



CMR INSTITUTE OF TECHNOLOGY

(UGC AUTONOMOUS)

**(Approved by AICTE, Affiliated to JNTU, Kukatpally, Hyderabad)
Kandlakoya, Medchal Road, Hyderabad**

2022-2023

**Department of Electronics and
Communication Engineering**

A Micro Project report on

**PYTHON PROGRAM TO IMPLEMENT A CLASS FOR LIBRARY AND INCLUDE
FUNCTIONS REQUIRED FOR IT**

Submitted to the CMR Institute of Technology in partial fulfillment of the requirement
for the award of the Laboratory of

SCRIPTING LANGUAGES LAB

Of

III-B.Tech. I-Semester

in

Electronics and Communication Engineering Department

Submitted By

S.SAI RAM	21R05A0401
A.SAI RAM	21R05A0405
P.LINGAM REDDY	21R05A0403
CH. DHEERAJ	21R05A0404
G.VISHNU TEJA	21R05A0405
B.SANDEEP	21R05A0406

Under the Guidance Of

Mr.A.Shiva Prasad



CMR INSTITUTE OF TECHNOLOGY
(UGC AUTONOMOUS)

(Approved by AICTE, Affiliated to JNTU, Kukatpally, Hyderabad)
Kandlakoya, Medchal Road, Hyderabad

2022-2023

CMR INSTITUTE OF TECHNOLOGY

(UGC AUTONOMOUS)

(Approved by AICTE, Affiliated to JNTU, Kukatpally, Hyderabad) Kandlakoya, Medchal
Road, Hyderabad.

Department of Electronics and Communication Engineering

DECLARATION

We hereby declare that the micro project report entitled **“PYTHON PROGRAM TO IMPLEMENT A CLASS FOR LIBRARY AND INCLUDE FUNCTIONS REQUIRED FOR IT”** is carried out by us during the academic year 2022-2023 in partial fulfilment of the award of Bachelor of Technology in Electronics and Communication Engineering from CMR Institute of Technology affiliated to Jawaharlal Nehru Technological University Hyderabad. We have not submitted the same to any other university or organization for the award of any other degree.

S.SAI RAM	21R05A0401
A.SAI RAM	21R05A0405
P.LINGAM REDDY	21R05A0403
CH. DHEERAJ	21R05A0404
G.VISHNU TEJA	21R05A0405
B.SANDEEP	21R05A0406



CMR INSTITUTE OF TECHNOLOGY

(UGC AUTONOMOUS)

(Approved by AICTE, Affiliated to JNTU, Kukatpally, Hyderabad) Kandlakoya,
Medchal Road, Hyderabad.

Department of Electronics and Communication

Engineering

CERTIFICATE

This is to certify that a Micro Project entitled with: **“PYTHON PROGRAM TO IMPLEMENT A CLASS FOR LIBRARY AND INCLUDE FUNCTIONS REQUIRED FOR IT**

” is being Submitted By

S.SAI RAM	21R05A0401
A.SAI RAM	21R05A0405
P.LINGAM REDDY	21R05A0403
CH. DHEERAJ	21R05A0404
G.VISHNU TEJA	21R05A0405
B.SANDEEP	21R05A0406

In partial fulfillment of the requirement for award of the of III-B.Tech I- Semester in ECE towards a record of “SCRIPTING LANGUAGES LAB” a bonafide work carried out under our guidance and supervision.

Signature of Faculty

Signature of HOD

ABSTRACT:

Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods. A Class is like an object constructor, or a "blueprint" for creating objects.

A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by their class) for modifying their state

A library management system keeps track of the books present in the library. It is an important piece of software which is a must at schools and colleges. We will build a library management system using Tkinter to make it interactive.

What is Tkinter?

