Python OOP

Download Starter Code

Serial Generator

Classes are a great way to combine data and functionality.

We'll use a class to make a "serial number generator" — you should be able to initialize it with a start number, like this:

```
>>> serial = SerialGenerator(start=100)
```

And then every time you ask for a new number, it should return the next sequential number:

```
>>> serial.generate() 100 >>> serial.generate() 101 >>> serial.generate() 102
```

You should provide a function to reset the number back to the original start number:

```
>>> serial.reset() >>> serial.generate() 100
```

We've given you a starter file, ***serial.py***, with doctests for this functionality. Add the missing functionality.

Do make sure you put in docstrings for your methods!

Random Word

You'll need to make a class that works like this:

- it is instantiated with a path to a file on disk that contains words, one word per line
  - it reads that file, and makes an attribute of a list of those words
  - it prints out "[num-of-words-read] words read"

  (it doesn't need to do all of this directly in the ***__init__*** method; it might be a good idea for the ***__init__*** method to call other functions to do some of this.)
- it provides a method, ***random()***, which returns a random word from that list of words

  Note: the ***random*** method **should not** re-read the list of words each time; it should work with the already-read-in list of words.

For example, assume you have a file at ***/Users/student/words.txt*** that looks like this:

```
cat dog porcupine
```

Working with your class should work like this:

```
>>> wf = WordFinder("/Users/student/words.txt") 3 words read >>> wf.random()
'cat' >>> wf.random() 'cat' >>> wf.random() 'porcupine' >>> wf.random()
'dog'
```

Some notes:

1. You'll need to learn how to read files in Python — you can look at the documentation at http://python.org as a good place to start
2. When Python reads files line-by-line, it still keeps the "newline" character at the end of each line. Make sure you take that off so that when you find a random word, you return it as "cat", not "cat\n"

3. You can make a list of words yourself, or use the words.txt file in the starter code — if you're not on Windows, your computer already has a giant list of English words! On OSX, this is at ***/usr/share/dict/words***

Subclass The RandomWordFinder

Our RandomWordFinder is nice, but we've received a new requirement: sometimes, we'll be provided with files that have blank lines, as well as lines that start with a `#` symbol to make a comment.

For example, we could have a file like this:

```
# Veggies kale parsnips # Fruits apple mango
```

When we work with this, we want it to return one of the actual foods, like "kale" or "apple", but never to return the blank lines or comments.

We can't just change the original Word Finder, though — our requirements have changed, so it would be unfair for users of that class to have this behavior suddenly change.

Make a subclass, ***SpecialWordFinder***, that uses WordFinder, but changes needed parts so it can work. Try to do this so as little code as needed is duplicated.

Advanced Further Study

Bonus!

There's a special method, ***__repr__***, which can be defined on a class. It is used to provide a nicer appearance in debugging messages or in the console when you look at an instance of a class.

Read about ***__repr__*** methods, and then make one so that when you examine your ***serial*** instance, it shows something useful, like this:

```
>>> serial <SerialGenerator start=100 next=101>
```

Testing The RandomWordFinder

Making a doctest for classes like these can be a bit tricky — since they return a random word, how can you be sure what the test would return?

Think of a way you could write doctests for these, then write some!

View our Solution