**We Create a To Do/Task List: Python Program**

In this Python project, we will build a GUI-based To Do List using the Tkinter module and File I/O. It is a beginner-level python project, and you should have basic knowledge of file I/O and Tkinter module before we begin and we will be able to apply them in real life.

To-Do Lists are actually a list of your daily tasks that you keep on your person to remind yourself of the tasks you have to accomplish that day.

In this project, we are going to make a GUI based python to-do list with options to add and delete items in your list.

The objective of this python project is to create a To-Do List. To build this, you will need a little understanding of File I/O and the Tkinter module.

Project Prerequisites:

To build python to-do list project, we will only need the **Tkinter** module to create the GUI, which comes pre-installed with Python.

The source code of python to-do list Project: [**To Do List Python Code**](https://techvidvan.s3.amazonaws.com/python-projects/to-do-list-python-code.zip)

***To Do/Task List Python Project File Structure:***

The files required in to-do list project are:

* **tasks.txt** – The text file where all our tasks will be stored
* **main.py** – The python script file

Here are the steps you will need to execute to build this python project:

1. Importing all the necessary libraries
2. Initializing the window and placing all the components in it
3. Creating the add\_item and delete\_item functions.

***Let’s take a closer look at these steps:***

1. **Importing the library:**

# Importing all the necessary modules

from tkinter import \*

1. **Initializing the window and placing all the components in it:**

# Initializing the python to do list GUI window

root = Tk()

root.title('TechVidvan To-Do List')

root.geometry('300x400')

root.resizable(0, 0)

root.config(bg="PaleVioletRed")

# Heading Label

Label(root, text='TechVidvan Python To Do List', bg='PaleVioletRed', font=("Comic Sans MS", 15), wraplength=300).place(x=35, y=0)

# Listbox with all the tasks with a Scrollbar

tasks = Listbox(root, selectbackground='Gold', bg='Silver', font=('Helvetica', 12), height=12, width=25)

scroller = Scrollbar(root, orient=VERTICAL, command=tasks.yview)

scroller.place(x=260, y=50, height=232)

tasks.config(yscrollcommand=scroller.set)

tasks.place(x=35, y=50)

# Adding items to the Listbox

with open('tasks.txt', 'r+') as tasks\_list:

for task in tasks\_list:

tasks.insert(END, task)

tasks\_list.close()

# Creating the Entry widget where the user can enter a new item

new\_item\_entry = Entry(root, width=37)

new\_item\_entry.place(x=35, y=310)

# Creating the Buttons

add\_btn = Button(root, text='Add Item', bg='Azure', width=10, font=('Helvetica', 12),

command=lambda: add\_item(new\_item\_entry, tasks))

add\_btn.place(x=45, y=350)

delete\_btn = Button(root, text='Delete Item', bg='Azure', width=10, font=('Helvetica', 12),

command=lambda: delete\_item(tasks))

delete\_btn.place(x=150, y=350)

# Finalizing the window

root.update()

root.mainloop()