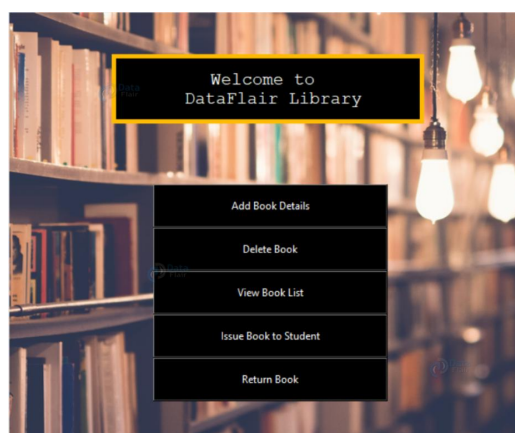
Python offers various utilities to design the GUI viz Graphical User Interface, and one such utility is Tkinter which is most commonly used. It is indeed one of the fastest and easiest ways to build GUI applications. Moreover, Tkinter is cross-platform, hence the same code works on macOS, Windows, and Linux.

INTRODUCTION:

A library management system keeps track of the books present in the library. It is an important piece of software which is a must at schools and colleges.

Python is the most popular programming language on the planet. Python is an object-oriented language, which means that it can model real-world entities. It is dynamically-typed because it carries out type-checking at runtime.

The library management system in python which we are going to build will look something like this.



Yes, this is the library management project which we are going to build. I know this may be a long article but be with me as at the end of the article you will be equipped with the knowledge to design and build great applications with decent UI.

Project Prerequisites

tkinter – Please run below command to install tkinter

pillow – Please run below command to install tkinter

pip install pillow

pymysql – Please run below command to install tkinter

```
pip install pymysql
```

Description of Project Files

Below are the project files you will get once you download and extract the Library project:

- **main.py** – which does function call to all other python files
- **AddBook.py** – To add the book
- **ViewBooks.py** – To View the list of books in the library
- **DeleteBook.py** – To Delete a book from library
- **IssueBook.py** – To Issue a book from library
- **ReturnBook.py** – To Return a book to the library

Description of Tables

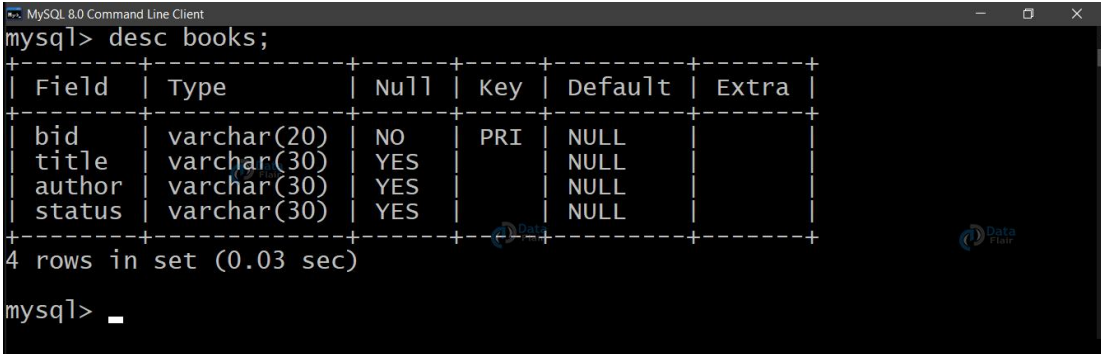
Create Tables

```
create database db;
```

```
create table books(bid varchar(20) primary key, title varchar(30), author varchar(30), status varchar(30));
```

```
create table books_issued(bid varchar(20) primary key, issuedto varchar(30));
```

1. books

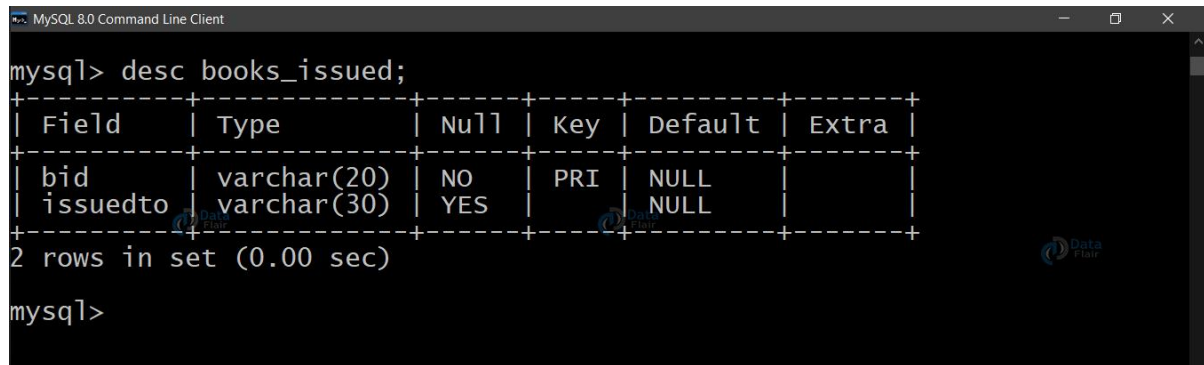


```
mysql> desc books;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| bid   | varchar(20)   | NO   | PRI | NULL    |       |
| title | varchar(30)   | YES  |     | NULL    |       |
| author | varchar(30)   | YES  |     | NULL    |       |
| status | varchar(30)   | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.03 sec)

mysql> _
```

