TEXT EDITOR

AIM:- To create a text editor using tkinkter and the file handling operations in python.

THEORY:-

TKINKTER- The tkinter library in Python is a standard library module used for creating graphical user interfaces (GUIs). It is the standard GUI toolkit for Python and provides a way to create windows, dialogs, buttons, menus, and other common GUI elements. tkinter is a thin object-oriented layer on top of the Tcl/Tk GUI toolkit, which is a cross-platform GUI toolkit that is widely used in various programming languages.

Key Features of tkinter

1. **Widgets**:
    - tkinter includes a variety of widgets such as buttons, labels, text boxes, frames, menus, and more. These can be used to create the user interface of an application.
2. **Geometry Management**:
    - tkinter provides different geometry management methods such as pack, grid, and

place to control the layout of widgets in the window.

3. **Event Handling**:
    - It supports event-driven programming, allowing developers to bind functions or methods to various events such as button clicks, keyboard presses, and mouse movements.
4. **Canvas**:
    - The Canvas widget allows for drawing shapes, lines, and other custom graphics. It is useful for creating custom drawings and animations.
5. **Dialogs and Messages**:
    - Built-in dialogs for common tasks such as file selection, color selection, and displaying messages.
6. **Cross-Platform**:
    - tkinter is cross-platform, meaning that applications built with it can run on Windows, macOS, and Linux.

Q] WHAT IS A TEXT EDITOR?

A text editor is a software application designed for editing plain text files. Unlike word processors, which are typically used for formatted documents, text editors handle text without any additional formatting or styling. They are essential tools for programmers, writers, and anyone who needs to work with text files.

Key Features of Text Editors

1. **Basic Editing Functions**:
    - **Text Manipulation**: Cut, copy, paste, delete, and undo/redo actions.
    - **Find and Replace**: Search for specific text and replace it with new text.
2. **Syntax Highlighting** (in advanced text editors):
    - **Color Coding**: Displays text in different colors based on its syntactic role, which helps in writing code by making it easier to read and debug.
3. **Line Numbering**:
    - **Numbering Lines**: Helps in navigating and referencing specific lines of code or text.
4. **Search and Navigation**:
    - **Search Functionality**: Allows for quick searching of text within the file.
    - **Go to Line**: Directly jump to a specific line number.

5. **File Handling**:
    - **Multiple Files**: Open and edit multiple files simultaneously using tabs or separate windows.
    - **Save and Load**: Save files in various formats and load files from disk.
6. **Customization**:
    - **Themes and Fonts**: Change the appearance of the editor to suit personal preferences.
    - **Plugins and Extensions**: Extend functionality with additional features.

EXAMPLES

1] Notepad (Windows): A basic text editor for simple text file editing without advanced features.
2] TextEdit (macOS): A basic editor with minimal formatting options, suitable for simple text editing.
3] Sublime Text: A fast and customizable text editor with support for multiple programming languages and syntax highlighting.
4] Visual Studio Code (VS Code): A feature-rich, extensible editor with integrated development tools, syntax highlighting, and debugging support.
5] Atom: An open-source, highly customizable editor known for its rich plugin ecosystem and user-friendly interface.

Q] WHAT IS THE DIFFERENCE BETWEEN TEXT EDITOR AND WORD PROCESSOR?

Feature	Text Editor	Word Processor
Primary Use	Editing plain text files	Creating and formatting documents
File Format	Typically handles plain text (.txt)	Handles formatted text (.doc, .docx, .rtf)
Formatting Options	Minimal or none; mainly plain text	Extensive formatting options (fonts, colors, styles)
Features	Syntax highlighting (in advanced editors), basic text manipulation	Rich text formatting, spell check, tables, images, headers, footers
Document Structure	Linear, simple text structure	Complex document structure with sections, styles, and layout
Example Software	Notepad, Sublime Text, Visual Studio Code	Microsoft Word, Google Docs, LibreOffice Writer
Intended Users	Programmers, system administrators, users needing quick text edits	Writers, journalists, office workers, and anyone needing document formatting
Additional Tools	Often lacks advanced tools; focus on text	Includes tools for document creation like templates, bibliography, and mail merge

Q] TALK ABOUT FILE HANDLING FUNCTIONS IN PYTHON.

