INTRODUCTION

This project tries to implement a Library Management System (LMS). On the frontend, it has a Graphical User Interface (GUI) using Python and on the backend, it uses MySQL.

We know that for any institute, library is an essential part. It is the place which provides book to all its members, either for reading or allowing to keep it for few days. On the other hand, its equally not easy to manually maintain the records of all books present in library and the books being issued or returned to library from time to time.

Keeping this huge importance and this mammoth task of library in mind, I decided to make my project related to library management, to handle this huge task – The GUI Library Management System.

This Project has been developed using **Python** (v3.8) programming language in the frontend and **MySQL** (v8.0) database at the backend.

This system will control all details related to a LIBRARY like Book Issue/Return, Add/Delete/Edit new Member/Books, View all issued books etc.

In the database, we store:

1. All the books present in library
2. All the members of library
3. Books issued by the members

Here, only those can issue the books, who are the members of library.

The tables storing the above-mentioned data are "*books*"," *members*" and "*issued*" respectively, created in database "*library*". The "*books*" table has column "*book_id*" as primary key and "*members*" table has column "*member_id*" as primary key. Through these two columns, the tables are individually linked with table "*issued*" using concept of Foreign key. The tables of MySQL acts as only a medium for storing and retrieving data. All the major input/output work is being done in Python.

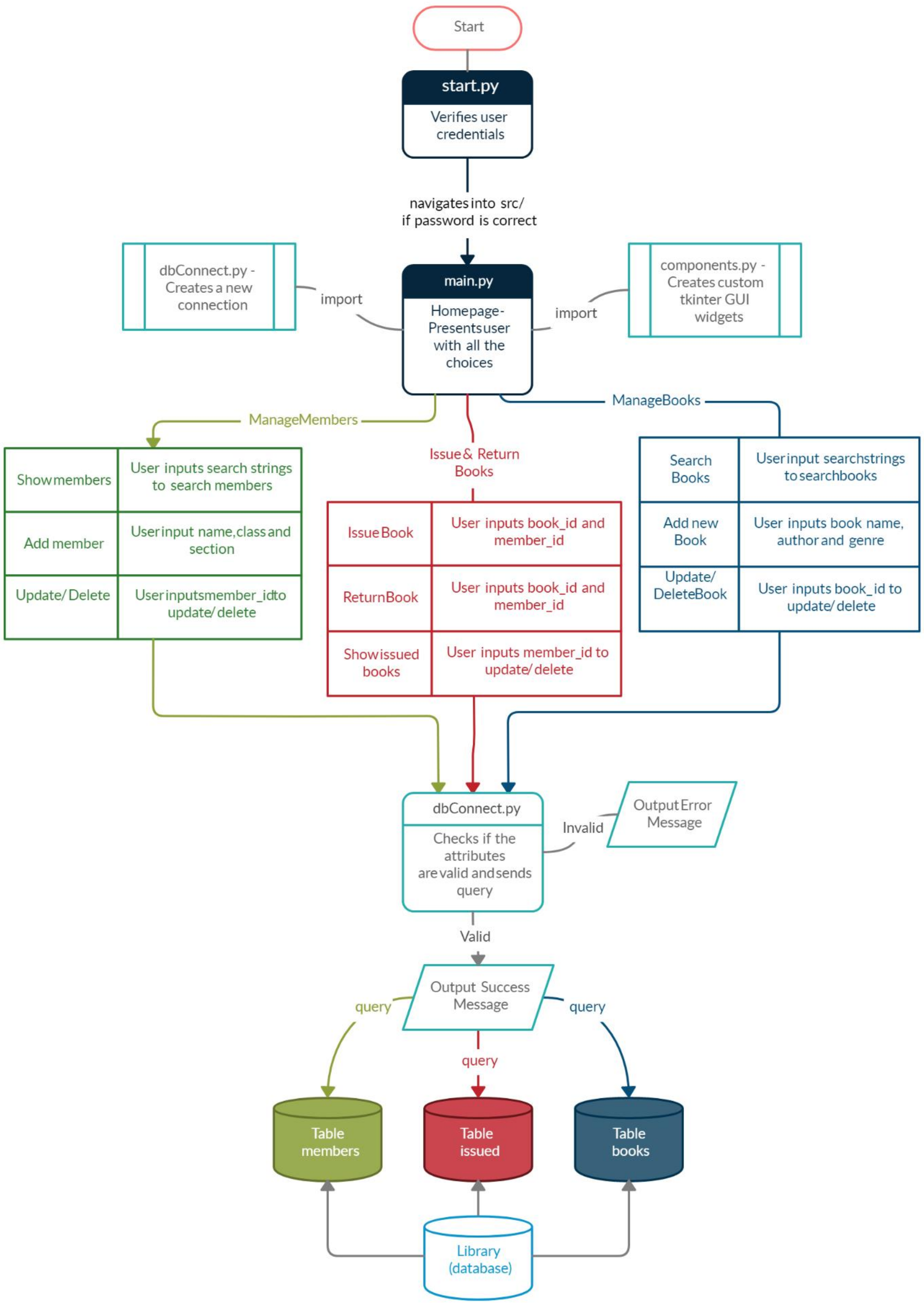
The frontend has been developed using **tkinter** library of python used for creating GUI apps. The connection to the database is established through the **mysql-connector** module of python.

In Python, I have created two modules. The first one is for handling all the backend connections and CRUD operations on the database (*dbConnect.py*). The second module (*components.py*) consists of tkinter GUI components like buttons, text boxes, tables etc. These modules are then integrated in a single, main file, *main.py*. All these files are stored in the *src/* folder

The user can open the application by running *start.py* located in the root folder which verifies the user credentials by asking for their MySQL password and then runs *main.py* if the password is correct.

To showcase the utility of the project, the application also inserts some sample data into the database which will be used to perform some sample operations mentioned here.

Data Flow Diagram


Start
start.py
Verifies user credentials
navigates into src/ if password is correct
dbConnect.py - Creates a new connection
import
main.py
Homepage- Presents user with all the choices
import
components.py - Creates custom tkinter GUI widgets
ManageMembers
Issue & Return Books
ManageBooks
Show members
User inputs search strings to search members
Add member
User input name, class and section
Update/ Delete
User inputs member_id to update/ delete
Issue Book
User inputs book_id and member_id
Return Book
User inputs book_id and member_id
Show issued books
User inputs member_id to update/ delete
Search Books
User input search strings to search books
Add new Book
User inputs book name, author and genre
Update/ Delete Book
User inputs book_id to update/ delete
dbConnect.py
Checks if the attributes are valid and sends query
Invalid
Output Error Message
Valid
Output Success Message
query
query
query
Table members
Table issued
Table books
Library (database)

Some basic Information regarding the Project

Files Created:

1. start.py
2. src/
    a. main.py
    b. dbConnect.py
    c. components.py

Dependencies:

1. mysql-connector (For establishing connectivity between MySQL and Python)
2. datetime (For converting string into date)
3. tkinter (For creating the Graphical User Interface)
4. os & sys (For passing changing directories and passing data b/w files)
5. csv (For reading csv files to create sample database – only used for demonstration)

Database Created – **library**

Tables Created :

1. books
2. members
3. issued

Structure of Tables

books

```
+---------+-------------+------+-----+---------+----------------+
| Field   | Type        | Null | Key | Default | Extra          |
+---------+-------------+------+-----+---------+----------------+
| book_id | int(11)     | NO   | PRI | NULL    | auto_increment |
| book    | varchar(60) | YES  |     | NULL    |                |
| author  | varchar(60) | YES  |     | NULL    |                |
| genre   | varchar(15) | YES  |     | NULL    |                |
+---------+-------------+------+-----+---------+----------------+
```

issued

```
+---------------+---------+------+-----+---------+-------+
| Field         | Type    | Null | Key | Default | Extra |
+---------------+---------+------+-----+---------+-------+
| book_id       | int(11) | YES  | UNI | NULL    |       |
| member_id     | int(11) | YES  |     | NULL    |       |
| date_of_issue | date    | YES  |     | NULL    |       |
+---------------+---------+------+-----+---------+-------+
```

members

```
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| member_id | int(11)     | NO   | PRI | NULL    | auto_increment |
| name      | varchar(25) | YES  |     | NULL    |                |
| class     | varchar(5)  | YES  |     | NULL    |                |
| join_date | date        | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
```

Existing Contents of the Tables

1. books

```
+---------+-------------------------------------------+--------------------------+-------------+
| book_id | book                                      | author                   | genre       |
+---------+-------------------------------------------+--------------------------+-------------+
|       1 | The Fellowship of the Ring                | J.R.R. Tolkien           | Fiction     |
|       2 | The Two Towers                            | J.R.R. Tolkien           | Fiction     |
|       3 | The Return of the King                    | J.R.R. Tolkien           | Fiction     |
|       4 | Origin                                    | Dan Brown                | Fiction     |
|       5 | Concepts of Physics                       | H.C. Verma               | Course Book |
|       6 | The Hobbit                                | J.R.R. Tolkien           | Fiction     |
|       7 | A Game of Thrones                         | George R.R. Martin       | Fiction     |
|       8 | A Clash of Kings                          | George R.R. Martin       | Fiction     |
|       9 | A Storm of Swords                         | George R.R. Martin       | Fiction     |
|      10 | A Feast for Crows                         | George R.R. Martin       | Fiction     |
|      11 | A Dance with Dragons                      | George R.R. Martin       | Fiction     |
|      12 | It                                        | Stephen King             | Fiction     |
|      13 | Wings of Fire                             | Dr. A. P. J. Abdul Kalam | Biography   |
|      14 | Becoming                                  | Michelle Obama           | Biography   |
|      15 | Neverwhere                                | Neil Gaiman              | Non-Fiction |
|      16 | An Absolutely Remarkable Thing            | Hank Green               | Fiction     |
|      17 | Mathematics for Class 12                  | R.D. Sharma              | Course Book |
|      18 | Mathematics for Class 11                  | R.D. Sharma              | Course Book |
|      19 | Computer Science with Python              | Sumita Arora             | Course Book |
|      20 | Computer Science with Python              | Sumita Arora             | Course Book |
|      21 | 11/22/63                                  | Stephen King             | Fiction     |
|      22 | Frankenstein                              | Mary Shelley             | Fiction     |
|      23 | The Martian Chronicles                    | Ray Bradbury             | Non-Fiction |
|      24 | A Beautifully Foolish Endeavor            | Hank Green               | Non-Fiction |
|      25 | Harry Potter and the Philosopher's Stone  | J.K. Rowling             | Fiction     |
|      26 | Harry Potter and the Prisoner of Azkaban  | J.K. Rowling             | Fiction     |
|      27 | Harry Potter and the Order of the Phoenix | J.K. Rowling             | Fiction     |
|      28 | Harry Potter and the Half-Blood Prince    | J.K. Rowling             | Fiction     |
|      29 | Harry Potter and the Deathly Hallows      | J.K. Rowling             | Fiction     |
|      30 | The Da Vinci Code                         | Dan Brown                | Fiction     |
|      31 | An Astronaut's Guide to Life on Earth     | Chris Hadfield           | Biography   |
|      32 | A Brief History of Time                   | Stephen Hawking          | Non-Fiction |
|      33 | The Hitchhiker's Guide to the Galaxy      | Douglas Adams            | Fiction     |
|      34 | Digital Fortress                          | Dan Brown                | Fiction     |
+---------+-------------------------------------------+--------------------------+-------------+
```

2. issued

```
+---------+-----------+----------------+
| book_id | member_id | date_of_issue  |
+---------+-----------+----------------+
|       3 |         9 | 2020-11-20     |
|      30 |         6 | 2020-11-23     |
|       2 |         5 | 2020-11-23     |
|       5 |        17 | 2020-11-25     |
|      10 |         2 | 2020-11-29     |
|       1 |        16 | 2020-12-02     |
|      14 |        11 | 2020-12-03     |
|       9 |        17 | 2020-12-10     |
|      23 |        20 | 2020-12-12     |
|      29 |        19 | 2020-12-13     |
+---------+-----------+----------------+
```

