3.1 Database Basics

3.1 Database Basics

**You may attempt this section on the Jupyter notebook:
https://colab.research.google.com/drive/1PZbFaHDU_pk0yJsj7SYwQ6LHfOD290IN#scrollTo=MFxpU7nNr4qU

Databases are organised collections of data, this allows them to be displayed, maintained and searched easily. Our database will have 1 table, effectively just like a spreadsheet table. The headings on each of the columns are the fields, the individual pieces of data we want to store about the books in our collection. The information about a single book are called its attributes and are stored toegther in one record, which would be a single row in our database table. To make it easier to search and sort our database, we should also select a primary key, one field that will be unique for each book. Sometimes one of the fields we are already storing works for this purpose, if not then the database will create an ID number that it uses to uniquely identify each record.

The requirements for our database are that it can do the following things; Save data to a file, read data from that file, create new books, display our full database, allow the user to enter a search term and display a list of relevant results based on that term.

We can decompose the problem into the following steps:
1. Set up our structures
2. Create a record
3. Save the data to the database file
4. Read from the database file
5. Display the database to the user
6. Allow the user to search the database
7. Display the results

We want to save our data to the text file, as if it was a table. So each line will be a comma separated list of the values in one of our book records. Apart from the first line in the file, which will contain our field labels. There is a special file type for this, called CSV (comma separated values), that can be read using Word or Excel (and equivalents).

Import Statements

Should import the external modules we need to run the project first.

```
import csv
```

Global Variables

Should initialize the global variables (i.e variables used by many finctions or modules)
Following are the variables along with their purposes
1. current_ID : a variable to count up as we create records, this will be our primary key
2. new_additions : a list to hold any new records we make while our code is running, before we save them to the file
3. fields : a list of our fields, so that our dictionaries can be aligned with our text file
4. data : a list that will hold ALL of the data from the database, so that we can search and display it without having to read the file every time.

```
current_ID = 1
new_additions = []
filename = "library.csv"
fields = ['ID', 'Title', 'Author', 'Genre', 'Year', 'Location']
data = []
```

CSV File

Read the exisiting data stored in the csv file for further usage. DictReader will take the comma seperated line and produce a dictionary for us. Then we need to make sure our current_ID is the one after the last line of our database (remember, we don't want duplicates)

```
with open(filename, 'r') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        data.append(dict(row))
if len(data) > 0:
    current_ID = data[:-1][0].get("ID")
    current_ID = int(current_ID) + 1
```

Function to add a record

We are going to use dictionaries to store our records. They reference their elements using keys instead of indexes, which fit our database fields nicely. We are going to generate our own IDs. Each of these must be unique, so a variable is needed that we can add to as we make our records. This is a user focussed application so let's make it so our user can input the data for the first book. The strings, in quotes, on the right of the colon are the keys (the names of our fields) and the data on the right is the stored value, in our case whatever the user inputs in response to our appropriate prompts.

```
def addRecord():
  global current_ID
  new_record = {
      "ID": str(current_ID),
      "Title": input("What is the title of the book? >"),
      "Author": input("What is the author's first name?  >"),
      "Genre": input("What genre is the book?  >"),
      "Year": input("What year was the book released?  >"),
      "Location": input("Where is the book? > ")
  }
  new_additions.append(new_record)
  current_ID = current_ID + 1
  print("-"*15)
  print("New Record added successfully!")
```

Function to display the data

We need to display the fields first, then the data from our database file and finally any new additions that have been made since the program has been running. There are 2 oddities in what we do to display the data, one is that because all of our data is stored in dictionaries, we need to print the value and not the key. Also we want to add space to our outputs so that it aligns like a table.

```
def displayData():
  for item in fields:
      print("%-25s"%item, end='')
  print("\n")
  for row in data:
      for key, val in row.items():
          print("%-25s"%val, end='')
      print("\n")
  for row in new_additions:
      for key, val in row.items():
          print("%-25s"%val, end='')
      print("\n")
```

Function to Search and display data

The last bit of essential functionality required for our database is to allow the user to search it. There is lots of room for differentiation here, you can set some to the task of allowing users to search by certain fields, to challenge more competent students, or implement the more simple search done in this code. Using a new array to store any results, we search through both the data from the file and any new additions that have been made since our program started and if the search string is found anywhere in the values of that dictionary, it will be added to our results.

```
def searchData() :
  #Search the data
  search_term = input("What would you like to search for?").lower()
  results = []
  for row in data:
      for key, val in row.items():
          if search_term in val.lower():
              results.append(row)
  for item in new_additions:
      for key, val in item.items():
          if search_term in val.lower():
              results.append(item)
  #Display the results
  if len(results) > 0:
      for item in fields:
          print("%-25s"%item, end='')
      print("\n")
      for item in results:
          for key, val in item.items():
```

```
        for key, val in item.items():
            print("%-25s"%val, end='')
        print("\n")
else:
    print("Sorry no records found")
```

Execute the code

```
print("------Welcome to the Python Library organiser------")
choice = ""
while choice.lower() != "x":
    print("""What would you like to do?
    1 - Add a book
    2 - Display your Books
    3 - Search for a Book""")

    choice = input(">")

    if choice == "1":
        #Code to add a record
        addRecord()

    elif choice == "2":
        #Display the data
        displayData()

    elif choice == "3":
        #Search the data
        searchData()


    elif choice.lower() == "x":
        print("Thank you! Shutting down.")
    else:
        print("Sorry, I didnt recognise that option")
```

Save the newly added record in the CSV

To store the new records, we check if there is something in the new_additions list, this is for later when it will be possible to run your program without adding a book. We then open the csv file, by calling the open function with the filename variable we set up a moment ago. The second argument is a character, and this is to choose which mode to open in. There are 3 main modes for interacting with files; read, write and append. We want to append as any new data should go on to the end of the file. Append means 'add at the end', just like an appendix goes at the end of a book. We then make a csv writer, in this case one that specifically works with dictionaries. The CSV writer does things like adds commas and matches the fields to the dictionary keys. To be able to do this, it needs to know what file it is using, and also what fields it has. We then iterate over our new additions list and write each new item to our database file. The writer does all the finnicky parts, we just hand it a dictionary and it will create a row for us.

```
if len(new_additions) > 0:
    with open(filename, 'a') as csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=fields)
        writer.writeheader()
        for item in new_additions:
```