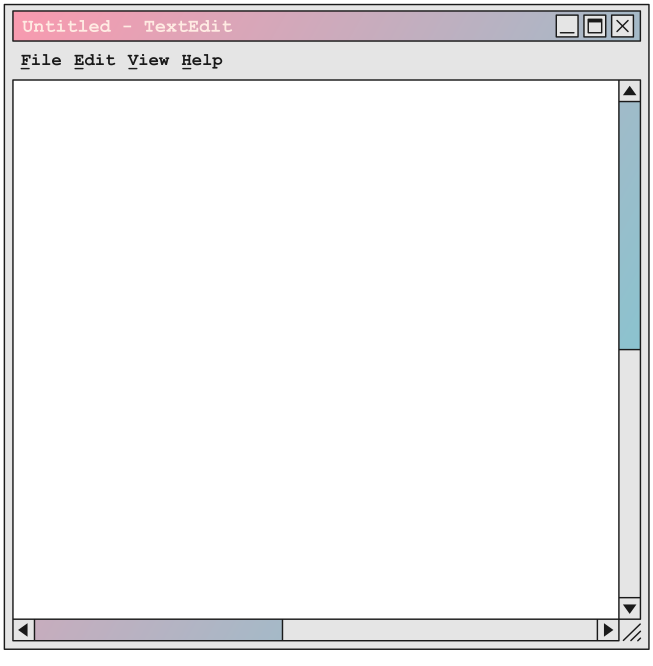
# Text Editor Mini project

Report-02

Implementation and Output

Submitted by
MERWIN PINTO SRN 202100102
ROLL NO 1



## Problem Statement:

Design and implement a basic text editor with fundamental features, such as text entry, editing, saving, and opening files. Include support for features like cut, copy, paste, undo, and redo. Develop a simple user interface for your editor and provide documentation on how to use these basic features effectively.

## Language used for Implementation:

The language chosen for implementing the text editor is ‘Python’ because of its extensive libraries and user friendliness. Python provided all in one solution for creation of GUI and also implement the functionalities as per the requirements. For implementing the GUI python library ‘Tkinter’ was used.

Python Program:

```
import tkinter as tk

from tkinter import filedialog

class TextEditorApp:

  def __init__(self):

    self.root = tk.Tk()

    self.root.title("Text Editor")


    # Create a toolbar

    toolbar = tk.Frame(self.root)

    toolbar.pack(side="top", fill="x")


    # Create toolbar buttons

    cut_button = tk.Button(toolbar, text="Cut", command=self.cut_text)

    cut_button.pack(side="left")

    copy_button = tk.Button(toolbar, text="Copy", command=self.copy_text)

    copy_button.pack(side="left")
```

```
paste_button = tk.Button(toolbar, text="Paste", command=self.paste_text)
paste_button.pack(side="left")
undo_button = tk.Button(toolbar, text="Undo", command=self.undo_text)
undo_button.pack(side="left")
redo_button = tk.Button(toolbar, text="Redo", command=self.redo_text)
redo_button.pack(side="left")
open_button = tk.Button(toolbar, text="Open", command=self.open_file)
open_button.pack(side="left")
save_button = tk.Button(toolbar, text="Save", command=self.save_file)
save_button.pack(side="left")

# Create a text widget for the main editing area
self.text_widget = tk.Text(self.root)
self.text_widget.pack(expand="yes", fill="both")

# Maintain a history of text changes for undo
self.history = []
self.history_index = -1

# Bind keyboard shortcuts
self.text_widget.bind("<Key>", self.on_text_change)
self.text_widget.bind("<Control-x>", lambda e: self.cut_text())
self.text_widget.bind("<Control-c>", lambda e: self.copy_text())
self.text_widget.bind("<Control-v>", lambda e: self.paste_text())
self.text_widget.bind("<Control-z>", lambda e: self.undo_text())
```

```
def on_text_change(self, event):
    self.record_change()

def cut_text(self):
    selected_text = self.text_widget.get("sel.first", "sel.last")
    self.copy_to_clipboard(selected_text)
    self.text_widget.delete("sel.first", "sel.last")
    self.record_change()

def copy_text(self):
    selected_text = self.text_widget.get("sel.first", "sel.last")
    self.copy_to_clipboard(selected_text)

def paste_text(self):
    clipboard_text = self.get_clipboard_text()
    self.text_widget.insert("insert", clipboard_text)
    self.record_change()

def undo_text(self):
    if self.history_index > 0:
        self.history_index -= 1
        self.text_widget.delete("1.0", "end")
        self.text_widget.insert("1.0", self.history[self.history_index])

def redo_text(self):
    if self.history_index < len(self.history) - 1:
```

```
        self.history_index += 1
        self.text_widget.delete("1.0", "end")
        self.text_widget.insert("1.0", self.history[self.history_index])

    def copy_to_clipboard(self, text):
        self.root.clipboard_clear()
        self.root.clipboard_append(text)

    def get_clipboard_text(self):
        return self.root.clipboard_get()

    def record_change(self):
        text = self.text_widget.get("1.0", "end")
        if self.history_index < len(self.history) - 1:
            self.history = self.history[:self.history_index + 1]
        self.history.append(text)
        self.history_index = len(self.history) - 1

    def open_file(self):
        file_path = filedialog.askopenfilename()
        if file_path:
            with open(file_path, 'r') as file:
                self.text_widget.delete("1.0", "end")
                self.text_widget.insert("1.0", file.read())

    def save_file(self):
```

