Lecture 19: Review of iterators, classes, and object-oriented programming
Recall that something is *iterable* if it supports the `iter` function—that is the method `__iter__` is defined—and returns an iterator. An *iterator* is something that

- supports the `next` function—that is, the method `__next__` is defined;
- throws a `StopIteration` when the iterator is empty; and
- returns itself under an `iter` call.

Iterators may be defined using *classes* or with *generators*.
class squares:

    def __init__(self, threshold=None):
        self._state = 1
        self._threshold = threshold

    def _below_threshold(self):
        return self._threshold is None or self._state**2 < self._threshold

    def __iter__(self):
        return self

    def __next__(self):
        if self._below_threshold():
            sq = self._state**2
            self._state += 1
            return sq
        else:
            raise StopIteration()
A Generator for Squares

```python
def squares_gen(threshold=None):
    i = 1
    while threshold is None or i**2 < threshold:
        yield i**2
        i += 1
```

Without getting too technical, the primary characteristics associated with object-oriented programming are

- inheritance;
- encapsulation; and
- polymorphism
```python
class Shape:
    # class definitions here

class Rectangle(Shape):
    # class definitions here

class Square(Rectangle):
    # class definitions here
```
class Shape:

class Rectangle(Shape):
    
def __init__(self, width, height):
        self._width = width
        self._height = height
class Shape:
    def area():
        pass

class Rectangle(Shape):
    def area():
        return self._width * self._height

class Square(Rectangle):
    def __init__(self, side):
        super().__init__(side, side)

>>> shape = Rectangle(10, 20)
>>> shape.area()
200
>>> shape = Square(10)
>>> shape.area()
100
class even_squares(squares):

    def __next__(self):
        sq = next(super())
        while (sq % 2 != 0):
            sq = next(super())
        return sq