2.

issued_books



```
mysql> desc books_issued;
```

Field	Type	Null	Key	Default	Extra
bid	varchar(20)	NO	PRI	NULL	
issuedto	varchar(30)	YES		NULL	

```
2 rows in set (0.00 sec)

mysql>
```

Library Project Code :

Let's start the detailed discussion of each and every file of our library management system python project:

1. main.py

Importing the Modules

To use the Tkinter we need to import the Tkinter module. As stated above, we have imported each file so that we can make function calls from our main file.

Code:

```
from tkinter import *
from PIL import ImageTk,Image #PIL -> Pillow
import pymysql
from tkinter import messagebox
from AddBook import *
from DeleteBook import *
from ViewBooks import *
from IssueBook import *
```

Connecting to the MySql server

Now we will connect to the server with the correct credentials associated with the MySql server installed on our system.

Code:

```
mypass = "root" #use your own password
mydatabase="db" #The database name
con = pymysql.connect
(host="localhost",user="root",password=mypass,database=mydatabase)
#root is the username here
cur = con.cursor() #cur -> cursor
```

Designing the Window

Now we will design the project window and add a background image. Make sure to keep the image in the same directory as the project is in order to avoid discrepancies.

Code:

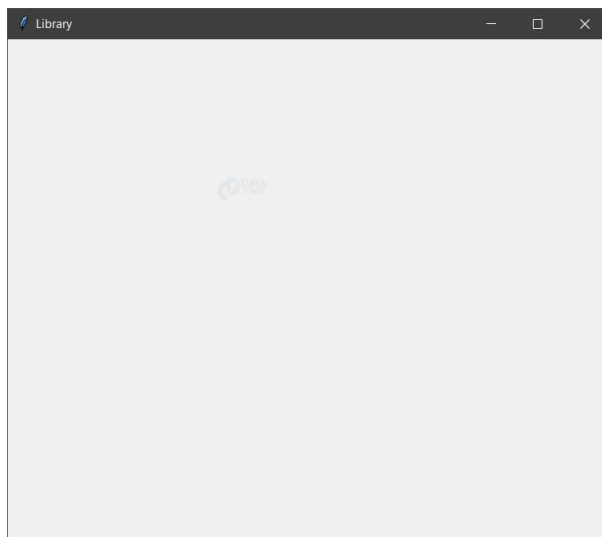
Code:

```
root = Tk()
root.title("Library")
root.minsize(width=400,height=400)
root.geometry("600x500")
```

Explanation:

The above code sets the title of the library project window as 'Library'. When you run the above code, you will get a window.

Output:



Adding a Background Image

Code:

```
same=True
n=0.25
# Adding a background image
background_image = Image.open("lib.jpg")
[imageSizeWidth, imageSizeHeight] = background_image.size
newImageSizeWidth = int(imageSizeWidth*n)
if same:
    newImageSizeHeight = int(imageSizeHeight*n)
else:
    newImageSizeHeight = int(imageSizeHeight/n)
```



```
background_image =
background_image.resize((newImageSizeWidth,newImageSizeHeight),Image.ANTIALIAS)
img = ImageTk.PhotoImage(background_image)
Canvas1 = Canvas(root)
Canvas1.create_image(300,340,image = img)
Canvas1.config(bg="white",width = newImageSizeWidth, height = newImageSizeHeight)
Canvas1.pack(expand=True,fill=BOTH)
```

Explanation:

We store our image in **background_image** with the help of **.open()** method. We fetch the image dimensions and adjust the image size according to our window size.

newImageHeight and **newImageWidth** contains the adjusted image dimensions.

