CAD LAB
Python PROJECT

TEXT EDITOR

Group Members:
Suchith - **121601016**
Suresh Khetawat - **121601013**

Introduction :-

A text editor is computer program that edits plain text. Text editors are provided with operating systems and software development packages and can be used to change configuration files.
Some text editors are small and simple, while others offer broad and complex functions.
Text editors are intended to open and save text files containing either plain text or anything that can be interpreted as plain text, including the markup for rich text or the markup for something else.

Features and Command Used :-

Our Project mainly based on GUI. Initially we started with making the layout of widget and organise it in the interface.

The making of interface include adding the menu bar and cascading it to the submenu points.

```
def layout(self):

    # file menu
    self.filemenu=tk.Menu(self.menu)
    self.menu.add_cascade(label='File',menu=self.filemenu)
    self.filemenu.add_command(label='New',command=self.Newfile,accelerator="Ctrl+N")
    self.filemenu.add_command(label='Open',command=self.Openfile,accelerator="Ctrl+O")
    self.filemenu.add_command(label='Save',command=self.Savefile,accelerator="Ctrl+S")
    self.filemenu.add_command(label='Save As',command=self.Saveasfile)
    self.filemenu.add_command(label='Exit',command=self.Exiteditor,accelerator="Ctrl+Q")

    # edit menu
    self.editmenu=tk.Menu(self.menu)
    self.menu.add_cascade(label='Edit',menu=self.editmenu)
    self.editmenu.add_command(label='Undo',command=self.Undo,accelerator="Ctrl+Z")
    self.editmenu.add_command(label='Redo',command=self.Redo,accelerator="Ctrl+R")
    self.editmenu.add_command(label='Cut',command=self.Cut,accelerator="Ctrl+X")
    self.editmenu.add_command(label='Copy',command=self.Copy,accelerator="Ctrl+C")
    self.editmenu.add_command(label='Paste',command=self.Paste,accelerator="Ctrl+V")
    self.editmenu.add_command(label='Select All',command=self.Select,accelerator="Ctrl+A")

    # search menu
    self.searchmenu=tk.Menu(self.menu)
    self.menu.add_cascade(label='Search',menu=self.searchmenu)
    self.searchmenu.add_command(label='Find',command=self.search,accelerator="Ctrl+F")
    # def addShortcuts(self):
```

And binding the submenu which are file menu, edit menu ,search with keyboard shortcuts using command below.

window.bind_all("<Control-button>",classname.filename)

```
# adding shortcuts for file menu
window.bind_all("<Control-o>",editor.Openfile)
window.bind_all("<Control-O>",editor.Openfile)

window.bind_all("<Control-S>",editor.Savefile)
window.bind_all("<Control-s>",editor.Savefile)

window.bind_all("<Control-n>",editor.Newfile)
window.bind_all("<Control-N>",editor.Newfile)

window.bind_all("<Control-q>",editor.Exiteditor)
window.bind_all("<Control-Q>",editor.Exiteditor)

# adding shortcuts for edit menu
window.bind_all("<Control-z>",editor.Undo)
window.bind_all("<Control-Z>",editor.Undo)

window.bind_all("<Control-R>",editor.Redo)
window.bind_all("<Control-r>",editor.Redo)

#self.text.bind("<Control-x>",self.Cut)
#self.text.bind("<Control-X>",self.Cut)
```

Opening and saving new file:

We have used python tkinter file dialog command to browse,select the file when we open the file and giving the name when we save the file.
Then after that if it's open we are taking the file as read mode and then reading the file and inserting this in the editor.
When we want to save the file we read the present text written in the text widget and then opening the file in write mode and then changing the content of the file.
The other thing we have added is save file function is that when a file is opened then there is no need to type the filename and any changes made to file are saved with that name.
At last we have added shortcut's like for opening new file ctrl+N,for opening file ctrl+o and for saving the file we have added ctrl+s.

The function for opening new file is same as saving a file without name and save as function is same as save function.

Exit Function:

This is somewhat tricky that we have checked that file is currently saved or not. If not it pops a message asking that if they want to save the file and then usual save function goes on. Even if the file name is null then also message pops up and after the process we will use command **root.destroy()** which makes tkinter to exit from the main loop.