File handling in Python involves the following key functions and concepts:

- **Opening a File**: Use the open() function to open a file, specifying the file path and mode ('r' for reading, 'w' for writing, 'a' for appending, etc.). This function returns a file object.
- **Reading from a File**: Methods such as read(), readline(), and readlines() are used to read the contents of a file. read() reads the entire file, readline() reads one line at a time, and readlines() reads all lines into a list.
- **Writing to a File**: Use write() to write a string to a file and writelines() to write a list of strings. These methods modify the file's contents.
- **Closing a File**: Always close a file with close() to free system resources and ensure changes are saved. Using a context manager with the with statement automatically handles file closing.
- **File Operations**: Functions like os.rename() and os.remove() are used for renaming and deleting files, respectively. os.path.exists() checks if a file exists.

CODE AND OUTPUT:-

```
import tkinter as tk
from tkinter import filedialog, messagebox, font, colorchooser
from tkinter import ttk

class EnhancedTextEditor:
    def __init__(self, root):
        self.root = root
        self.root.title("Enhanced Text Editor")
        self.root.geometry("800x600")

        # Set up the menu
        self.menu = tk.Menu(self.root)
        self.root.config(menu=self.menu)

        self.file_menu = tk.Menu(self.menu, tearoff=0)
        self.menu.add_cascade(label="File", menu=self.file_menu)
        self.file_menu.add_command(label="New", command=self.new_file)
        self.file_menu.add_command(label="Open", command=self.open_file)
        self.file_menu.add_command(label="Save", command=self.save_file)
        self.file_menu.add_command(label="Save As", command=self.save_as_file)
        self.file_menu.add_command(label="Exit", command=self.root.quit)

        # Set up the toolbar
        self.toolbar = tk.Frame(self.root, bd=1, relief=tk.RAISED)
        self.toolbar.pack(side=tk.TOP, fill=tk.X)

        # Font selection
        self.font_var = tk.StringVar()
        self.font_var.set("Arial")
        self.font_menu = ttk.Combobox(self.toolbar,
textvariable=self.font_var, values=self.get_fonts(), state="readonly")
        self.font_menu.pack(side=tk.LEFT, padx=2, pady=2)
        self.font_menu.bind("<<ComboboxSelected>>", self.change_font)

        # Font size selection
        self.font_size_var = tk.StringVar()
        self.font_size_var.set("12")
        self.font_size_menu = ttk.Combobox(self.toolbar,
textvariable=self.font_size_var, values=self.get_font_sizes(),
state="readonly")
        self.font_size_menu.pack(side=tk.LEFT, padx=2, pady=2)
        self.font_size_menu.bind("<<ComboboxSelected>>",
self.change_font_size)

        # Bold, Italic, Underline buttons
```

```
        self.bold_button = tk.Button(self.toolbar, text="Bold",
command=self.toggle_bold)
        self.bold_button.pack(side=tk.LEFT, padx=2, pady=2)
        self.italic_button = tk.Button(self.toolbar, text="Italic",
command=self.toggle_italic)
        self.italic_button.pack(side=tk.LEFT, padx=2, pady=2)
        self.underline_button = tk.Button(self.toolbar, text="Underline",
command=self.toggle_underline)
        self.underline_button.pack(side=tk.LEFT, padx=2, pady=2)

        # Text color button
        self.color_button = tk.Button(self.toolbar, text="Text Color",
command=self.change_text_color)
        self.color_button.pack(side=tk.LEFT, padx=2, pady=2)

        # Background color button
        self.bg_color_button = tk.Button(self.toolbar, text="Background
Color", command=self.change_bg_color)
        self.bg_color_button.pack(side=tk.LEFT, padx=2, pady=2)

        # Alignment buttons
        self.left_align_button = tk.Button(self.toolbar, text="Left",
command=self.align_left)
        self.left_align_button.pack(side=tk.LEFT, padx=2, pady=2)
        self.center_align_button = tk.Button(self.toolbar, text="Center",
command=self.align_center)
        self.center_align_button.pack(side=tk.LEFT, padx=2, pady=2)
        self.right_align_button = tk.Button(self.toolbar, text="Right",
command=self.align_right)
        self.right_align_button.pack(side=tk.LEFT, padx=2, pady=2)

        # Zoom buttons
        self.zoom_in_button = tk.Button(self.toolbar, text="Zoom In",
command=self.zoom_in)
        self.zoom_in_button.pack(side=tk.LEFT, padx=2, pady=2)
        self.zoom_out_button = tk.Button(self.toolbar, text="Zoom Out",
command=self.zoom_out)
        self.zoom_out_button.pack(side=tk.LEFT, padx=2, pady=2)

        # Undo/Redo buttons
        self.undo_button = tk.Button(self.toolbar, text="Undo",
command=self.undo)
        self.undo_button.pack(side=tk.LEFT, padx=2, pady=2)
        self.redo_button = tk.Button(self.toolbar, text="Redo",
command=self.redo)
        self.redo_button.pack(side=tk.LEFT, padx=2, pady=2)

        # Highlight color button
```