Now we resize the image using **.resize()** method using the new dimensions.

The **.PhotoImage()** method is used to display images (either grayscale or true color images) in labels, buttons, canvases, and text widgets.

We create the image on the **canvas1** using **.create_image()** method. We use **.pack()** method to organize widgets in blocks before placing them in the parent widget.

Setting up the Head Frame

Code:

```
headingFrame1 = Frame(root,bg="#FFBB00",bd=5)
headingFrame1.place(relx=0.2,relx=0.1,relwidth=0.6,relheight=0.16)
headingLabel = Label(headingFrame1, text="Welcome to \n DataFlair Library", bg='black',
fg='white', font=('Courier',15))
headingLabel.place(relx=0,relx=0, relwidth=1, relheight=1)
```

Explanation:

We create a frame that will hold our Label viz **headingLabel**. We increase the size and alter the font by passing one more parameter in the **Label** method viz **font**.

Adding the Buttons

Code:

```
btn1 = Button(root,text="Add Book Details",bg='black', fg='white', command=addBook)
btn1.place(relx=0.28,relx=0.4, relwidth=0.45,relheight=0.1)
btn2 = Button(root,text="Delete Book",bg='black', fg='white', command=delete)
btn2.place(relx=0.28,relx=0.5, relwidth=0.45,relheight=0.1)
btn3 = Button(root,text="View Book List",bg='black', fg='white', command=View)
```

```
btn3.place(relx=0.28,relx=0.6, relwidth=0.45,relheight=0.1)
```

```
btn4 = Button(root,text="Issue Book to Student",bg='black', fg='white', command =  
issueBook)
```

```
btn4.place(relx=0.28,relx=0.7, relwidth=0.45,relheight=0.1)
```

```
btn5 = Button(root,text="Return Book",bg='black', fg='white', command = returnBook)
```

```
btn5.place(relx=0.28,relx=0.8, relwidth=0.45,relheight=0.1)
```

```
root.mainloop()
```

Explanation:

The above code adds buttons to the window frame.

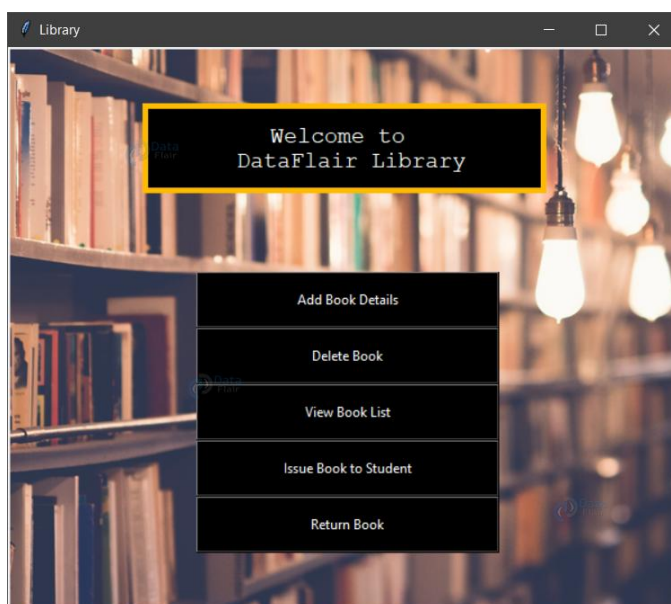
AddBook Details :

btn1 stores the button created on **root** with **text** = '**AddBook Details**'. As soon as someone clicks this button, we call the function **addBook** defined in the **AddBook.py**. We call a function by specifying the **command** parameter equal to the **name of the function**.

We place this button using the **.place()** method by defining the position as well as dimensions of the button.

Similarly, we define other buttons using the **Button** method and keep placing them by making minor changes in the **rely** parameter. You can notice that we are increasing it by 0.1 every time we define a new button.

Output:



2. AddBook.py



2.1. Importing the necessary modules

Code:

```
from tkinter import *
from PIL import ImageTk,Image
from tkinter import messagebox
import pymysql
```

2.2. Function – bookRegister()

This function executes an SQL command to insert data into the table and commit the changes.