3. members

```
+-----------+----------+-------+------------+
| member_id | name     | class | join_date  |
+-----------+----------+-------+------------+
|         1 | Divya    | 12 C1 | 2020-12-13 |
|         2 | Arya     | 11 Sc | 2017-01-20 |
|         3 | John     | 12 C2 | 2019-12-05 |
|         4 | Anoop    | 12 C2 | 2019-03-10 |
|         5 | Divakar  | 11 C1 | 2017-05-20 |
|         6 | Darlene  | 12 C2 | 2019-12-13 |
|         7 | Francis  | 12 Sc | 2020-02-12 |
|         8 | Sabina   | 11 A  | 2018-12-25 |
|         9 | Robert   | 11 Sc | 2017-05-05 |
|        10 | Rubina   | 12 A  | 2019-09-12 |
|        11 | Vikas    | 12 A  | 2018-03-13 |
|        12 | Mohan    | 12 C2 | 2017-05-05 |
|        13 | Bina     | 12 C2 | 2019-10-29 |
|        14 | Shama    | 11 Sc | 2018-05-25 |
|        15 | Arun     | 12 C2 | 2017-10-20 |
|        16 | Asha     | 12 A  | 2018-01-13 |
|        17 | Meera    | 11 Sc | 2017-10-20 |
|        18 | Fahad    | 11 A  | 2019-10-18 |
|        19 | Vidhisha | 12 Sc | 2019-01-05 |
|        20 | Navneet  | 12 Sc | 2020-05-12 |
+-----------+----------+-------+------------+
```

Source Code

start.py

```
from tkinter import *
import os
from tkinter import messagebox
import mysql.connector as sq

def login(e):
    passwd = entry.get().strip()
    try:
        db = sq.connect(host='localhost', user='root', password=passwd)
        root.destroy()
        os.chdir('src')
        os.system(f'python main.py {passwd}')
    except:
        messagebox.showinfo(title='Access Denied', message='The password is incorrect.'
)

FONT_BIG = ('arial',14,'bold')
FONT_SMALL = ('arial',12,'bold')
FONT_INP = ('verdana',11,'bold')

root = Tk()
root.title('User Authentication')
root.geometry('290x181+500+300')
COLOR='#d9ecff'
root.configure(bg=COLOR)

lbl = Label(root, text='Enter your\nMySQL password', bg=COLOR, font=FONT_BIG)
entry = Entry(root, width=20, font=FONT_INP)
btn = Button(root, text='Submit', font=FONT_SMALL, bg='#fff', command=lambda: login(1))

lbl.grid(row=0, column=0, pady=10, padx=10)
entry.grid(row=1, column=0, ipady=3, ipadx=3, pady=5, padx=30)
btn.grid(row=2, column=0, pady=20, padx=10)
entry.focus()
root.bind('<Return>', login)

root.mainloop()
```

dbConnect.py

```
import mysql.connector as sq
import csv
import os
from datetime import datetime
from mysql.connector.errors import IntegrityError

PATH = os.getcwd() + '\\data\\'
now = datetime.now()
DATE = now.strftime('%Y-%m-%d')

def new_connection(passwd):
    '''Establishes connection with MySQL database
       And adds sample data if needed'''
    global db
    db = sq.connect(host='localhost', user='root', password=passwd)
    cursor = db.cursor()
    cursor.execute('CREATE DATABASE IF NOT EXISTS library')
    cursor.execute('USE library')
    cursor.execute('SHOW TABLES LIKE \'books\'')
    result = cursor.fetchone()
    if not result:
        add_data()

def defineCursor(func):
    '''Decorator funtion to create and close cursor instances'''
    def wrapper(*args, **kwargs):
        cursor = db.cursor()
        result = func(cursor, *args, **kwargs)
        try:
            cursor.close()
        except:
            _ = cursor.fetchall()
            cursor.close()
        return result
    return wrapper

@defineCursor
def add_data(cursor):
    '''Adding sample data to the database'''
    cursor.execute('''create table if not exists books(
        book_id int primary key auto_increment,
        book varchar(60),
        author varchar(60),
        genre varchar(15)
    )''')
    cursor.execute('''create table if not exists members(
        member_id int primary key auto_increment,
        name varchar(25),
        class varchar(5),
        join_date date
    )''')
    cursor.execute('''create table if not exists issued(
        book_id int UNIQUE,
        member_id int,
        date_of_issue date
    )''')

    books_data = get_sample_data('sample_books.csv')
    members_data = get_sample_data('sample_members.csv')
    issued_data = get_sample_data('sample_issued.csv')

    try:
        for row in books_data:
            q = f'''INSERT INTO books(book, author, genre)
                    VALUES("{row[0]}", '{row[1]}', '{row[2]}')'''
            cursor.execute(q)
        for row in issued_data:
            q = f'''INSERT INTO issued
                    VALUES({row[0]}, {row[1]}, '{row[2]}')'''
            cursor.execute(q)
        for row in members_data:
            q = f'''INSERT INTO members(name, class, join_date)
                    VALUES('{row[0]}', '{row[1]}', '{row[2]}')'''
            cursor.execute(q)
    except:
        return

    db.commit()

def get_sample_data(file):
    data = []
    with open(PATH+file) as f:
        reader = csv.reader(f)
        for row in reader:
            data.append(row)
    return data

def shorten_string(s):
    i = s.find(' ', 18, -1)
    i = s.rfind(' ') if i == -1 else i
    return s[:i] + '\n' + s[i:]
```

```
@defineCursor
def add_new_book(cursor, name, author, genre):
    '''Add a new book to the database'''
    q = f'''INSERT INTO books(book, author, genre)
        VALUES("{name}", '{author}', '{genre}')'''
    try:
        cursor.execute(q)
        db.commit()
        return True
    except:
        return False

@defineCursor
def add_new_member(cursor, name, clss):
    q = f'''INSERT INTO members(name, class, join_date)
        VALUES('{name}', '{clss}', '{DATE}')'''

    try:
        cursor.execute(q)
        db.commit()
        return True
    except:
        return False

@defineCursor
def get_issued_books(cursor, book_id, member_id):
    '''Get the list of issued books along with issuer's info'''
    query = '''SELECT issued.book_id, book, issued.member_id, name, class, date_of_issu
e FROM issued, members, books
        WHERE members.member_id = issued.member_id
        AND books.book_id = issued.book_id'''

    if book_id and member_id:
        query += f' and issued.book_id = {book_id} and issued.member_id={member_id}'
    elif book_id:
        query += f' and issued.book_id = {book_id}'
    elif member_id:
        query += f' and issued.member_id={member_id}'

    query += ' ORDER BY date_of_issue'

    data = []
    try:
        cursor.execute(query)
        data = cursor.fetchall()
        return data
    except:
        return data

@defineCursor
def get_members(cursor, member_id, name, clss):
    q = f'''SELECT * FROM members
        WHERE name LIKE '%{name}%' AND class LIKE '%{clss}%' '''

    if member_id:
        q += f' AND member_id = {member_id}'

    data = []
    try:
        cursor.execute(q)
        data = cursor.fetchall()
        return data
    except:
        return data

@defineCursor
def get_search(cursor, book_id, name, author):
    '''Get results for searched books'''
    search_id = f' AND books.book_id = {book_id}'
    query = f'''SELECT books.book_id, book, author, genre,
            case
                when EXISTS (SELECT NULL FROM issued WHERE  books.book_id = issued.book
_id)
                then 'Yes'
                ELSE 'No'
            END AS 'Issued'
        FROM books
        WHERE book LIKE '%{name}%' AND author LIKE '%{author}%'
        '''
    if book_id:
        query += search_id

    try:
        cursor.execute(query)
        data = cursor.fetchall()
        return data
    except:
        return []

@defineCursor
def fill_issue_details(cursor, table, column_id):
    '''Fill lables on KeyRelease functions'''
    if not column_id:
        return
    if table == 'members':
        q = f'SELECT name, class FROM {table} WHERE member_id={column_id}'
    else:
        q = f'SELECT book, author FROM {table} WHERE book_id={column_id}'

    cursor.execute(q)
    row = cursor.fetchone()
    if cursor.rowcount == 1:
        col1, col2 = row
    else:
        col1 = col2 = ''
    if len(col1) > 18:
        col1 = shorten_string(col1)
    return col1, col2

@defineCursor
def fill_return_details(cursor, book_id, member_id):
    '''Fill return details if entered info is correct'''
    if member_id and book_id:
        q = f'WHERE issued.member_id = {member_id} AND issued.book_id = {book_id}'
    elif member_id:
        q = f'issued.member_id = {member_id}'
    elif book_id:
        q = f'issued.book_id = {book_id}'
    else:
        # no input
        return ['' for _ in range(6)]

    query = f'''SELECT issued.book_id, book, author, issued.member_id, name, class FROM
 issued, members, books
        WHERE {q}
        AND members.member_id = issued.member_id
        AND books.book_id = issued.book_id'''

    try:
        cursor.execute(query)
        row = list(cursor.fetchone())
        if row and len(row[1]) > 18:
            row[1] = shorten_string(row[1])
        return row
    except:
        return False

@defineCursor
def issue_book(cursor, book_id, member_id):
    '''Issue a book'''
    q = f"INSERT INTO issued VALUES({book_id},{member_id},'{DATE}')"
    try:
        cursor.execute(q)
        db.commit()
        return True
    except IntegrityError:
        return 'is issued'
    except:
        return False

@defineCursor
def return_book(cursor, book_id):
    '''Return a book'''
    q = f'''DELETE FROM issued WHERE book_id = {book_id}'''
    try:
        cursor.execute(q)
        db.commit()
        return True
    except:
        return False

@defineCursor
def update_column(cursor, table, _id, col1, col2, col3):
    if table == 'books':
        q = f'''UPDATE books
            SET book = '{col1}', author='{col2}', genre='{col3}'
            WHERE book_id = {_id}'''
    else:
        q = f'''UPDATE members
            SET name = '{col1}', class='{col2}', join_date='{col3}'
            WHERE member_id = {_id}'''
    try:
        cursor.execute(q)
        db.commit()
        return True
    except:
        return False

@defineCursor
def fill_column_details(cursor, table, _id):
    if table == 'members':
        q = f'''SELECT name, class, join_date FROM members
            WHERE member_id = {_id}'''
    else:
        q = f'''SELECT book, author, genre FROM books
            WHERE book_id = {_id}'''

    try:
        cursor.execute(q)
        row = cursor.fetchone()
        if row:
            return row
    except:
        pass
    return ['' for _ in range(3)]

@defineCursor
def delete_column(cursor, table, _id):
    '''Delete a row from the database'''
    col = 'book_id' if table == 'books' else 'member_id'
    q = f'DELETE FROM {table} WHERE {col} = {_id}'
    try:
        cursor.execute(q)
        db.commit()
        return True
    except:
        return False

def close_connection():
    db.close()
```

components.py

```
from tkinter import *
from tkinter import ttk

FONT_ENTRY = ('verdana',10,'bold')
COLOR = '#f6f6f6'
CLR_GRAY = '#5d5d66'

BTN_FONT = ('arial',15,'bold')
FONT_BIG = ('arial',15,'bold')
FONT_SMALL = ('arial',12,'bold')
FONT_REALLY_BIG = ('arial',19,'bold')


class HomeButton(Button):
    def __init__(self, parent, **options):
        Button.__init__(self, parent, options, relief=GROOVE, font=BTN_FONT, width=20, 
bg=COLOR)

    def set_grid(self, **kwargs):
        self.grid(kwargs, pady=10, padx=25)

class MyEntry(Entry):
    def __init__(self, parent, **options):
        Entry.__init__(self, parent, options, font=FONT_ENTRY)

    def val(self):
        return self.get().strip()

    def set_val(self, val):
        self.delete(0, END)
        self.insert(0, val)

class MyLabel(Label):
    def __init__(self, parent, **options):
        Label.__init__(self, parent, options, font=FONT_SMALL)

class MyTree(ttk.Treeview):
    def __init__(self, parent, **options):
        self.tree_frame = Frame(parent)
        self.tree_frame.grid(options)
        self.tree_scroll = Scrollbar(self.tree_frame)
        self.tree_scroll.pack(side=RIGHT, fill=Y)

        self.tree = ttk.Treeview(self.tree_frame, yscrollcommand=self.tree_scroll.set)
        self.tree_scroll.config(command=self.tree.yview)
        self.tree.pack(expand=True)

    def set_columns(self, columns, headings, widths):
        self.tree['columns'] = tuple(columns)
        self.tree.column('#0', width=0, stretch=NO)
        self.tree.heading('#0', text='')

        for column, heading, width in zip(columns, headings, widths):
            self.tree.column(column, anchor=CENTER, width=width)
            self.tree.heading(column, text=heading)

    def insert_data(self, data):
        i = 0
        for row in data:
            self.tree.insert(parent='', index='end', iid=i, text='', value=row)
            i += 1
```