Save Function:

```
if self.file is None:
    self.file=tkf.asksaveasfile(mode='w',defaultextension='.txt',
                        title='Save')

    if self.file is None:
        return;
        self.name=self.file.name
# getting the text
    self.textwritten=str(self.text.get('1.0','end-1c'))
    print(self.file)
    print(self.textwritten)
    self.file.write(self.textwritten)
    self.file.close()
else:
    # the file is closed so we have to open it
    self.file=open(self.name,'w')
    # chaning the mode from read to write
    # there is no need of taking the name but just to be safe
    self.name=self.file
    # now i have to update the text written in the editor
    self.textwritten=str(self.text.get('1.0','end-1c'))
    # before rewriting the we have to delete everything
    print(self.textwritten)
    # writing this in file
    # before writing erasing the file content
   # self.file.
    self.file.write(self.textwritten)
    self.file.close()
```

Open Function:

```
def Openfile(self,*args):
    self.file=tkf.askopenfile(mode='r',filetypes=[('Text files','*.txt')],
                              title='Open File')

    if(self.file is None):
        return;

    print(self.file)
    # writing the name
    self.name=self.file.name
    # taking the text
    self.textwritten=self.file.read()
    print(self.textwritten)
    self.text.delete('1.0','end')
    self.text.insert('1.0',self.textwritten)
    self.file.close()
```

Some General features necessary for the text-editor:-

The basic function for the text editor such as copy, paste, undo, redo are added by using the command given below

```
def function(self,*args):
    self.text.event_generate("<<function>>")
```

```
# edit menu functions
def Undo(self,*args):
    self.text.event_generate("<<Undo>>")

def Redo(self,*args):
    self.text.event_generate("<<Redo>>")

def Cut(self,*args):
    self.text.event_generate("<<Cut>>")

def Copy(self,*args):
    self.text.event_generate("<<Copy>>")
```

Exit Function

```
def Exiteditor(self,*args):
    print(len(str(self.text.get('1.0','end-1c'))))
    print(len(str(self.textwritten)))
    if (self.file is None) or (str(self.text.get('1.0','end-1c')) is not self.textwritten):
        answer=messagebox.askquestion("Alert!!","Do you want close the file Without Saving ?")
        if answer=='yes':
            window.destroy()
        elif self.file is None:
            self.file=tkf.asksaveasfile(mode='w',defaultextension='.txt',
                        title='New File')

            if self.file is None:
                return;
            self.name=self.file.name
            # getting the text
            self.textwritten=str(self.text.get('1.0','end-1c'))
            print(self.file)
            print(self.textwritten)
            self.file.write(self.textwritten)
            self.file.close()

        elif str(self.text.get('1.0','end-1c')) is not self.textwritten:
            # the file is closed so we have to open it
            self.file=open(self.name,'w')
            # chaning the mode from read to write
            # there is no need of taking the name but just to be safe
            self.name=self.file
            # now i have to update the text written in the editor
            self.textwritten=str(self.text.get('1.0','end-1c'))
            # before rewriting the we have to delete everything
            print(self.textwritten)
            # writing this in file
            # before writing erasing the file content
            #self.file.
            self.file.write(self.textwritten)
            self.file.close()
    else:
        window.destroy()
```

Search Function:-

As we discussed of adding the function to search the word and highlighting with background and foreground colour we want. To get this, we applied the below command.

self.text.tag_config('found', foreground='white', background='red')

Search is done by using tag by like when we search for some words it takes the initial string element and search and name a tag after search for the whole string with the length of the string.

```
def search(self,*args):
    self.text.tag_remove('found', '1.0', tk.END)
    target = simpledialog.askstring('Find', 'Search String:')
    if target:
        idx = '1.0'
        while 1:
            idx = self.text.search(target, idx, nocase=1, stopindex=tk.END)
            if not idx: break
            lastidx = '%s+%dc' % (idx, len(target))
            self.text.tag_add('found', idx, lastidx)
            idx = lastidx
        self.text.tag_config('found', foreground='white', background='red')
```

Format Menu:

In this we have added options like bold,italic,underline,changing the background color and font color. For background color we have used color chooser which gives options to choose the color and then binded it to the text widget.

As for the font colour change and the background colour change it asks the colour to choose by combination of 3 different colours. There are some default font provided by font families which are used.