**Explanation:**

* To initialize the to-do list window, you need to initialize the **Tk()** class to a variable. Then, you need to set the following attributes:
  1. **title()** method is used to give a title to the project window.
  2. **geometry()** method is used to set the initial geometry of window.
  3. **resizable()** method is used to allow/deny the user the permission to resize the python to-do list window. It takes truthy and falsy values and arguments to the parameters width and height.
  4. **config()** method is used to configure some extra attributes of the window, like bg or background to set the background color.
  5. **update()** and **.mainloop()** methods are used to put the window in a loop to prevent it from closing nanoseconds after it opens.
  6. Note: These lines of code will be considered the last lines that will be read by the interpreter to run the script.
* The **Label** class is used to create a label that displays the static screen on the window and set the following parameters to it:
  1. **master** attribute, positional argument root in this case, is used to specify the parent widget of the widget.
  2. **text** attribute is used to mention the text that will be displayed on the label.
  3. **font** is used to specify the font family, size and effects on the text.
  4. **wraplength** attribute is the length after which the text will be wrapped on the window.
* The **Button** class is used to create a button on the screen that executes a function as a command when pressed. You need to give it the parameters as mentioned in the code to add it to the screen.
  1. **command** attribute is the function that will run when the button is pressed. You do not need to use the lambda keyword if the function requires no arguments.
  2. **width** attribute is the width of the button in pixels.
* The **Entry** class is used to add an input widget to the window that accepts input data from the user. The attributes and methods of this class used in this python to do list project are:
  1. The **.get()** method is used to get the text inputted by the user in the widget. It requires no arguments for this widget.
* The **Listbox** class is used to add a list box to the window that displays multiple items on the screen. Its attributes, parameters and methods are:
  1. **selectbackground** attribute is used to mention the color of the background once an item is selected.
  2. **width** attribute represents the number of characters visible.
  3. **height** attribute represents the number of items visible on the screen.
  4. **yscrollcommand** attribute is used to pack the widget to a scrollbar that will navigate up and down the widget.
  5. **insert()** method is used to insert an element to the widget. It takes 2 arguments, index and element.
     1. index can either be a number or an acceptable tkinter constant, END, in this case.
     2. The element is the item to be added to the widget.
  6. **get()** method is used to get an element from the widget. It takes only one – index argument.
     1. The index is either a number or an acceptable tkinter constant, which in this case is ACTIVE to get the selected element.
  7. **yview()** method is used to make the widget vertically scrollable.
* The **Scrollbar** class is used to add a scroll bar to the widget on the window to navigate up and down, or right to left in said widget. Its attributes (undiscussed) are:
  1. The **orient** attribute is used to mention whether the scrollbar will scroll vertically or horizontally.
* The with open(“<filename>.<extension>”, “<mode>”) as alias is used to open a file in the <mode> mode, to perform actions on it.
  + The modes are:
    1. “r”, for reading a file
    2. “w”, for writing a file
    3. “a”, for appending text to the file
    4. “x”, to create a file
    5. “r+”, for reading and writing a file
    6. “t”, to open the file in text mode (default)
    7. “b”, to open the file in binary mode.
  + The actions that can be performed on it are:
    1. To close – **.close()** method
    2. To rewrite the file completely – **.write(‘’)** method when opened in “w” or “r+” mode
    3. To add text to the file **– .write(text\_to\_add)** method when opened in “a” mode
    4. To reduce the size**– .truncate(int(<size in bytes>))** method
    5. To store all the lines in a list **– .readlines()** method
    6. To remove a line **– .remove(line\_to\_remove)** method

1. **Creating the add\_item and delete\_item functions:**

# Adding and Deleting items functions

def add\_item(entry: Entry, listbox: Listbox):

new\_task = entry.get()

listbox.insert(END, new\_task)

with open('tasks.txt', 'a') as tasks\_list\_file:

tasks\_list\_file.write(f'\n{new\_task}')

def delete\_item(listbox: Listbox):

listbox.delete(ACTIVE)

with open('tasks.txt', 'r+') as tasks\_list\_file:

lines = tasks\_list\_file.readlines()

tasks\_list\_file.truncate()

for line in lines:

if listbox.get(ACTIVE) == line[:-2]:

lines.remove(line)

tasks\_list\_file.write(line)

tasks\_list\_file.close()

**Explanation:**

1. In this step, we will create two functions, one to add items to the python to-do list and the other to delete an item from the list.
2. In the **add\_item** function, the user will give it to arguments, one an Entry object and the other a Listbox object.
   1. In this function, we will give the Entry object’s user-provided text to a variable and then insert that variable to the last of our listbox object, and the last line of our text file.
3. In the delete\_item function, the user will give the function one argument that will be a listbox object.
   1. In this function, we will delete the item selected in the listbox object. We will first remove that element from the listbox and then go through every text file line to see if anything matches. If it does, we remove it too.

To Do List in Python

In this tutorial, we’re going to learn how to create a To-Do list in Python using Tkinter. But first, we need to understand what Tkinter is. Now, Python allows multiple options to implement GUI (Graphical User Interface). Tkinter is one of the most popular methods to create simple GUI based applications, using the Tk GUI toolkit.

