5. [SOLUTIONS] Final Long Questions

Library System

You are to develop a database for a library. The library has a collection of books, which are described by the following data:

- Title
- Author(s)
- Genre(s)
- ISBN, a unique 13-digit number which identifies the book
- Publication date

The library also has some data about the books for their own information, including:

- Borrow status, indicating whether a book is borrowed or not
- Borrow date, if borrowed
- Return date, if borrowed

1. Design a data structure for a book. Also consider how you would represent a collection of books which would represent the library's stock. Assume that this collection is a global variable, available to all other functions which will be written.

[4 marks]

```
book = {
  "title": "To Kill a Mockingbird",
  "author": "Harper Lee",
  "genre": ["fiction", "american"],
  "isbn": 9789023493617,
  "publicationDate": 1960,
  "borrowed": True,
  "borrowedStart": "01/05/2019",
  "borrowedEnd": "15/-5/2019"
}

# There are of course other ways of doing this. For example, nesting the borrowing-related statuses in an internal dictionary, or nesting all the book details inside a "book" parameter at the root with the borrowed statuses
# To store multiple books, considering the diversity of operations we will need to perform on them, the most likely data type would be a list of dictionaries
```

2. The library staff often want to check their stock. To do so, they will want to review the collection of books in their database ordered by different fields. Write a function called `bookSort(fieldName)` that takes a field name and sorts the collection of books by that field. Possible fields are title, authors (just sort by the first author) and ISBN. The function should return a collection sorted by the given keyword.

[6 marks]

```
def bookSort(fieldName):

  # Sorting helper function
  def sortFunction(book):
    return book[fieldName]

  possibleFields = ["title", "author", "isbn"]
  if fieldName in possibleFields:
    outputList = sorted(books, key=sortFunction)
```

```
  return outputList
```

3. The database allows users to search for particular books. They can search for books which match a title, author, ISBN or publish date. Write a function `bookSearch(value)` where `value` is the search term. The function should return all books where any of the fields above matches the search term.

[4 marks]

```
def bookSearch(value):
  foundBooks = []
  possibleFields = ["title", "author", "genre", "isbn", "publicationDate"]
  for book in books:
    for field in possibleFields:
      if value in book[field] and book not in foundBooks:
        foundBooks = foundBooks + [book]
  return foundBooks
```

4. The library collects the following information on its clients:
    - Library card number, a unique 8-digit number
    - Name
    - Address
    - Books currently borrowed
    - Number of books allowed to borrow (clients who have been registered with the library for a longer period of time are allowed to borrow more books after set periods)

   Assume a date is a previously defined data structure with the following associated functions:
    - `compare(date1, date2)`, which takes two dates and returns:
        - 1 if `date1>date2`
        - 0 if `date1==date2`
        - -1 if `date1<date2`
    - `difference(date1, date2)`, which calculates the number of days occurring between date1 and date2 (if date2 is prior to date1, the number will be negative)
    - `add(date, days)`, which calculates a date given a starting date and a number of days to add to it

   Write a function called `borrow(client, book, date)` which allocates the book to a user, updates the borrow status, sets the borrow date and calculates a return date which is two weeks from the borrow date. No books should be allocated if the book is already borrowed or if the user has already borrowed the maximum number of books. Consider how client data is likely to be stored.

[6 marks]

```
# Client data - multiple clients can be sorted in a list of dictionaries or a dictionary of
dictionaries with the library card number as the key for each user
# <<book>> represents a book data structure, as above. This could also be the ISBN
client = {
  "libraryNumber": 12345678,
  "name": "John Doe",
  "address": "555 Nowhere Street",
  "books": [<<book>>, <<book>>],
  "booksAllowed": 3
}

def borrow(client, book, date):
  if book["borrowed"] is False and len(client["books"]) < client["booksAllowed"]:
    book["borrowed"] = True
    book["borrowedStart"] = date
    book["borrowedEnd"] = date.add(14)
```

```
    client["books"] = client["books"] = [book]
```

5. Clients who do not return books within the allocated time will face fines, which are £2 per day overdue up to a maximum of £250 per book. Write a function `fines(user, date)` to calculate how much a client owes up to the date specified based on all the books they have borrowed. Note that it could be £0.

