Python Advanced Topics

Dr. Ami Tusharkant Choksi
Associate Professor,
Computer Engineering Department,
C.K.Pithawala College of Engineering & Technology, Surat, Gujarat State, India.
email: ami.choksi@ckpcet.ac.in

April 25-May 01, 2020

Iterator I

- Iterator in python is any python type that can be used with a 'for in loop'.
- Python lists, tuples, dicts and sets are all examples of inbuilt iterators.
- In a class, define an `__iter__()` method which returns an object with a `__next__()` method
- If the class defines `__next__()`, then `__iter__()` just return itself
- Any object that wants to be an iterator must implement above two methods.
- The backbone for *for ...in* statement

Iterator II

Listing 1: Iterator Example

```
#{Iterator Example}
for element in [1,2,3]:
        print(element,end=":")#1:2:3:
print()
for element in (4,5,6):
        print(element,end=":")#4:5:6:
print()
for key in {'one':1,'two':2}:
        print(key,end=":")#one:two:
print()
for char in "789":
        print(char,end=":")#7:8:9:
print()
for char in 'ami':
        print(char,end=":")#a:m:i:
print()
```

Iterator III

```
17 for line in open("myfile.txt"):
18         print(line,end=":")
19 #Hello and welcome
20 #:to
21 #:the world of
22 #:computers:
```

Listing 2: Contents of myfile.txt

```
1 Hello and welcome
2 to
3 the world of
4 computers
```

- for statement calls iter() on the container object
- iter() function returns an iterator object that defines the method next()
- next() accesses elements in the container one at a time

Iterator IV

- When there are no more elements, next() raises a StopIteration exception
- tells the for loop to terminate.

Listing 3: String Iterator example using iter()

```
1 #String iterator using iter()
2 s='abc'
3 it=iter(s)
4 print(it)#<str_iterator object at 0x7fd69cce1390>
5 print(next(it))#a
6 print(next(it))#b
7 print(next(it))#c
```

Iterator V

Listing 4: Tuple Iterator example using iter()

```
1 s=(1,2,3)
2 it=iter(s)
3 print(it)
4 print(next(it))#1
5 print(next(it))#2
6 print(next(it))#3
```

Listing 5: List Iterator example using iter()

```
1 #List Iterator example using iter()
2 st=[4,5,6]
3 it=iter(st)
4 print(it)
5 print(next(it))#4
6 print(next(it))#5
7 print(next(it))#6
```

Iterator VI

Listing 6: get Next key using iter() for Dictionary

```
1  dict2={'name':'ami','work':'ckpcet','exp':'20'}
2
3  it=iter(dict2)
4  print(next(it))#name
5  print(next(it))#work
6  print(next(it))#exp}]
7  #get \textsc{Next key using iter() for Dictionary}
8  dict2={'name':'ami','work':'ckpcet','exp':'20'}
9
10 it=iter(dict2)
11 print(next(it))#name
12 print(next(it))#work
13 print(next(it))#exp
```

- To prevent the iteration to go on forever, we can use the `StopIteration` statement.

Listing 7: Iter for class

```
# A simple Python program to demonstrate
# working of iterators using an example type
# that iterates from 10 to given value

# An iterable user defined type
class Test:

# Cosntructor
def __init__(self, limit):
        self.limit = limit

# Called when iteration is initialized
def __iter__(self):
        self.x = 10
        return self

```

Iterator VIII

```
17 # To move to next element. In Python 3,
18 # we should replace next with __next__
19 def next(self):
20
21         # Store current value of x
22         x = self.x
23
24         # Stop iteration if limit is reached
25         if x > self.limit:
26                 raise StopIteration
27
28         # Else increment and return old value
29         self.x = x + 1;
30         return x
31
32 # Prints numbers from 10 to 15
33 for i in Test(15):
34         print(i, end="-")
```

Iterator IX

```
35
36 # Prints nothing
37 for i in Test(5):
38         print(i, end="-")
39 #Output: 10-11-12-13-14-15-
```

Listing 8: Next characters of Alphabet using iter()

```
1  #print next characters using iter()
2  class MyCharacters:
3  def __iter__(self):
4          self.a = 'a'
5          return self
6
7  def __next__(self):
8          x = self.a
9          self.a = chr(ord(self.a) + 1)
10         return x
```

Iterator X

```
11
12 myclass = MyCharacters()
13 myiter = iter(myclass)
14
15 print(next(myiter))#a
16 print(next(myiter))#b
17 print(next(myiter))#c
18 print(next(myiter))#d
19 print(next(myiter))#e
```

zip I

- Python's `zip()` function creates an *iterator* that will aggregate elements from two or more iterables.

