Gaurav Gupta

Iterators

- Iterator vs Iterable
- Understanding with list example
- Iterable Requirements
- Iterator Requirements

tuteur.py@gmail.com

Gaurav Gupta

Iterator vs Iterable

- An iterator is an object that allows the next method to be called upon it and returns values.
- In iterable is an object that has the __iter__ method, which returns an iterator.
- Ex: **list** is an **iterable**

  calling the **__iter__** method return an iterator

tuteur.py@gmail.com

Gaurav Gupta

Iterable Requirements

- Should support an **__iter__** method which returns an iterator object upon calling.

- Example:

```
l = [1, 2, 3]
dir(l)
it1 = l.__iter__()
it2 = iter(l)
```

tuteur.py@gmail.com

Gaurav Gupta

Iterator requirements

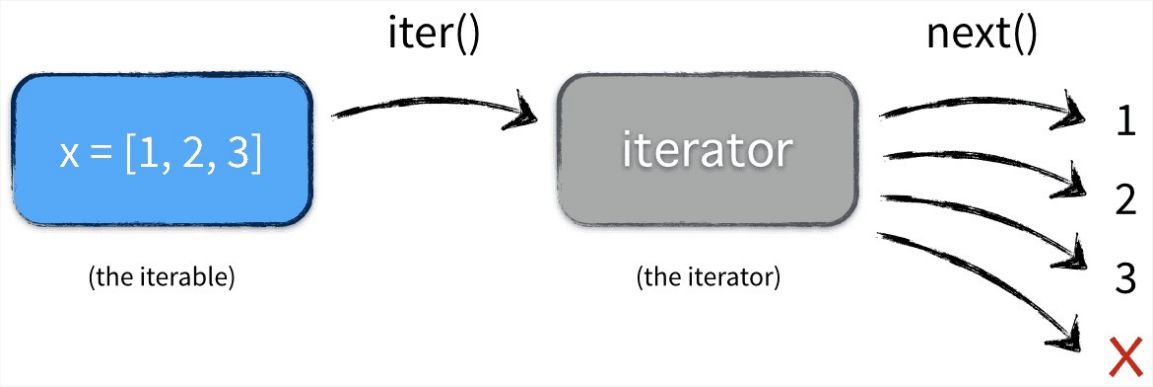
- An iterator should support the **__next__** method.
- Should raise a **StopIteration** exception upon reaching the last element to be iterated.
- Example:

```
l = [1, 2, 3]
itr = iter(l)
itr.__next__()
next(itr)
```

tuteur.py@gmail.com

Gaurav Gupta

Understanding with list example



Gaurav Gupta

World without for loops

-
```
l = [1, 2, 3]
i = 0
while i < len( l ):
    print( l[i] )
    i += 1
```

-
```
It = iter( l )
try:
    while True:
        print( next( it ) )
except:
    pass
```

Gaurav Gupta

Generator and Iterator behavior

- Generator objects also support iterator protocol.
- They have the method __next__ to allow iteration
- Example:

```
def my_range():
        for value in range(10):
                yield(value)
itr = my_range()
dir(itr)
```

tuteur.py@gmail.com