[3 marks]

```
def fines(user, date):
  fines = 0
  for book in user["books"]:
    if compare(date, book["borrowedEnd"]) < 0:
      fine = difference(date, book["borrowedEnd"]) * 2
      if fine >= 250:
        fines = fines + 250
      else:
        fines = fines + fine
  return fines
```

6. Write a function called `return(client, book, date)` which does the inverse of `borrow()` and also displays to the librarian whether the client owes fines (and if so, how much).

[3 marks]

```
def return(client, book, date):
  book["borrowed"] = False
  client["books"] = client["books"].remove(book)
  print("This client owes (in £):")
  print(fines(client, date))
```

7. Sometimes a client forgets which books they've borrowed. They can come into the library and search for their library card number to display all the information stored on them, including the books they've borrowed. Write a function `clientSearch(id)` which takes a library card number and prints all the information sorted on that client. For the books, all the information, including the associated dates, should also be displayed.

[3 marks]

```
def clientSearch(id):
  for client in clients:
    if client["libraryNumber"] == id:
      for book in client["books"]:
        print("Title: " + book["title"])
        print("Author: " + book["author"])
        print("Genre: " + book["genre"])
        print("ISBN: " + book["isbn"])
        print("Published: " + book["publicationDate"])
        print("Borrowed: " + book["borrowedStart"])
        print("Return due: " + book["borrowedEnd"])
```

8. The library staff need to keep on top of all the books which are loaned out. Write two functions, `booksInCirculation()` and `booksOverdue(date)` which each print out a collection of books and the library card number of the client associated with them. `booksInCirculation()` should print information for all currently borrowed books, `booksOverdue()` should print information for all currently borrowed books which are overdue according to the date specified.

[6 marks]

```
def booksInCirculation():
  for client in clients:
    if client["books"] != []:
      print("Client Card ID: " + str(client["libraryNumber"]))
```

```
      for book in client["books"]:
        print("Title: " + book["title"])
        print("Author: " + book["author"])
        print("Genre: " + book["genre"])
        print("ISBN: " + book["isbn"])
        print("Published: " + book["publicationDate"])

def booksOverdue(date):
  for client in clients:
    if fines(cleint, date) > 0:
      print("Client Card ID: " + str(client["libraryNumber"]))
      for book in client["books"]:
        if compare(book["borrowedEnd"], date) < 0:
          print("Title: " + book["title"])
          print("Author: " + book["author"])
          print("Genre: " + book["genre"])
          print("ISBN: " + book["isbn"])
          print("Published: " + book["publicationDate"])
```

9. Write a program for librarians to use which allows them to do the following:
    - Sort and search library stock
    - Issue and accept returns of book loans
    - List all books in circulation and all overdue books

   Consider how you are going to handle user input. The program should keep taking instructions until an exit parameter is provided.

[15 marks]

```
print("The following numbers correspond to instructions:")
print(" 0 - Quit")
print(" 1 - Sort books")
print(" 2 - Search books")
print(" 3 - Borrow a book")
print(" 4 - Return a book")
print(" 5 - List books in circulation")
print(" 6 - List overdue books")

instruction = int(input("Enter a number: "))

while instruction != 0:
  if instruction == 1:
    field = input("Provide a field to sort by: ")
    print(bookSort(field))
  elif instruction == 2:
    value = input("Provide a value to search for: ")
    print(bookSearch(value))
  elif instruction == 3:
    id = int(input("Provide a borrower's card ID: "))
    borrower = None
    for client in clients:
      if client["libraryNumber"] == id:
        borrower = client
    if borrower != None:
      isbn = int(input("Provide a book's ISBN: "))
      borrowedBook = None
      for book in books:
```

```
      if book["isbn"] == isbn:
        borrowedBook = book
    if isbn != None:
      date = input("Insert date: ")
      borrow(borrower, borrowedBook, date)
    else:
      print("Book not found")
  else:
    print("User not found")
    elif instruction == 4:
      id = int(input("Provide the returning client's card ID: "))
      returner = None
      for client in clients:
        if client["libraryNumber"] == id:
          returner = client
      if returner != None:
        isbn = int(input("Provide the book's ISBN: "))
        borrowedBook = None
        for book in returner["books"]:
          if book["isbn"] == isbn:
            borrowedBook = book
        if isbn != None:
          date = input("Insert date: ")
          return(returner, borrowedBook, date)
        else:
          print("Book not found")
      else:
        print("User not found")
elif instruction == 5:
  booksInCirculation()
elif instruction == 6:
  date = input("Insert date: ")
  booksOverdue(date)

print("The following numbers correspond to instructions:")
print(" 0 - Quit")
print(" 1 - Sort books")
print(" 2 - Search books")
print(" 3 - Borrow a book")
print(" 4 - Return a book")
print(" 5 - List books in circulation")
print(" 6 - List overdue books")

instruction = int(input("Enter a number: "))
```