Listing 9: zip example

```
numbers = [1, 2, 3]
letters = ['a', 'b', 'c']
zipped = zip(numbers, letters)
print(type(zipped))#<class 'zip'>
print(list(zipped))#[(1, 'a'), (2, 'b'), (3, 'c')]
```

- `zip()` with no arguments create empty zip.

Listing 10: Empty zip

```
#empty zip
zipu = zip()
print(type(zipu))#<class 'zip'>
print(list(zipu))#[]
```

zip II

- Iterate zip object using next

Listing 11: Iterate zip object using next

```
1 #zip object iterate using next()
2 numbers = [1, 2, 3]
3 letters = ['a', 'b', 'c']
4 zipm = zip(numbers, letters)
5
6 print(next(zipm))#(1, 'a')
7 print(next(zipm))#(2, 'b')
8 print(next(zipm))#(3, 'c')
```

zip III

Listing 12: 3 parameters zip

```
1 #3 parameters zip
2 numbers = [1, 2, 3]
3 letters = ['a', 'b', 'c']
4 exp=[1.1,2.5,3.5]
5 zipm = zip(numbers, letters,exp)
6 print(next(zipm))#(1, 'a', 1.1)
7 print(next(zipm))#(2, 'b', 2.5)
8 print(next(zipm))#(3, 'c', 3.5)
```

- for unequal length of zip parameters, it considers least length to create zip

zip IV

Listing 13: Unequal length of zip parameters

```
1 #unequal length arguments to zip
2 numbers = [1, 2, 3]
3 letters = ['a', 'b', 'c']
4 exp=[1.1,2.5]
5 zipm = zip(numbers, letters,exp)
6 print(next(zipm))#(1, 'a', 1.1)
7 print(next(zipm))#(2, 'b', 2.5)
```

- for unequal length of zip parameters, if we want to make zip of longest length parameter, use `zip_longest()` from itertools, library, and for less length parameter, fill that with any character say, '?'

zip V

Listing 14: Longest length parameter of zip, fill value with '?'

```
#longest length argements we want to consider for zip, fill value wi
from itertools import zip_longest
numbers=range(3)
letters=['1','b','c']
longest=[1.1,2.3,3.4,4.5,5.6]
zipm=zip_longest(numbers, letters, longest, fillvalue='?')
print(next(zipm))#(0, '1', 1.1)
print(next(zipm))#(1, 'b', 2.3)
print(next(zipm))#(2, 'c', 3.4)
print(next(zipm))#('?', '?', 4.5)
```

Listing 15: Traversing Lists in Parallel

```
#traversing list in parallel
letters = ['a', 'b', 'c']
numbers = [0, 1, 2]
for l, n in zip(letters, numbers):
print(f'Letter: {l}')
print(f'Number: {n}')
'''

Output:
Letter: a
Number: 0
Letter: b
Number: 1
Letter: c
Number: 2
'''
```

Listing 16: Unzipping a Sequence

```
numbers = [1, 2, 3]
letters = ['a', 'b', 'c']
zipm = zip(numbers, letters)
numbers, letters = zip(*zipm)
print(numbers)#(1, 2, 3)
print(letters)#('a', 'b', 'c')}]
#\textsc{Unzipping a Sequence}
numbers = [1, 2, 3]
letters = ['a', 'b', 'c']
zipm = zip(numbers, letters)
numbers, letters = zip(*zipm)
print(numbers)#(1, 2, 3)
print(letters)#('a', 'b', 'c')
```

References

1. iteraor example, https://www.geeksforgeeks.org/iterators-in-python/
2. StopIteration, https://www.w3schools.com/python/python_iterators.asp
3. ord function, https://stackoverflow.com/questions/1724473/how-could-i-print-out-the-nth-letter-of-the-alphabet-in-p
4. Python Iterators, https://www.w3schools.com/python/python_iterators.asp
5. zip, https://realpython.com/python-zip-function/