Creating a GUI using Tkinter is actually very easy. Follow these steps:

1. We need to first import the Tkinter module in our source code.
2. Next, we can create a main window of the GUI and give it a suitable name.
3. Now, we can create any number of features in our GUI window, like message box, buttons, etc.
4. We can also specify an event trigger on each of the GUI features.

Let’s first check how to import the Tkinter GUI toolkit: -

from tkinter import \*

**Creating a To-Do List in Python**

Now, we’ll learn how to implement the same to create a To-Do list in Python:-

**import** tkinter

**from** *tkinter* **import** \*

**from** *tkinter* **import** messagebox

l=**[]**

c=1

**def** entryError**()** :

**if** insertField.get**()** == "" :

messagebox.showerror**(**"Error in input. Please input again"**)**

**return** 0

**return** 1

**def** insertTask**()**:

**global** c

value = entryError**()**

**if** **(**value == 0**)**:

**return**

var=insertField.get**()**+"\n"

l.append**(**var**)**

TextArea.insert**(**'end -1 chars', str**(**c**)** + "---> " + var**)**

c=c+1

del\_tf**()**

**def** del\_nf**()** :

nf.delete**(**0.0, END**)**

**def** del\_tf**()** :

insertField.delete**(**0, END**)**

**def** delete**()** :

**global** c

**if** **(**len**(**l**)**==0**)**:

messagebox.showerror**(**"There are no tasks"**)**

**return**

number = nf.get**(**1.0, END**)**

**if** **(**number=="\n"**)**:

messagebox.showerror**(**"input error"**)**

**return**

**else** :

task\_no = int**(**number**)**

del\_nf**()**

l.pop**(**task\_no - 1**)**

c=c-1

TextArea.delete**(**1.0, END**)**

**for** i **in** range**(**len**(**l**))**:

TextArea.insert**(**'end -1 chars',str**(**i + 1**)** + "---> " + l**[**i**])**

**if** **(**\_\_name\_\_ == "\_\_main\_\_"**)**:

window = Tk**()**

window.configure**(**background = "green"**)**

window.title**(**"To-Do List"**)**

window.geometry**(**"300x300"**)**

enterTask = Label**(**window, text = "Please enter your task", bg = "green"**)**

insertField = Entry**(**window**)**

Submit = Button**(**window, text = "Submit", fg = "Black", bg = "light green", command = insertTask**)**

TextArea = Text**(**window, height = 4, width = 25, font = "arial 13"**)**

taskNumber = Label**(**window, text = "Specify the task number that you want to remove, below", bg = "cyan"**)**

nf = Text**(**window, height = 1, width = 2, font = "arial 13"**)**

delete = Button**(**window, text = "Delete", fg = "Black", bg = "orange", command = delete**)**

Exit = Button**(**window, text = "Do you want to close?", fg = "Black", bg = "Red", command = exit**)**

enterTask.grid**(**row = 0, column = 2**)**

insertField.grid**(**row = 1, column = 2, ipadx = 50**)**

Submit.grid**(**row = 2, column = 2**)**

TextArea.grid**(**row = 3, column = 2, padx = 10, sticky = W**)**

taskNumber.grid**(**row = 4, column = 2, pady = 5**)**

nf.grid**(**row = 5, column = 2**)**

delete.grid**(**row = 6, column = 2, pady = 5**)**

Exit.grid**(**row = 7, column = 2**)**

window.mainloop**()**

After executing the above code, we’ll find the To-Do/Task list application has been created and launched. Now, we’re be able to use our own Python based To-Do/Task list.

In today’s fast-paced world, staying organized is crucial to manage our tasks and responsibilities effectively. A To-Do List App can be an invaluable tool for keeping track of everything you need to accomplish. In this blog post, we’ll guide you through the process of creating a simple To-Do List App using Python.

**Prerequisites:**

Before we dive into the coding part, make sure you have Python installed on your system. You can download and install it from the official Python website: Python Downloads.

**Setting Up the Project:**

Create a New Folder: Start by creating a new folder for your project. Name it something like “ToDoApp.”