Code:

```
def bookRegister():

    bid = bookInfo1.get()

    title = bookInfo2.get()

    author = bookInfo3.get()

    status = bookInfo4.get()

    status = status.lower()

    insertBooks = "insert into "+bookTable+" values
    ('"+bid+"','"+title+"','"+author+"','"+status+"')"

    try:

        cur.execute(insertBooks)

        con.commit()
```

```

messagebox.showinfo('Success','Book added successfully')

except:

messagebox.showinfo("Error","Can't add data into Database")

print(bid)

print(title)

print(author)

print(status)

root.destroy()

```

Explanation:

We fetch the data in the input field by **.get()** method. Hence after fetching each of the data fields value we are ready to execute an SQL command to insert the data.

General Syntax:

insert into <Table Name> values (<the values of the respective columns>)

Modified :

```

insertBooks = "insert into "+bookTable+" values
('"+bid+"','"+title+"','"+author+"','"+status+"')"

```

We put this code in a try-except block in order to handle the situation effectively.

Now, we execute the **insertBooks** command by **.execute()** method associated with cur. We commit the changes by **.commit()** method associated with con as discussed above.

We use **messagebox()** function to acknowledge the user of success or failure.

2.3. Function – addBook()

This function connects to the MySQL server and creates a window for accommodating new text fields. We fetch details of a new book from the user and then call **bookRegister()** function to register the books into the table.

Code:

```

def addBook():
global bookInfo1 ,bookInfo2, bookInfo3, bookInfo4, Canvas1, con, cur, bookTable, root
root = Tk()
root.title("Library")
root.minsize(width=400,height=400)
root.geometry("600x500")
mypass = "root"
mydatabase="db"

```

```

con =
pymysql.connect( host="localhost",user="root",password=mypass,database=mydatabase)
cur = con.cursor()
# Enter Table Names here
bookTable = "books" # Book Table
Canvas1 = Canvas(root)
Canvas1.config(bg="#ff6e40")
Canvas1.pack(expand=True,fill=BOTH)
headingFrame1 = Frame(root,bg="#FFBB00",bd=5)
headingFrame1.place(relx=0.25,rely=0.1,relwidth=0.5,relheight=0.13)
headingLabel = Label(headingFrame1, text="Add Books", bg='black', fg='white',
font=('Courier',15))
headingLabel.place(relx=0,rely=0, relwidth=1, relheight=1)
labelFrame = Frame(root,bg='black')
labelFrame.place(relx=0.1,rely=0.4,relwidth=0.8,relheight=0.4)
# Book ID
lb1 = Label(labelFrame,text="Book ID : ", bg='black', fg='white')
lb1.place(relx=0.05,rely=0.2, relheight=0.08)
bookInfo1 = Entry(labelFrame)
bookInfo1.place(relx=0.3,rely=0.2, relwidth=0.62, relheight=0.08)
# Title
lb2 = Label(labelFrame,text="Title : ", bg='black', fg='white')
lb2.place(relx=0.05,rely=0.35, relheight=0.08)
bookInfo2 = Entry(labelFrame)
bookInfo2.place(relx=0.3,rely=0.35, relwidth=0.62, relheight=0.08)
# Book Author
lb3 = Label(labelFrame,text="Author : ", bg='black', fg='white')
lb3.place(relx=0.05,rely=0.50, relheight=0.08)
bookInfo3 = Entry(labelFrame)
bookInfo3.place(relx=0.3,rely=0.50, relwidth=0.62, relheight=0.08)
# Book Status
lb4 = Label(labelFrame,text="Status(Avail/issued) : ", bg='black', fg='white')
lb4.place(relx=0.05,rely=0.65, relheight=0.08)
bookInfo4 = Entry(labelFrame)
bookInfo4.place(relx=0.3,rely=0.65, relwidth=0.62, relheight=0.08)
#Submit Button
SubmitBtn = Button(root,text="SUBMIT",bg='#d1ccc0', fg='black',command=bookRegister)
SubmitBtn.place(relx=0.28,rely=0.9, relwidth=0.18,relheight=0.08)
quitBtn = Button(root,text="Quit",bg='#f7f1e3', fg='black', command=root.destroy)
quitBtn.place(relx=0.53,rely=0.9, relwidth=0.18,relheight=0.08)
root.mainloop()

```

Variables

- **bookInfo1** – contains book ID
- **bookInfo2** – contains Title of the book
- **bookInfo3** – contains Author of the book
- **bookInfo4** – contains status of the book (available or issued)
- **Con** – MySql console
- **Cur** – cursor of the console

Buttons

- **Submit** – to commit the changes
- **Exit**

Explanation:

We declare certain variables as global in order to use them in the **bookRegister()** function.

