Library Practice Prompt

The following prompt is a practice prompt, and will be based around creating a library system. Please, follow the prompts to complete the assignment. Please use all variable names that we have requested in the prompts. When we have not requested a particular variable name, you are welcome to choose whatever variable name you would like. Once you have finished your solution for these prompts, you should be able to answer the questions in the Coursera practice quizzes that rely on this notebook.

Please, use this space as a way to become comfortable with using jupyter notebooks if you are not already familiar with them. Select the section called "User Interface Tour" within the "Help" tab in the Notebook Menubar if you are unfamiliar with how to use the tool. To run a block of code, you can either select the "Run Cells" in the Notebook Menubar "Cell" tab or you can use the standard shortcut: Shift + Enter. Remember that you can run cells out of order. This is not likely to affect the tasks that you will be completing, but you may overwrite a variable's value in a way you aren't expecting if the same variable is assigned in different cells and you run the cells out of order.

1. A class called `Book` has already been partially defined. You are tasked with completing the class definition. `Book` takes two input variables in the following order, `title` and `author`, which are strings. There should be two instance variables, `title`, and `author`. You should create 1 method for `Book`. The method will be the `__str__` method. When an instance of the book class is printed out, it should use the following format:

    ```
    "Author: {author name here}, Title: {title of book here}."
    ```

In [45]:
```
class Book():
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def __str__(self):
        #fill this in
        return "Author: {}, Title: {}.".format(self.author, self.title)
```

2. A class called `Library` has already been partially defined. You are tasked with completing the class definition. `Library` takes one input variable, `name`, which is a string that represents the name of the library. There should be two attributes/instance variables, `name`, and `books`.

2a. You will need to set the default value for `books` to be an empty dictionary.

2b. You should create a method called `add_book`, which has one parameter which is an instance of the `Book` class. In `add_book`, you will check to see if a book is already in the dictionary `books`.

```
If the book is not already in the library instance, then you should add the book to the library, using a tuple of the aut
hor and title as the key, and setting the value as another dictionary with two key value pairs. One key is 'checked_out'
which is set to 0. The other is 'available' which is set to 1.
However, if the book is already in the library instance, then you will increase the value of that book's 'available' key-
value pair by 1.
The method should return None. An example of what a "book" in the books dictionary looks like is in the following cell, a
bove the partially defined Library class.
```

3. Create another method for the `Library` class called `can_check_out` which takes in a `Book` instance, checks to see if any copies of that book are available, and either returns True if it can be checked out or returns False if it cannot be checked out. If the book is not in the library at all, it should return the string:

    ```
    "The book you are looking for has not been added to this library yet."
    ```

In [78]:
```
#example = {("Merriam-Webster", "Webster's Dictionary"): {"checked_out": 0, "available": 1}}
#print(example[("Merriam-Webster", "Webster's Dictionary")])
#print(example[("Merriam-Webster", "Webster's Dictionary")]['available'])

class Library:
    def __init__(self, name, books = {}):
        books = {}
        self.name = name
        self.books = books



        #fill in here to define the books attribute/instance variable


    # add_book takes a Book instance, and checks to see if the Book is not in self.books. If it's not
    # then a new key value pair is added to the dictionary. If it is in books, then the data stored
    # in the key 'available' is updated for that book.


    def add_book(self, book_inst):
        if (book_inst.author, book_inst.title) not in self.books:
            # fill in here
            self.books[(book_inst.author, book_inst.title)] = {"checked_out": 0, "available": 1}

        else:
            # fill in here
            self.books[(book_inst.author, book_inst.title)]["available"] += 1

    # can_check_out will take a Book instance as input. Checks to see if the book is in the dictionary
    # of books. If it is, then it checks to see if the value of available is 1 or more. If it is, then it
    # returns True. If it does not, then it returns False. If the book cannot be found, then a string is returned
    # to the user to inform them of this occurrence.
    def can_check_out(self, book_inst):
        book = (book_inst.author, book_inst.title)
        if book in self.books:
            # fill in here
            if self.books[(book_inst.author, book_inst.title)]["available"] >= 1:
                return True
            else:
                return False
        else:
            return "The book you are looking for has not been added to this library yet."
```

4. Finally, we have included a text file with inventory for a library. Each line in the file has an author name and title for a book, separated by the "-" character. Your task is to read in the data from the file called `list_of_books.txt`, create a list of `Book` instances for all lines in the text file and assign that to the variable `book_list`. There is an instance of the `Library` class called `hatcher`; you must add all of the `Book` instances to `hatcher`. Be sure to remove any whitespace from the beginning or end of a book title or author name.