main.py

```
from tkinter import *
import dbConnect as db
from tkinter import ttk
from tkinter import messagebox
import sys
from components import *

TABLE_MEMBERS = 'members'
TABLE_BOOKS = 'books'
GENRES = ['Fiction', 'Non-Fiction', 'Biography', 'Course Book']

def show_issued_books():
    '''Show issued books window'''
    show_window = Toplevel(root)
    show_window.title('All issued books')
    show_window.geometry("740x380+400+200")
    show_window.resizable(False, False)

    lb_title = Label(show_window, text='Issued Books', font=FONT_REALLY_BIG)
    lb_book_id = MyLabel(show_window, text='Book Id')
    lb_student_id = MyLabel(show_window, text='Student Id')

    entry_book = MyEntry(show_window, width=10)
    entry_student = MyEntry(show_window, width=10)

    entry_book.bind("<KeyRelease>", lambda e: populate_table())
    entry_student.bind("<KeyRelease>", lambda e: populate_table())

    lb_title.grid(row=0, column=0, columnspan=2, pady=15)
    lb_book_id.grid(row=1, column=0)
    lb_student_id.grid(row=1, column=1)
    entry_book.grid(row=2, column=0)
    entry_student.grid(row=2, column=1)

    def populate_table():
        cols = ['book_id', 'book_name', 'student_id', 'student_name', 'class', 'date']
        col_names = ['Book Id', 'Book', 'Student Id', 'Issued by', 'Class', 'Issue Date
']
        widths = [50, 250, 70, 140, 80, 100]

        tree = MyTree(show_window, row=4, column=0, columnspan=2, padx=20, pady=15)
        tree.set_columns(columns=cols, headings=col_names, widths=widths)

        data = db.get_issued_books(entry_book.val(), entry_student.val())
        tree.insert_data(data)
    populate_table()

def search_books():
    '''Search books window'''
    search_window = Toplevel(root)
    search_window.title('Search Books')
    search_window.geometry("740x500+400+200")
    search_window.resizable(False, False)

    entry_name = MyEntry(search_window, width=28)
    entry_author = MyEntry(search_window, width=22)
    entry_id = MyEntry(search_window, width=6)

    lb_length = Label(search_window, font=FONT_SMALL, fg=CLR_GRAY)
    lb_length.grid(row=5, column=0)

    btn_search = Button(search_window, text='Search', width=10, font=FONT_BIG,
        command=lambda: populate_table())
    btn_search.grid(row=3, column=0, columnspan=5, pady=15)

    Label(search_window, text='Search Books', font=FONT_REALLY_BIG).grid(
        row=0, column=0, columnspan=4, pady=15)
    MyLabel(search_window, text='Enter Id:').grid(row=1, column=0)
    MyLabel(search_window, text='Enter Name:').grid(
        row=1, column=1, columnspan=2, pady=2)
    MyLabel(search_window, text='Enter Author:').grid(
        row=1, column=3, columnspan=2, pady=2)
    entry_id.grid(row=2, column=0, ipady=2, padx=45)
    entry_name.grid(row=2, column=1, columnspan=2, ipady=2, padx=30, pady=2)
    entry_author.grid(row=2, column=3, columnspan=2, ipady=2, padx=30, pady=2)

    def populate_table():
        book_id = entry_id.val()
        name = entry_name.val()
        author = entry_author.val()
        tree = MyTree(search_window, row=4, column=0, columnspan=4, padx=20, pady=15)

        cols = ['id', 'name', 'author', 'fiction', 'issued']
        col_names = ['Id', 'Name', 'Author', 'Type', 'Is issued']
        widths = [50, 250, 200, 100, 90]
        tree.set_columns(columns=cols, headings=col_names, widths=widths)

        data = db.get_search(book_id, name, author)
        lb_length.configure(text=f'{len(data)} results')
        tree.insert_data(data)
    populate_table()

def fill_non_specific_info(window):
    '''Fill the common fields in issue/return windows'''
    Label(window, text='Book Details', font=FONT_BIG).grid(
        row=0, column=0, pady=15, columnspan=2)
    MyLabel(window, text='Book id').grid(row=1, column=0, pady=5, padx=40)
    MyLabel(window, text='Name: ').grid(row=2, column=0, pady=5)
    MyLabel(window, text='Author: ').grid(row=3, column=0, pady=5)
    Label(window, text='Student Details', font=FONT_BIG).grid(
        row=5, column=0, pady=15, columnspan=2)
    MyLabel(window, text='Student id').grid(row=6, column=0, pady=5)
    MyLabel(window, text='Name: ').grid(row=8, column=0, pady=5)
    MyLabel(window, text='Class: ').grid(row=9, column=0, pady=5)

def return_book():
    '''Return book window'''
    return_window = Toplevel(root)
    return_window.title('Return a Book')
    return_window.geometry("380x530+400+180")
    return_window.resizable(False, False)

    book_id = StringVar()
    student_id = StringVar()

    fill_non_specific_info(return_window)

    keypress_event = lambda: fill_return_details()

    lb_book_name = MyLabel(return_window)
    lb_author = MyLabel(return_window)
    entry_book_id = MyEntry(return_window, width=20)
    find_student = Button(return_window, text='Find student', font=FONT_SMALL, command=
keypress_event)

    lb_member_name = MyLabel(return_window)
    lb_member_class = MyLabel(return_window)
    entry_member_id = MyEntry(return_window, width=20)
    find_book = Button(return_window, text='Find book issued', font=FONT_SMALL, command
=keypress_event)
    on_return = lambda: return_book_in_db()
    btn_return = Button(return_window, text='Return Book', width=25, font=BTN_FONT, com
mand=on_return)

    find_student.grid(row=4, column=0, columnspan=2, pady=5)
    find_book.grid(row=10, column=0, columnspan=2, pady=5)
    entry_book_id.grid(row=1, column=1, pady=5,sticky=W, ipady=2)
    lb_book_name.grid(row=2, column=1, pady=5)
    lb_author.grid(row=3, column=1, pady=5)
    entry_member_id.grid(row=6, column=1, pady=5,sticky=W, ipady=2)
    lb_member_name.grid(row=8, column=1, pady=5)
    lb_member_class.grid(row=9, column=1, pady=5)
    btn_return.grid(row=11, column=0, columnspan=2, padx=30, pady=25)

    def return_book_in_db():
        returned = db.return_book(entry_book_id.val())
        if returned:
            messagebox.showinfo(title='Success',
                message='The book has been returned.')
            return_window.destroy()
        else:
            messagebox.showerror(title='Error',
                message='Sorry, the book could not be returned.')
            return_window.lift(root)

    def fill_return_details():
        row = db.fill_return_details(
            entry_book_id.val(), entry_member_id.val())

        if not row:
            msg='Either the entered book is not issued\nOr the given member has not\nis
sued any books currently'
            row = ['' for _ in range(6)]
            messagebox.showwarning(title='Error', message=msg)
            return_window.lift(root)

        entry_book_id.set_val(row[0])
        entry_member_id.set_val(row[3])
        lb_book_name.configure(text=row[1])
        lb_author.configure(text=row[2])
        lb_member_name.configure(text=row[4])
        lb_member_class.configure(text=row[5])

def issue_book():
    '''Issue book window'''
    issue_window = Toplevel(root)
    issue_window.title('Issue new Book')
    issue_window.geometry("380x450+400+220")
    issue_window.resizable(False, False)

    fill_non_specific_info(issue_window)

    lb_book = MyLabel(issue_window)
    lb_author = MyLabel(issue_window)
    lb_member = MyLabel(issue_window)
    lb_class = MyLabel(issue_window)

    entry_book_id = MyEntry(issue_window, width=20)
    entry_member_id = MyEntry(issue_window, width=20)

    entry_book_id.bind("<KeyRelease>",
        lambda e: fill_details(TABLE_BOOKS, e, lb_book, lb_author))
    entry_member_id.bind("<KeyRelease>",
        lambda e: fill_details(TABLE_MEMBERS, e, lb_member, lb_class))

    btn_issue = Button(issue_window, text='Issue Book', width=25, font=BTN_FONT,
        command=lambda: issue_book_in_db())

    entry_book_id.grid(row=1, column=1, pady=5,sticky=W, ipady=2)
    lb_book.grid(row=2, column=1, pady=5)
    lb_author.grid(row=3, column=1, pady=5)
    entry_member_id.grid(row=6, column=1, pady=5,sticky=W, ipady=2)
    lb_member.grid(row=8, column=1, pady=5)
    lb_class.grid(row=9, column=1, pady=5)
    btn_issue.grid(row=11, column=0, columnspan=2, padx=30, pady=25)

    def fill_details(table, e, label1, label2):
        col1, col2 = db.fill_issue_details(table, e.widget.val())
        label1.configure(text=col1)
        label2.configure(text=col2)

    def issue_book_in_db():
        book_id = entry_book_id.val()
        mem_id = entry_member_id.val()

        issued = db.issue_book(book_id, mem_id)
        if issued == 'is issued':
            messagebox.showwarning(title='Error',
                message='The book is already issued by someone')
            issue_window.lift(root)
        elif issued == False:
            messagebox.showerror(title='Error',
                message='There was some problem.\nThe book could not be issued.')
            issue_window.destroy()
        else:
            messagebox.showinfo(title='Success',
                message='The book has been issued successfully.')
            issue_window.destroy()

def edit_book():
    '''Delete book window'''
```

```
edit_window = Toplevel(root)
edit_window.title('Update/Delete Book')
edit_window.geometry("420x320+500+220")
edit_window.resizable(False, False)

var_genre = StringVar(edit_window)
var_genre.set('')

entry_id = MyEntry(edit_window, width=28)
entry_name = MyEntry(edit_window, width=28)
entry_author = MyEntry(edit_window, width=28)
drp_genre = OptionMenu(edit_window, var_genre, *GENRES)
drp_genre.config(font=FONT_SMALL)

entry_id.bind('<KeyRelease>', lambda e:fill_details())

btn_fr = Frame(edit_window)
btn_fr.grid(row=5, column=0, columnspan=2, pady=20)
btn_update = Button(btn_fr, text='Update', width=12, font=BTN_FONT,
    command=lambda: update_book())
btn_delete = Button(btn_fr, text='Delete', width=12, font=BTN_FONT,
    command=lambda: delete_book())
btn_update.grid(row=0, column=0, padx=20)
btn_delete.grid(row=0, column=1)

drp_genre.grid(row=4, column=1)
entry_id.grid(row=1, column=1, pady=10, ipady=3)
entry_name.grid(row=2, column=1, pady=10, ipady=3)
entry_author.grid(row=3, column=1, pady=10, ipady=3)

Label(edit_window, text=' '*7+'Update / Delete Book', font=FONT_BIG).grid(
    row=0, column=0, columnspan=2, pady=10)
MyLabel(edit_window, text='Book Id:').grid(row=1, column=0, pady=10, padx=30)
MyLabel(edit_window, text='Name:').grid(row=2, column=0, pady=10)
MyLabel(edit_window, text='Author:').grid(row=3, column=0, pady=10)
MyLabel(edit_window, text='Genre:').grid(row=4, column=0, pady=10)

def fill_details():
    book_id = entry_id.val()
    name, author, genre = db.fill_column_details(TABLE_BOOKS, book_id)
    entry_name.set_val(name)
    entry_author.set_val(author)
    var_genre.set(genre)

def update_book():
    book_id = entry_id.val()
    name = entry_name.val()
```

```
        author = entry_author.val()
        genre = var_genre.get()

        if book_id and name and author:
            updated = db.update_column(TABLE_BOOKS, book_id, name, author, genre)
            if updated:
                messagebox.showinfo(title='Success',
                    message='Book details have been Updated.')
                edit_window.destroy()
                return

        messagebox.showerror(title='Error',
            message='Please enter proper values.')
        edit_window.lift(root)

    def delete_book():
        book_id = entry_id.val()
        deleted = db.delete_column(TABLE_BOOKS, book_id)
        if deleted:
            messagebox.showinfo(title='Success',
                message='The Book has been deleted.')
            edit_window.destroy()
        else:
            messagebox.showerror(title='Error',
                message='There was some problem.\nTry again later.')
            edit_window.lift(root)

def add_new_book():
    '''Add a new book window'''
    add_window = Toplevel(root)
    add_window.title('Add New Book')
    add_window.geometry("400x290+400+220")
    add_window.resizable(False, False)

    genre = StringVar(add_window)
    genre.set(GENRES[0])

    lb_title = Label(add_window, text='Add New Book', font=FONT_BIG)
    lb_name = MyLabel(add_window, text='Name:')
    lb_author = MyLabel(add_window, text='Author:')
    lb_genre = MyLabel(add_window, text='Genre:')

    entry_name = MyEntry(add_window, width=30)
    entry_author = MyEntry(add_window, width=30)
    drp_genre = OptionMenu(add_window, genre, *GENRES)
    drp_genre.config(font=FONT_SMALL)
    btn_submit = Button(add_window, text='Submit', width=25, font=BTN_FONT,
        command=lambda: add_to_db())

    lb_title.grid(row=0, column=0, columnspan=3, pady=10)
    lb_name.grid(row=1, column=0, pady=10)
    entry_name.grid(row=1, column=1, pady=10, ipady=3, columnspan=2)
    lb_author.grid(row=2, column=0, pady=10)
    entry_author.grid(row=2, column=1, pady=10, ipady=3, columnspan=2)
    lb_genre.grid(row=3, column=0, padx=15, pady=10)
    drp_genre.grid(row=3, column=1)

    btn_submit.grid(row=4, column=0, columnspan=3, padx=30, pady=30)

    def add_to_db():
        name = entry_name.val()
        author = entry_author.val()
        sel_genre = genre.get()
        if name and author:
            added = db.add_new_book(name, author, sel_genre)
            if added:
                messagebox.showinfo(title='Success',
                    message='The book has been added to the database.')
                add_window.destroy()
            else:
                messagebox.showerror(title='Error',
                    message='There was an error in adding the book.')
                add_window.lift(root)
        else:
            messagebox.showwarning(title='Invalid',
                    message='Please enter the correct values.')
            add_window.lift(root)

def add_member():
    ''' Adds new member into the database '''
    add_mem_window = Toplevel(root)
    add_mem_window.title('Add New Member')
    add_mem_window.geometry("400x280+400+220")
    add_mem_window.resizable(False, False)

    entry_name = MyEntry(add_mem_window, width=20)
    entry_class = MyEntry(add_mem_window, width=20)
    entry_section = MyEntry(add_mem_window, width=20)

    entry_name.grid(row=1, column=1, ipady=3)
    entry_class.grid(row=2, column=1, ipady=3)
    entry_section.grid(row=3, column=1, ipady=3)
    btn_submit = Button(add_mem_window, text='Submit', width=25, font=BTN_FONT,
         command=lambda: add_to_db())

    btn_submit.grid(row=4, column=0, columnspan=2, padx=25, pady=20)
    Label(add_mem_window, text='Add New Member', font=FONT_BIG).grid(
        row=0, column=0, pady=10, columnspan=2)
    MyLabel(add_mem_window, text='Name:').grid(
        row=1, column=0, pady=10, padx=25)
    MyLabel(add_mem_window, text='Class:').grid(
        row=2, column=0, pady=10)
    MyLabel(add_mem_window, text='Section: ').grid(
        row=3, column=0, pady=10)

    def add_to_db():
        name = entry_name.val()
        clss = entry_class.val() + ' ' + entry_section.val().upper()

        inserted = db.add_new_member(name, clss)
        if inserted:
            messagebox.showinfo(title='Success',
                message='Member details have been added.')
            add_mem_window.destroy()
        else:
            messagebox.showerror(title='Error',
                message='There was some problem.\nTry again later.')
            add_mem_window.lift(root)

def edit_member():
    ''' Update member details or delete a member from database '''
    edit_mem_window = Toplevel(root)
    edit_mem_window.title('Add New Member')
    edit_mem_window.geometry("420x420+600+220")
    edit_mem_window.resizable(False, False)

    entry_id = MyEntry(edit_mem_window, width=20)
    entry_name = MyEntry(edit_mem_window, width=20)
    entry_class = MyEntry(edit_mem_window, width=20)
    entry_section = MyEntry(edit_mem_window, width=20)
    entry_date = MyEntry(edit_mem_window, width=20)

    entry_id.grid(row=1, column=1, ipady=3)
    entry_name.grid(row=2, column=1, ipady=3)
    entry_class.grid(row=3, column=1, ipady=3)
    entry_section.grid(row=4, column=1, ipady=3)
    entry_date.grid(row=5, column=1, ipady=3)

    entry_id.bind('<KeyRelease>', lambda e: fill_details())
    btn_update = Button(edit_mem_window, text='Update', width=12, font=BTN_FONT,
         command=lambda: update_member())
    btn_delete = Button(edit_mem_window, text='Delete', width=12, font=BTN_FONT,
         command=lambda: delete_member())

    btn_update.grid(row=6, column=0, pady=20, padx=25)
    btn_delete.grid(row=6, column=1, pady=20, padx=10)

    Label(edit_mem_window, text=' '*10+'Update / Delete Member', font=FONT_BIG).grid(
        row=0, column=0, padx=50, pady=15, columnspan=2)
    MyLabel(edit_mem_window, text='Id:').grid(
        row=1, column=0, pady=10)
    MyLabel(edit_mem_window, text='Edit Name:').grid(
        row=2, column=0, pady=10)
    MyLabel(edit_mem_window, text='Edit Class:').grid(
        row=3, column=0, pady=10, padx=35)
    MyLabel(edit_mem_window, text='Edit Section:').grid(
        row=4, column=0, pady=10)
    MyLabel(edit_mem_window, text='Edit Date of\nJoining:\n(YYYY-MM-DD)').grid(
        row=5, column=0, pady=10)

    def fill_details():
        mem_id = entry_id.val()
        name, clss, date = db.fill_column_details(TABLE_MEMBERS, mem_id)

        clss = clss.split(maxsplit=1) if clss else ['','']
        entry_name.set_val(name)
        entry_class.set_val(clss[0])
        entry_section.set_val(clss[-1])
        entry_date.set_val(date)

    def update_member():
        mem_id = entry_id.val()
        name = entry_name.val()
        clss = entry_class.val() + ' ' + entry_section.val()
        date = entry_date.val()

        if mem_id and name and clss and date:
            updated = db.update_column(TABLE_MEMBERS, mem_id, name, clss, date)
            if updated:
                messagebox.showinfo(title='Success',
                    message='Member details have been Updated.')
                edit_mem_window.destroy()
            else:
                messagebox.showerror(title='Error',
                    message='Please enter proper values\nMake sure the date is in corre
ct format.')
```