Initialize Virtual Environment (Optional but Recommended): Open your terminal and navigate to the project folder. Create a virtual environment to isolate the project dependencies using the following command:

python -m venv venv

**Activate the virtual environment:**

On Windows:

venv\Scripts\activate

On macOS and Linux:

source venv/bin/activate

3. Install Dependencies: Now, let’s install the required libraries using the following command:

pip install PyQt5

Designing the User Interface

For this To-Do List App, we’ll be using the PyQt5 library to create the graphical user interface (GUI). The app will have a simple window with an input field for adding tasks and a list to display them.

Create a file named todo\_app.py in your project folder and start building the interface:

import sys

from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QHBoxLayout, QLineEdit, QPushButton, QListWidget

class ToDoApp(QWidget):

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

super().\_\_init\_\_()

self.setWindowTitle("To-Do List App")

self.setGeometry(100, 100, 400, 300)

self.tasks = []

self.layout = QVBoxLayout()

self.input\_field = QLineEdit()

self.add\_button = QPushButton("Add Task")

self.task\_list = QListWidget()

self.layout.addWidget(self.input\_field)

self.layout.addWidget(self.add\_button)

self.layout.addWidget(self.task\_list)

self.add\_button.clicked.connect(self.add\_task)

self.setLayout(self.layout)

def add\_task(self):

task = self.input\_field.text()

if task:

self.tasks.append(task)

self.task\_list.addItem(task)

self.input\_field.clear()

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

app = QApplication(sys.argv)

window = ToDoApp()

window.show()

sys.exit(app.exec\_())

Running the App

Save the todo\_app.py file and run it using your terminal:

python todo\_app.py

A simple window should appear with an input field and a button. Enter tasks into the input field and click the “Add Task” button to populate the list. Your basic To-Do List App is now up and running!

Creating a To-Do List App in Python using PyQt5 is a great way to learn about GUI programming and handling user input. This basic version can be extended with features like task deletion, marking tasks as complete, and even persisting tasks to a file.

Experiment, customize, and explore further to make your To-Do List App even more functional and user-friendly.

Happy coding and stay organized!

from tkinter import \*

import os

def read\_tasks\_from\_file():

with open('tasks.txt', 'r') as tasks\_file:

content = tasks\_file.readlines()

for x in content:

x = x.strip()

tasks\_dict[x.split(",")[0]] = x.split(",")[1]

return tasks\_dict

def complete():

tasks\_dict[tasks.get(ACTIVE)] = 'True'

tasks.itemconfig(list(tasks\_dict).index(tasks.get(ACTIVE)), foreground='grey')

tasks.selection\_clear(0, END)

def incomplete():

tasks\_dict[tasks.get(ACTIVE)] = 'False'

tasks.itemconfig(list(tasks\_dict).index(tasks.get(ACTIVE)), foreground='black')

tasks.selection\_clear(0, END)

def add\_item():

new\_task = new\_item\_entry.get()

check = True

for task in tasks\_dict:

if(new\_task.lower().strip() == task.lower().strip()):

check = False

if(new\_task != '' and check):

tasks\_dict[new\_task] = 'False'

tasks.insert(END, new\_task)

new\_item\_entry.delete(0, END)

status.config(text="Task Added")

else:

if(new\_task == ''):

status.config(text="Please enter a task")

else:

status.config(text="Task already present")

def delete\_item():

del tasks\_dict[tasks.get(ACTIVE)]

tasks.delete(ACTIVE)

status.config(text="Task Deleted")

def on\_closing():

with open('tasks.txt', 'w') as final:

for element in tasks\_dict:

final.write(element+","+tasks\_dict[element]+"\n")

root.destroy()

if \_\_name\_\_ == '\_\_main\_\_':

tasks\_dict = {}

if not (os.path.exists('tasks.txt')):

#Creating the txt file

file = open('tasks.txt', 'x')

file.close()

tasks\_dict = read\_tasks\_from\_file()

# Initializing the python to do list GUI window

root = Tk()

root.title('To-Do')

root.geometry('300x460')

root.resizable(0, 0)

root.config(bg="PaleVioletRed")