CODE:-

```
# text editor try
import tkinter as tk
import tkinter.filedialog as tkf
from tkinter import messagebox
from tkinter import simpledialog
from tkinter import colorchooser
from tkinter import font


class TextEditor:

    def __init__(self,window):
```

```
    # will make the borders and boundaries
    self.text=tk.Text(window,undo=True)
    self.text.pack(fill=tk.X)
    self.menu=tk.Menu(window)
    window.config(menu=self.menu)

    # some initializations
    self.name=None
    self.file=None
    self.textwritten=None
    self.word=tk.StringVar()
    self.count=tk.StringVar()
    self.s=None

def layout(self):

    # file menu
    self.filemenu=tk.Menu(self.menu)
    self.menu.add_cascade(label='File',menu=self.filemenu)
    self.filemenu.add_command(label='New',command=self.Newfile,accelerator="Ctrl+N")
    self.filemenu.add_command(label='Open',command=self.Openfile,accelerator="Ctrl+O")
    self.filemenu.add_command(label='Save',command=self.Savefile,accelerator="Ctrl+S")
    self.filemenu.add_command(label='Save As',command=self.Saveasfile)
    self.filemenu.add_command(label='Exit',command=self.Exiteditor,accelerator="Ctrl+Q")

    # edit menu
    self.editmenu=tk.Menu(self.menu)
    self.menu.add_cascade(label='Edit',menu=self.editmenu)
    self.editmenu.add_command(label='Undo',command=self.Undo,accelerator="Ctrl+Z")
    self.editmenu.add_command(label='Redo',command=self.Redo,accelerator="Ctrl+R")
    self.editmenu.add_command(label='Cut',command=self.Cut,accelerator="Ctrl+X")
    self.editmenu.add_command(label='Copy',command=self.Copy,accelerator="Ctrl+C")
    self.editmenu.add_command(label='Paste',command=self.Paste,accelerator="Ctrl+V")
    self.editmenu.add_command(label='Select All',command=self.Select,accelerator="Ctrl+A")

    # search menu
    self.searchmenu=tk.Menu(self.menu)
    self.menu.add_cascade(label='Search',menu=self.searchmenu)
    self.searchmenu.add_command(label='Find',command=self.search,accelerator="Ctrl+F")
    # def addShortcuts(self):

    # format menu
```

```
    self.formatmenu=tk.Menu(self.menu)

    fsubmenu = tk.Menu(self.formatmenu, tearoff=0)
    ssubmenu = tk.Menu(self.formatmenu, tearoff=0)

    for option in font.families():
        fsubmenu.add_command(label=option, command = lambda: font.Font.configure(family=option))
    for value in range(1,31):
        ssubmenu.add_command(label=str(value), command = lambda: font.Font.configure(size=value))

    self.menu.add_cascade(label='Format',menu=self.formatmenu)
    self.formatmenu.add_command(label='Change Background',command=self.changebackground)
    self.formatmenu.add_command(label='Change font color',command=self.changefc)

    self.formatmenu.add_cascade(label="Font", underline=0, menu=fsubmenu)
    self.formatmenu.add_cascade(label="Size", underline=0, menu=ssubmenu)

    self.formatmenu.add_command(label="Bold", command=self.bold, accelerator="Ctrl+B")
    self.formatmenu.add_command(label="Italic", command=self.italic, accelerator="Ctrl+I")
    self.formatmenu.add_command(label="Underline", command=self.underline, accelerator="Ctrl+U")

    # help menu
    self.help=tk.Menu(self.menu)
    self.menu.add_cascade(label='Help',menu=self.help)
    self.help.add_command(label='About',command=self.about)


# file menu functions
def Newfile(self,*args):
    self.file=tkf.asksaveasfile(mode='w',defaultextension='.txt',
                                title='New File')

    if self.file is None:
        return;
    self.name=self.file.name
    # getting the text
    self.textwritten=str(self.text.get('1.0','end-1c'))
    print(self.file)
    print(self.textwritten)
    self.file.write(self.textwritten)
    self.file.close()
```

```
def Openfile(self,*args):
    self.file=tkf.askopenfile(mode='r',filetypes=[('Text files','*.txt')],
                                          title='Open File')

    if(self.file is None):
        return;

    print(self.file)
    # writing the name
    self.name=self.file.name
    # taking the text
    self.textwritten=self.file.read()
    print(self.textwritten)
    self.text.delete('1.0','end')
    self.text.insert('1.0',self.textwritten)
    self.file.close()
def Savefile(self,*args):
    if self.file is None:
        self.file=tkf.asksaveasfile(mode='w',defaultextension='.txt',
                                title='Save')

        if self.file is None:
            return;
            self.name=self.file.name
    # getting the text
        self.textwritten=str(self.text.get('1.0','end-1c'))
        print(self.file)
        print(self.textwritten)
        self.file.write(self.textwritten)
        self.file.close()
    else:
        # the file is closed so we have to open it
        self.file=open(self.name,'w')
        # chaning the mode from read to write
        # there is no need of taking the name but just to be safe
        self.name=self.file
        # now i have to update the text written in the editor
        self.textwritten=str(self.text.get('1.0','end-1c'))
        # before rewriting the we have to delete everything
        print(self.textwritten)
        # writing this in file
        # before writing erasing the file content
       # self.file.
```

```
            self.file.write(self.textwritten)
            self.file.close()

    def Saveasfile(self,*args):
        self.file=tkf.asksaveasfile(mode='w',defaultextension='.txt',
                                    title='Save As')

        if self.file is None:
            return;
        self.name=self.file.name
        # getting the text
        self.textwritten=str(self.text.get('1.0','end-1c'))
        print(self.file)
        print(self.textwritten)
        self.file.write(self.textwritten)
        self.file.close()




def Exiteditor(self,*args):
        print(len(str(self.text.get('1.0','end-1c'))))
        print(len(str(self.textwritten)))
        if (self.file is None) or (str(self.text.get('1.0','end-1c')) is not self.textwritten):
