Goals for this exercise

Thanks for trying out our take-home exercise! There aren't any gotchas or brain teasers -- we're primarily looking to get a sense for how you solve problems with code that:

- is readable
- works correctly
- is well structured / maintainable
- anticipates and handles edge cases
- leverages capabilities of language and libraries

We don't want or expect folks to spend a huge amount of time on the the exercise: ideally two to three hours. To that end, focus on first getting something roughly working, then polish and expand after that's in place with time you've got left.

If you find that it's taking you significantly more than two to three hours to complete, leave some notes about what's left to finish in your Readme and send along what you've got.

Be aware that we'll potentially build on your solution in later interview rounds, so designing your program to allow for future modifications is a good idea.

If you have any questions about the problem itself or logistics for the exercise, please email the recruiting coordinator you've been in touch with. Note that we won't be able to offer technical advice or hints.

To consider before you start

We'll get to the actual problem in a moment, but let's first highlight some things to consider that might affect your approach.

Your solution should be a command-line application

We ask you to implement your program as a command-line application. That means that it should run and be interacted with from a command prompt. Don't build a webapp or other GUI, or rely on IDE configuration or functionality for running your application.

We should be able to build and run your solution

In addition to reviewing the code, we'll build and run your program to test it out. To accomplish this we'll rely on the instructions you give in your Readme. We should be able to follow the instructions on an average Linux/Mac command line, or from the WSL in Windows.

If you use a compiled language please provide the source and any build instructions rather than including a pre-compiled executable.

You should pick an appropriate language

Speaking of languages, we recommend picking a language that best showcases your experience and skills. Try to avoid choosing a language that you haven't worked with much before.

We also recommend choosing a language that will make doing things like working with lists and strings straightforward. Some examples include Python, Ruby, Javascript/Node.js, and Go -- but anything else that accomplishes those goals is also fine.

Lower level languages might prove challenging, especially if we extend your solution in later interview rounds. For that reason, we generally recommend not using languages like C, or C++. We've found that people's experience using Java can be mixed, if you have

significant real-world experience with it you'll likely be fine, but we recommend looking for an alternative if you've had primarily academic experience with it.

You will need to use libraries

We expect and encourage you to use libraries when implementing your solution. That could be either standard libraries for your language or external dependencies like python packages, node modules, etc. If using external dependencies, please use the standard mechanisms for your language, and include any needed installation instructions in your Readme.

When you're done

Once you're ready to send us your solution…

Make sure it works

Especially towards the end of your work, it's easy to make last minute changes that introduce problems, so once you're "pencils down" make a point of double checking that your program works correctly and matches the sample output provided.

Also double-check that any build/run instructions are correct. If we run into any problems while testing your program out, we'll reach out for clarification, but this will delay our ability to process your application.

Don't share or publish

We want each candidate to have a chance to show us their best work, so please don't share your solution with others or publish it anywhere public like github.

Send it in!

Create a zipfile with your code, your Readme, and anything else needed to build and run the program — then send it in based on the instructions you received.

Please don't include your name in your Readme, source, or your zipfile's filename. We'll take a look and should get back to you shortly!

The problem

We'd like to build a tool to help the BookBub editorial team classify books by genre. To do this we'll calculate a score based on how often genre-related words or phrases occur in the book's description.

Let's assume that we have two kinds of data files available: a list of books, and a list of keyphrases.

Book List

The book list will be JSON-formatted and include a **title** and a **description**. For example:

```
[
     {
          "title": "The Cat in the Hat",
          "description": "This bestselling thriller has
     captivated millions, and teaches the importance of
     listening to fish."
     },
     {
          "title": "The Hunger Games",
          "description": "A charming cookbook that helps make
     beating hunger fun!"
     }
]
```

Keyphrase List

This will be CSV-formatted, and gives a point value to pairs of genres and words/phrases that might indicate a book is in a given genre. Note that multiple keyphrases can be included for each genre, and some keyphrases might occur for multiple genres. For example:

```
Genre,Keyword,Points
action,katana,6
action,sweet motorcycle,4
biography,mists of time,2
biography,born and raised,3
how to,sweet motorcycle,6
```

Calculating a score

We'll use the following to calculate "genre scores" for each book:

- count the total number of times any keyphrase for a genre occurs in that book's description, we'll call this **n**
- take the average of the point scores of the *unique* keyphrases occuring in that book's description, we'll call this **k**
- a book's score for a genre is then: **n * k**

Note that keyphrases can contain spaces, so "sweet motorcycle" would match "Jane rode a sweet motorcycle" but not "It would be sweet to ride a motorcycle".

Example calculation

Let's say we have the following book and keyphrase data:

```
[
    {
        "title": "Harry Potter",
```

```
        "description": "The story of a hat with a grudge that
    makes a school full of children's lives miserable.  Will
    the lovable groundskeeper be able to keep the school open?
    Find out!"
    }
]
```

```
Genre,Keyword,Points
action,hat,6.5
action,school,4
biography,groundskeeper,2
biography,born and raised,3
```

To determine the "action" score for the book, we would do the following:

- action-related keyphrases occur **3** times in the description
- both the "hat" and "school" keyphrases matched, so we take the average of their point values: **5.25**
- so the total score is then: **3 * 5.25** -> **15.75**

Requirements

Your task is to create a program that does the following:

- runs from the command-line, taking two arguments for the book list file and keyphrase file to use — please use arguments to the command itself rather than prompting the user for input
- takes each book present in the data file, then prints a genre score summary for each book, with the books listed alphabetically by title
- the genre score summary should include the genre and calculated score, ordered by score, for up to three of the highest-scoring genres for the book

To test your program, you can run it against these sample files:

- Book list

- Keyphrase list

It should then print the following output:

```
Hunger Games
action, 15
sci-fi, 8

Infinite Jest
literary fiction, 12
```

Please include a Readme that describes:

- all necessary steps to build and run your program from your source code, including how to install any needed libraries or dependencies
- the version of the language you used, and of any required tools or commands, IE Python 3.7.6
- any notable/interesting tradeoffs or edge cases that you ran into
- any functionality that isn't yet complete, or other future refinements you'd like to make
- approximately how long you took to complete the exercise

When evaluating your solution, we'll use other book/keyphrase files, so make sure it works generically and not just for the samples above.

That's it, good luck and we look forward to hearing back from you!