CSCI 195 – Intro to Programming w/Python
Parsing the Bible Text into a Python data structure

We have been given a version of the King James Bible whose contents have been encoded using a simple text format. The following is a sample of the file's contents.

```
Ge@1:1@In the beginning God created the heaven and the earth.
Ge@1:2@And the earth was without form, and void; and darkness was upon the face of the deep. And the Spirit of God moved upon the face
of the waters.
…
Ge@1:31@And God saw every thing that he had made, and, behold, it was very good. And the evening and the morning were the sixth day.
Ge@2:1@Thus the heavens and the earth were finished, and all the host of them.
…
Ge@50:26@So Joseph died, being an hundred and ten years old: and they embalmed him, and he was put in a coffin in Egypt.
Exo@1:1@Now these are the names of the children of Israel, which came into Egypt; every man and his household came with Jacob.
…
```

As you can see, each line of the file contains a single verse; at the beginning of the line is an abbreviation of the book, chapter and verse, encoded in the format **Book@Chapter:Verse Number@Verse Text**.

Our goal will be to create a **dictionary**, with the keys being strings representing the book. Each entry in the dictionary will be a list, with entry `c-1` in the list representing chapter `c` (e.g., entry 0 represents chapter 1, entry 1 represents chapter 2, and so on). The **contents** of each chapter will be a list of strings, containing the text of each verse. As with the chapters, entry `v-1` contains the text of verse `v`.

1. Using `{ }` for dictionary, and `[ ]` for list, show what the contents of our dictionary will be after the first two verses of Genesis 1 have been read and parsed by our program.

2. Write a statement that declares an empty dictionary named `bible`.

3. Write a statement that opens the file named **kjv.atv** using a `with` statement, and then iterates over each line in that file with a `for` loop. Use the `bible_file` as the variable for the opened file, and `line` as the loop variable.

4. Assume that `line` stores a single line from the file. Write a series of statements that (1) strips off any trailing whitespace from `line`, storing the stripped text back in `line`; (2) splits the variable `line` on the @ sign, storing the result into a variable named `parts`; and (3), stores the components of `parts` into variables named `book`, `reference` and `verse_text`, in that order.

5. Assuming that `reference` stores a chapter and verse combination (such as 1:1 or 2:50), write a series of statements that (1) splits `reference` based on the **:** character, storing the result into a variable named `parts`; (2) uses the `int` function to convert the *first* entry in `parts` into an integer, storing the result into a variable named `chapter`; and (3) uses the `int` function to convert the *second* entry in `parts` into an integer, storing the result into a variable named `verse`.

6. Write an `if/else` statement that checks to see if there is an entry in the dictionary `bible` for the string variable `book`. Remember that you can use the `in` operator to check if a particular key exists in a dictionary.
   - If `book` **does exist** as a key in the `bible` dictionary, set the variable named `book_chapters` to be the value in `bible` associated with the key `book` (the variable, not a literal string).
   - If `book` **does not exist** as a key in `bible`, (1) set a variable `book_chapters` to be an **empty list**, and then store `book_chapters` as the value for the key `book` in the `bible` dictionary.

7. Write an `if/else` statement that checks whether the number of entries in the list named `book_chapters` is **at least as big** as the variable `chapter`, which we assigned a value to in step 5. Fill in the body of the `if/else` according to the following logic:
   - If the test is **true**, we know that there is already an entry in `book_chapters` corresponding to the chapter represented by `chapter`, and so we should append the variable `verse_text` to the list associated with `chapter` (remember chapter numbers are 1 based, but list indexes are 0 based)
   - If the test is **false**, this is the first verse we've seen from this chapter. So we need to add a new list containing only `verse_text` to the end of the `book_chapters` list.

8. Take the code from the previous steps and enter it into a new file named **parse_bible.py** in the in-class-exercise folder named **bible_parser** (Start a new file by clicking the New button from the Files tab, enter the name **parse_bible.py** and then press Enter).

   Add a print statement at the end as follows:

   ```
   print bible["Ge"][0][0]
   ```

9. Execute the program, checking for any syntax errors:

   ```
   python parse_bible.py
   ```

   If all goes well, you should see the contents of Genesis 1:1. If you don't, check the logic of each of the statements that you've entered. You can use

   ```
   python –m pdb parse_bible.py
   ```

   to start the program in a *debugger*. You can then use `n` followed by `Enter` to execute the program one line at a time, and use statements like `print book` to see the values of variables.

10. Add a second print statement to print out the contents of the last verse:
    ```
    print bible["Rev"][21][20]
    ```
    and ensure the output is as expected

11. Now that we know our data structure is complete, we'll add some interactivity. First, write a statement that asks the user to enter a book in the bible and stores the user's response as a variable named `book`, and then sets the variable `book_chapters` to be the list of chapters in the dictionary `bible` associated with the book entered by the user.

12. Assume the variables `book` and `book_chapters` has been set using the code you wrote above. Determine what the following series of statements does:

    ```
    list = []
    for c in book_chapters:
      list.append(len(c))

    print "{0}: {1}".format(book, max(list))
    ```

13. Consider the following "transcript" of an interactive session, with user input shown in **bold**:

    ```
    python parse_bible.py
    Enter the desired book: John
    Enter the desired chapter [1 - 21]: 3
    Enter the desired verse [1 - 36]: 16
    John 3:16 For God so loved the world, that he gave his only begotten
    Son, that whosoever believeth in him should not perish, but have
    everlasting life.
    ```

    Add code at the end of **parse_bible.py** to implement the functionality shown above. Make note of how the program prompts the user with the appropriate limits for chapter number and verse number.

14. Add code to the end of your program that prints out the name of the book containing the **fewest** chapters. Here's a process you can use to figure this out:

    Create a variable named `fewest_chapters` and set its value to the **length** of the list associated with the string key `Ge` (it actually doesn't really matter which book you pick here)

    Create a second variable named `book_with_fewest_chapters`, and set it to the string value `Ge` (must be the same as what you picked above).

    ```
    Loop through the keys in the bible dictionary; each time through the loop:
      Compare the length of the list associated with the current key to fewest_chapters
      If that length is less than the value of fewest_chapters,
        update the values of both fewest_chapters and book_with_fewest_chapters
    ```

    After the loop is done executing, print out the values of `book_with_fewest_chapters` and `fewest_chapters`.

    The correct answer is Obad (Obadiah) with a single chapter.

According to the web site bibleresources.org/how-to-study-bible, the "principle of first mention" indicates that:

> "It is important to look for the place in the Bible that a subject, attitude or principle is mentioned for the first time, and see what it meant there."

We'll want to modify our program so that it asks the user to enter a word, and the reference and first where that word first occurs is printed. For example:

```
Enter a word you wish to find the first occurrence of: grace
First mention of grace is in Ge 6:8
```

15. In order to do this, we're going to have to keep track of the ordering of the books. Rather than entering this information ourselves, we can keep track of it while we parse the bible. To do this:
    - Initialize an empty list named `books` before opening the file *kjv.atv*
    - When you come across a book that is not yet in the dictionary named `bible` , append it to `books`
16. Add the appropriate `raw_input` statement to the end of your program to ask the user for the word to search for.

17. Add code to find the first occurrence of the given word. I did this by writing 3 nested loops:

```
Loop over each book in the books list
  Loop over the chapters in the current book
    Loop over the verses in the current chapter
      Test to see if the desired word is contained within the current verse, using the in operator
```

I used variables `found` (Boolean, initially False and set to True when the desired word has been found), `chapter_num` and `verse_num` to help me in my implementation