i in li_students:
   print(i, end = " ")
print()
li_iterator = MyIterator(li_students)
for i in li_iterator:
   print(i, end = " ")
```

```
C:\Users\Dell>python classtest.py
Actual list [23, 55, 11, 77, 99, 55]
23 11 99
----------------------------------
Actual list ['a', 'aba', 'python', 'aaa', 'pes']
a python pes
----------------------------------
Actual list sindhu indhu bindhu
sindhu bindhu
C:\Users\Dell>
```

Try this!

- **Can you make some changes in the code to make sure that iterator will skip two elements at a time?**
- **Once you iterate through the element using the next function of MyIterator type, are you allowed to access those elements again?**
- **Can we shift self.i = 0 to the constructor function and remove the definition of __iter__() to get the same output?**

-END-