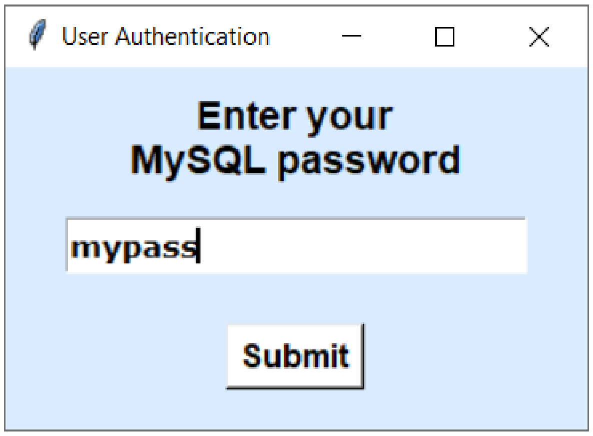
```
                edit_mem_window.lift(root)
            else:
                messagebox.showerror(title='Error',
                    message='Please enter proper values.')
                edit_mem_window.lift(root)

        def delete_member():
            mem_id = entry_id.val()
            deleted = db.delete_column(TABLE_MEMBERS, mem_id)
            if deleted:
                messagebox.showinfo(title='Success',
                    message='Member details have been deleted.')
                edit_mem_window.destroy()
            else:
                messagebox.showerror(title='Error',
                    message='There was some problem.\nTry again later.')
                edit_mem_window.lift(root)

    def show_members():
        ''' Window to show and search members '''
        members_window = Toplevel(root)
        members_window.title('Show All Members')
        members_window.geometry("640x420+400+200")
        members_window.resizable(False, False)

        entry_id = MyEntry(members_window, width=6)
        entry_name = MyEntry(members_window, width=28)
        entry_class = MyEntry(members_window, width=10)

        Label(members_window, text='Library Members', font=FONT_REALLY_BIG).grid(
            row=0, column=0, columnspan=3, pady=15)
        MyLabel(members_window, text='Search by Id:').grid(
            row=1, column=0)
        MyLabel(members_window, text='Search by Name:').grid(
            row=1, column=1, pady=2)
        MyLabel(members_window, text='Search by Class:').grid(
            row=1, column=2, pady=2)
        entry_id.grid(row=2, column=0, ipady=2, padx=10)
        entry_name.grid(row=2, column=1, ipady=2, padx=10, pady=10)
        entry_class.grid(row=2, column=2, ipady=2, padx=10)

        event_click = lambda e: populate_table()
        entry_id.bind('<KeyRelease>', event_click)
        entry_name.bind('<KeyRelease>', event_click)
        entry_class.bind('<KeyRelease>', event_click)

        def populate_table():
```

```
        mem_id = entry_id.val()
        name = entry_name.val()
        clss = entry_class.val()

        cols = ['member_id', 'name', 'class', 'date']
        col_names = ['Member Id', 'Name', 'Class', 'Join Date']
        widths = [100, 250, 100, 120]

        tree = MyTree(members_window, row=3, column=0, columnspan=3, padx=20, pady=10)
        tree.set_columns(columns=cols, headings=col_names, widths=widths)

        data = db.get_members(mem_id, name, clss)
        tree.insert_data(data)

    populate_table()

'''Home page'''
root = Tk()
root.title('Library Management System')
root.geometry("900x380+350+200")
root.resizable(False,False)
root.configure(bg=COLOR)

app_title = Label(root, text='Library Management\nSystem', bg=COLOR, font=FONT_REALLY_B
IG)

issue_btn = HomeButton(root, text='Issue Book', command=issue_book)
return_btn = HomeButton(root, text='Return Book', command=return_book)
show_btn = HomeButton(root, text='Show Issued Books', command=show_issued_books)
search_btn = HomeButton(root, text='Search for Books', command=search_books)
add_btn = HomeButton(root, text='Add New Book', command=add_new_book)
edit_btn = HomeButton(root, text='Update/Delete Book', command=edit_book)
show_members_btn = HomeButton(root, text='Show All Members', command=show_members)
add_member_btn = HomeButton(root, text='Add a Member', command=add_member)
edit_member_btn = HomeButton(root, text='Update/Delete Member', command=edit_member)

Label(root, text='Issue & Return\nBooks', font=FONT_BIG, bg=COLOR, fg=CLR_GRAY).grid(ro
w=1, column=1)
Label(root, text='Manage\nBooks', font=FONT_BIG, bg=COLOR, fg=CLR_GRAY).grid(row=1, col
umn=0)
Label(root, text='Manage\nMembers', font=FONT_BIG, bg=COLOR, fg=CLR_GRAY).grid(row=1, c
olumn=2)

app_title.grid(row=0, column=1, columnspan=1, pady=25)
issue_btn.set_grid(row=2, column=1)
return_btn.set_grid(row=3, column=1)
show_btn.set_grid(row=4, column=1)
```

```
search_btn.set_grid(row=2, column=0)
add_btn.set_grid(row=3, column=0)
edit_btn.set_grid(row=4, column=0)
show_members_btn.set_grid(row=2, column=2)
add_member_btn.set_grid(row=3, column=2)
edit_member_btn.set_grid(row=4, column=2)

try:
    PASSWD = sys.argv[1]
    db.new_connection(passwd=PASSWD)
    root.mainloop()
    db.close_connection()
except:
    messagebox.showwarning(title='Error',
        message='Could not connect to the database.')
```