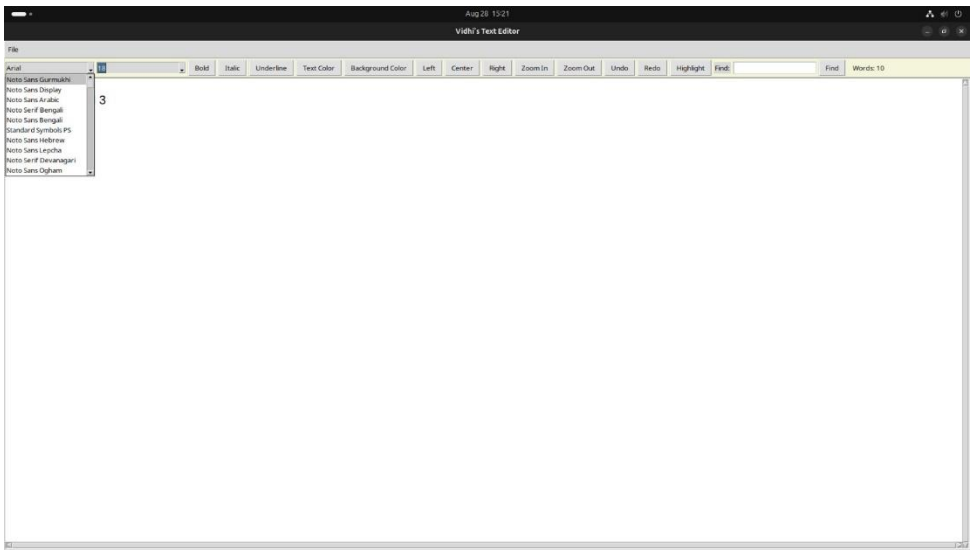
```
        self.highlight_button = tk.Button(self.toolbar, text="Highlight",
command=self.change_highlight_color)
        self.highlight_button.pack(side=tk.LEFT, padx=2, pady=2)

        # Find text entry
        self.find_label = tk.Label(self.toolbar, text="Find:")
        self.find_label.pack(side=tk.LEFT, padx=2, pady=2)
        self.find_entry = tk.Entry(self.toolbar)
        self.find_entry.pack(side=tk.LEFT, padx=2, pady=2)
        self.find_button = tk.Button(self.toolbar, text="Find",
command=self.find_text)
        self.find_button.pack(side=tk.LEFT, padx=2, pady=2)

        # Word count label
        self.word_count_label = tk.Label(self.toolbar, text="Words: 0")
        self.word_count_label.pack(side=tk.LEFT, padx=10, pady=2)

        # Set up the text widget
        self.text = tk.Text(self.root, wrap='word', undo=True)
        self.text.pack(expand=1, fill='both')
        self.text.bind('<KeyRelease>', self.update_word_count)

        # Set up the scrollbars
        self.scrollbar_y = tk.Scrollbar(self.text)
        self.scrollbar_x = tk.Scrollbar(self.text, orient='horizontal')

        self.text.config(yscrollcommand=self.scrollbar_y.set)
        self.text.config(xscrollcommand=self.scrollbar_x.set)

        self.scrollbar_y.pack(side='right', fill='y')
        self.scrollbar_x.pack(side='bottom', fill='x')

        self.scrollbar_y.config(command=self.text.yview)
        self.scrollbar_x.config(command=self.text.xview)

    def get_fonts(self):
        return list(font.families())

    def get_font_sizes(self):
        return [str(size) for size in range(8, 72, 2)]

    def change_font(self, event=None):
        font_name = self.font_var.get()
        font_size = self.font_size_var.get()
        self.text.configure(font=(font_name, font_size))

    def change_font_size(self, event=None):
        self.change_font()
```

```
    def toggle_bold(self):
        self.apply_tag("bold", "bold", font=("Arial", 12, "bold"))

    def toggle_italic(self):
        self.apply_tag("italic", "italic", font=("Arial", 12, "italic"))

    def toggle_underline(self):
        self.apply_tag("underline", "underline", font=("Arial", 12,
"underline"))

    def apply_tag(self, tag_name, tag_id, **config):
        if self.text.tag_names("sel.first"):
            if tag_name in self.text.tag_names("sel.first"):
                self.text.tag_remove(tag_name, "sel.first", "sel.last")
            else:
                self.text.tag_add(tag_name, "sel.first", "sel.last")
                self.text.tag_configure(tag_name, **config)

    def change_text_color(self):
        color = colorchooser.askcolor()[1]
        if color:
            self.text.tag_add("color", "sel.first", "sel.last")
            self.text.tag_configure("color", foreground=color)

    def change_bg_color(self):
        color = colorchooser.askcolor()[1]
        if color:
            self.text.config(bg=color)

    def change_highlight_color(self):
        color = colorchooser.askcolor()[1]
        if color:
            self.text.tag_add("highlight", "sel.first", "sel.last")
            self.text.tag_configure("highlight", background=color)

    def zoom_in(self):
        current_font = font.Font(self.text, self.text.cget("font"))
        new_size = min(current_font.actual()["size"] + 2, 72)
        self.text.configure(font=(current_font.actual()["family"], new_size))

    def zoom_out(self):
        current_font = font.Font(self.text, self.text.cget("font"))
        new_size = max(current_font.actual()["size"] - 2, 8)
        self.text.configure(font=(current_font.actual()["family"], new_size))

    def undo(self):
        self.text.event_generate("<<Undo>>")
```