            answer=messagebox.askquestion("Alert!!","Do you want close the file Without Saving ?")
            if answer=='yes':
                window.destroy()
            elif self.file is None:
                self.file=tkf.asksaveasfile(mode='w',defaultextension='.txt',
                                    title='New File')

                if self.file is None:
                    return;
                self.name=self.file.name
                # getting the text
                self.textwritten=str(self.text.get('1.0','end-1c'))
                print(self.file)
                print(self.textwritten)
                self.file.write(self.textwritten)
                self.file.close()

            elif str(self.text.get('1.0','end-1c')) is not self.textwritten:
```

```
            # the file is closed so we have to open it
            self.file=open(self.name,'w')
            # chaning the mode from read to write
            # there is no need of taking the name but just to be safe
            self.name=self.file
            # now i have to update the text written in the editor
            self.textwritten=str(self.text.get('1.0','end-1c'))
            # before rewriting the we have to delete everything
            print(self.textwritten)
            # writing this in file
            # before writing erasing the file content
            #self.file.
            self.file.write(self.textwritten)
            self.file.close()
    else:
        window.destroy()
# edit menu functions
def Undo(self,*args):
    self.text.event_generate("<<Undo>>")

def Redo(self,*args):
    self.text.event_generate("<<Redo>>")

def Cut(self,*args):
    self.text.event_generate("<<Cut>>")

def Copy(self,*args):
    self.text.event_generate("<<Copy>>")

def Paste(self,*args):
    self.text.event_generate("<<Paste>>")

def Select(self,*args):
    self.text.tag_add('sel','1.0','end')

def search(self,*args):
    self.text.tag_remove('found', '1.0', tk.END)
    target = simpledialog.askstring('Find', 'Search String:')
    if target:
        idx = '1.0'
        while 1:
            idx = self.text.search(target, idx, nocase=1, stopindex=tk.END)
            if not idx: break
```

```
            lastidx = '%s+%dc' % (idx, len(target))
            self.text.tag_add('found', idx, lastidx)
            idx = lastidx
        self.text.tag_config('found', foreground='white', background='red')


# format menu functions
def changebackground(self):
    (triple, hexstr) = colorchooser.askcolor()
    if hexstr:
        self.text.config(bg=hexstr)

def changefc(self):
    (triple, hexstr) = colorchooser.askcolor()
    if hexstr:
        self.text.config(fg=hexstr)

def bold(self, *args):  # Works only if text is selected
    try:
        current_tags = self.text.tag_names("sel.first")
        print(current_tags)
        if "bold" in current_tags:
            self.text.tag_remove("bold", "sel.first", "sel.last")
        else:
            print("here")
            self.text.tag_add("bold", "sel.first", "sel.last")
            bold_font = font(self.text, self.text.cget("font"))
            print(type(bold_font))
            bold_font.configure(weight="bold")
            self.text.tag_configure("bold", font=bold_font)
    except:
        pass

def italic(self, *args): # Works only if text is selected
    try:
        current_tags = self.text.tag_names("sel.first")
        if "italic" in current_tags:
            self.text.tag_remove("italic", "sel.first", "sel.last")
        else:
            self.text.tag_add("italic", "sel.first", "sel.last")
            italic_font = font(self.text, self.text.cget("font"))
            italic_font.configure(slant="italic")
            self.text.tag_configure("italic", font=italic_font)
```

```
        except:
            pass


    def underline(self, *args):       # Works only if text is selected
        try:
            current_tags = self.text.tag_names("sel.first")
            if "underline" in current_tags:
                self.text.tag_remove("underline", "sel.first", "sel.last")
            else:
                self.text.tag_add("underline", "sel.first", "sel.last")
                underline_font = font(self.text, self.text.cget("font"))
                underline_font.configure(underline=1)
                self.text.tag_configure("underline", font=font.underline_font)
        except:
            pass

    def about(self,*args):
        messagebox.showinfo(title='About',message='This is Text editor implemented in python')



window=tk.Tk()
editor=TextEditor(window)
editor.layout()

# adding shortcuts for file menu
window.bind_all("<Control-o>",editor.Openfile)
window.bind_all("<Control-O>",editor.Openfile)

window.bind_all("<Control-S>",editor.Savefile)
window.bind_all("<Control-s>",editor.Savefile)

window.bind_all("<Control-n>",editor.Newfile)
window.bind_all("<Control-N>",editor.Newfile)

window.bind_all("<Control-q>",editor.Exiteditor)
window.bind_all("<Control-Q>",editor.Exiteditor)

# adding shortcuts for edit menu
window.bind_all("<Control-z>",editor.Undo)
window.bind_all("<Control-Z>",editor.Undo)
```

```
window.bind_all("<Control-R>",editor.Redo)
window.bind_all("<Control-r>",editor.Redo)

#self.text.bind("<Control-x>",self.Cut)
#self.text.bind("<Control-X>",self.Cut)

#self.text.bind("<Control-c>",self.Copy)
#self.text.bind("<Control-C>",self.Copy)

#self.text.bind("<Control-v>",self.Paste)
#self.text.bind("<Control-V>",self.Paste)

window.bind_all("<Control-a>",editor.Select)
window.bind_all("<Control-A>",editor.Select)

window.bind_all("<Control-F>",editor.search)
window.bind_all("<Control-f>",editor.search)

window.bind_all("<Control-b>",editor.bold)
window.bind_all("<Control-B>",editor.bold)

window.bind_all("<Control-I>",editor.italic)
window.bind_all("<Control-i>",editor.italic)

window.bind_all("<Control-U>",editor.underline)
window.bind_all("<Control-u>",editor.underline)

window.mainloop()
```