Output

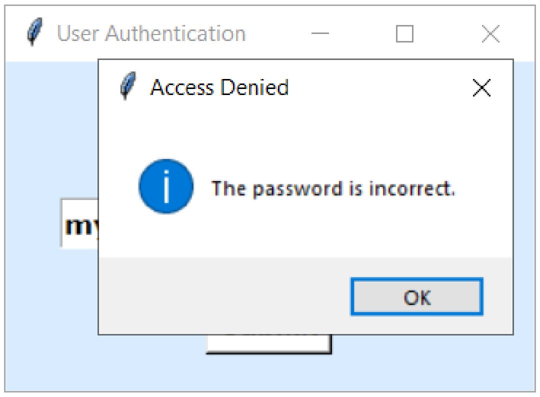
1. Login page– start.py



If password is correct

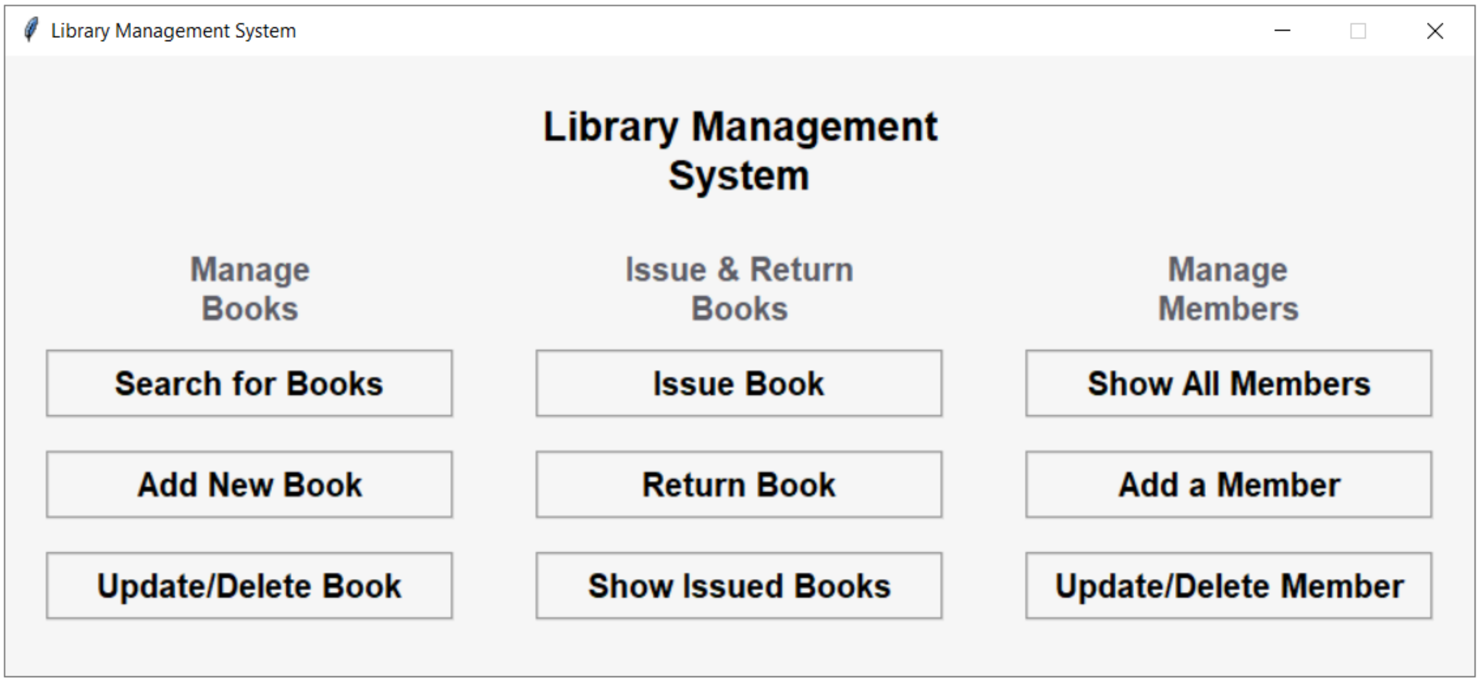
If password is correct, it opens src/main.py (see next page)

If password is incorrect



The password is incorrect.

2. Main Application – main.py



Library Management System

Library Management System

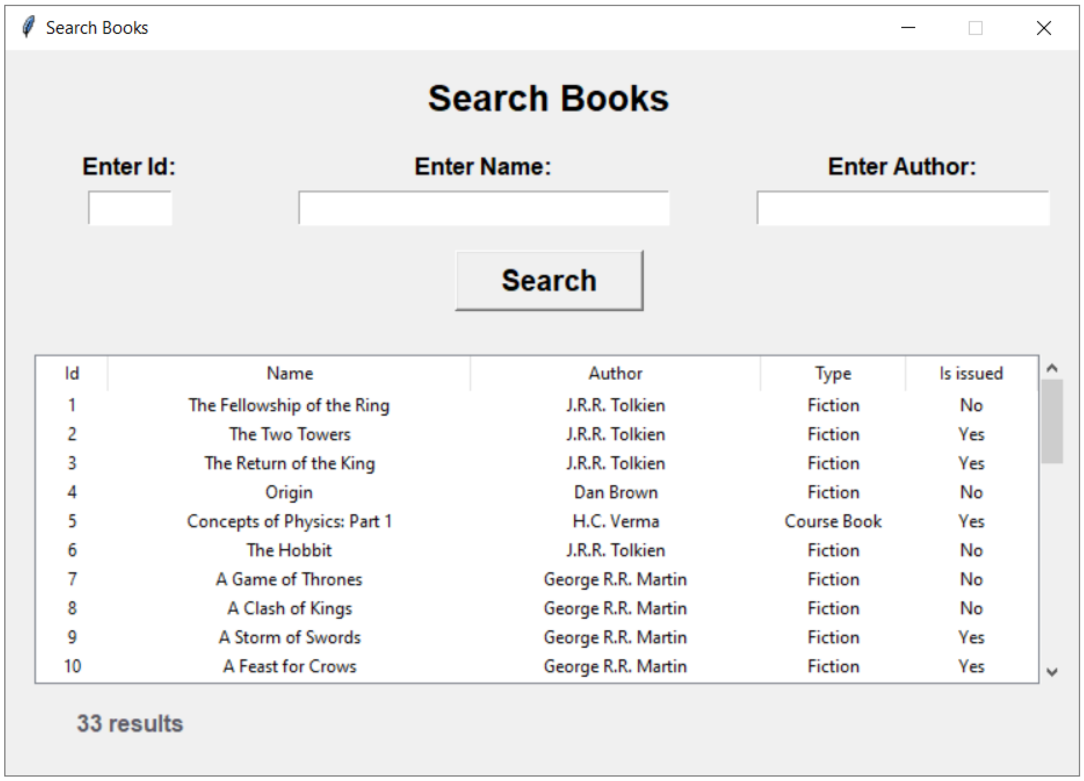
Manage Books

Issue & Return Books

Manage Members

Manage Books

1. Search for Books



Search Books

Search Books

Enter Id:

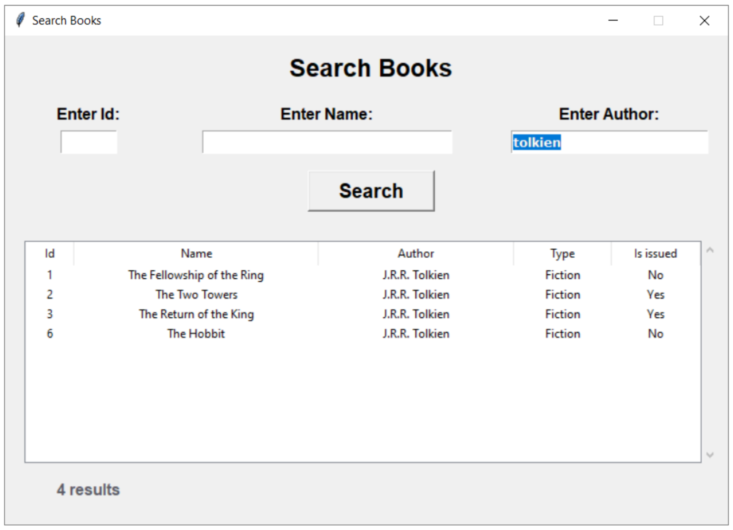
Enter Name:

Enter Author:

Id	Name	Author	Type	Is issued
1	The Fellowship of the Ring	J.R.R. Tolkien	Fiction	No
2	The Two Towers	J.R.R. Tolkien	Fiction	Yes
3	The Return of the King	J.R.R. Tolkien	Fiction	Yes
4	Origin	Dan Brown	Fiction	No
5	Concepts of Physics: Part 1	H.C. Verma	Course Book	Yes
6	The Hobbit	J.R.R. Tolkien	Fiction	No
7	A Game of Thrones	George R.R. Martin	Fiction	No
8	A Clash of Kings	George R.R. Martin	Fiction	No
9	A Storm of Swords	George R.R. Martin	Fiction	Yes
10	A Feast for Crows	George R.R. Martin	Fiction	Yes

33 results

With different search strings:


Search Books

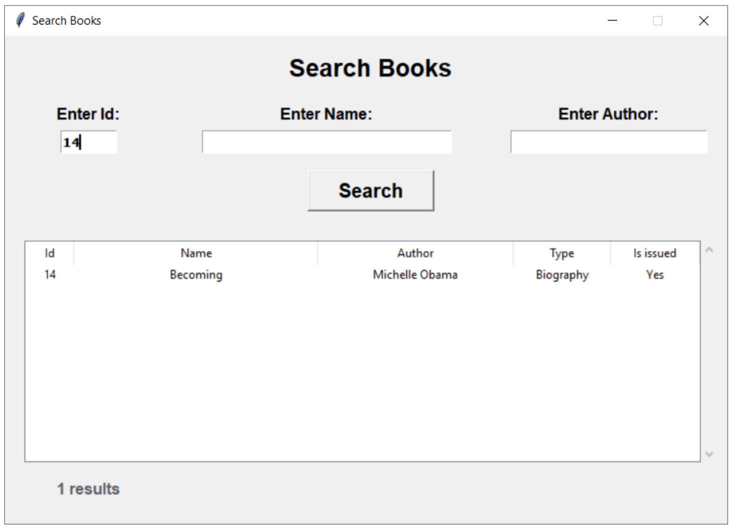
Enter Id:

Enter Name:

Enter Author: tolkien

Id	Name	Author	Type	Is issued
1	The Fellowship of the Ring	J.R.R. Tolkien	Fiction	No
2	The Two Towers	J.R.R. Tolkien	Fiction	Yes
3	The Return of the King	J.R.R. Tolkien	Fiction	Yes
6	The Hobbit	J.R.R. Tolkien	Fiction	No

4 results


Search Books

Enter Id: 14

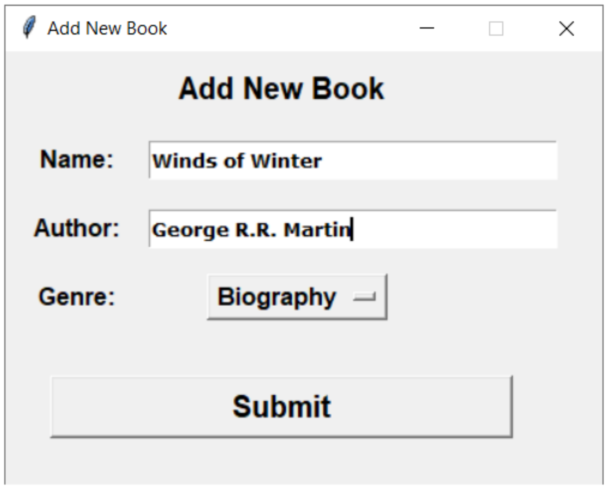
Enter Name:

Enter Author:

Id	Name	Author	Type	Is issued
14	Becoming	Michelle Obama	Biography	Yes

1 results

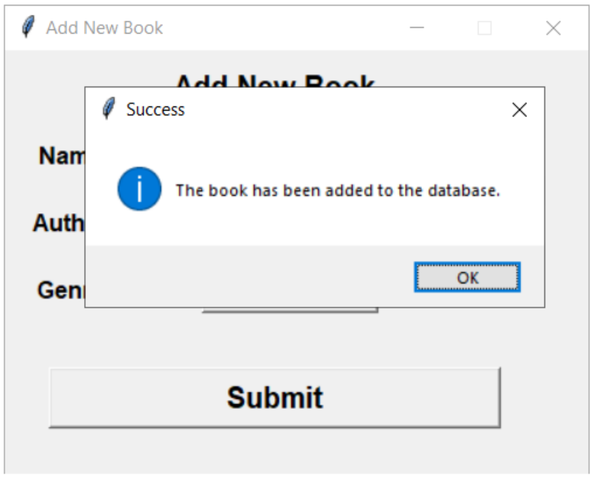
2. Add New Book


Add New Book

Name: Winds of Winter

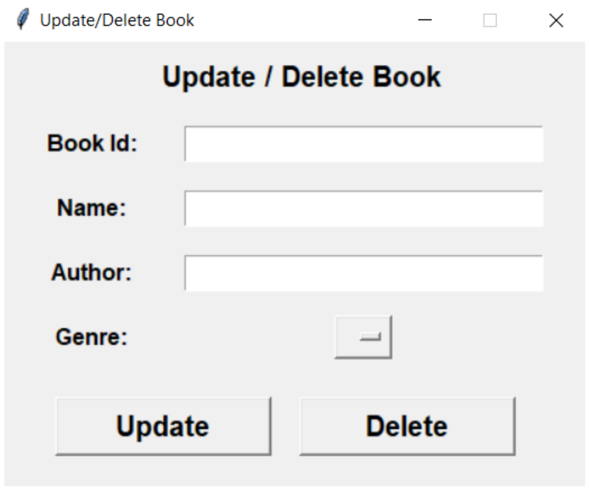
Author: George R.R. Martin

Genre: Biography


Success

The book has been added to the database.

