PIC 16, Winter 2018 – Preparation 3F

Assigned 1/24/2018. To be completed by class 1/26/2018.

Intended Learning Outcomes

By the end of this preparatory assignment, students should be able to:

- implement iterator behavior in custom classes so they can be looped through using the same convenient syntax as built-in containers; and
- use generators to create simple, custom iterators.

Tasks.

- ☐ Clear your mind. Relax.
- ☐ Read 9.9.
- ☐ (Optional) Watch https://youtu.be/7scnUQEKZ2A and https://youtu.be/ZbVkWu7Djyk
- ☐ The concepts may seem confusing and convoluted at first – but the explanation in the tutorial is actually very clear and concise, so try to follow it line by line. If that doesn't work, here are some (verbose) thoughts that might help.
    - We're learning to write the blueprint for our own container objects, that is, objects that contain multiple elements that we can access individually and iterate over in a `for` loop. Python lists, tuples, sets, and dictionaries are all built-in container objects. Now we're trying to create our own.
    - Our custom container might have a built-in container object as an instance variable. For instance, the tutorial's `Reverse` class has an instance variable `data` that is already a container. In this case, we just need to write some code that tells a `for` loop how to iterate over that instance variable.
    - The simplest sort of container object will have its own `next` method that, when called, returns to the `for` loop the next element in the container. When there are no more elements in the container, it raises a `StopIteration` exception (see 8.4) instead of returning an element. The `for` loop terminates when it gets this exception.
    - In general, however, the container object doesn't need to have its own `next` method. Instead, it may assign the job of picking the next element to a separate object, called an iterator.
    - In general, an iterator is any object that defines a suitable `next` method. When an iterator object's `next` method is invoked, the method should return the next element of some collection – whatever that may mean. How the `next` method is written defines the order in which the elements of a collection are iterated over in a `for` loop.
    - Your collection appoints an iterator by defining an `__iter__` method that returns an instance of an iterator object.
    - If the collection has its own `next` method, the collection's `__iter__` method can return `self`; the container will serve as its own iterator. Note that in the tutorial's example, the `Reverse` class is both the container *and* the iterator object. But in general, the iterator can be a separate object from the container.
    - If you are understand all this, you might ask "Wouldn't it be simpler if containers were just required to have their own `next` method that the `for` loop would call? What is the use of first getting an iterator object from an `__iter__` method (which could just be the container object itself) and then invoking *that* object's `next` method?" You'll see in the assignment.

- If 9.9 is still difficult to follow or if you have trouble with any of the remaining sections (9.10 and 9.11), I suggest you read http://anandology.com/python-practice-book/iterators.html.
- Remove the `next` method and `index` instance variable from the `Reverse` class from the tutorial. Write a new class, `ReverseIterator`, that serves as an iterator for the `Reverse` class. Modify the `__iter__` method of the `Reverse` class accordingly so that the test code from the tutorial (`for char in Reverse('spam'): print char`) still works.
- Read 9.10.
- (Optional) Watch https://youtu.be/np152YN7T6Q
- Following the example, write a generator `every_other(data)` that `yield`s every other element of the data. The test code:

```
for char in every_other("supercalifragilisticexpialidocious"):
    print char,
```

  should print: `s p r a i r g l s i e p a i o i u`
- Read 9.11. Note the first example might come in handy in the `MathVector` assignment.
- (Optional) Watch https://youtu.be/13qEGstK4BY
- (Challenging) Create the same generator `every_other` (as above) in a single line using a lambda function and a generator expression.