We create another window and connect to the MySQL server and pass on the cursor control to **cur**. It means that whatever we want to write on the MySQL shell, we will do it through **cur**. In order to **commit(write changes in the table)** the changes we will use **con**.

We draw the canvas and paint it with a good color. I have used **Orange**, you can use any color you like.

We create and place the **headingLabel** inside the **headingFrame1** and give the title as “**Add Books**”.

The **labelFrame** basically creates a **black box** (in our case) to accommodate the input fields to fetch the book details.

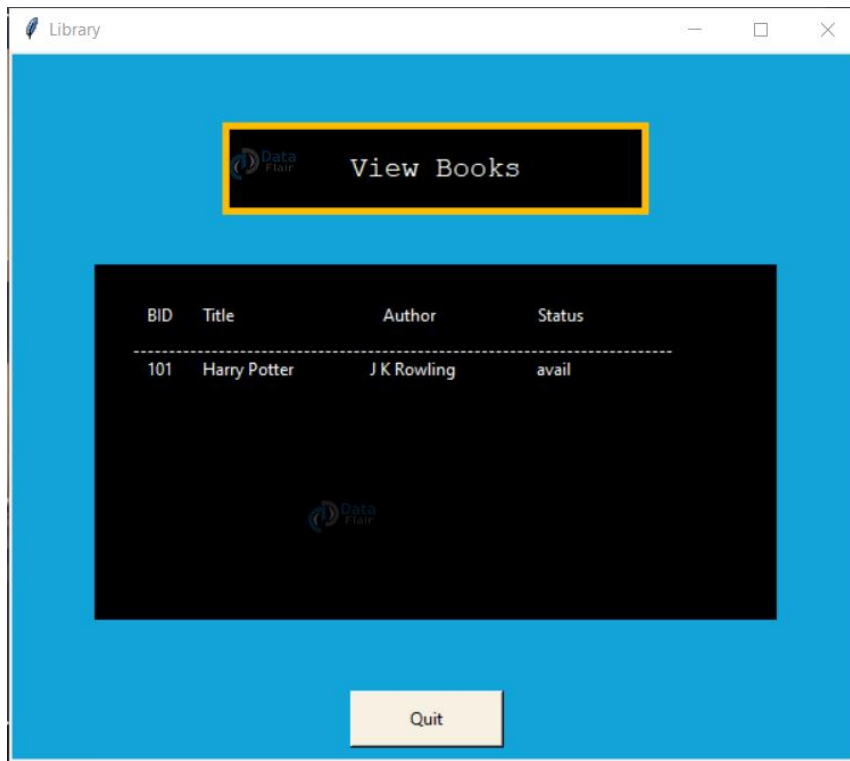
We create and place a **Label** on our black box which displays the text ‘**Book ID:**’. Just after **Label** is placed we create and place the **Entry** box acting as our input field.

Similarly, we create and place input fields associated with labels: Title, Author, and Status.

In the end, we create a button to **Finally!!** submit the details given by the user and a button to exit just in case the user did not intend to enter any details in library management.

As an action of a click on the **SubmitBtn**, we call the **bookRegister()** function to insert the details into the **books** table.

ViewBooks.py



3.1. Importing the necessary modules

Code:

```
from tkinter import *
from PIL import ImageTk,Image
from tkinter import messagebox
import pymysql
```

3.2. Connection to MySql server

Code:

```
mypass = "root"
mydatabase="db"
con =
pymysql.connect( host="localhost",user="root",password=mypass,database=mydatabase)
cur = con.cursor()
# Enter Table Names here
bookTable = "books"
```

3.3. Function – View()

This function in our library project creates a window for displaying the records in the table.