3. Update Book

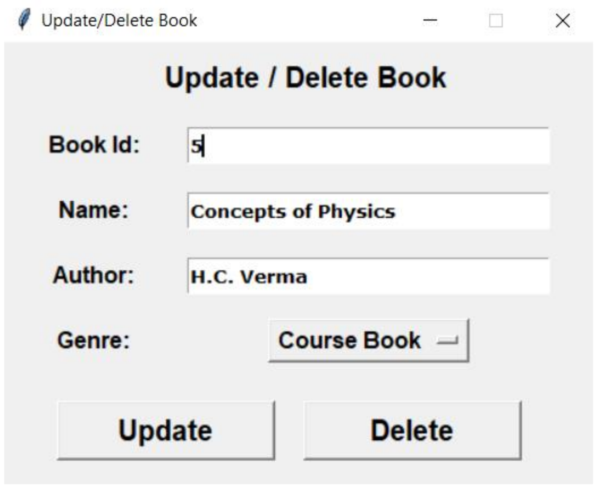

Update / Delete Book

Book Id:

Name:

Author:

Genre:

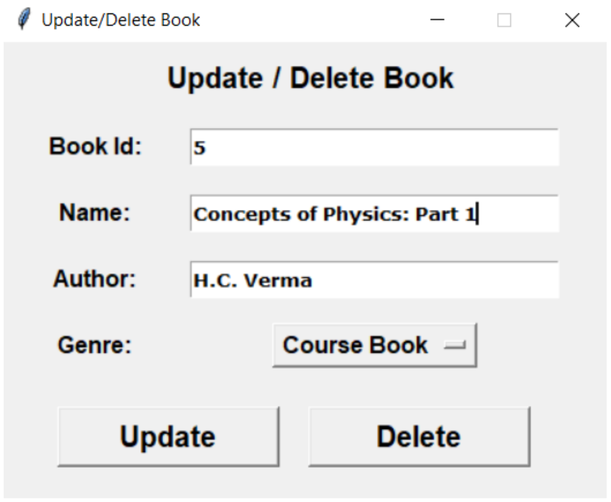

Update / Delete Book

Book Id: 5

Name: Concepts of Physics

Author: H.C. Verma

Genre: Course Book


Update/Delete Book

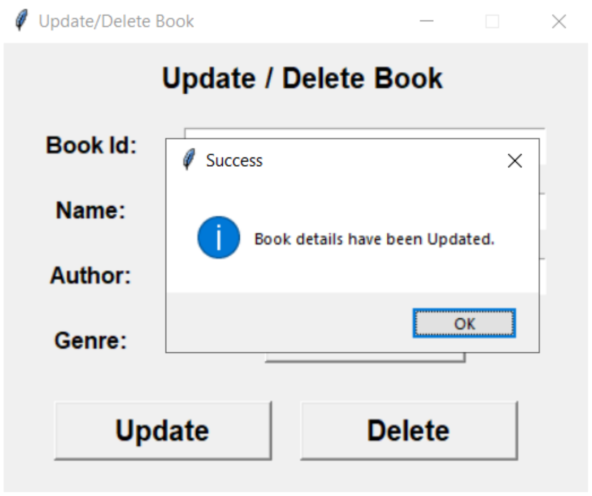
Update / Delete Book

Book Id: 5

Name: Concepts of Physics: Part 1

Author: H.C. Verma

Genre: Course Book

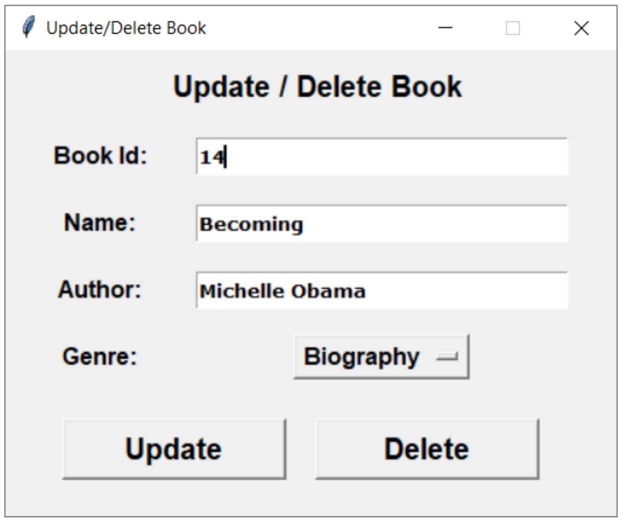

Update/Delete Book

Update / Delete Book

Success

Book details have been Updated.

4. Delete Book


Update/Delete Book

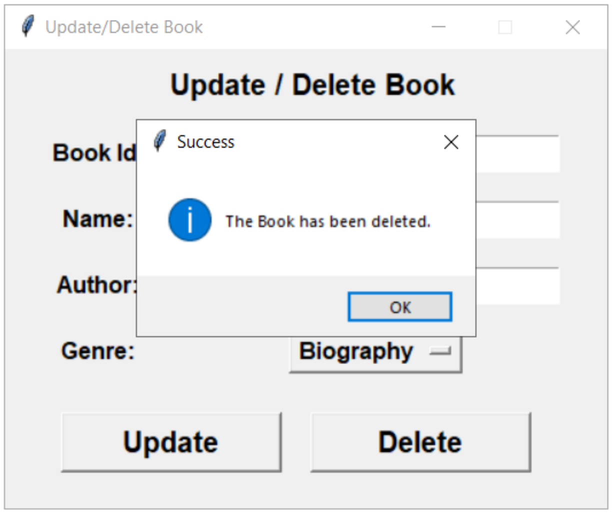
Update / Delete Book

Book Id: 14

Name: Becoming

Author: Michelle Obama

Genre: Biography


Update/Delete Book

Update / Delete Book

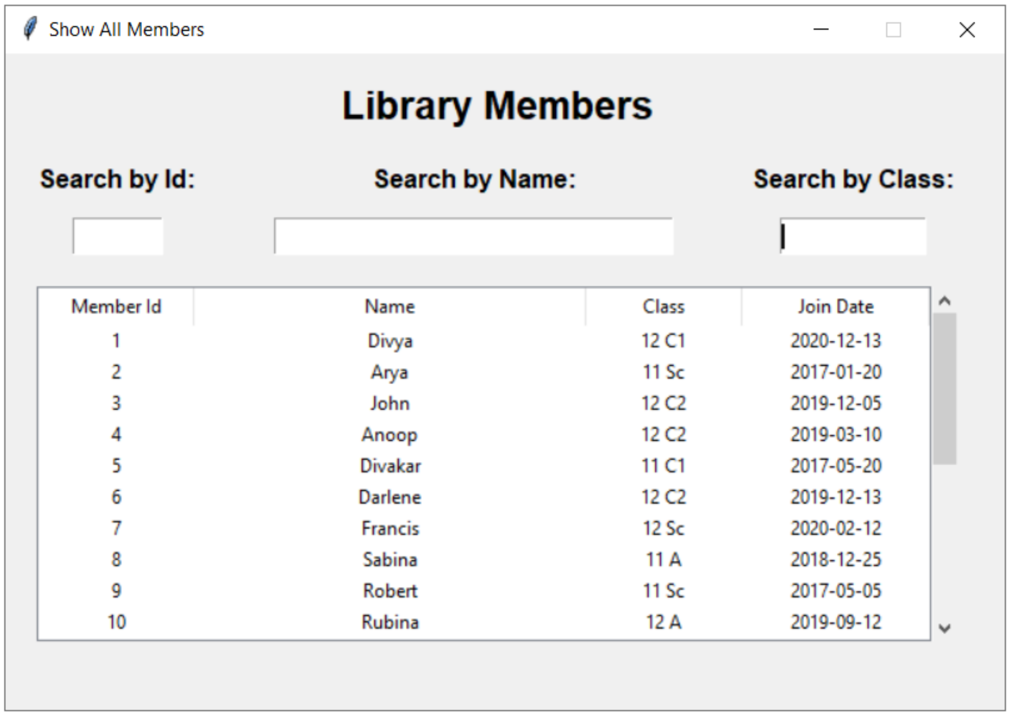
Success

The Book has been deleted.

Genre: Biography

Manage Members

1. Show all members



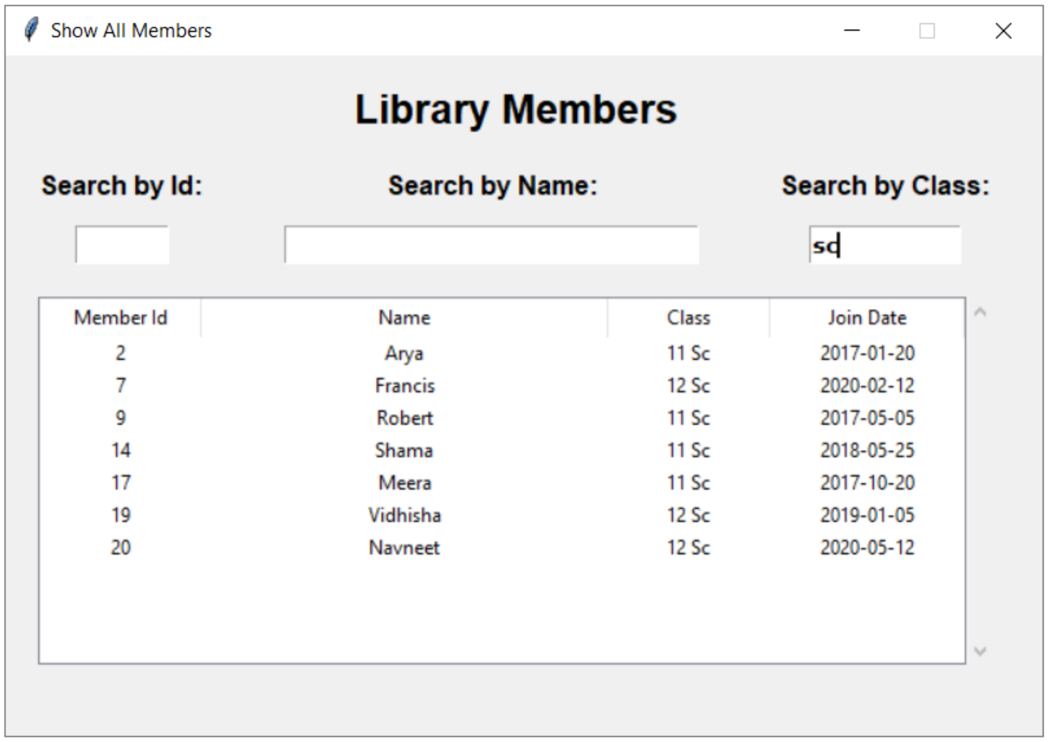
Show All Members

Library Members

Search by Id: | Search by Name: | Search by Class:

Member Id	Name	Class	Join Date
1	Divya	12 C1	2020-12-13
2	Arya	11 Sc	2017-01-20
3	John	12 C2	2019-12-05
4	Anoop	12 C2	2019-03-10
5	Divakar	11 C1	2017-05-20
6	Darlene	12 C2	2019-12-13
7	Francis	12 Sc	2020-02-12
8	Sabina	11 A	2018-12-25
9	Robert	11 Sc	2017-05-05
10	Rubina	12 A	2019-09-12

With search strings:



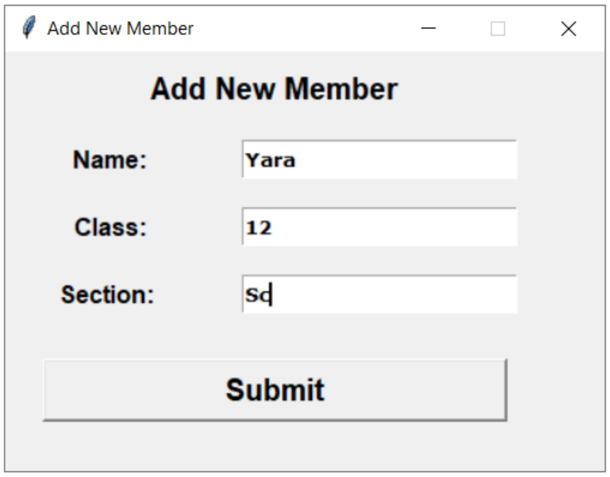
Show All Members

Library Members

Search by Id: | Search by Name: | Search by Class: sc

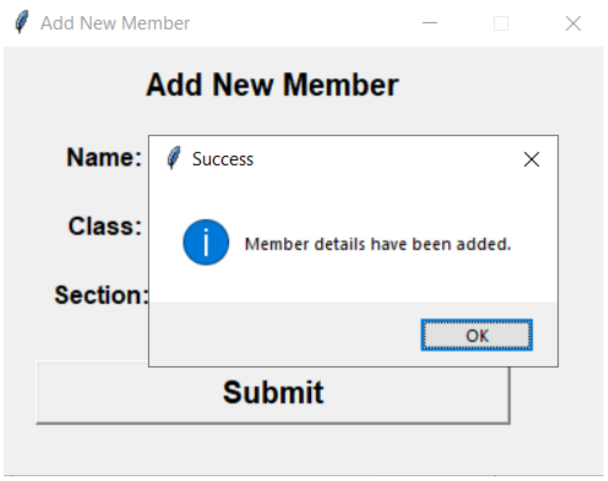
Member Id	Name	Class	Join Date
2	Arya	11 Sc	2017-01-20
7	Francis	12 Sc	2020-02-12
9	Robert	11 Sc	2017-05-05
14	Shama	11 Sc	2018-05-25
17	Meera	11 Sc	2017-10-20
19	Vidhisha	12 Sc	2019-01-05
20	Navneet	12 Sc	2020-05-12