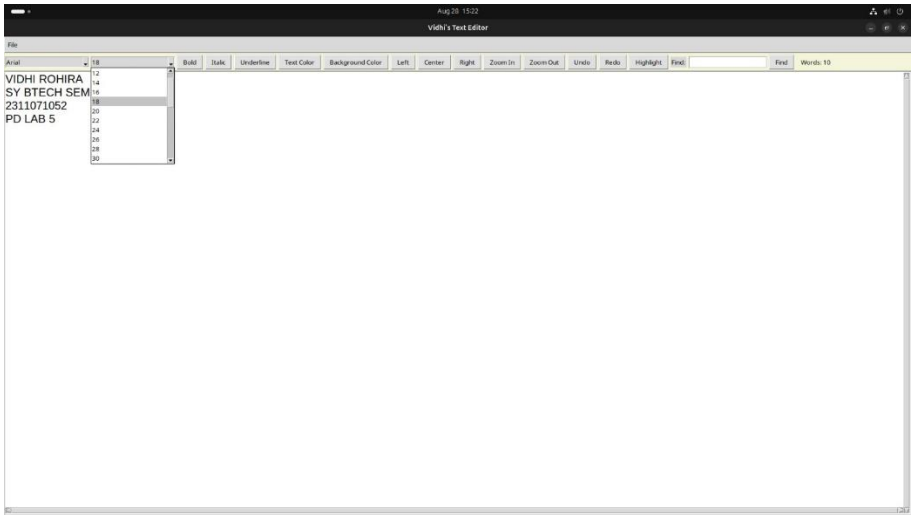
```
    def redo(self):
        self.text.event_generate("<<Redo>>")

    def align_left(self):
        self.apply_alignment("left", "left")

    def align_center(self):
        self.apply_alignment("center", "center")

    def align_right(self):
        self.apply_alignment("right", "right")

    def apply_alignment(self, tag_name, justify):
        self.text.tag_add(tag_name, "1.0", tk.END)
        self.text.tag_configure(tag_name, justify=justify)

    def find_text(self):
        search_term = self.find_entry.get()
        if search_term:
            self.text.tag_remove("highlight", "1.0", tk.END)
            idx = '1.0'
            while True:
                idx = self.text.search(search_term, idx, nocase=True,
stopindex=tk.END)
                if not idx:
                    break
                end_idx = f"{idx}+{len(search_term)}c"
                self.text.tag_add("highlight", idx, end_idx)
                idx = end_idx
            self.text.tag_configure("highlight", background="yellow",
foreground="black")

    def update_word_count(self, event=None):
        text_content = self.text.get(1.0, tk.END).strip()
        word_count = len(text_content.split())
        self.word_count_label.config(text=f"Words: {word_count}")

    def new_file(self):
        self.text.delete(1.0, tk.END)
        self.update_word_count()

    def open_file(self):
        file_path = filedialog.askopenfilename(defaultextension=".txt",
                                               filetypes=[("Text files",
"*.txt"),
                                                          ("All files",
"*.*")])
```

```
        if file_path:
            with open(file_path, 'r') as file:
                content = file.read()
                self.text.delete(1.0, tk.END)
                self.text.insert(tk.END, content)
                self.update_word_count()

    def save_file(self):
        file_path = filedialog.asksaveasfilename(defaultextension=".txt",
                                                 filetypes=[("Text files",
"*.txt"),
                                                            ("All files",
"*.*")])
        if file_path:
            try:
                with open(file_path, 'w') as file:
                    content = self.text.get(1.0, tk.END)
                    file.write(content)
            except Exception as e:
                messagebox.showerror("Error", f"Failed to save file: {e}")

    def save_as_file(self):
        self.save_file()

if __name__ == "__main__":
    root = tk.Tk()
    editor = EnhancedTextEditor(root)
    root.mainloop()
```