METHODS USED FOR DEBUGGING:

Spyder (editor) helped me a lot for debugging.The functionality of executing blocks was very helpful.Furthermore its ability to hold values made debugging a lot faster so i did not have to execute the whole program or input repeatedly. I have also added print statements to get to know what's happening in the run time of the program.

This made debugging easier and allowed me to focus on one task at a time instead of handling the whole program.

LIBRARY USED :-

As project in mainly done using python tkinter library In this library we have used file dialog,message box, simple dialog,color chooser and font and mainly python text widget.

CONCLUSION:-

Here we conclude that the our project on the topic “Text editors” with the extreme satisfaction and contentment. This report contains brief definitions of the text editor together with its features and functions.
Also we have sincerely included the references from where we have taken help.
We tried it to add as many as features and presently some features like font doesn’t work.

REFERENCES:

Video Tutorials of python tkinter:
https://www.youtube.com/playlist?list=PL6gx4Cwl9DGBwibXFtPtflztSNPGuIB_d

Python Tkinter Text widget:
https://www.tutorialspoint.com/python/tk_text.htm

Python Tkinter Text:
https://www.tutorialspoint.com/python/tk_fonts.htm

Tkinter Documentation:
http://www.tkdocs.com/

Tkinter text widget Documentation:
http://www.tkdocs.com/tutorial/text.html