**To-Do List Documentation**

**1. Introduction**

This documentation provides a detailed overview of a To-Do List application implemented in Python using the Tkinter library. The application offers basic functionalities such as adding tasks, marking tasks as completed, removing tasks, and displaying the current list of tasks.

**2. Project Structure**

**2.1. Import Statements**

Python code :

import tkinter as tk

from tkinter import messagebox

* The code imports the **tkinter** library for GUI development.
* **messagebox** is imported from **tkinter** to display messages in the application.

**2.2. Class: ToDoListGUI**

Python code:

GUI listbox with taclass ToDoListGUI:

def \_\_init\_\_(self, master):

# ... (Constructor and initialization)

def add\_task(self):

# ... (Add task to the list)

def mark\_as\_completed(self):

# ... (Mark task as completed)

def remove\_task(self):

# ... (Remove task from the list)

def display\_tasks(self):

# ... (Display current list of tasks)

def update\_listbox(self):

# ... (Update GUI listbox with tasks)sks)

* The main class represents the To-Do List application and defines methods for various functionalities.
* Methods include adding tasks, marking tasks as completed, removing tasks, displaying tasks, and updating the GUI listbox.

**2.3. Function: main()**

Python code :

def main():

# ... (Initialize the Tkinter root window and start the event loop)

if \_\_name\_\_ == "\_\_main\_\_":

main()

The **main()** function initializes the Tkinter root window and creates an instance of the **ToDoListGUI** class.

* The script checks if it is the main module and, if so, calls the **main()** function to start the application.

**3. Class Methods**

**3.1. \_\_init\_\_(self, master)**

* **Description:** Initializes the To-Do List GUI with the specified master (root) window.
* **Parameters:**
  + **master**: The master (root) window for the application.
* **Components:**
  + Entry widget for adding tasks.
  + Buttons for adding, marking as completed, removing, and displaying tasks.
  + Listbox for displaying the tasks.

**3.2. add\_task(self)**

* **Description:** Adds a new task to the To-Do List.
* **Functionality:**
  + Retrieves the task from the entry widget.
  + Validates if the task is not empty.
  + Appends the task to the list of tasks and updates the task listbox.
  + Shows a warning message if the task is empty.

**3.3. mark\_as\_completed(self)**

* **Description:** Marks the selected task as completed.
* **Functionality:**
  + Retrieves the index of the selected task from the listbox.
  + Marks the corresponding task as completed in the tasks list.
  + Updates the task listbox.
  + Shows a warning message if no task is selected.

**3.4. remove\_task(self)**

* **Description:** Removes the selected task from the To-Do List.
* **Functionality:**
  + Retrieves the index of the selected task from the listbox.
  + Removes the corresponding task from the tasks list.
  + Updates the task listbox.
  + Shows an information message with the removed task's details.
  + Shows a warning message if no task is selected.

**3.5. display\_tasks(self)**

* **Description:** Displays the current list of tasks.
* **Functionality:**
  + Creates a formatted string displaying each task with its status (completed or not).
  + Shows an information message with the formatted task display.
  + Shows a message if the task list is empty.

**3.6. update\_listbox(self)**

* **Description:** Updates the task listbox with the current list of tasks.
* **Functionality:**
  + Clears the existing items in the task listbox.
  + Inserts each task with its status into the task listbox.

**4. Function: main()**

**4.1. Description**

* Initializes the Tkinter root window and creates an instance of the **ToDoListGUI** class.
* Starts the Tkinter main event loop.

**4.2. Execution**

* The script checks if it is the main module and, if so, calls the **main()** function to start the To-Do List application.

**5. Execution and Usage**

**5.1. Preferred Development Environment**

* You can use any Python-supported development environment such as VS Code, PyCharm, Jupyter, or a simple text editor.

**5.2. Necessary Packages and Libraries**

* Ensure that you have a Python environment installed.
* The code uses the built-in **tkinter** library, which is usually included in standard Python installations.

**5.3. Execute the Code**

* Copy the provided code into a Python file (e.g., **todo\_app.py**).
* Open a terminal or command prompt.
* Navigate to the directory containing the Python file.
* Run the script using **python todo\_app.py**.
* The To-Do List GUI should appear, allowing you to interact with it.

**6. Implementation**

**6.1. Potential Enhancements**

* Consider adding data persistence (e.g., saving tasks to a file).
* Implement user authentication for personalized to-do lists.
* Improve the GUI layout and styling.

**6.2. Test Scenarios**

* Add tasks, mark them as completed, and remove tasks.
* Test warning messages by intentionally leaving the task entry empty.
* Verify the display of tasks with the "Display Tasks" button.
* Test the removal of tasks after marking them as completed.

**6.3. Exploration**

* Explore customization options provided by Tkinter for GUI elements.
* Research additional Tkinter features to enhance the application.

**6.4. Learning Opportunities**

* Learn more about Tkinter's event-driven programming model.
* Experiment with adding additional features or functionalities to the application.

Feel free to experiment, modify, and expand the code to further your understanding of GUI programming with Tkinter!

Top of Form