DATABASE Report

Libraries:

from tkinter import \*: This line imports all symbols from the tkinter library. It includes all the functions, classes, and constants available in the tkinter module. The \* (asterisk) wildcard in import \* means importing everything from the module. While convenient, it might not be considered best practice due to potential namespace conflicts.



from tkinter import messagebox: This line specifically imports the messagebox module from the tkinter library. The messagebox module provides a set of functions to create and display various types of message boxes or dialog boxes in Tkinter applications. These boxes can be used to show information, ask for confirmation, display warnings, or prompt the user for input.



from tkinter import ttk: This line imports the ttk module from the tkinter library. The ttk module contains themed Tkinter widgets. These widgets have a more modern and consistent appearance compared to the standard Tkinter widgets. They include buttons, labels, frames, etc., with enhanced visuals and styles.



from PIL import ImageTk, Image: This line imports specific modules (ImageTk and Image) from the PIL (Python Imaging Library) module. The PIL library allows for image processing tasks in Python. ImageTk provides functionality to display images in Tkinter GUIs, and Image allows opening, manipulating, and saving different image file formats.



Tuple Definition: The fonts variable is assigned a tuple. Tuples in Python are ordered collections of items, enclosed in parentheses (), and can contain multiple elements separated by commas.

Font Style and Size:

('Comic Sans MS', 12) consists of two elements:

'Comic Sans MS': This represents the font style or font family name. In this case, it specifies the font as "Comic Sans MS". This is a specific font type that will be used for text.

12: This indicates the font size. It sets the font size to 12 points.



Function Definition:

def exitapp(event): defines a function named exitapp. The function takes an event parameter, indicating that this function is intended to handle an event triggered within the Tkinter application.



Event Handling:

The function appears to handle an event related to the closure or termination of the Tkinter application window (library\_app). The specific event might be related to a user action, such as pressing a certain key, clicking a specific widget, or a window manager event like closing the application window.

library\_app.destroy():

Inside the exitapp function, library\_app.destroy() calls the destroy() method on the library\_app object. The destroy() method in Tkinter is used to close or destroy a window or widget. In this case, it's being applied to the library\_app window to close the entire application.



Purpose:

This function seems to serve the purpose of handling a specific event (likely related to the termination of the application) by closing the main window (library\_app) and effectively ending the Tkinter application.

The Function:

getlastid(directory), aims to retrieve the last identifier or ID from a file located at the specified directory.

Function Definition:

def getlastid(directory): defines a function named getlastid that takes a parameter directory, representing the path to a file.



Reading the File:

with open(directory) as file: opens the file specified by the directory.



last\_line = file.readlines()[-1] reads all lines from the file and extracts the last line.



file.close() ensures that the file is closed after reading its content.



Return the Last ID:

return last\_line[: 3] returns the first three characters of the last line. This assumes that the last line contains an identifier of some sort, and the function aims to extract that identifier.



Exception Handling (Try-Except Block):

try: begins a block of code where exceptions might occur.



Except: catches any exceptions that might occur within the try block.



If an exception occurs while trying to read the file (such as the file not existing or being empty), it moves to the except block.

File Creation and Default Return:

Inside the except block, file = open(directory, 'a') opens the file in append mode if it doesn't exist, creating an empty file.



file.close() ensures the file is closed after creating it or in case it was already open.



return '000' returns a default string '000' as the last ID if an exception occurred, either due to file not existing or being unable to read the last line.



This function:

gettitleandauthor(directory, lst), aims to read lines from a file located at the specified directory, extract information (title and author) from each line, and store it in a list lst.

Function Definition:

def gettitleandauthor(directory, lst): defines a function named gettitleandauthor that takes two parameters: directory (representing the file path) and lst (an existing list).



Reading the File and Extracting Information:

with open(directory) as file: opens the file specified by the directory.



The function then iterates through each line in the file using a for loop: for line in file.



id\_, title, author, status = line.split(' - ') splits each line of the file into separate components using the delimiter ' - '. It assumes that each line in the file is structured with the format: "ID - Title - Author - Status". It unpacks these components into variables id\_, title, author, and status. Then, it appends a list [title, author] (containing title and author information) to the existing list lst.

A black background with white text

Description automatically generated

Exception Handling (Try-Except Block):

try: begins a block of code where exceptions might occur.



Except: catches any exceptions that might occur within the try block.



If an exception occurs while trying to read the file (such as the file not existing or being unable to split lines), it moves to the except block.

File Creation and Closing:

Inside the except block, file = open(directory, 'a') opens the file in append mode if it doesn't exist, creating an empty file.



file.close() ensures the file is closed after creating it or in case it was already open.



Return the List:

return lst returns the updated list lst that contains title and author information extracted from the file lines.



This function attempts to read lines from a file specified by directory, split each line into parts based on the delimiter ' - ', extract title and author information, and append it as a list to the provided lst. If there's an issue reading the file or splitting lines, it catches the exception, creates an empty file if needed, and returns the existing or updated list lst.

This function:

getbooksinfo(directory, dic), reads lines from a file located at the specified directory, extracts information (such as title, author, status) from each line, and stores it in a dictionary dic.

Function Definition:

def getbooksinfo(directory, dic): defines a function named getbooksinfo that takes two parameters: directory (representing the file path) and dic (an existing dictionary).



Reading the File and Extracting Information:

with open(directory) as file: opens the file specified by the directory.



