**Project Name:**

**Text Editor with Python**

**Personal Details:**

**Name**: VINNAKOTA AKHIL SAI SRINIVAS

**Intern ID-** CC40693 **Designation-** Python Development Intern  
**Batch-** Jul 2023

**Mail id:** [20981a4960@raghuenggcollege.in](mailto:20981a4960@raghuenggcollege.in)

Context Introduction and usage:

In this Python project, we will create a GUI-based Text Editor using only the Tkinter module in Python. It is a beginner-level project, and be able to use some amazing basic GUI components in real life. Let’s get started!,A notepad is a text-only editor that only works with .txt files but can still read and edit file formats that can be edited. This project is a very simple one to make, but you are still going to have a lot of fun creating it.

Objective:

The objective of this is to create a GUI-based Text Editor in Python. To build this, you will need a basic understanding of Tkinter library but some knowledge of rarer and intermediate widgets like the Text widget and the Scrollbar widget is also required.

Project Prerequisites:

To build this text editor in python, we will need the following libraries:

**1. Tkinter –** To create the GUI.  
**2. PIL (Python Image Library) –** To give the GUI window an icon.  
**3. OS –** To get the path of the file.

The PIL library does not come pre-installed with Python, so you will have to run the following command in the terminal to install it.

### Python Text Editor Project File Structure:

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

Step 1.

Importing the libraries.  
Step 2.

Initializing the GUI window.  
Step 3.

Defining the functions for every menu option.  
Step 4.

Creating and placing the Menu widget on the top and placing components in them.  
Step 5.

Setting and placing the Text and Scrollbar widgets.

#### **Step 1. Importing the libraries:**

from tkinter import \*

import tkinter.filedialog as fd

import tkinter.messagebox as mb

from PIL import Image, ImageTk

import os

#### **Step 2. Initializing the GUI window (and adding an icon to it) to Create Python Text Editor Project**

# Initializing the window to create python text editor

root = Tk()

root.title("Untitled - Notepad")

root.geometry('800x500')

root.resizable(0, 0)

root.columnconfigure(0, weight=1)

root.rowconfigure(0, weight=1)

icon = ImageTk.PhotoImage(Image.open('Notepad.png'))

root.iconphoto(**False**, icon)

# Finalizing the window

root.update()

root.mainloop()

**Explanation:**

* The **Tk()** class is used to initialize the main window to a variable.
  + The **.title()** method is used to give a title to the window.
  + The **.geometry()** method is used to set the dimensions of the window, in pixels.
  + The **.resizable()** method is used to allow/deny the user the permission to resize the window.
  + The **.columnconfigure()** and **.rowconfigure()** methods are used to specify the maximum number of columns and rows that will be available in the window.
  + The **.iconphoto()** method is used to set a PhotoImage object as the title bar icon of the window. The first argument default specifies whether the particular image will be the default for all windows, and the second argument image is the PhotoImage object that will become the title bar icon.
* In the **PIL** library, we will use:
  + The **open** method of the **Image** class to open the image file, and
  + The **PhotoImage** class of the **ImageTk** module converts the opened image into a PhotoImage object so that it can be acceptable to the Tkinter toolkit as a title bar icon.

#### **Step 3. Defining the functions for every menu option:**

# Creating all the functions of all the buttons in the NotePad

def open\_file():

file = fd.askopenfilename(defaultextension='.txt', filetypes=[('All Files', '\*.\*'), ("Text File", "\*.txt\*")])

if file != '':

root.title(f"{os.path.basename(file)}")

text\_area.delete(1.0, END)

with open(file, "r") as file\_:

text\_area.insert(1.0, file\_.read())

file\_.close()

else:

file = None

def open\_new\_file():

root.title("Untitled - Notepad")

text\_area.delete(1.0, END)

def save\_file():

global text\_area

file = text\_area.get(1.0, END)

if file == '':

file = None

else:

file = open(file, "w")

file.write(text\_area.get(1.0, END))

file.close()

if file is None:

file = fd.asksaveasfilename(initialfile='Untitled.txt', defaultextension='.txt',

filetypes=[("Text File", "\*.txt\*"), ("Word Document", '\*,docx\*'), ("PDF", "\*.pdf\*")])

else:

file = open(file, "w")

file.write(text\_area.get(1.0, END))

file.close()

root.title(f"{os.path.basename(file)} - Notepad")

def exit\_application():

root.destroy()

def copy\_text():

text\_area.event\_generate("<<Copy>>")

def cut\_text():

text\_area.event\_generate("<<Cut>>")

def paste\_text():

text\_area.event\_generate("<<Paste>>")

def select\_all():

text\_area.event\_generate("<<Control-Keypress-A>>")

def delete\_last\_char():

text\_area.event\_generate("<<KP\_Delete>>")

def about\_notepad():

mb.showinfo("About Notepad", "This is just another Notepad, but this is better than all others")

def about\_commands():

commands = """

Under the File Menu:

- 'New' clears the entire Text Area

- 'Open' clears text and opens another file

- 'Save As' saves your file in the same / another extension

Under the Edit Menu:

- 'Copy' copies the selected text to your clipboard

- 'Cut' cuts the selected text and removes it from the text area

- 'Paste' pastes the copied/cut text

- 'Select All' selects the entire text

- 'Delete' deletes the last character

"""

mb.showinfo(title="All commands", message=commands, width=60, height=40)

**Explanation:**

* In the **filedialog** module, we are going to use:
  + The **askopenfilename** function is used to ask the user to open a filename, whose text we will place in our text area.
  + The **asksaveasfilename** function is used to ask the user to save a file in a particular location as a particular file type.
* In the **messagebox** module, we will use:
  + The **showinfo** function is used to show an informational message as an external window.
    - title attribute – title of the box
    - message attribute – message to be displayed on the box
    - The width attribute specifies the number of characters that can be in one line.
    - The height attribute specifies the number of maximum lines in the message box.
* The **.event\_generate()** method, applicable on all tkinter widgets, is used to generate a window event and arranges for it to be processed as if it were in a windows terminal.
  + The argument it requires is a key or a combination in the form of “<<Keystroke(s)>>”.  
    You can view more information on this here: Key names

#### **Step 4. Creating and placing the Menu widget on the top and placing components in them:**

menu\_bar = Menu(root)

# Adding the File Menu and its components to create Python Text Editor

file\_menu = Menu(menu\_bar, tearoff=False, activebackground='DodgerBlue')

file\_menu.add\_command(label="New", command=open\_new\_file)

file\_menu.add\_command(label="Open File", command=open\_file)

file\_menu.add\_command(label="Save As", command=save\_file)

file\_menu.add\_separator()

file\_menu.add\_command(label="Close File", command=exit\_application)

menu\_bar.add\_cascade(label="File", menu=file\_menu)

# Adding the Edit Menu and its components

edit\_menu = Menu(menu\_bar, tearoff=False, activebackground='DodgerBlue')

edit\_menu.add\_command(label='Copy', command=copy\_text)

edit\_menu.add\_command(label='Cut', command=cut\_text)

edit\_menu.add\_command(label='Paste', command=paste\_text)

edit\_menu.add\_separator()

edit\_menu.add\_command(label='Select All', command=select\_all)

edit\_menu.add\_command(label='Delete', command=delete\_last\_char)

menu\_bar.add\_cascade(label="Edit", menu=edit\_menu)

# Adding the Help Menu and its components

help\_menu = Menu(menu\_bar, tearoff=False, activebackground='DodgerBlue')

help\_menu.add\_command(label='About Notepad', command=about\_notepad)

help\_menu.add\_command(label='About Commands', command=about\_commands)

menu\_bar.add\_cascade(label="Help", menu=help\_menu)

root.config(menu=menu\_bar)

**Explanation:**

* The **Menu** class is used to add a menu to the parent widget.
  + The **tearoff** attribute is used to allow/deny the menu to be dragged to another position in the parent widget.
  + The **activebackground** attribute is used to specify the color of the background of the selected cascade/element in the Menu bar.
  + The **.add\_cascade()** method is used to add a sub-menu to the main menu.
    - The label attribute designates the text that will be visible on the window for the sub-menu.
    - The menu attribute is used to specify the Menu object that will be added as the sub-menu. [This attribute is also available to the Tk() object to add a super menu to the window]
* The **.add\_separator()** method is used to add a Separator (a ttk widget) component to the menu it is associated with.
* The **.add\_command()** method is used to add a new item to the menu object.

#### **Step 5. Setting and placing the Text and Scrollbar widgets**

# Setting the basic components of the window

text\_area = Text(root, font=("Times New Roman", 12))

text\_area.grid(sticky=NSEW)

scroller = Scrollbar(text\_area, orient=VERTICAL)

scroller.pack(side=RIGHT, fill=Y)

scroller.config(command=text\_area.yview)

text\_area.config(yscrollcommand=scroller.set)

**Explanation:**

* The **Text** class is used to add a multiline input field to the parent widget.
  + The **yscrollcommand** attribute sets a Scrollbar object to navigate the widget up and down.
  + The **.get()** method, which takes the start and end arguments, is used to get the values in the object from the ‘start’ index to the ‘end’ index.
  + The **.delete()** method is used to delete the value in the object from the start to the end indices.
  + The **.insert()** method, which takes the arguments index and text, is used to insert the text in the text argument at the ‘index’ index.
  + The **.yview()** method is used to make the widget vertically scrollable.
* The **Scrollbar** class is used to add a Scrollbar to the parent widget.
  + The **orient** attribute is used to designate whether the object will vertically scroll the widget mentioned in the command attribute, or it will horizontally scroll the widget.
  + The **.set()** method is used to allow the Scrollbar object to scroll the text.
* The **.pack()** method is one of the 3 Tkinter Geometry handlers. This method is used to pack a widget to a particular edge of the parent widget.
  + The **side** attribute is used to mention the edge to which the widget will be packed.
  + The **fill** attribute is used to state if the widget will fill any extra space than was allocated to it by the packer.

### Python Text Editor Project Output