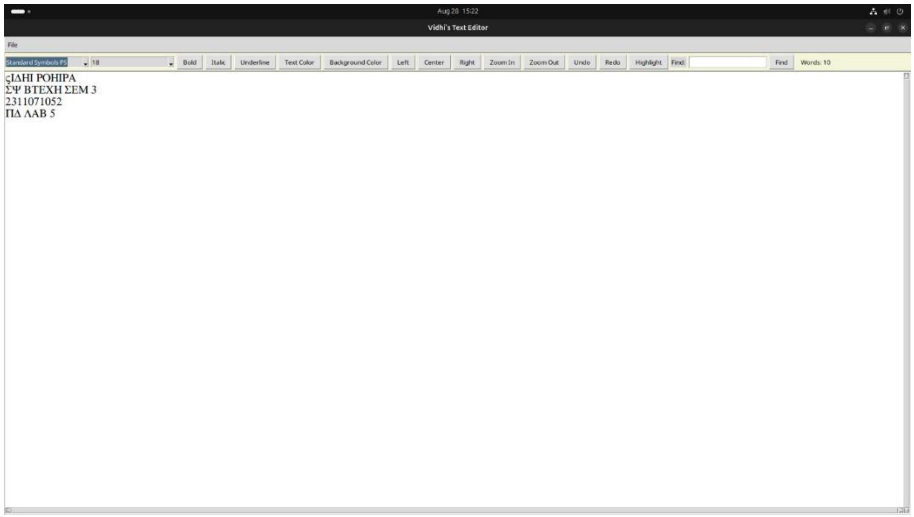
OUTPUT:-



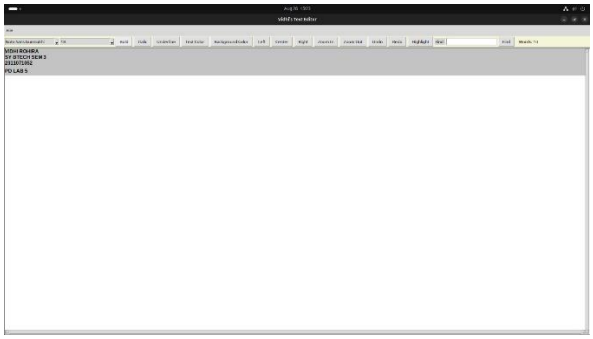
OFFERS CHANGE IN FONT ↟

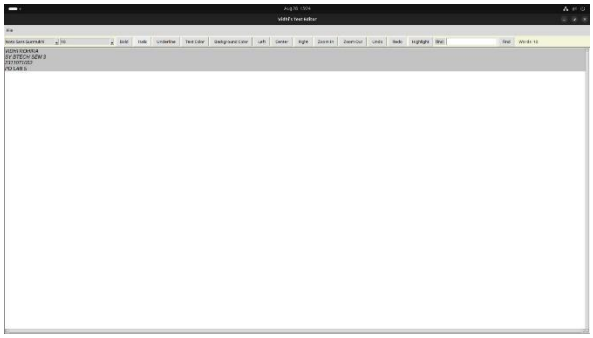


OFFERS CHANGE IN FONT SIZE ↟

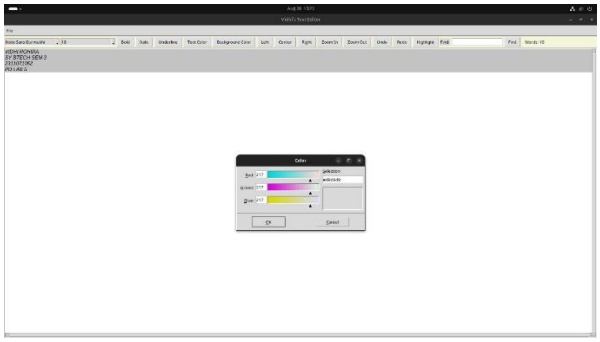


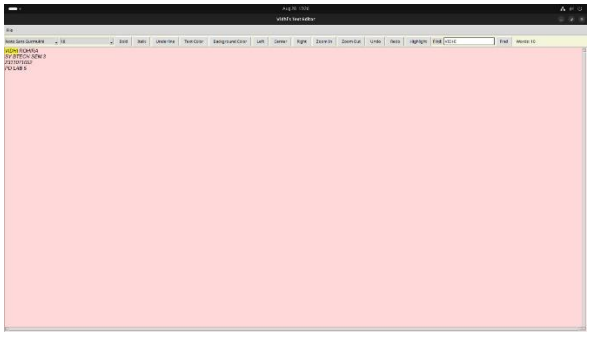
DIFFERENT FONT TYPE ↟



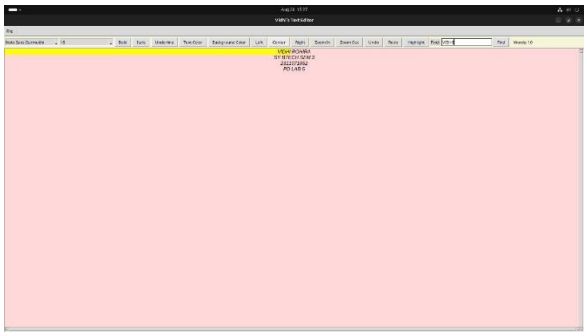


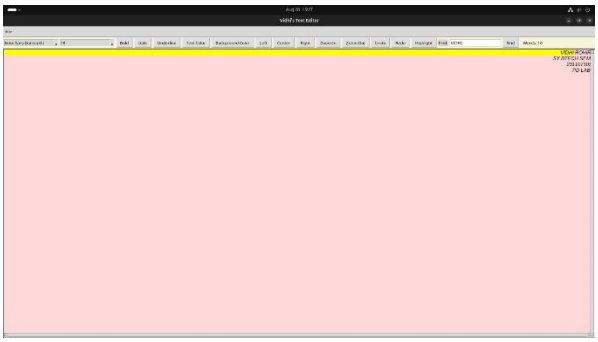
CAN MAKE WORDS BOLD OR ITALIC ↟



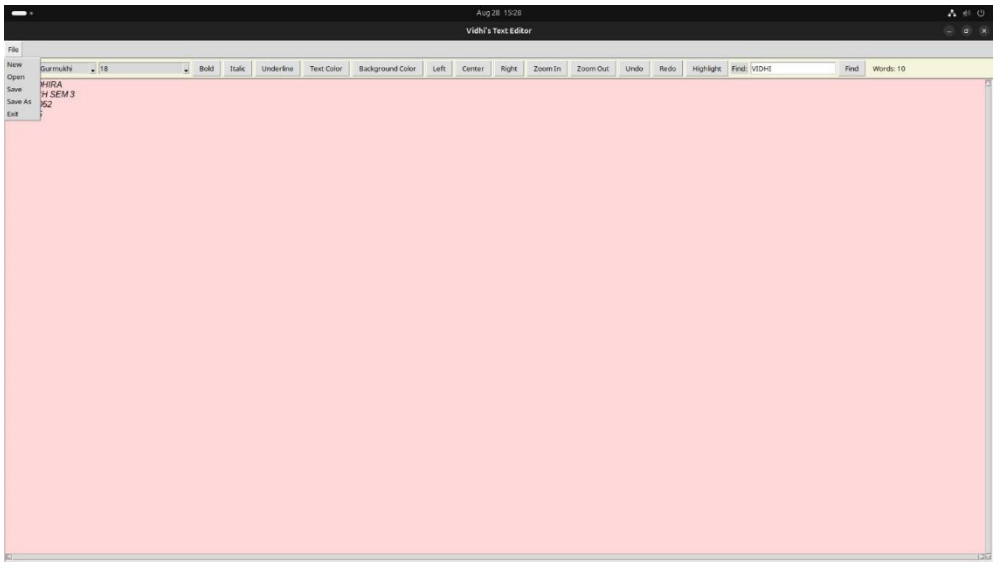


OFFERS BAGROUND THEMES ↟





SEARCHING WORDS FEATURE, CENTRE AND RIGHT ALLIGNMENT ↟



EDIT FILE AND SAVE FILE FEATURES ↟