The function then iterates through each line in the file using a for loop: for line in file.



id\_, title, author, status = line.split(' - ') splits each line of the file into separate components using the delimiter ' - '. It assumes that each line in the file is structured with the format: "ID - Title - Author - Status". It unpacks these components into variables id\_, title, author, and status. Then, it assigns a value to the dictionary dic where the key is id\_ and the value is a list [title, author, status[:-1]]. The status[:-1] slicing removes the newline character (\n) from the end of the status string.



Closing the File:

After processing all lines in the file, file.close() ensures that the file is properly closed.



Return the Dictionary:

return dic returns the updated dictionary dic that contains book information with IDs as keys and lists of title, author, and status as values.



This function reads lines from a file specified by directory, splits each line into parts based on the delimiter ' - ', extracts book information (title, author, status) from each line, and stores it in the provided dictionary dic with book IDs as keys. Finally, it returns the updated dictionary dic containing the book information.

This function:

BookList() appears to create a window that displays a list of books using Tkinter's GUI components.



Opening and Closing File:

bookFile = open('image/BookList.txt') attempts to open a file named 'BookList.txt' located in the 'image/' directory.

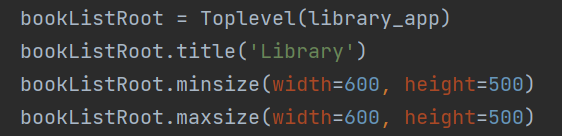


bookFile.close() immediately closes the file. This section seems to primarily check if the file exists without performing any operations with its content.



Creating a New Window (Toplevel):

bookListRoot = Toplevel(library\_app) creates a new window (a Toplevel widget) associated with the main window (library\_app). Configures the title, size (minimum and fixed), dimensions, and icon for the new window.



Canvas Widget:

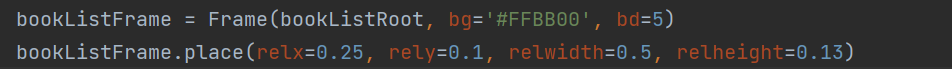
CanvasBookList = Canvas(bookListRoot) creates a canvas widget inside the new window (bookListRoot), allowing for drawing and display. Configures the canvas background color and packs it to fill the available space.

A black background with white text

Description automatically generated

Frame for Heading:

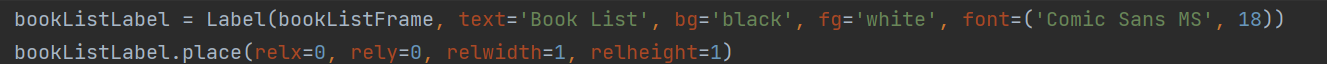
bookListFrame = Frame(bookListRoot) creates a frame inside the new window for displaying the heading. Places the frame at a specific relative position and size within the window. Configures the frame's background color, border width, and positioning of the label inside it.



Heading Label:

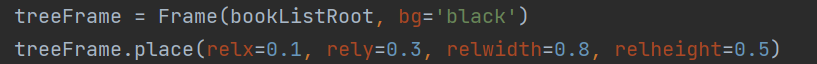
bookListLabel = Label(bookListFrame) creates a label with the text 'Book List' inside the frame.

Configures the label's appearance: text color, background color, font, and placement within the frame.



Frame for Treeview (Table-Like Widget):

treeFrame = Frame(bookListRoot) creates another frame inside the new window for displaying a Treeview (a table-like widget). Places the frame at a specific relative position and size within the window. Configures the frame's background color.

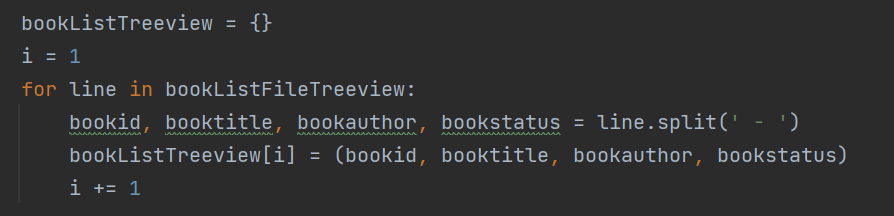


Reading and Processing File Data:

with open('image/BookList.txt') as bookListFileTreeview: opens the file 'BookList.txt' for reading.

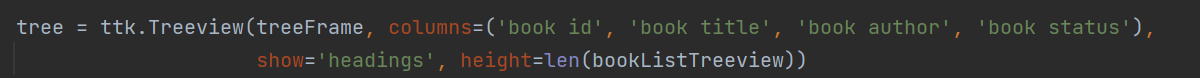


The code reads each line of the file, splits it based on the delimiter ' - ', and assigns the components (bookid, booktitle, bookauthor, bookstatus) to variables. It stores this data in a dictionary named bookListTreeview, using an incremental index (i) as keys and tuples containing book information as values.

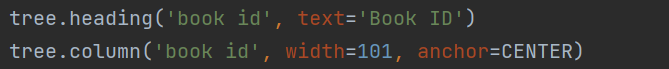


Creating a Treeview Widget:

tree = ttk.Treeview(treeFrame, columns=('book id', 'book title', 'book author', 'book status'), show='headings', height=len(bookListTreeview)) creates a Treeview widget with specified columns and headings to display book information.



It configures the column headings, widths, and alignment for displaying book ID, title, author, and status. A loop populates Treeview with the book information stored in the bookListTreeview dictionary.

