Creating Your Own Iterators

Occasionally you will want to create your own custom iterators. Python makes this very easy to do. As mentioned in the previous section, all you need to do is implement the __iter__ and __next__ methods in your class. Let's create an iterator that can iterate over a string of letters:

```
class MyIterator:

    def __init__(self, letters):
        """
        Constructor
        """
        self.letters = letters
        self.position = 0

    def __iter__(self):
        """
        Returns itself as an iterator
        """
        return self

    def __next__(self):
        """
        Returns the next letter in the sequence or
        raises StopIteration
        """
        if self.position >= len(self.letters):
            raise StopIteration
        letter = self.letters[self.position]
        self.position += 1
        return letter

if __name__ == '__main__':
    i = MyIterator('abcd')
    for item in i:
        print(item)
```

For this example, we only needed three methods in our class. In our initialization, we pass in the string of letters and create a class variable to refer to them. We also initialize a position variable so we always know where we're at in the string. The __iter__ method just returns itself, which is all it

really needs to do. The __next__ method is the meatiest part of this class. Here we check the position against the length of the string and raise StopIteration if we try to go past its length. Otherwise we extract the letter we're on, increment the position and return the letter.

Let's take a moment to create an infinite iterator. An infinite iterator is one that can iterate forever. You will need to be careful when calling these as they will cause an infinite loop if you don't make sure to put a bound on them.

```
class Doubler:
    """
    An infinite iterator
    """

    def __init__(self):
        """
        Constructor
        """
        self.number = 0

    def __iter__(self):
        """
        Returns itself as an iterator
        """
        return self

    def __next__(self):
        """
        Doubles the number each time next is called
        and returns it.
        """
        self.number += 1
        return self.number * self.number

if __name__ == '__main__':
    doubler = Doubler()
    count = 0

    for number in doubler:
        print(number)
        if count > 5:
            break
        count += 1
```

In this piece of code, we don't pass anything to our iterator. We just instantiate it. Then to make sure we don't end up in an infinite loop, we add a counter before we start iterating over our custom iterator. Finally we start iterating and break out when the counter goes above 5.