Code:

```
def View():
    root = Tk()
    root.title("Library")
    root.minsize(width=400,height=400)
    root.geometry("600x500")
```

```

Canvas1 = Canvas(root)
Canvas1.config(bg="#12a4d9")
Canvas1.pack(expand=True,fill=BOTH)
headingFrame1 = Frame(root,bg="#FFBB00",bd=5)
headingFrame1.place(relx=0.25,rely=0.1,relwidth=0.5,relheight=0.13)
headingLabel = Label(headingFrame1, text="View Books", bg='black', fg='white', font =
('Courier',15))
headingLabel.place(relx=0,rely=0, relwidth=1, relheight=1)
labelFrame = Frame(root,bg='black')
labelFrame.place(relx=0.1,rely=0.3,relwidth=0.8,relheight=0.5)
y = 0.25
Label(labelFrame, text="%-10s%-40s%-30s%-20s"%( 'BID','Title','Author','Status'),
bg='black',fg='white').place(relx=0.07,rely=0.1)
Label(labelFrame, text = "-----",
bg='black',fg='white').place (relx=0.05,rely=0.2)
getBooks = "select * from "+bookTable
try:
cur.execute(getBooks)
con.commit()
for i in cur:
Label(labelFrame,text="%-10s%-30s%-30s%-20s"%(i[0],i[1],i[2],i[3]) ,bg='black',
fg='white').place(relx=0.07,rely=y)
y += 0.1
except:
messagebox.showinfo("Failed to fetch files from database")
quitBtn = Button(root,text="Quit",bg='#f7f1e3', fg='black', command=root.destroy)
quitBtn.place(relx=0.4,rely=0.9, relwidth=0.18,relheight=0.08)
root.mainloop()
Explanation:

```

We create a new window to display the list of books and their status.

Just like we did in the previous file, we create a **headingFrame** and title it with a Label to display ‘**View Books**’.

Again we create a black box to accommodate the records returned by the **getBooks** query.

We manually display the name of the columns associated with our books table.

We execute the query stored in **getBooks** using **cur.execute()** display each record one by one as a Label. We place each **Label** You can notice that each time a record is displayed the value of **y** increases by ‘**0.1**’.

In order to handle any discrepancies, we place this code in a try-except block.

In the end, we create and place a button, **quitBtn** to exit from the current window of the library project.

Functions:

Function issue():

Primarily, we fetch the desired book ID and Issuer's name and store it into **bid** and **issueto** respectively.

After that we store all the Book ID from the **books** table in **allBid** by executing the SQL query stored in **extractBid**.

We check for the existence of the desired **bid** in **allBid**. If it exists and is available, we set the **status** as **True**.

If the book is available we update the **books_issued** table with the book id (**bid**) and Issuer's name (**issueto**) and update the **books** table by changing the status of the issued book to '**issued**'.

Function – issueBook():

We create and place a **headingFrame** and two input fields for taking input of the desired books' ID and issuers' name. After which we create and add two buttons named **issueBtn** and **quitBtn** to facilitate submission of our issue request and closing the present window of library management system respectively.

Function – returnn():

This function is similar to the **issue()** function we designed for **issueBook.py**.

In this library management system project, we fetch the desired book ID and store it into **bid**.

After that we store all the Book IDs from the **books_issued** table in **allBid** by executing the SQL query stored in **extractBid**.

We check for the existence of the desired **bid** in **allBid**. Also, we check the status of the same book and if it is '**issued**', we set the **status** as **True**.

Then, we remove the record from **books_issued** table and update the **books** table by changing the status of the issued book to '**avail**'.

6.4. Function – returnBook()

We create and place a **headingFrame** and an input field for taking input of the books' ID.

Then, we create and add two buttons named **SubmitBtn** and **quitBtn** to facilitate submission of our return request and closing the present window respectively.

CONCLUSION :

we have successfully designed a Library management system using python and tkinter with a decent UI. In order to make things easy, this tutorial divided the various tasks into different python files. In the real world, we follow such practices to make things easy to build. Hence, whenever making a project, divide your goals into modules and integrate them at a later stage.

Apart from this, you can now take a step forward to extend the project by making a history tab which keeps track of the previous books issued. Moreover, you can integrate a login system to authenticate a user before making changes to the database.