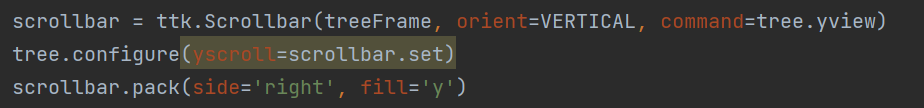


Displaying the Treeview and Scrollbar:

tree.pack(side='left', anchor=N, fill='x') places the Treeview widget in the left part of treeFrame, allowing it to expand horizontally (fill='x').

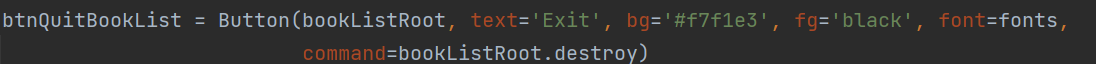


scrollbar = ttk.Scrollbar(treeFrame, orient=VERTICAL, command=tree.yview) creates a vertical scrollbar linked to the Treeview's y-axis movement. tree.configure(yscroll=scrollbar.set) configures the Treeview to use the scrollbar for vertical scrolling. The scrollbar is packed on the right side of treeFrame.



Exit Button:

btnQuitBookList = Button(bookListRoot, text='Exit', bg='#f7f1e3', fg='black', font=fonts, command=bookListRoot.destroy) creates an 'Exit' button using a Button widget. It specifies the appearance (text, colors, font), and when clicked (command=bookListRoot.destroy), it closes the bookListRoot window, effectively quitting the book list display.



Running the GUI:

bookListRoot.mainloop() starts the event loop for the bookListRoot window, allowing the GUI to be displayed and handle user interactions.



Handling FileNotFoundError:

except FileNotFoundError: initiates a block of code that will execute if Python encounters a FileNotFoundError while trying to open a file. The FileNotFoundError occurs when the specified file (in this case, 'BookList.txt') is not found in the given directory or path.



Messagebox Display:

messagebox.showerror() is a function from the tkinter.messagebox module used to display an error message dialog box in the Tkinter GUI. title='No book' sets the title of the error message box as 'No book'. message='There is no book registered in this library.' sets the content or message displayed within the error message box to indicate that there are no books registered in the library.



This function:

SearchBook() seems to handle the creation of a search interface within a Tkinter-based application to search for books.



Global Variable Declaration:

global SearchIDEntry declares a global variable SearchIDEntry. This variable likely holds an entry widget where users can input a book ID for searching.



File Check and GUI Setup:

Inside the try block, it attempts to open the 'BookList.txt' file to check if it exists. If the file exists, it proceeds to set up the search interface GUI.

A black background with white text

Description automatically generated

Creating a New Window (Toplevel):

bookSearchRoot = Toplevel(library\_app) creates a new window (a Toplevel widget) associated with the main window (library\_app). Configures the title, size (minimum, maximum, and fixed), dimensions, and icon for the new window.

A screen shot of a computer code

Description automatically generated

Canvas Widget and Heading:

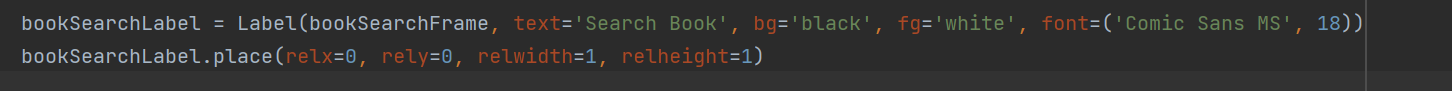
CanvasBookSearch = Canvas(bookSearchRoot) creates a canvas widget inside the new window (bookSearchRoot), allowing for drawing and display. Configures the canvas background color and packs it to fill the available space. Creates a frame (bookSearchFrame) for displaying the heading 'Search Book' with specific background color and text style.

A black background with white text

Description automatically generated

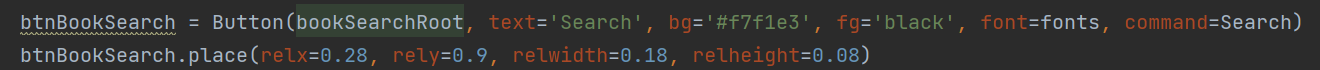
Book ID Entry Widget:

SearchIDEntry = Entry(bookSearchInfoFrame) creates an Entry widget inside a frame (bookSearchInfoFrame) for users to input a book ID. Place the Entry widget within the frame and configures its position and size.

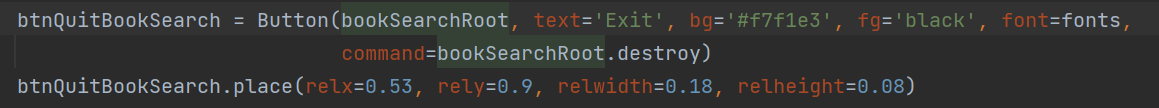


Search and Exit Buttons:

btnBookSearch and btnQuitBookSearch are buttons created for 'Search' and 'Exit' respectively, both placed in specific positions within the window.

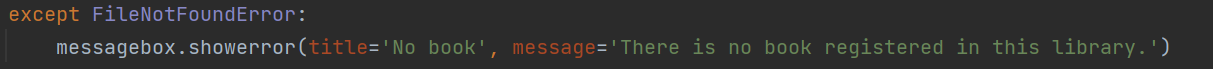


btnBookSearch likely triggers a function named Search when clicked (command=Search), while btnQuitBookSearch closes the search window when clicked (command=bookSearchRoot.destroy).



Handling File Not Found Error:

If the 'BookList.txt' file is not found, the FileNotFoundError exception is caught, and a message box is displayed using messagebox.showerror() to notify the user that there are no books registered in the library.