2. Add New member


Add New Member

Add New Member

Name:	Yara
Class:	12
Section:	sc

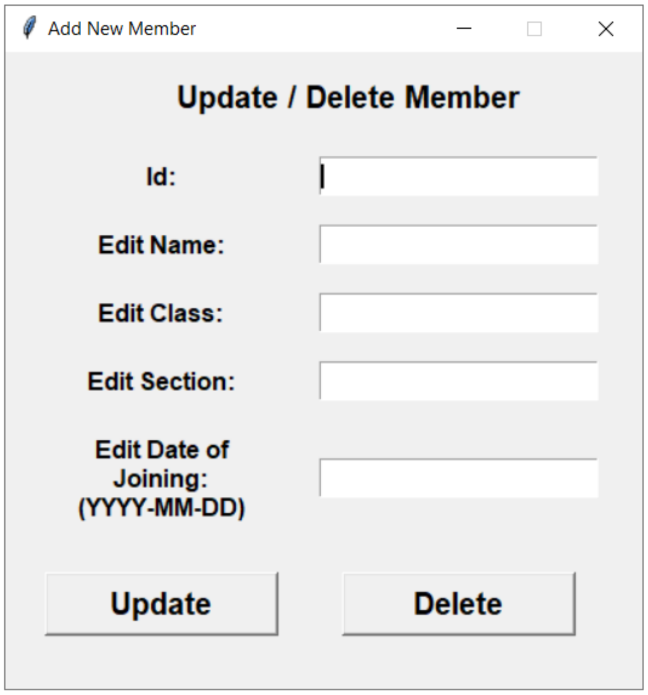

Add New Member

Add New Member

Success

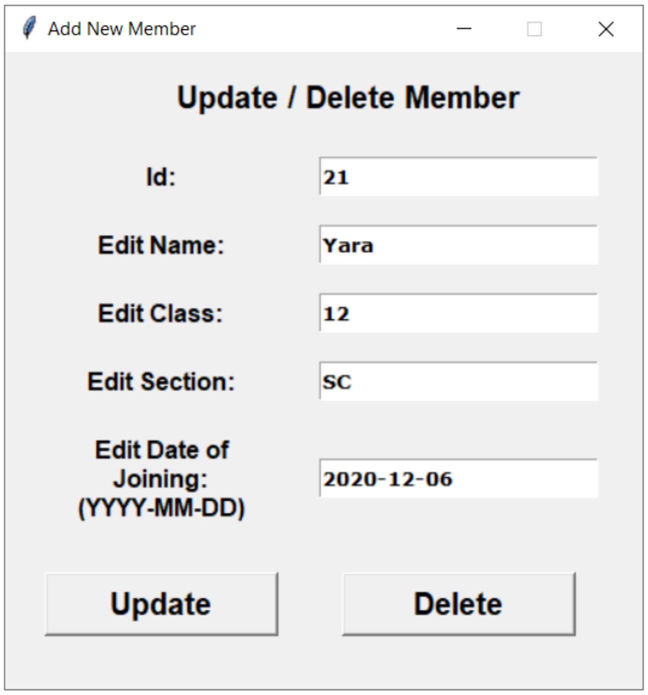
Member details have been added.

3. Update Member Details


Add New Member

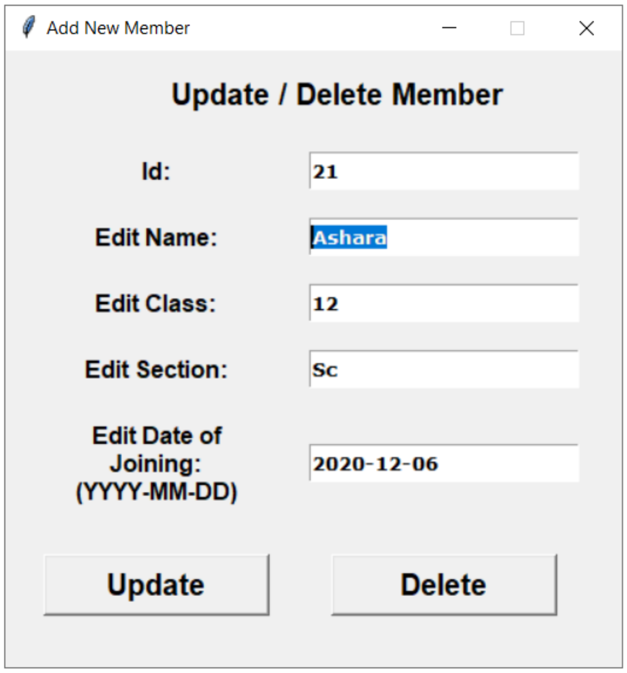
Update / Delete Member

Id:	
Edit Name:	
Edit Class:	
Edit Section:	
Edit Date of Joining: (YYYY-MM-DD)	


Add New Member

Update / Delete Member

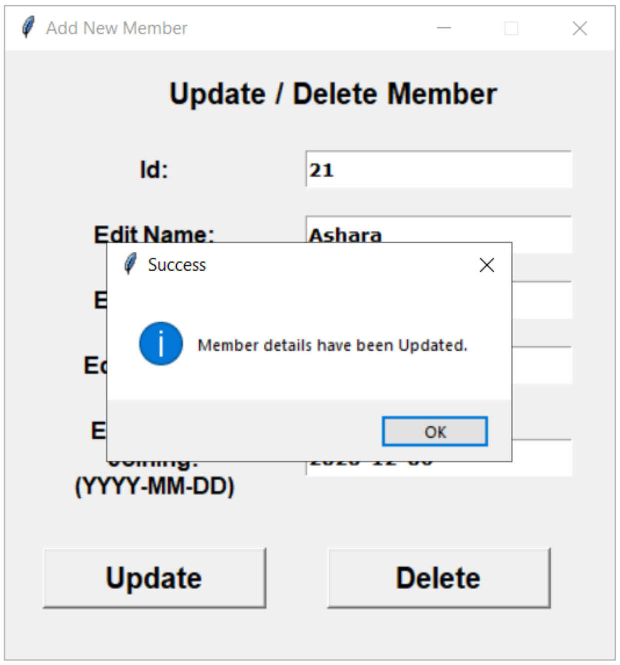
Id:	21
Edit Name:	Yara
Edit Class:	12
Edit Section:	SC
Edit Date of Joining: (YYYY-MM-DD)	2020-12-06



Add New Member

Update / Delete Member

Id:	21
Edit Name:	Ashara
Edit Class:	12
Edit Section:	Sc
Edit Date of Joining: (YYYY-MM-DD)	2020-12-06



Add New Member

Update / Delete Member

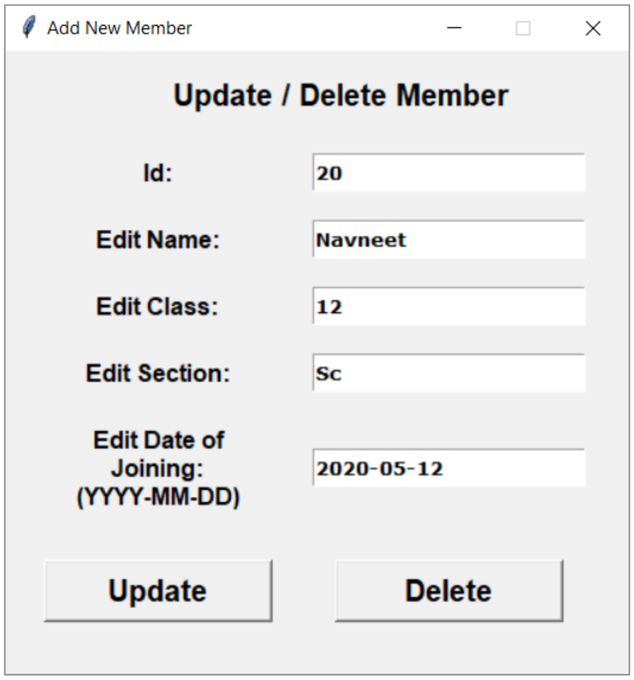
Id: 21

Edit Name: Ashara

Success

Member details have been Updated.

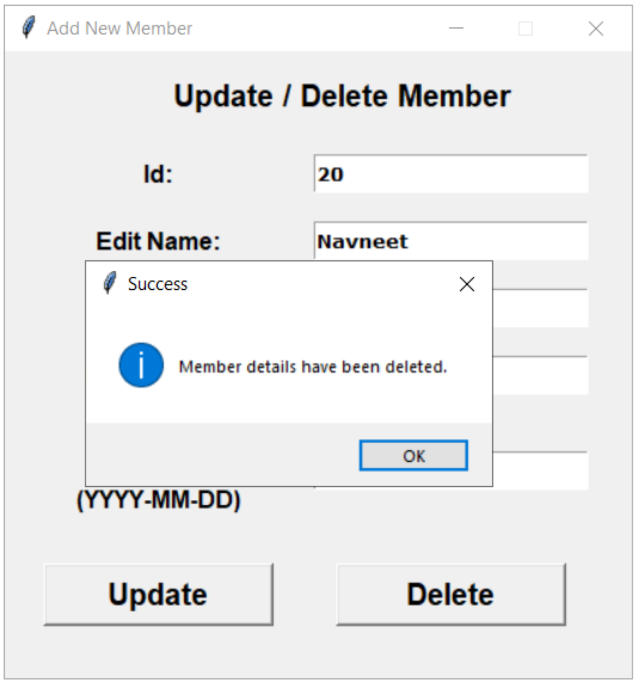
4. Delete a member



Add New Member

Update / Delete Member

Id:	20
Edit Name:	Navneet
Edit Class:	12
Edit Section:	Sc
Edit Date of Joining: (YYYY-MM-DD)	2020-05-12



Add New Member

Update / Delete Member

Id: 20

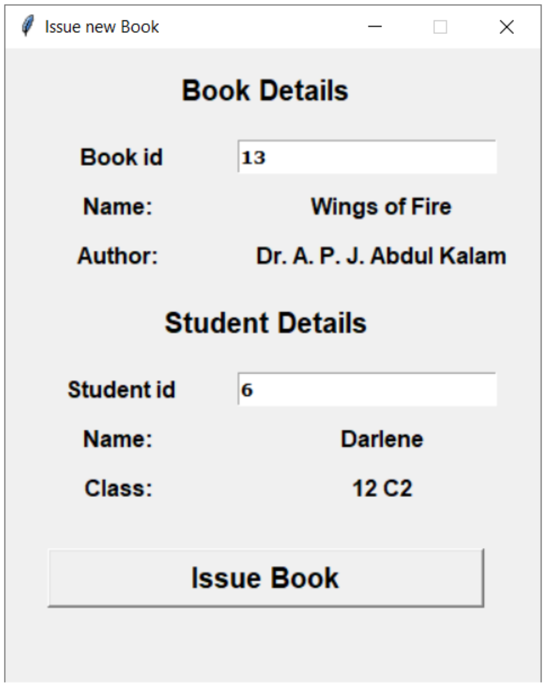
Edit Name: Navneet

Success

Member details have been deleted.

Issue & Return Books

1. Issue a book

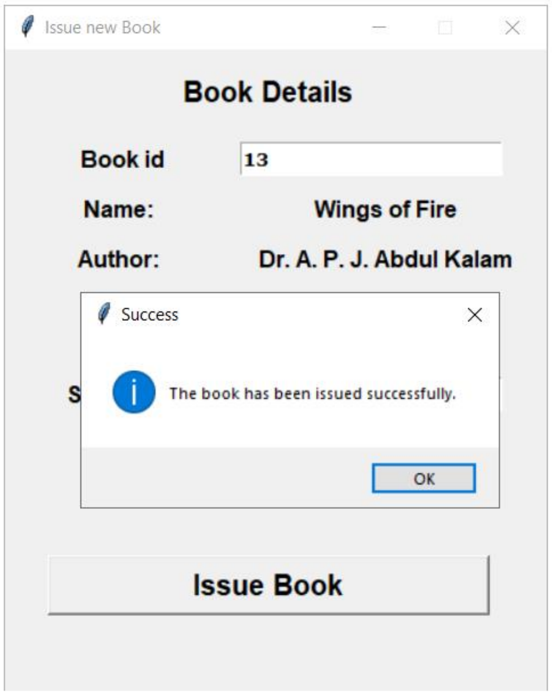

Issue new Book

Book Details

Book id	13
Name:	Wings of Fire
Author:	Dr. A. P. J. Abdul Kalam

Student Details

Student id	6
Name:	Darlene
Class:	12 C2


Issue new Book

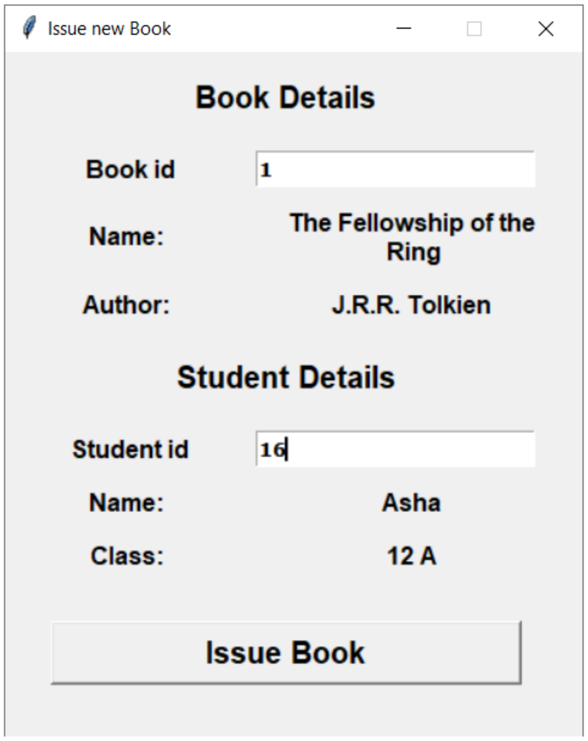
Book Details

Book id	13
Name:	Wings of Fire
Author:	Dr. A. P. J. Abdul Kalam

Success

The book has been issued successfully.

NOTE: If the book is already issued by someone,

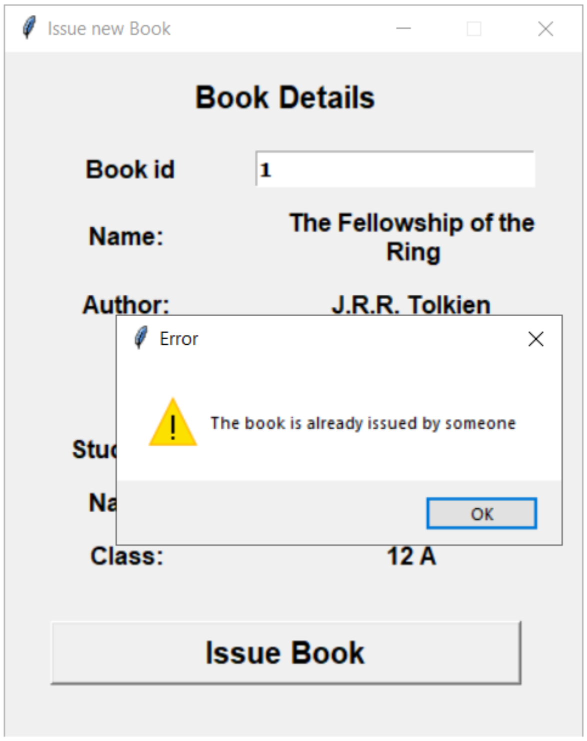

Issue new Book

Book Details

Book id	1
Name:	The Fellowship of the Ring
Author:	J.R.R. Tolkien

Student Details

Student id	16
Name:	Asha
Class:	12 A


Issue new Book

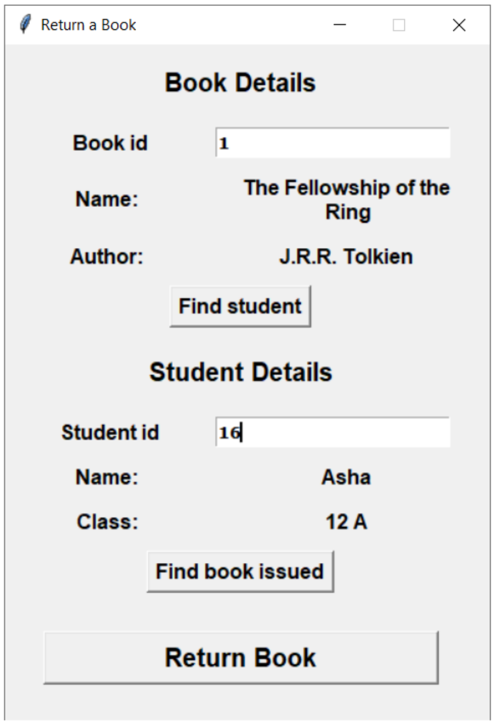
Book Details

Book id	1
Name:	The Fellowship of the Ring
Author:	J.R.R. Tolkien

Error

The book is already issued by someone

2. Return a book


Return a Book

Book Details

Book id: 1

Name: The Fellowship of the Ring

Author: J.R.R. Tolkien

Find student

Student Details

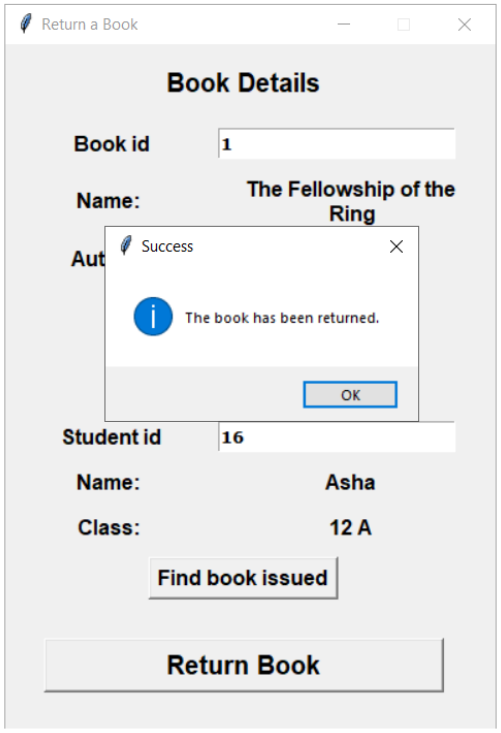
Student id: 16

Name: Asha

Class: 12 A

Find book issued

Return Book


Return a Book

Book Details

Book id: 1

Name: The Fellowship of the Ring

Success

The book has been returned.

OK

Student id: 16

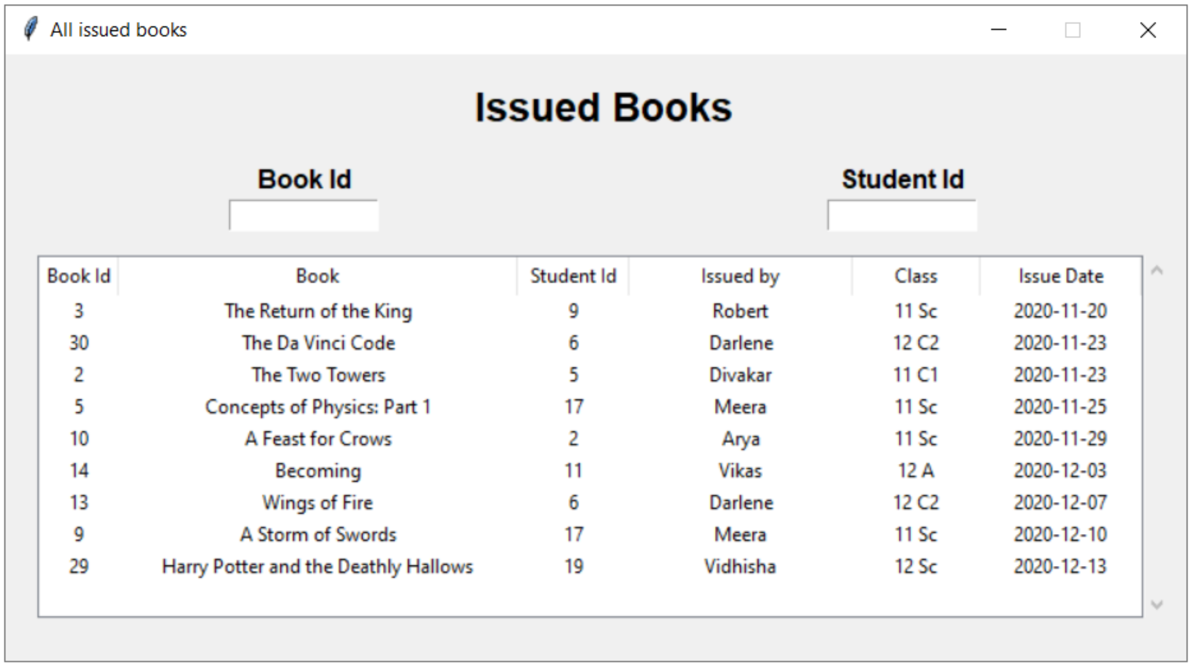
Name: Asha

Class: 12 A

Find book issued

Return Book

3. Show all issued books


All issued books

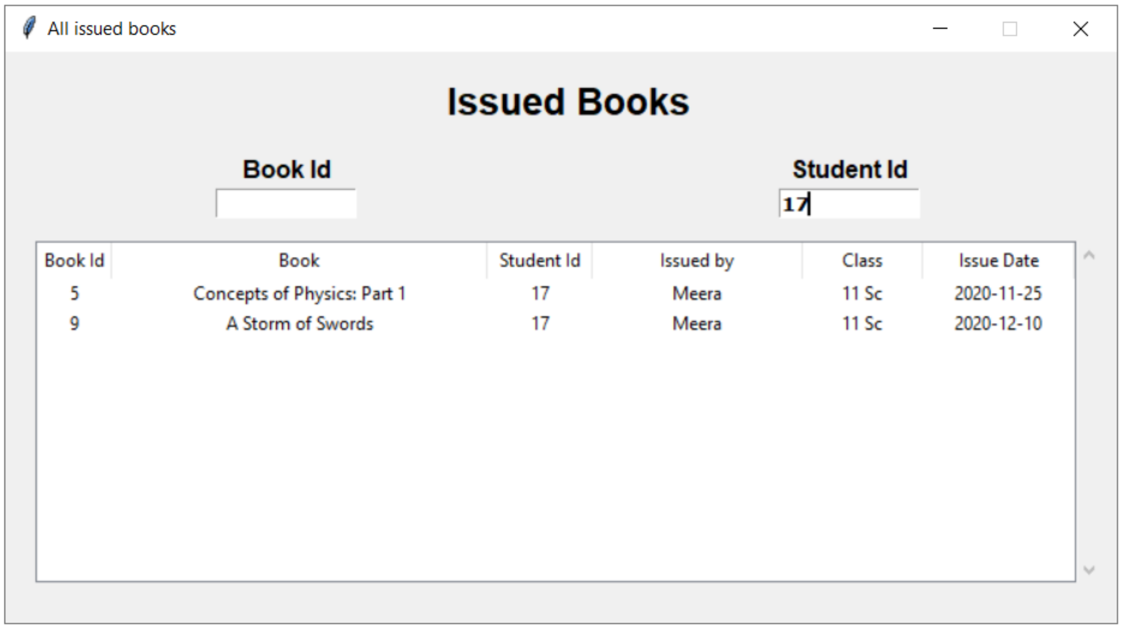
Issued Books

Book Id

Student Id

Book Id	Book	Student Id	Issued by	Class	Issue Date
3	The Return of the King	9	Robert	11 Sc	2020-11-20
30	The Da Vinci Code	6	Darlene	12 C2	2020-11-23
2	The Two Towers	5	Divakar	11 C1	2020-11-23
5	Concepts of Physics: Part 1	17	Meera	11 Sc	2020-11-25
10	A Feast for Crows	2	Arya	11 Sc	2020-11-29
14	Becoming	11	Vikas	12 A	2020-12-03
13	Wings of Fire	6	Darlene	12 C2	2020-12-07
9	A Storm of Swords	17	Meera	11 Sc	2020-12-10
29	Harry Potter and the Deathly Hallows	19	Vidhisha	12 Sc	2020-12-13

With search strings:



Issued Books

Book Id

Student Id

17

Book Id	Book	Student Id	Issued by	Class	Issue Date
5	Concepts of Physics: Part 1	17	Meera	11 Sc	2020-11-25
9	A Storm of Swords	17	Meera	11 Sc	2020-12-10