# Heading Label

Label(root, text='Python To Do List', bg='PaleVioletRed', font=("Comic Sans MS", 15), wraplength=300).place(x=35, y=0)

# Listbox with all the tasks with a Scrollbar

tasks = Listbox(root, selectbackground='Gold', bg='Silver', font=('Helvetica', 12), height=12, width=25)

scroller = Scrollbar(root, orient=VERTICAL, command=tasks.yview)

scroller.place(x=260, y=50, height=232)

tasks.config(yscrollcommand=scroller.set)

tasks.place(x=35, y=50)

count = 0

#Populate the ListBox

for task in tasks\_dict:

tasks.insert(END, task)

if(tasks\_dict[task] == 'True'):

tasks.itemconfig(count, foreground='grey')

count += 1

# Creating the Entry widget where the user can enter a new item

new\_item\_entry = Entry(root, width=37)

new\_item\_entry.place(x=35, y=310)

# Creating the Buttons

Button(root, text='Add Item', bg='Azure', width=10, font=('Helvetica', 12),

command=add\_item).place(x=45, y=350)

Button(root, text='Delete Item', bg='Azure', width=10, font=('Helvetica', 12),

command=delete\_item).place(x=150, y=350)

Button(root, text='Complete', bg='Azure', width=10, font=('Helvetica', 12),

command=complete).place(x=45, y=390)

Button(root, text='Incomplete', bg='Azure', width=10, font=('Helvetica', 12),

command=incomplete).place(x=150, y=390)

status = Label(root, text='', fg='white', bg='PaleVioletRed', font=("Comic Sans MS", 12), wraplength=300)

status.place(x=65, y=430)

# Finalizing the window

root.protocol("WM\_DELETE\_WINDOW", on\_closing)

root.mainloop()

# Importing all the necessary modules

from tkinter import \*

# Initializing the python to do list GUI window

root = Tk()

root.title('To-Do List')

root.geometry('300x400')

root.resizable(0, 0)

root.config(bg="PaleVioletRed")

# Heading Label

Label(root, text='Python To Do List', bg='PaleVioletRed', font=("Comic Sans MS", 15), wraplength=300).place(x=35, y=0)

# Listbox with all the tasks with a Scrollbar

tasks = Listbox(root, selectbackground='Gold', bg='Silver', font=('Helvetica', 12), height=12, width=25)

scroller = Scrollbar(root, orient=VERTICAL, command=tasks.yview)

scroller.place(x=260, y=50, height=232)

tasks.config(yscrollcommand=scroller.set)

tasks.place(x=35, y=50)

# Adding items to the Listbox

with open('tasks.txt', 'r+') as tasks\_list:

for task in tasks\_list:

tasks.insert(END, task)

tasks\_list.close()

# Creating the Entry widget where the user can enter a new item

new\_item\_entry = Entry(root, width=37)

new\_item\_entry.place(x=35, y=310)

# Creating the Buttons

add\_btn = Button(root, text='Add Item', bg='Azure', width=10, font=('Helvetica', 12),

command=lambda: add\_item(new\_item\_entry, tasks))

add\_btn.place(x=45, y=350)

delete\_btn = Button(root, text='Delete Item', bg='Azure', width=10, font=('Helvetica', 12),

command=lambda: delete\_item(tasks))

delete\_btn.place(x=150, y=350)

# Finalizing the window

root.update()

root.mainloop()

# Adding and Deleting items functions

def add\_item(entry: Entry, listbox: Listbox):

new\_task = entry.get()

listbox.insert(END, new\_task)

with open('tasks.txt', 'a') as tasks\_list\_file:

tasks\_list\_file.write(f'\n{new\_task}')

def delete\_item(listbox: Listbox):

listbox.delete(ACTIVE)

with open('tasks.txt', 'r+') as tasks\_list\_file:

lines = tasks\_list\_file.readlines()

tasks\_list\_file.truncate()

for line in lines:

if listbox.get(ACTIVE) == line[:-2]:

lines.remove(line)

tasks\_list\_file.write(line)

tasks\_list\_file.close()