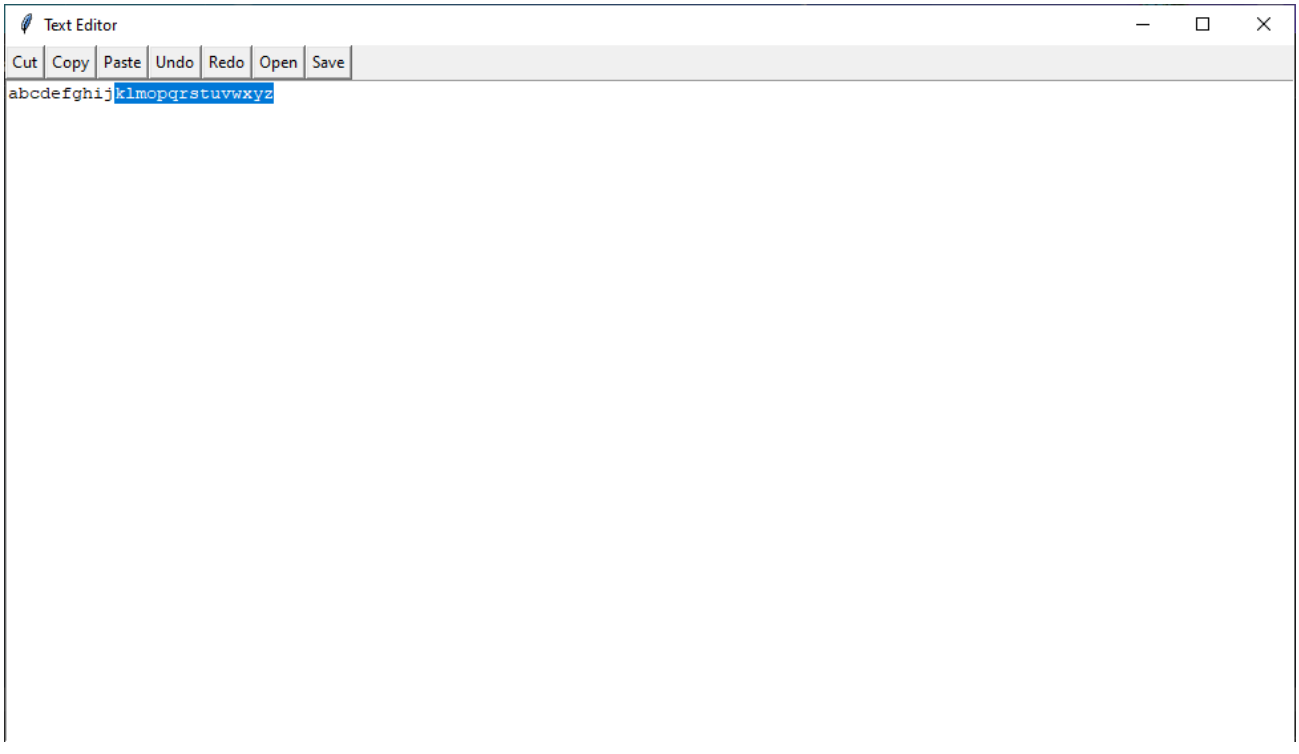
```
        file_path = filedialog.asksaveasfilename(defaultextension='.txt')
        if file_path:
            with open(file_path, 'w') as file:
                file.write(self.text_widget.get("1.0", "end"))

if __name__ == "__main__":
    app = TextEditorApp()
    app.root.mainloop()
```

The above code creates a GUI for a text editor. The text editor has Cut, Copy, Paste, Undo, Redo, Open and Save options. Also the inherent keyboard shortcuts like ‘Ctrl+Z’ for Cut option are also functional. When the user chooses the open and save option the directory window is opened allowing the user to select where to store and under what name.

**Output:**

- Initial GUI of Text Editor:

- Text Editor While Copying:

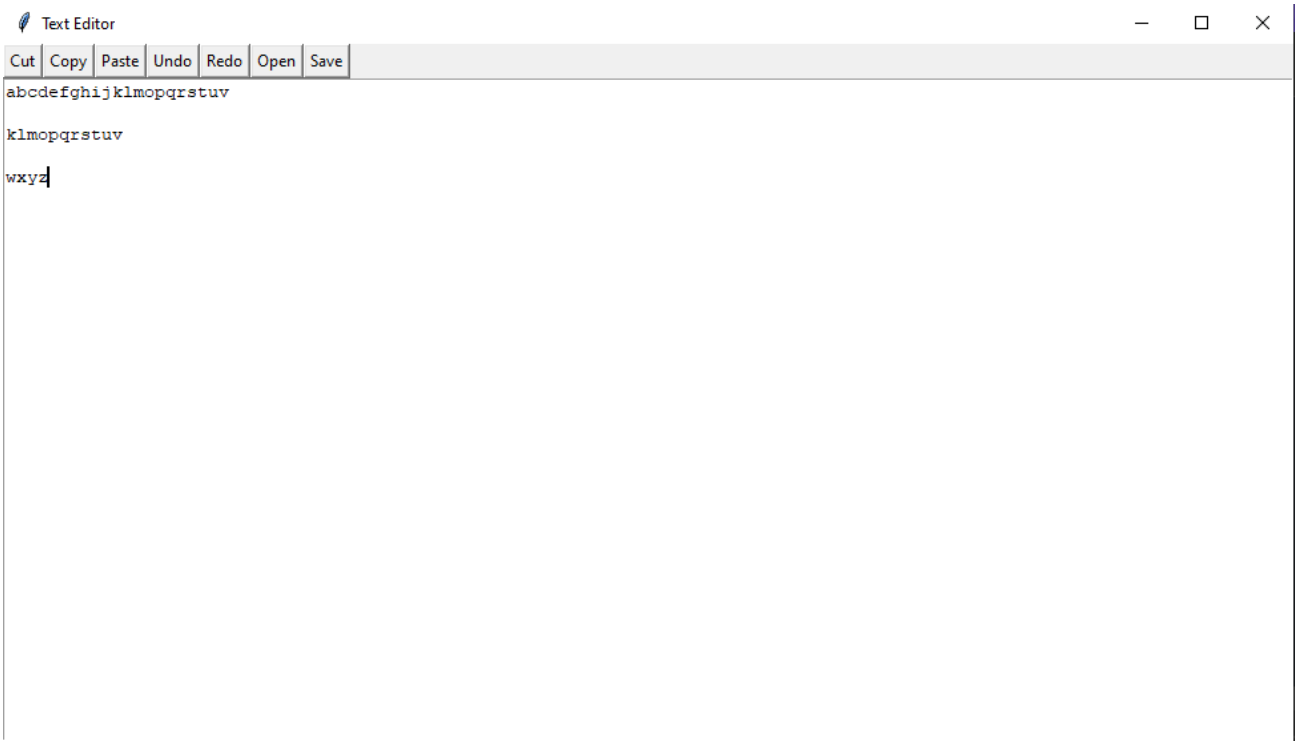



- Text Editor after Paste:


---

- Text Editor When Cut is used:

(From above state of text editor we will now cut 'wxyz’ from the line number 2 and paste it on new line)

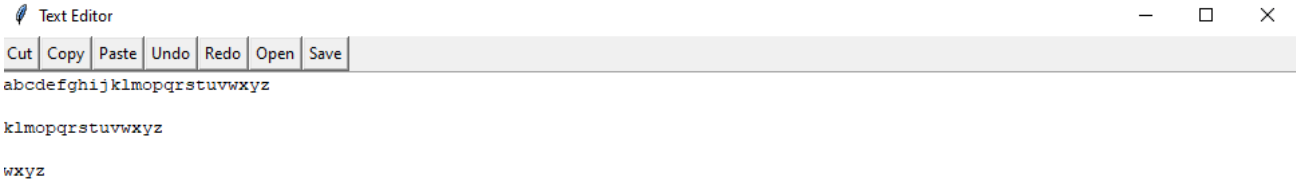



- Undo Option:

From above example we will undo pasting of ‘zyx’ from line number 3.

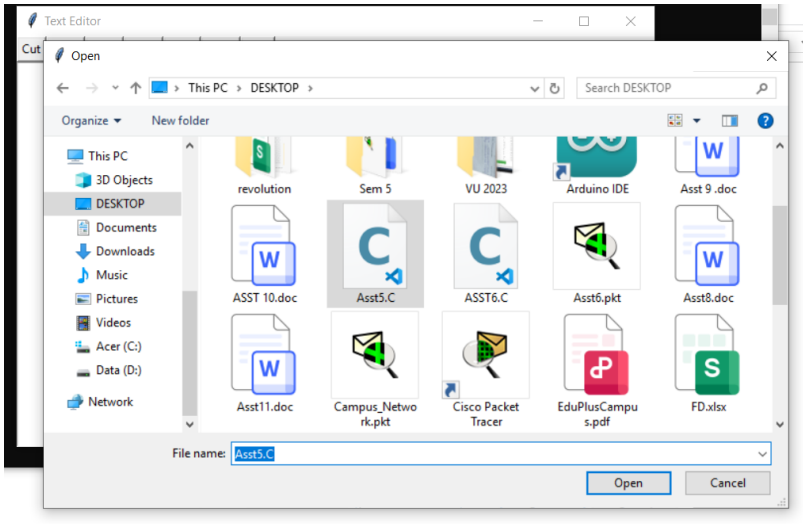
- Redo Option:

We will redo the previous undo option:




- Open and Save option:

Both these options when pressed open the windows directory to choose the file and location where to save.



## Conclusion:

In the above report we saw how the problem statement was implemented. ‘Python’ programming language was used for implementation because of it’s extensive libraries and user friendliness. The code implements GUI for text editor and created functional Cut, Copy, Paste, Undo, Redo, Open and Save options.