Running the GUI:

bookSearchRoot.mainloop() starts the event loop for the bookSearchRoot window, allowing the GUI to be displayed and handle user interactions.



This function attempts to create a search interface window where users can input a book ID to search for book details. It handles both the case where the file exists and where it doesn't, providing an error message if no books are found. The interface includes entry fields for user input, buttons for search and exit functionality, and a canvas for layout and display within the Tkinter GUI.

The function:

Search() is designed to handle the search functionality within the book library application.



Fetching Search Input:

searchID = SearchIDEntry.get().strip(): Retrieves the input from the SearchIDEntry field (which presumably contains the book ID). The .strip() method removes any leading or trailing whitespace from the input.



Checking the Length of Input:

if len(searchID) == 3: Verifies if the length of the entered book ID is three characters. This might be a predefined length for book IDs in the system.



Search and Display Book Information:

If the entered book ID is three characters long, it proceeds with searching for the book.

dicSearch = {}: Initializes an empty dictionary for storing book information.



booksInfoSearch = getbooksinfo('image/BookList.txt', dicSearch): Retrieves book information from the 'BookList.txt' file using the getbooksinfo function, storing it in booksInfoSearch.

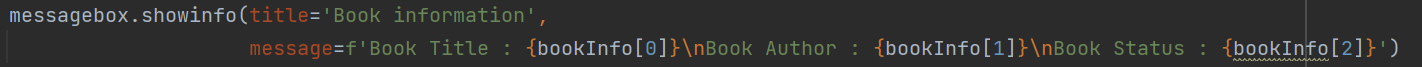


bookInfo = booksInfoSearch[searchID]: Retrieves the book information (title, author, status) corresponding to the entered book ID.



Showing Book Information in Message Box:

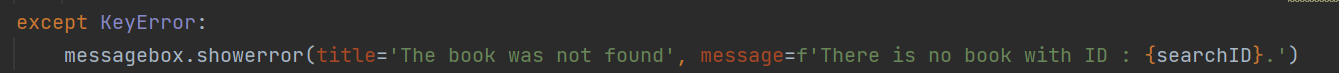
messagebox.showinfo(): Displays a message box containing the retrieved book information (title, author, status) for the book associated with the entered ID.



Handling Exceptions (KeyError):

If the entered book ID is not found in the book information dictionary (booksInfoSearch), it raises a KeyError.

except KeyError: Catches the KeyError exception indicating that the entered book ID was not found in the library. Displays an error message in a message box indicating that there's no book with the entered ID.



Handling Invalid ID Length:

If the length of the entered book ID is not three characters, it displays an error message in a message box, indicating an invalid book ID.



Resetting Entry Field and Focus:

SearchIDEntry.delete(0, END): Clears the SearchIDEntry field after processing the search.



SearchIDEntry.focus\_force(): Sets the focus back to the SearchIDEntry field, allowing for immediate input without clicking.



This function aims to handle user input, search for book information based on the entered ID, and display the book details in a message box. It also provides error messages for invalid input or when the entered ID does not match any book in the library.

Borrow book:

The BorrowBook() function sets up a window for borrowing books in the library.



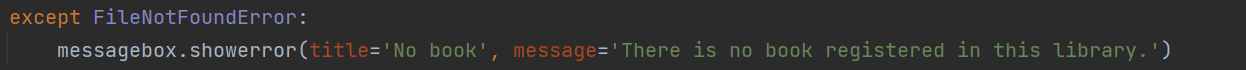
Global Variable Declaration:

global BorrowIDEntry: Declares the BorrowIDEntry variable as a global variable. This suggests that BorrowIDEntry will be used and possibly modified within other functions.



Handling No Books (FileNotFoundError):

The function is wrapped in a try-except block to handle the possibility of a specific exception - FileNotFoundError. It attempts to open the 'BookList.txt' file. If the file is not found, it shows an error message using messagebox.showerror() indicating that there are no books registered in the library.



Creating the Borrow Book Window:

If the file is found (no FileNotFoundError), the function proceeds to create the borrow book window. bookBorrowRoot = Toplevel(library\_app): Creates a new Toplevel window associated with the main window (library\_app) for the borrowing functionality. Configures the title, dimensions, and icon for the new window.

A computer screen shot of text

Description automatically generated

Canvas Widget:

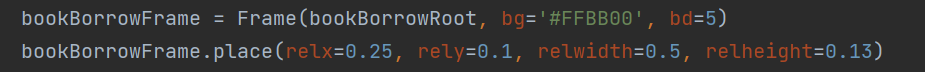
CanvasBookBorrow = Canvas(bookBorrowRoot): Creates a canvas widget for drawing and display inside the borrow window. Configures the canvas background color and packs it to fill the available space.

A black background with white text

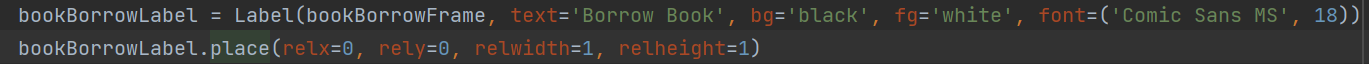
Description automatically generated

Heading Frame and Label:

bookBorrowFrame = Frame(bookBorrowRoot): Creates a frame inside the borrow window for displaying the heading. Configures the frame's background color and border width.

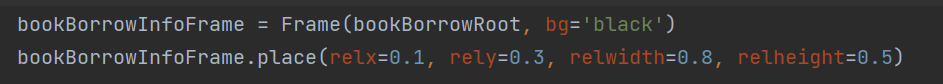


bookBorrowLabel = Label(bookBorrowFrame): Creates a label with the text 'Borrow Book' inside the frame. Configures the label's appearance: text color, background color, font, and placement within the frame.

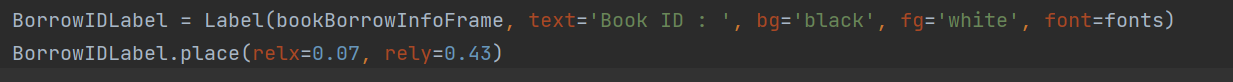


Info Frame and Entry for Book ID:

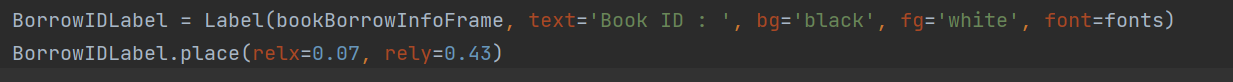
bookBorrowInfoFrame = Frame(bookBorrowRoot): Creates another frame inside the borrow window for displaying book information and borrowing-related elements. Configures the frame's background color.



BorrowIDLabel = Label(bookBorrowInfoFrame): Creates a label with the text 'Book ID' inside the info frame.



BorrowIDEntry = Entry(bookBorrowInfoFrame): Creates an entry widget for entering the book ID inside the info frame.

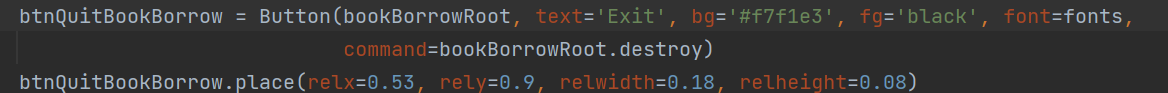


Borrow and Quit Buttons:

btnBookBorrow = Button(bookBorrowRoot): Creates a 'Borrow' button using a Button widget. It is associated with a function named Borrow.



btnQuitBookBorrow = Button(bookBorrowRoot): Creates an 'Exit' button using a Button widget. It is associated with the destruction of the bookBorrowRoot window.



Running the GUI:

bookBorrowRoot.mainloop(): Initiates the event loop for the bookBorrowRoot window, allowing the GUI to be displayed and handle user interactions.



This function sets up a window for borrowing books in the library, providing input fields for entering book IDs and buttons for borrowing and quitting. If there are no books in the library (FileNotFoundError), it shows an error message to inform the user.

Def borrow:

Input Validation:

It gets the input for the book ID and checks if its length is 3 characters.

A black background with white text

Description automatically generated

Reading Book Information:

It uses a function getbooksinfo() passing a file path ('image/BookList.txt') and an empty dictionary (dicBorrow) to retrieve book information.



The function getbooksinfo() likely reads the file content and organizes it into a dictionary (booksInfoBorrow) where book IDs are keys and book information (title, author, and status) are stored as values.

Checking Availability:

It checks if the book with the provided ID is marked as 'not taken' in the booksInfoBorrow dictionary.



Updating Book Status:

If the book is 'not taken', it updates the status of the book in the text file ('image/BookList.txt') from 'not taken' to 'taken' by modifying the file contents directly.

A black background with white text

Description automatically generated

Displaying Messages:

Depending on whether the book was successfully borrowed or if it's already borrowed or not found, it shows corresponding pop-up messages using messagebox.



Error Handling:

It handles the situation where the book ID is not found in the dictionary (KeyError exception).

Cleanup:

Finally, it clears the entry field (BorrowIDEntry) and refocuses on it for the next input.

A black background with white text

Description automatically generated

Return book:

This ReturnBook() function appears to handle the process of returning books in a library management system.



Error Handling:

The function starts with a try-except block. It attempts to open a file named 'image/BookList.txt'. If the file doesn't exist (FileNotFoundError), it shows an error message indicating that there are no books registered in the library.

A black background with white text

Description automatically generated

Main Window Creation:

Assuming the file exists (no exception occurred), it opens a new window (Toplevel) for returning a book.

Sets up the window title, size, geometry, and icon.

A screen shot of a computer program

Description automatically generated

GUI Setup:

Configures a canvas and frames within the new window for a visually organized interface. Creates a frame for the return book section and a label for its heading. Generates another frame for book information, particularly for the book ID entry (ReturnIDEntry).

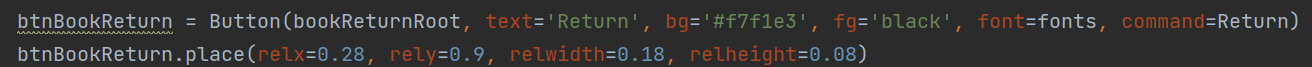
A black background with white text

Description automatically generated

Buttons:

Adds two buttons:

Return button wired to the Return() function, presumably used to trigger the book return process.



'Exit' button to close the 'Return Book' window (bookReturnRoot.destroy()).



User Interaction:

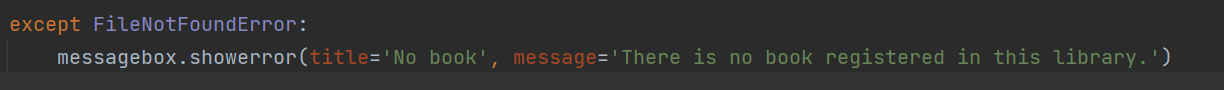
This function allows the user to enter the book ID they wish to return using the ReturnIDEntry entry field.

Visualization:

The code uses specific colors, fonts, and geometry to provide a visually appealing and user-friendly interface for returning books.

Event Loop:

Starts the main event loop (bookReturnRoot.mainloop()) to handle user interactions within this particular window until it's closed.



This function sets up a window where users can input a book ID to return, creating a structured and visually appealing interface for this purpose. If there are no books registered, it presents an error message instead of opening the return window.

Return:

This Return() function is responsible for handling the book return process within a library management system.



Get ID:

Retrieves the book ID entered by the user through the ReturnIDEntry field, stripping any extra whitespace.



Validation:

Checks if the entered book ID has a length of 3 characters.



If the length is correct:

Attempts to retrieve book information associated with the provided ID by calling getbooksinfo() function and passing the 'image/BookList.txt' file and an empty dictionary.



Book Information Retrieval:

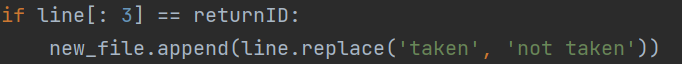
If the book information for the given ID is found:

Checks if the book status is 'taken'. This presumably means the book has been borrowed.



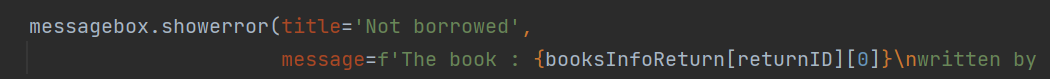
If the book is indeed 'taken':

Opens the BookList file, modifies the line corresponding to the book's ID to mark it as 'not taken' (indicating the book has been returned). Updates the file with the modified information. Displays a success message confirming the book return with details like book title, author, and ID.



If the book status is not 'taken':

Shows an error message indicating that the book hasn't been borrowed before.



Exception Handling:

If the book ID is not found in the database (KeyError), it displays an error message indicating that there is no book with that ID.



Invalid ID Handling:

If the entered book ID doesn't match the required length of 3 characters, it shows an error message indicating an invalid ID.



Cleanup and Focus:

Clears the ReturnIDEntry field after the book return process is completed. Sets the focus back on the ReturnIDEntry field for the user's convenience.

A black background with white text

Description automatically generated

This function is designed to handle the return of books by verifying the provided book ID, checking its status, updating the database file accordingly, and providing appropriate feedback messages to the user based on the book's status and the validity of the entered ID.

Add Book:

This AddBook() function creates a GUI window that allows users to add new books to the library system.



Initialization:

It initializes a new window (Toplevel) named bookAddRoot that serves as the window for adding books.

Sets the window title, size constraints, dimensions, and icon.

A computer screen with text and numbers

Description automatically generated with medium confidence

Canvas Setup:

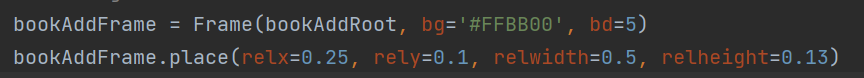
Creates a canvas (CanvasBookAdd) within the window to hold elements, setting its background color and making it expandable to fill the available space.

A black background with white text

Description automatically generated

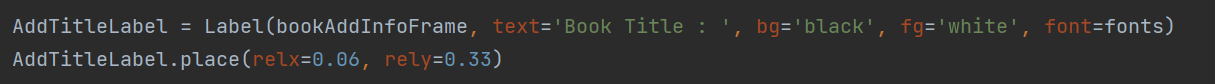
Heading:

Constructs a frame (bookAddFrame) within bookAddRoot to hold the heading label (bookAddLabel) displaying "Add Book". This frame provides a distinctive background color and border.



Book Information Entry:

Sets up another frame (bookAddInfoFrame) to hold input fields for book title and author details. Creates labels (AddTitleLabel, AddAuthorLabel) for book title and author respectively, with appropriate formatting and placement within bookAddInfoFrame. Sets up entry fields (AddTitleEntry, AddAuthorEntry) where users can input the book's title and author, respectively.

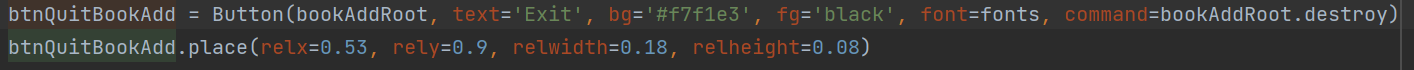


Buttons:

Creates a button (SubmitBtn) labeled "Add" that triggers the Add() function (not provided here) when clicked. This function is presumably responsible for handling the addition of the book based on the input.



Another button (btnQuitBookAdd) labeled "Exit" allows users to close this window by destroying the bookAddRoot window.



Event Loop:

Starts the main event loop (bookAddRoot.mainloop()) to ensure the window remains open and responsive, allowing users to interact with the elements created until the window is closed.

A close up of a black background

Description automatically generated

This function sets up a user-friendly interface for adding book information, including title and author, and provides buttons to trigger the book addition process or exit the window.

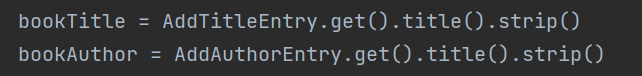
Add:

This Add() function serves to handle the addition of new books to the library.



Get Input Values:

Retrieves the book title and author entered in the respective entry fields (AddTitleEntry and AddAuthorEntry). Removes any leading/trailing whitespace and ensures the titles and authors are in title case.



Input Validation:

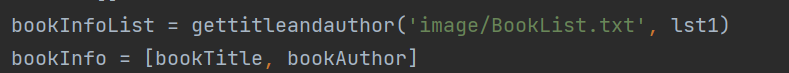
Checks if both the book title and author are not empty. If either the title or author is empty, it displays an error message indicating that the entered information is not correct.

A black background with white text

Description automatically generated

Checking Existing Book Information:

Retrieves the existing list of book titles and authors from the file 'image/BookList.txt' using the gettitleandauthor() function. Compares the entered book's title and author with the existing list.



If the book information already exists in the list, it displays an error message stating that the book has already been added. Clears the entry fields and sets focus back to the title entry.



Generating Book ID:

Retrieves the last book ID from the file 'image/BookList.txt' using the getlastid() function.



Increments the ID to generate a new unique book ID:

If the ID is between 10 and 99, it increases it by 1.

A black background with white text

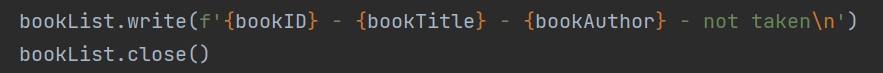
Description automatically generated

If the ID is less than 10, it adds a leading '0' to make it a three-digit ID.

If the ID is already 99 or above, it increments it without leading zeros.

Adding Book Information to File:

Opens the file in append mode and writes the new book's information (ID, title, author, status) in the format 'ID - Title - Author - Status\n'. Displays a success message showing the details of the added book (title, author, ID).



Clearing and Focusing Entry Fields:

Clears both the title and author entry fields after successful addition. Sets focus back to the title entry for the user's convenience in adding another book.

A screen shot of a computer

Description automatically generated

This function ensures that the entered book information is valid, checks if the book is not a duplicate, generates a unique ID, adds the book details to the file, and provides appropriate user feedback throughout the process.

Delete book:

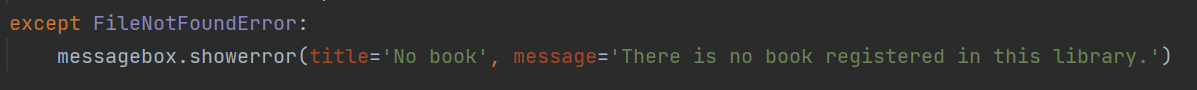
This DeleteBook() function manages the deletion of books from the library system.



Error Handling for Empty Library:

Attempts to open the file 'image/BookList.txt' within a try-except block.

If the file is not found (raises FileNotFoundError), it displays an error message indicating that there are no books registered in the library.



GUI Setup for Book Deletion:

If the file exists (no error is raised), it creates a new window (Toplevel) named bookDeleteRoot to handle the book deletion interface. Sets up the window title, size restrictions, geometry, and icon.

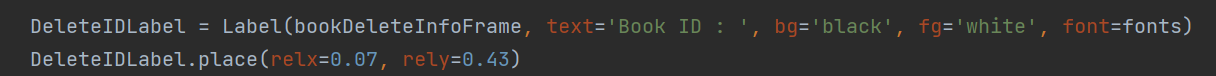
A screen shot of a computer program

Description automatically generated

Configures a canvas (CanvasBookDelete) to serve as the background for the deletion window.

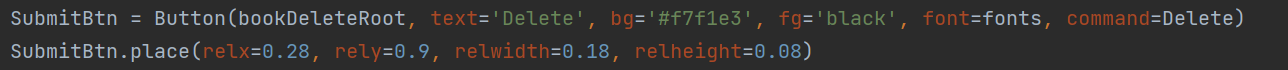
GUI Elements for Book Deletion:

Creates a frame (bookDeleteFrame) for the title "Delete Book" with specific styling and placement. Place a label (bookDeleteLabel) inside the frame to display the title using specified font and color. Sets up another frame (bookDeleteInfoFrame) for user input, providing space for entering the book ID to be deleted. Creates a label (DeleteIDLabel) for displaying "Book ID" and an entry field (DeleteIDEntry) for users to input the ID of the book they want to delete.

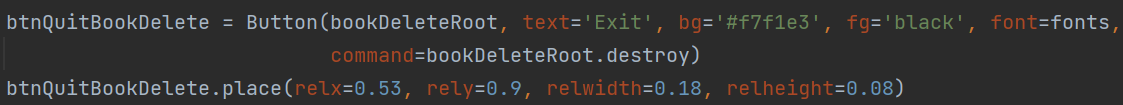


Submit and Quit Buttons:

Adds a "Delete" button (SubmitBtn) to trigger the deletion process.



Adds an "Exit" button (btnQuitBookDelete) to close the deletion window.



Main Loop:

Enter the main event loop (bookDeleteRoot.mainloop()) to handle user interactions within this deletion window.



This function ensures that the book deletion interface is presented to the user, allowing them to input the ID of the book they want to remove. Additionally, it handles the scenario where the library contains no books by displaying an appropriate error message.

Delete :

This Delete() function manages the deletion of a book from the library system based on the provided book ID.



Get ID and Validation:

Retrieves the book ID entered by the user from the DeleteIDEntry field. Checks if the entered ID consists of 3 characters. If not, it shows an error message indicating an invalid ID format.

A black background with white text

Description automatically generated

Deleting the Book:

Attempts to retrieve the book information using the getbooksinfo() function from the file 'image/BookList.txt' and stores it in booksInfoDelete as a dictionary. Checks if the entered deleteID exists within the keys of booksInfoDelete.



If the deleteID is found:

Initializes an empty list new\_file to store the modified content of the book list file. Opens the book list file in read mode ('r') and iterates through its lines.

A black background with white text

Description automatically generated

Processes each line based on the provided deleteID:

Keeps lines with IDs lower than deleteID. Ignores the line with the matching deleteID. Adjusts IDs of lines higher than deleteID to fill the gap left by the deleted book. Writes the modified content back to the 'image/BookList.txt' file.



Handling Exceptions and User Feedback:

If the entered deleteID is not found in booksInfoDelete or if any exceptions occur during this process, it displays an error message indicating that there is no book with that ID or that the book was not found. After deletion or in case of an invalid ID, it clears the DeleteIDEntry field and refocuses on it for further input.

User Notification:

If the book is successfully deleted, it displays a success message indicating the details of the deleted book.



This function ensures proper deletion of a book from the library file based on the provided ID and provides user feedback regarding the deletion status through message boxes.

Top of Form

Main App:

library\_app = Tk(): Creates the main window of the application.



library\_app.title('Library'): Sets the title of the window to 'Library'.



library\_app.minsize(width=400, height=300): Defines the minimum size that the window can be resized to, ensuring it doesn't become too small.



library\_app.maxsize(width=900, height=800): Sets the maximum size that the window can be resized to, preventing it from becoming too large and maintaining a reasonable size.



library\_app.geometry('600x500'): Specifies the initial size of the window as 600 pixels wide and 500 pixels high.



library\_app.iconbitmap('image/Icon.ico'): Sets the icon for the window, using an icon file named 'Icon.ico' located in the 'image' directory. This icon typically appears in the window's title bar and taskbar.



These lines of code initialize and configure the main window ('library\_app') for the library application, setting its title, size constraints, initial dimensions, and icon.

Background:

Loading and Resizing the Background Image:

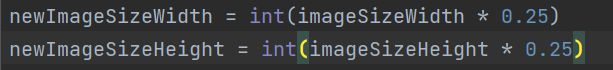
backgroundImage = Image.open('image/background.jpg'): Opens the image file named 'background.jpg' located in the 'image' directory using PIL's Image.open() method.



imageSizeWidth, imageSizeHeight = backgroundImage.size: Retrieves the width and height of the opened image.



newImageSizeWidth = int(imageSizeWidth \* 0.25), newImageSizeHeight = int(imageSizeHeight \* 0.25): Calculates new dimensions for the image by reducing its size to 25% of the original size.



backgroundImage = backgroundImage.resize ((newImageSizeWidth, newImageSizeHeight)): Resizes the image to the calculated new dimensions.



Converting Image to Tkinter-Compatible Format:

img = ImageTk.PhotoImage(backgroundImage): Converts the resized PIL image (backgroundImage) to a format compatible with Tkinter using ImageTk.PhotoImage().



Creating and Configuring Canvas:

CanvasMain = Canvas(library\_app): Creates a canvas widget in the main application window (library\_app).



CanvasMain.create\_image(350, 540, image=img): Places the converted image (img) onto the canvas at the specified coordinates (350, 540).



CanvasMain.config(bg='white', width=newImageSizeWidth, height=newImageSizeHeight): Configures the canvas background color as white and sets its dimensions to match the resized image.



CanvasMain.pack(expand=True, fill=BOTH): Packs the canvas into the main window, allowing it to expand and fill both horizontally and vertically.



Creating a Heading Label:

welcomeFrame = Frame(library\_app, bg='#FFBB00', bd=5): Creates a frame for the heading with a yellow background color.



welcomeFrame.place(relx=0.2, rely=0.05, relwidth=0.6, relheight=0.16): Places the frame in the specified relative position and size within the main window.



welcomeLabel = Label(welcomeFrame, text='Welcome to Library', bg='black', fg='white', font=('Comic Sans MS', 18)): Creates a label with the text 'Welcome to Library', black background, white foreground, and a specified font.



welcomeLabel.place(relx=0, rely=0, relwidth=1, relheight=1): Places the label inside the frame, covering its entire area.

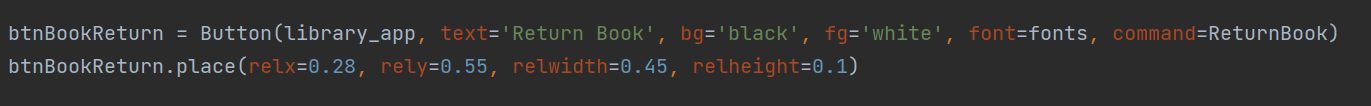


Buttons:

Button Creation and Placement:

btnBookList, btnBookSearch, btnBookBorrow, btnBookReturn, btnBookAdd, btnBookDelete, btnQuit: These lines create different buttons for actions within the library application.

Each button is configured with specific text, background color, foreground color, font style (fonts), and a command to be executed when clicked (command=...).



Placement Using place Method:

.place(relx, rely, relwidth, relheight): Positions and sizes the buttons relative to the main window (library\_app) using the place() method. relx and rely specify the relative x and y coordinates within the window. relwidth and relheight define the relative width and height of the buttons, respectively.



Button Commands:

command=BookList, command=SearchBook, command=BorrowBook, command=ReturnBook, command=AddBook, command=DeleteBook: These commands are functions or methods that will be executed when the corresponding button is clicked.

Exit Handling:

library\_app.bind('<Escape>', exitapp): Binds the 'Escape' key to the exitapp function, which typically handles the exiting or closing of the application.



Main Loop:

mainloop(): Starts the main event loop of the Tkinter application, which listens for events (like button clicks, key presses, etc.) and responds accordingly.



Pictures:

A screenshot of a computer

Description automatically generated A computer screen shot of a book list

Description automatically generated

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated