

ITERATORS AND GENERATORS 3

COMPUTER SCIENCE 61A

November 8, 2016

1 Iterators

We've been using iterators all along! Examine this `for` loop:

```
>>> counts = [1, 2, 3]
>>> for item in counts:
    print(item)
1
2
3
```

That `for` loop actually gets unpackaged as follows:

```
>>> counts = [1, 2, 3]
>>> items = iter(counts)
>>> try:
    while True:
        item = next(items)
        print(item)
    except StopIteration
        pass # Do nothing
1
2
3
```

1. What does calling `iter` on an iterable return?
2. What does calling `next` on an iterator return?
3. What is the difference between an iterable and a iterator? What methods does each have?
4. Write an iterator class `Reverse` which takes in a list `lst` and iterates through it in the reverse direction.

```
class Reverse:
    def __init__(self, lst):
```

```

        def __iter__(self):
```

```

            def next(self):
```

5. Write an iterator class that counts down from a given number.

```
class Countdown:
    def __init__(self, n):
```

```

        def __iter__(self):
```

```

            def next(self):
```

6. Write an iterator `zip` that takes in two iterators `iter1` and `iter2`. It will pair the elements of the two iterators until either `iter1` or `iter2` runs out of elements. Once one of the iterators runs out of elements, `Zip` will return `None`

```
class Zip:
```

7. Recall Learning to Count from Midterm 1 Review? Here is the problem statement again:

Tammy always brings a timer with her to take exams. Unfortunately, Tammy bought the timer before the number 6 was discovered. Her timer skips every number that contains the number 6. So it would display the first 20 seconds as follows:

0 1 2 3 4 5 7 8 9 10 11 12 13 14 15 17 18 19 20 21

Only 19 seconds have passed, but the timer shows 21. Write an iterator class that behaves like Tammy's timer.

Hint: write a helper function to determine if `n` contains 6.

```
def has_six(n):
```

```
class Timer:
```

```
    def __init__(self):
```

```
        def __iter__(self):
```

```
            def next(self):
```

8. Now write an iterator class that takes in a Tammy second and counts down from the actual amount of time that has passed. For example, if we pass in 21, the iterator will count down as follows:

19 18 17 16 15 14 ... 5 4 3 2 1 0

Hint: Use the `Countdown` iterator and `has_six` function. You may also find it helpful to write new helper functions.

2 Generators

A generator is an iterator made by calling a generator function. A generator function is one that yields instead of returning a value.

Here is an example of `Countdown`:

```
class Countdown:
    def __init__(self, start):
        self.start = start
    def __iter__(self):
        v = self.start
        while v > 0:
            yield v
            v -= 1
```

1. How can you tell if a function is a generator function?

2. Given the generator function `f`, what will `f()` return? Will calling `f()` cause an error?

```
def gen():
    start = 0
    while start != 10:
        yield start
        start = start / 0
        start += 1
```

3. Write a generator function `map` that takes in an iterator, `iter`, and a function `fn`. It will yield the result of applying `fn` to each element in `iter`

```
def map(iter, fn):
    """ Function that yields result of applying fn to each
        element of fn
    """
    >>> iter = iter([1, 2])
    >>> fn = lambda x: x + 1
    >>> map = map(iter, fn)
    >>> m.next()
    2
    >>> m.next()
    3
    >>> m.next()
    Traceback (most recent call last):
    ...
    StopIteration
    """
```

3 Challenge Questions

1. (From Final Fall 2012)

The `iter` generator for the `BinaryTree` class should `yield` the entries of the tree (and each subtree) starting with the root, and yield all of the entries of the left branch before any of the entries of the right branch.

```
class Iterable(BinaryTree):  
    def __iter__(self):  
        """ Yield entries of tree.  
  
        >>> t = IterableBTree(3, IterableBTree(1),  
                             IterableBTree(5))  
        >>> list(t)  
        [3, 1, 5]  
        """
```

2. (From Final Summer 2013)

Write a function `group_iterator` that takes another iterator of key-value tuples as its argument. It should return a new iterator that yields key-value tuples: one tuple per unique key in the original iterator. The value for each tuple should be a list containing all values corresponding to that key in the original iterator. You may assume that the original iterator has been sorted such that all pairs with the same key are next to each other. You may not assume anything about the length of the provided iterator.

```
def group_iterator(orig):
    """Groups elements from the provided iterator by keys.
    >>> x = [('steven', 1), ('steven', 2), ('eric', 3), ('eric',
    ' ', 5), ('eric', 4)]
    >>> grouped = group_iterator(iter(x))
    >>> next(grouped)
    ('steven', [1, 2])
    >>> next(grouped)
    ('eric', [3, 5, 4])
    >>> next(grouped)
    Traceback
    ...
    StopIteration
    """
```