In [79]:
```
f = open('list_of_books.txt').readlines()
book_list = []
for book in f:
    #fill in here
    if '\n' in book:
        book_list.append(book.replace('\n', ''))

    #book_list.append(instance) # the variable instance should be assigned an instance of the Book class.

hatcher = Library("Hatcher Graduate Library")
for book in book_list:
    book = book.split('-')
    author = book[0].strip()
    title = book[1].strip()
    hatcher.add_book(Book(title, author))
```

In [80]:
```
print(hatcher.can_check_out(Book("Webster's Dictionary", "Merriam-Webster")))
print(hatcher.can_check_out(Book('How to Lie with Statistics', 'Darrell Huff')))
```

```
The book you are looking for has not been added to this library yet.
True
```

Tests you can run to check if you have likely answered the prompts correctly.

These tests will not contain answers to the questions you will fill in later. To provide some additional feedback, these tests will check some of the variables or values involved in the correct solutions.

In [81]:
```
import unittest
class MyTests(unittest.TestCase):
    def test_prompt_one_attributes(self):
        book_example = Book("Webster's Dictionary", "Merriam-Webster")
        self.assertEqual(book_example.author, "Merriam-Webster", "Testing that the author attribute is set correctly")
        self.assertEqual(book_example.title, "Webster's Dictionary", "Testing that the title attribute is set correctly")
    def test_prompt_one_method(self):
        book_example = Book("Webster's Dictionary", "Merriam-Webster")
        self.assertEqual(book_example.__str__(), "Author: Merriam-Webster, Title: Webster's Dictionary.", "Testing that the __str_

    def test_prompt_two_books(self):
        example_ints = Library("Ugli")
        self.assertEqual(example_ints.books, {}, "Testing whether the books attribute is set correctly. Expecting an empty diction

    def test_prompt_two_add_book(self):
        example_inst = Library("Ugli")
        book_example = Book("Webster's Dictionary", "Merriam-Webster")
        example_inst.add_book(book_example)
        self.assertEqual(len(example_inst.books), 1, "Testing that when add_book is called, that something is added to the books a
        self.assertEqual(example_inst.books, {('Merriam-Webster', "Webster's Dictionary"): {'checked_out': 0, 'available': 1}}, "T

    def test_prompt_three_book_is_available(self):
        example_inst = Library("Ugli")
        book_example = Book("Webster's Dictionary", "Merriam-Webster")
        example_inst.add_book(book_example)
        self.assertTrue(example_inst.can_check_out(book_example), "Testing that when the book is available, that True is returned"
    def test_prompt_three_book_is_not_available(self):
        example_inst = Library("Ugli")
        book_example = Book("Webster's Dictionary", "Merriam-Webster")
        example_inst.add_book(book_example)
        example_inst.books[("Merriam-Webster", "Webster's Dictionary")]["available"] = 0
        example_inst.books[("Merriam-Webster", "Webster's Dictionary")]["checked_out"] = 1
        self.assertFalse(example_inst.can_check_out(book_example), "Testing that when the book is not available, that False is ret
    def test_prompt_three_book_is_not_there(self):
        example_inst = Library("Ugli")
        book_example = Book("Webster's Dictionary", "Merriam-Webster")
        self.assertEqual(example_inst.can_check_out(book_example), "The book you are looking for has not been added to this librar

    def test_prompt_four_len(self):
        self.assertEqual(len(hatcher.books), 4, "Testing that the number of books in hatcher is accurate")
    def test_prompt_four_contents(self):
        self.assertEqual(hatcher.books[('Darrell Huff', 'How to Lie with Statistics')]['available'], 2, "Testing that the book 'Ho



unittest.main(argv=['first-arg-is-ignored'], exit=False, verbosity=2)
```

```
test_prompt_four_contents (__main__.MyTests) ... ok
test_prompt_four_len (__main__.MyTests) ... FAIL
test_prompt_one_attributes (__main__.MyTests) ... ok
test_prompt_one_method (__main__.MyTests) ... ok
test_prompt_three_book_is_available (__main__.MyTests) ... ok
test_prompt_three_book_is_not_available (__main__.MyTests) ... ok
test_prompt_three_book_is_not_there (__main__.MyTests) ... ok
test_prompt_two_add_book (__main__.MyTests) ... ok
test_prompt_two_books (__main__.MyTests) ... ok

======================================================================
FAIL: test_prompt_four_len (__main__.MyTests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-81-4701d543309e>", line 40, in test_prompt_four_len
    self.assertEqual(len(hatcher.books), 4, "Testing that the number of books in hatcher is accurate")
AssertionError: 3 != 4 : Testing that the number of books in hatcher is accurate

----------------------------------------------------------------------
Ran 9 tests in 0.008s

FAILED (failures=1)
```

Out[81]: <unittest.main.TestProgram at 0x7f44c027ac50>

In [ ]: