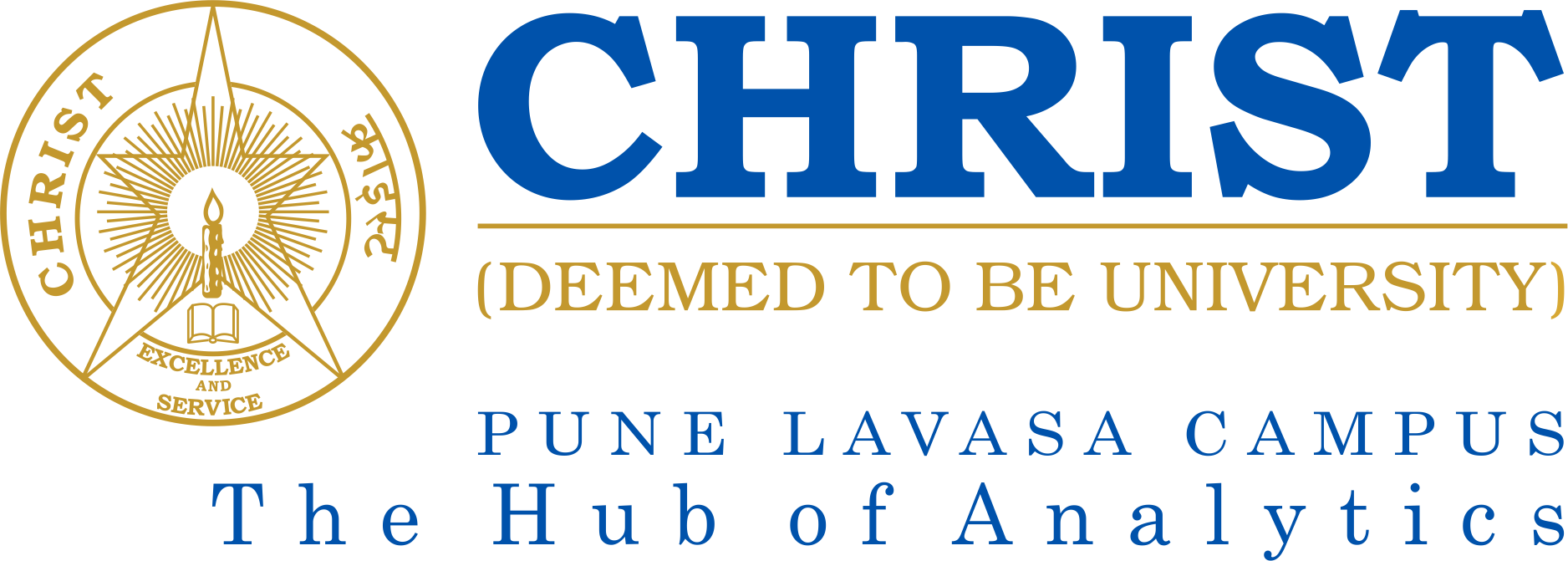
****

**CAC - 1**

**Data Analytics**

**MINI PROJECT REPORT**

***Prof.***

***Kandula Balagangadhar Reddy***

***By***

***Jason Seraphim***

***Jai Sharma***

***2B.Sc DS A***

**To-Do List Application**

This is a Python code that implements a simple To-Do List application using the Tkinter library for creating the graphical user interface (GUI). The application allows users to add tasks with due dates, mark tasks as complete, delete tasks, and save the tasks to an Excel file. The code is well-commented to explain the functionality of each component. Here is a detailed documentation of the code:

**Observation**

The given description outlines the requirements for a To-Do List application. Here is a detailed breakdown of the requirements:

1. **User Data Entry**: The application should allow users to enter tasks or data. This means providing an interface where users can input the task name, due date, and any other relevant information.

2. **Data Deletion**: Users should be able to delete tasks or data from the list. This can be achieved by providing a way for users to select a task and remove it from the list.

3. **Displaying All Data**: The application should display a list of all the entered data. This can be done by creating a section or view where all the tasks or data are shown in a user-friendly format.

4. **Search Functionality**: Users should be able to search for specific tasks or data using key values such as the task name or the date when the task was entered. The application should provide a search feature where users can enter their search criteria, and the matching tasks or data will be displayed.

5. **Setting Due Dates**: Users should be able to set due dates for their tasks. This means allowing users to input a specific date or select a date from a calendar. The due dates can be associated with each task and displayed alongside the task details.

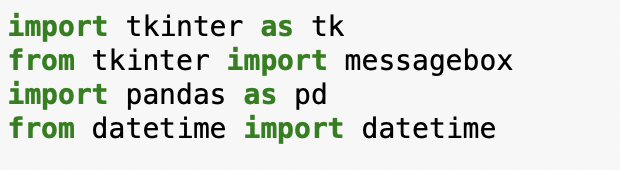
To implement these requirements, you can use the Tkinter library to create the GUI components and SQLite as a database for storing the tasks or data. The GUI will include entry fields for task name, due date, and search criteria, buttons for adding and deleting tasks, a listbox or table for displaying tasks, and a search button for executing the search functionality.

The data entered by the user will be stored in the SQLite database, and queries will be used to retrieve and display the data based on the user's actions, such as adding, deleting, or searching for tasks. The due dates can be stored as a separate field in the database table and associated with each task.

By combining these elements, you can create a functional To-Do List application that allows users to enter, delete, search, and set due dates for their tasks.

**IMPORTS**

1. **‘tkinter’**: The main library used for creating GUI applications in Python.
2. **‘messagebox’** (from tkinter): A submodule of tkinter that provides a simple way to display message boxes for notifications or errors.
3. **‘pandas’**: A powerful data manipulation library used for working with structured data, in this case, for handling tasks data and saving it to an Excel file.
4. **‘datetime’**: A module that provides classes for working with dates and times



**FUNCTIONS**

**Save Tasks Function (‘save\_tasks(tasks)’):**

* This function takes a list of tasks as input and saves them to an Excel file named "new.csv".
* It first creates a DataFrame using pandas with the given tasks and column names ("Task", "Due Date", "Status").
* Then, it uses the ‘**to\_csv’** function to export the DataFrame to a CSV file without including the index.
* If an error occurs during the saving process, a message box is displayed with the error details.
* Otherwise, a message box is shown to inform the user that the tasks were saved successfully

.

**Add Task Function (‘add\_task()’):**

* This function is called when the "Add Task" button is clicked.
* It retrieves the task description and due date from the respective entry widgets.
* If the task description is empty, an error message box is displayed, and the function returns.
* If the due date is empty, an error message box is displayed, and the function returns.
* If both the task and due date are provided, the task is appended to the ‘**tasks’** list as a new item containing the task description, due date, and initial status of "Incomplete".
* The task and due date entry widgets are then cleared.
* Finally, the ‘**update\_tasks\_display()’** function is called to refresh the displayed list of tasks.

**Delete Task Function (‘delete\_task()’):**

* This function is called when the "Delete Task" button is clicked.
* It retrieves the index of the selected task in the tasks list from the ‘**tasks\_listbox’** widget using the ‘**curselection()’** method.
* If a task is selected, the task is removed from the ‘**tasks’** list using the ‘**pop()’** method with the retrieved index.
* The ‘**update\_tasks\_display()’** function is then called to update the tasks displayed in the listbox.

**Mark Complete Function (‘mark\_complete()’):**

* This function is called when the "Mark Complete" button is clicked.
* It retrieves the index of the selected task in the tasks list from the ‘**tasks\_listbox’** widget using the ‘**curselection()’** method.
* If a task is selected, the status of the task at the retrieved index is updated to "Complete".
* The ‘**update\_tasks\_display()’** function is then called to update the tasks displayed in the listbox.

**Update Tasks Display Function (‘update\_tasks\_display()’):**

* This function updates the tasks displayed in the ‘**tasks\_listbox’** widget.
* It clears the current contents of the listbox using the ‘**delete()’** method with the arguments 0 and ‘**tk.END’**.
* Then, it iterates over each task in the ‘**tasks’** list and inserts a formatted string representation of the task into the listbox using the ‘**insert()’** method.
* The formatted string includes the task description, due date, and status.
* The task is formatted as "{task[0]} - Due: {task[1]} - Status: {task[2]}".
* The listbox is updated with all the tasks.

**Main Window Setup:**

* The main window is created using ‘**tk.Tk()’**, and the window title is set to "To-Do List" using the ‘**title()’** method.
* The main window is the container for all the GUI elements.

**Task Entry Widgets:**

* A label widget (‘**task\_label’**) is created with the text "Task:" and packed into the main window using the ‘**pack()’** method.
* An entry widget (‘**task\_entry’**) is created with a width of 30 characters and packed into the main window.
* These widgets allow the user to enter the task description.

**Due Date Entry Widgets:**

* A label widget (‘**due\_date\_label’**) is created with the text "Due Date (YYYY-MM-DD):" and packed into the main window.
* An entry widget (‘**due\_date\_entry’**) is created with a width of 30 characters and packed into the main window.
* These widgets allow the user to enter the due date of the task.

**Add, Delete, and Mark Complete Buttons:**

* Three button widgets (‘**add\_button’**, ‘**delete\_button’**, ‘**mark\_complete\_button’**) are created with the respective labels "Add Task", "Delete Task", and "Mark Complete".
* The command option is set to call the corresponding functions ‘**add\_task()’**, ‘**delete\_task()’**, and ‘**mark\_complete()’** when the buttons are clicked.
* The buttons are packed into the main window.

**Tasks Listbox:**

* A listbox widget (‘**tasks\_listbox’**) is created with a width of 50 characters and packed into the main window.
* This listbox will display the tasks.
* Initially, it is empty, and the tasks will be updated using the ‘**update\_tasks\_display()’** function.

**Sample List of Tasks:**

* A sample list of tasks is created as a list of lists (‘**tasks’**).
* Each task is represented by a list containing the task description, due date, and status ("Incomplete" by default).

**Update Tasks Display:**

* The ‘**update\_tasks\_display()’** function is called to populate the ‘**tasks\_listbox’** with the initial sample list of tasks.

**Save and Exit Button:**

* A button widget (‘**save\_exit\_button’**) is created with the label "Save and Exit".
* The command option is set to call the ‘**save\_and\_exit()’** function when the button is clicked.
* The button is packed into the main window.

**Save and Exit Function (‘save\_and\_exit()’):**

* This function is called when the "Save and Exit" button is clicked.
* It calls the ‘**save\_tasks()’** function to save the tasks to an Excel file.
* If the saving process is successful, the ‘destroy()’ method is called on the main window to close the application.

**Main Loop:**

* The main loop is started using the ‘mainloop()’ method on the main window.
* This loop listens for user interactions and updates the GUI accordingly.

**Summary:**

This code implements a To-Do List application using Tkinter and pandas libraries. It provides functionality to add, delete, and mark tasks as complete, as well as save the tasks to an Excel file. The GUI elements, such as labels, entry widgets, buttons, and listbox, are created and packed into the main window. The functions handle the logic behind each user action and update the display accordingly. The code provides a basic structure that can be extended and customized further to meet specific requirements.