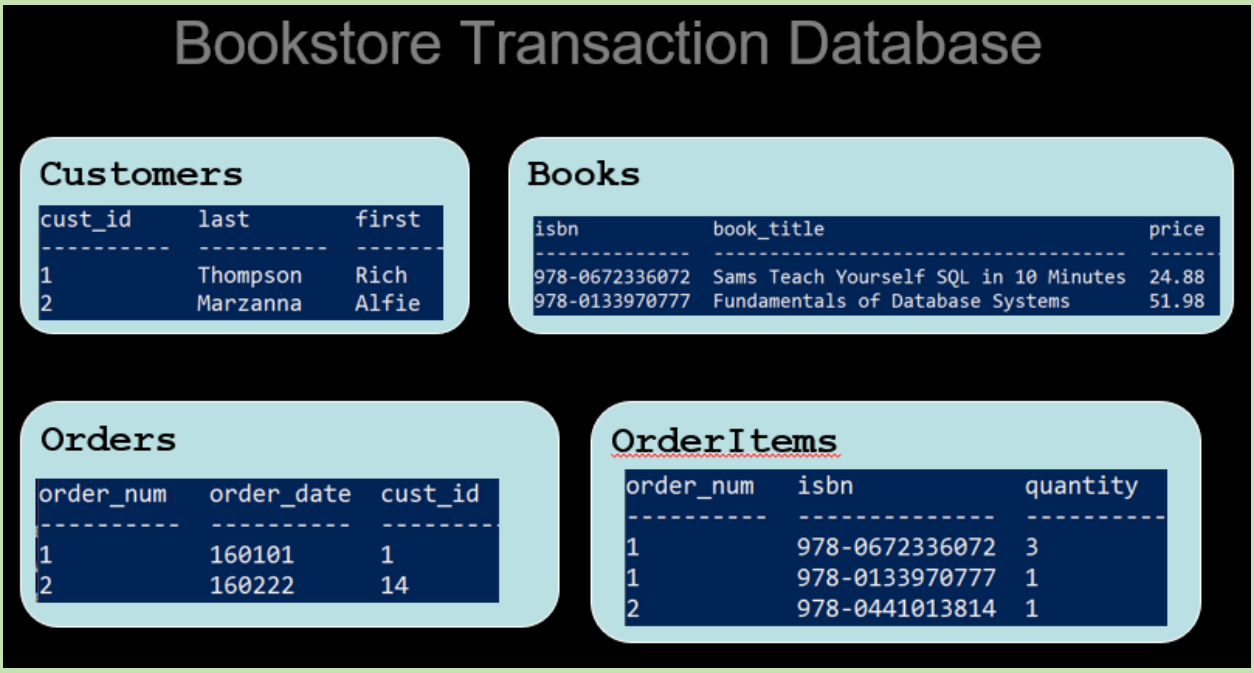
A Simple Recommender System
ISTA 331 Hw1, Due Thursday 2/6/2019 at 11:59 pm

Introduction. This homework will introduce you to the paradigm-altering concept of recommender systems. Recommender systems enable targeted advertising, which has changed the face of business across the planet and made Amazon into a global hegemon. This homework uses item-based collaborative filtering, which is vastly faster than user-based, which means it's vastly better.

This homework will also refresh your SQL. I have created two bookstore transaction databases, `bookstore.db` and `small.db`. `bookstore.db` has real ISBN's, real titles, and real prices taken from Amazon. `small.db` is a much tinier version with shorter, fake ISBN's so that the illustrations in this spec are comprehensible. Both have the following schema:



`order_date` and `isbn` are `TEXT`, everything else is what it looks like. Primary keys are the leftmost columns, except for the `OrdersItems` table, which has the leftmost two columns combining to form the primary key.

Instructions. Download and unpack `hw1.zip`. `hw1_partial.py` has some code that will help you see what you're your code is doing and show you what you have created when you're done. Rename it as `hw1.py` and put your code in this module. Below is the spec for 9 functions. Implement them in the provided starter module and upload your final product to the D2L Assignments folder.

Testing. Run `hw1_test.py` from the command line to see your current correctness score. Each of the 9 functions is worth 11.1% of your correctness score. You can examine the test module in a text editor to understand better what your code should do. The test module is part of the spec. The test file we will use to grade

your program will be different and may uncover failings in your work not evident upon testing with the provided file. Add any necessary tests to make sure your code works in all cases.

Documentation. Your module must contain a header docstring containing your name, your section leader's name, the date, `ISTA 331 Hw1`, and a brief summary of the module. Each function must contain a docstring. Each function docstring should include a description of the function's purpose, the name, type, and purpose of each parameter, and the type and meaning of the function's return value.

Grading. Your module will be graded on correctness, documentation, and coding style. Code should be clear and concise. You will only lose style points if your code is a real mess. Include inline comments to explain tricky lines and summarize sections of code.

Collaboration. Collaboration is allowed. You are responsible for your learning. Depending too much on others will hurt you on the tests. "Helping" others too much harms them in reality. Cite any sources/collaborators in your header docstring. Leaving this out is dishonest.

Resources.
https://www.sqlite.org/index.html
https://docs.python.org/3/library/sqlite3.html
https://en.wikipedia.org/wiki/Netflix_Prize
http://pandas.pydata.org/pandas-docs/stable/

These are the complete tables in the small database:

```
sqlite> SELECT * FROM Books;
isbn        book_title                            price
----------  ------------------------------------  ----------
971         Sams Teach Yourself SQL in 10 Minutes  24.88
972         Fundamentals of Database Systems       51.98
973         Python Data Science Handbook: Essenti  28.12
974         Python More                            28.12
```

```
sqlite> SELECT * FROM Customers;
cust_id     last        first
----------  ----------  ----------
1           Thompson    Rich
2           Marzanna    Alfie
3           Knut        Dan
```

```
sqlite> SELECT * FROM OrderItems;
order_num   isbn        quantity
----------  ----------  ----------
1           971         3
1           973         1
2           971         1
2           972         1
2           974         1
3           971         1
3           974         1
```

```
sqlite> SELECT * FROM Orders;
order_num   order_date  cust_id
----------  ----------  --------
1           160101      1
2           160222      2
3           160224      3
```

Function specifications.

`get_purchase_matrix`: This function takes a `connection` object to a bookstore transaction db and returns a dictionary that maps customer id's to a sorted lists of books they have purchased (i.e. ISBN's) without duplicates. It must be sorted for the test to work. For the small db, it looks like this:

```
******************** Purchase Matrix ********************
{1: ['971', '973'], 2: ['971', '972', '974'], 3: ['971', '974']}
```

You will need a query to get the customer id's. In a loop, you will add each customer to the dictionary you are building, using a query to get the ISBN's of the books that they have purchased. We will talk about this query in class.

`get_empty_count_matrix`: This function takes a `connection` object to a bookstore db and returns a `DataFrame` with index and columns that are the ISBN's of the books available in the bookstore and `int 0's` for data:

```
******************** Empty Count Matrix *****************
     971  972  973  974
971    0    0    0    0
972    0    0    0    0
973    0    0    0    0
974    0    0    0    0
```

`fill_count_matrix`: This function takes an empty count matrix and a purchase matrix and fills the count matrix. Go through each customer's list of ISBN's. For each ISBN in the list, increment the appropriate spot on the diagonal. We are keeping track of the number of users who have purchased each book on the diagonal (we have already ignored any additional purchases of the same book by the same customer). For each pair of books purchased by the same customer, increment the appropriate positions in the matrix. For instance, customer 2 bought both 971 and 972. Therefore, increment both `matrix.loc['971', '972']` and `matrix.loc['972', '971']`. These off-diagonal positions hold the number of times both books represented by the position's row and column labels have been purchased by the same person. We are keeping the matrix symmetric because we have enough to think about as it is. We could store the info in a triangular matrix, but forget that.

```
******************** Full Count Matrix ******
     971  972  973  974
971    3    1    1    2
972    1    1    0    1
973    1    0    1    0
974    2    1    0    2
```

Try to reconstruct the count matrix from the above purchase matrix by hand until you understand what's going on.
`make_probability_matrix`: This function takes a count matrix and returns a conditional probability matrix. The value at each position `[row_isbn, col_isbn]` is the probability that a customer has purchased the column book given that that customer has purchased the row book. We calculate this probability `cm.loc[row_isbn, col_isbn] / cm.loc[row_isbn, row_isbn]`. I did it using integer indices instead of row and column labels, but it will work either way. Set the diagonal terms to `-1`. This ensures that when we make a list of most similar books to a given book, that the given book won't appear in the list.

```
******************** Probability Matrix ******
     971       972       973       974
971 -1.0  0.333333  0.333333  0.666667
972  1.0 -1.000000  0.000000  1.000000
973  1.0  0.000000 -1.000000  0.000000
974  1.0  0.500000  0.000000 -1.000000
```

`sparse_p_matrix`: This function takes a probability matrix and returns a dictionary that maps ISBN's to a list of the 15 books most likely to be purchased by a customer who has purchased the key book in descending order of likelihood. I went across each row of the p-matrix and made a list of 2-element lists, the first inner element being the probability, the second being the associated column book, and sorted the list, setting the keyword argument `reverse` to `True`. Then I made a list of the second elements in each of the inner lists and boom.

```
******************** Sparse Probability Matrix ********************
{'971': ['974', '973', '972', '971'], '972': ['974', '971', '973', '972'],
'973': ['971', '974', '972', '973'], '974': ['971', '972', '973', '974']}
```

`get_cust_id`: This function takes a `connection` object and returns an integer customer id or `None`, depending upon user input. Grab the customer info from the db and print it as in this example:

```
CID       Name
-----     -----
    1     Thompson, Rich
    2     Marzanna, Alfie
    3     Knut, Dan
---------------
Enter customer number or enter to quit:
```

The `"Enter customer…: "` is a prompt passed to the `input` function. There is a space after the colon.

`purchase_history`: This function takes a customer id, a list of ISBN's that the customer has purchased, and a `connection` to the db, and returns a string containing the customer's purchase history as titles instead of ISBN's. The string ends with a newline character. It prints like this:

```
Purchase history for Alfie Marzanna
-----------------------------------
Sams Teach Yourself SQL in 10 Minutes
Fundamentals of Database Systems
Python More
----------------------------------------
```

There are 40 dashes in the last line. Each title is truncated at 80 characters if necessary.

`get_recent`: This function takes a customer id and a `connection`. It returns an ISBN chosen randomly from the customers most recent order. Make a list of the ISBN's in the most recent order and use `random.randrange` to randomly index into the list to grab a book.

`get_recommendation`: This function takes a customer id, a sparse probability matrix, purchase history ISBN list, and a `connection`. Use the previous function to randomly grab a book from the customer's most recent purchase. Get the customer's name. Get the two books most similar to the recently purchased book, not including any books already purchased by the customer. We don't want to recommend a book that we have already sold the customer. That would make us look stupid. If there is only one book in our 15 most likely books that the customer hasn't purchased, recommend only one. If there are none, recommend `"Out of ideas, go to Amazon\n"`. Return your recommendation as a string that prints as such:

```
Recommendations for Alfie Marzanna
----------------------------------
Python Data Science Handbook: Essential Tools for Working with Data
```

If the title is too long, truncate it at 80 characters.