Top Trumps

Top Trumps is a card game. Each pack of Top Trumps is themed - about cars, or dinosaurs, or airplanes, or TV shows, for example - and each card gives a picture and some numerical data about a single example of those things. So, a pack of Top Trumps about cars would have a different make/model of car on each card, and the data might include top speed, engine size, gross weight, 0-60mph time, and so on.

In a game of Top Trumps, all the cards are dealt out to the players. One player starts and each round takes the following form:

1. The player chooses what appears to be the best/strongest item of data from the top card.
2. He/she tells the other player both the data category and the value - e.g. top speed and 140mph
3. The player with the "best" value on that data category wins the round, and both cards are placed at the bottom of the winning player's hand of cards.
    a. The "best" is sometimes the largest - as in top speed - and sometimes the smallest - as in the best 0-60mph time.
4. The winning player is the one who starts the next round back at step one.

The game finishes when one player has all the cards.

In this question, we'll be considering a Top Trump game with:

- Two players
- A pack of 50 cards
- The theme is cars
- The data categories on each card include are as follows, where the "best" value for each category is noted:
    - Top speed (largest)
    - Engine size (in cubic centimetres) (largest)
    - Weight (smallest)
    - 0-60mph time (smallest)

The question consists of a number of parts.

1. Design Python data structures to represent a pack of Top Trump cards, containing all the necessary information to support playing the game. (Read the whole question before answering this question, so you are sure about what "playing the game" means - in particular, part 7.).

[5 marks]

```
# There are two main data structure
# The pack is a list of dictionaries, where each dictionary is one card
# A card dictionary has four integer entries:
#    "topSpeed", "engineSize", "weight", "timeTo60"
# Here is a small instance of a pack:

cards = [ { "topSpeed" : 100, "engineSize" : 3500, "weight" : 1300, "timeTo60" : 6 },
          { "topSpeed" : 92,  "engineSize" : 2400, "weight" : 1100, "timeTo60" : 7 },
          { "topSpeed" : 110, "engineSize" : 1750, "weight" : 900, "timeTo60" : 8 },
          { "topSpeed" : 80,  "engineSize" : 1100, "weight" : 1000, "timeTo60" : 9 }]

# A final data structure records whether the fields should be ordered in descending order,
# that is, the smaller a value, the earlier it would be in the ranking.

descending = { "topSpeed" : True, "engineSize" : True, "weight" : False, "timeTo60" : False
}
```

2. Assume you have a pack of 50 cards in your data format, held in the variable `cards`. Write a function `rankCategoryValues` that takes the pack as a parameter and the name of a data category as a second parameter, and returns a dictionary. The dictionary should map the value of the given data category from each card (the dictionary key) to the ranking in the whole pack of that value (the dictionary value). Hence each field value has a ranking from 1 to 50 - you can assume there are no duplicate values in a single data category. For example, if the fastest car in the pack can travel at 220mph, then the dictionary will contain a key value pair of 220 and 1; if the second fastest is 218mph, the dictionary will have a key of 218 and value of 2 key-value pair; and if the slowest car is 55mph, there

will be an key of 55 and value of 50 key-value pair. Hint: sort the pack of cards before you start to build the dictionary.

[10 marks]

```
def rankCategoryValues( cards, cat ):
    def sortKey( card ):
        return card[ cat ]

    cards.sort( key=sortKey, reverse=descending[ cat ] )
    print( cards )

    ranking = {}
    for index in range( len( cards ) ):
        ranking[ cards[ index ][ cat ] ] = index

    return ranking
```

3. Write a function getRanking that taking as parameters a single card, a data category name, and an appropriate collection of dictionaries, each one created by calling the function created in (2) for each different data category. Would this third parameter be best as a list or a dictionary? The function should return the ranking of that card according to the given data category.

[3 marks]

```
def getRanking( card, catName, rankings ):
    thisCatRanking = rankings[ catName ]
    thisCardCatValue = card[ catName ]
    thisCardCatRank = thisCatRanking[ thisCardCatValue ]
    return thisCardCatRank
```

4. Write a function taking a card and the collection of dictionaries, the same as in (3) above, and returning the name of the best data category on the card - that is, the data category whose value has the highest ranking compared to the ranking of the values of all the other data categories.

[6 marks]

```
def bestCatOnCard( card, rankings ):
    bestCat = "topSpeed"
    bestCatRank = rankings[ bestCat ][ card[ bestCat ] ]
    for key in card.keys():
        newKeyRank = rankings[ key ][ card[ key ] ]
        if newKeyRank < bestCatRank:
            bestCat = key
            bestCatRank = newKeyRank
    return bestCat
```

5. Write a function that takes a card as parameter and prints out the card details neatly. You choose the format.

[3 marks]

```
def printCard( card ):
    for key in card.keys():
        print( key, ": ", card[ key ] )
```

6. Write a function “deal" that takes a pack of cards, and returns two lists, each containing half the pack. You may simply split the pack in two.

[3 marks]

```
def deal( pack ):
    hand1 = pack[ : len( pack )//2 ]
    hand2 = pack[ len( pack )//2 : ]
```

```
    return ( hand1, hand2 )
```

7. Write a program to play Top Trumps, with one human player and one computer player. The game should be played as described above, considering the following:

    - Assume you have a pack of 50 cards, in your format, randomly sorted, in the variable cards.
    - On each round, both the computer's next card and the human's next card should be displayed on the screen
    - If the human is to start the round, he/she is asked to specify the data category to be used
    - If the computer is to start the round, it uses the functions written earlier to pick the "best" data category on its card
    - Once the category is chosen, by human or computer, the computer then
        - indicates who has won
        - sets who should start the next round
        - moves the two cards to the bottom of the winner's pack
    - Play continues until one player has all the cards

[20 marks]

```
rankings = {}
for cat in descending.keys():
    print( cat )
    rankings[ cat ] = rankCategoryValues( cards, cat )
    print( rankings[ cat ] )

pHand, cHand = deal( cards )

humanToPlay = True
more = "y"

while len( pHand ) != 0 and len( cHand ) != 0 and more == "y":
    pCard = pHand[ 0 ]
    cCard = cHand[ 0 ]
    print( "Player's Card" )
    printCard( pCard )

    if humanToPlay:
        chosenCat = input( "Choose a category (type it in): " )
    else:
        chosenCat = bestCatOnCard( cCard, rankings )
        print( "Computer has chosen category " + chosenCat + "." )

    pValue = pCard[ chosenCat ]
    cValue = cCard[ chosenCat ]
    print( "Computer value is: ", cValue )

    pRank = rankings[ chosenCat ][ pValue ]
    cRank = rankings[ chosenCat ][ cValue ]

    if pRank < cRank:
        # Player has won the round
        humanToPlay = True
        pHand.append( pCard )
        pHand.append( cCard )
    else:
        # Computer has won
```

```
    humanToPlay = False
    cHand.append( pCard )
    cHand.append( cCard )

del( cHand[ 0 ] )
del( pHand[ 0 ] )

print( "Player cards = ", len( pHand ) )
print( "Computer cards = ", len( cHand ) )

more = input( "Do you want to continue? (y/n): " )
```