Experiment No.1

Name: Veeransh Shah
Roll No: 221070063

Aim:

The aim of this Python code is to create a simple shell-like interface using the Tkinter library. The shell provides a text-based user interface where the user can input commands, and the output of those commands is displayed in a scrolled text widget. The shell supports basic commands such as execution of shell commands, changing directories (cd command), clearing the output, and exiting the shell.

Theory:

1. Initialization: The Shell class inherits from tk.Tk and initializes the shell with user and machine information, setting up the title of the shell window.

2. Widgets: The shell consists of three main widgets:
   1.A scrolled text widget (output_text) to display the output of commands in a scrollable window.
   2.An entry widget (command_entry) where the user can input commands.
   3.A button (execute_button) to trigger the execution of commands.

3. Command Execution:
   1.The execute_command method is responsible for executing the entered command. It handles special commands like 'exit', 'cd', and 'clear'.
   2.If the command is 'exit', the shell window is destroyed.
   3.If the command starts with 'cd', it attempts to change the current working directory.
   4.If the command is 'clear', it clears the output in the scrolled text widget.
   5.Other commands are executed using the subprocess module, and the output is displayed in the scrolled text widget.

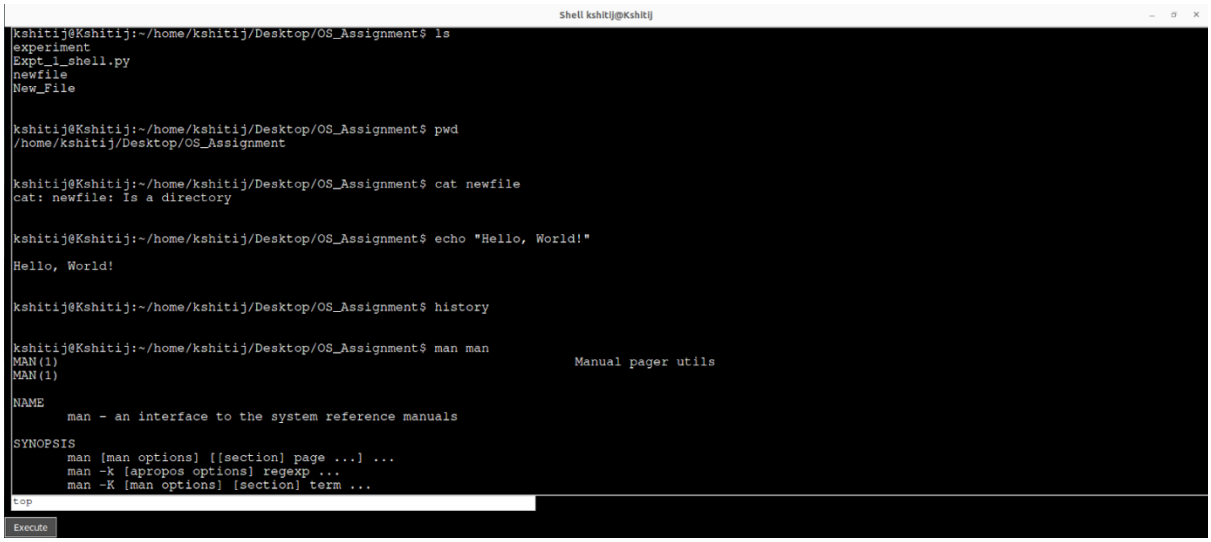
4. Directory Change:
   1.The change_dir method is called when the user inputs a 'cd' command. It attempts to change the current working directory using os.chdir.
   2.If the directory change is successful, it updates the prompt text in the scrolled text widget.
   3.If the specified directory is not found, it displays an error message.

CODE:

```
import tkinter as tk
from tkinter import scrolledtext
import subprocess
import os

class Shell(tk.Tk):
    def __init__(self):
        super().__init__()
        self.user_info = os.environ['USER']
        self.machine_info = os.uname().nodename
        self.title(f"Shell {self.user_info}@{self.machine_info}")
        self.create_widgets()

    def create_widgets(self):
        self.configure(bg="black")

        self.output_text = scrolledtext.ScrolledText(self, wrap=tk.WORD,
width=160, height=34, bg="black", fg="white", insertbackground="white",
font=("Courier", 15))
        self.output_text.grid(row=1, column=0, padx=10, pady=0,
sticky="w", columnspan=3)

        self.command_entry = tk.Entry(self, width=80, bg="white",
fg="black", insertbackground="black", relief=tk.FLAT,font=("Courier",
12))
        self.command_entry.grid(row=2, column=0, padx=10, pady=(0, 10),
sticky="w", columnspan=3)

        self.command_entry.bind("<Return>", lambda event:
self.execute_command())

        execute_button = tk.Button(self, text="Execute",
command=self.execute_command, bg="#4E4E4E", fg="white", relief=tk.FLAT)
        execute_button.grid(row=3, column=0, pady=(0, 10), sticky="w",
columnspan=3)

    def execute_command(self):
        command = self.command_entry.get()
        if command.lower() == 'exit':
            self.destroy()
```

```
            return
        self.command_entry.delete(0, tk.END)

        if command.startswith('cd '):
            new_dir = command.split(' ',1)[1]
            self.change_dir(new_dir)
            return

        if command.lower() == 'clear':
            self.output_text.delete(1.0,tk.END)
            return

        try:
            result = subprocess.check_output(['bash', '-c', command],
stderr=subprocess.STDOUT, text=True)
        except subprocess.CalledProcessError as e:
            result = e.output

        prompt_text =
f"{self.user_info}@{self.machine_info}:~{os.getcwd()}$"
        self.output_text.insert(tk.END, f"{prompt_text}
{command}\n{result}\n\n")
        self.output_text.see(tk.END)

    def change_dir(self, new_dir):
        try:
            os.chdir(new_dir)
            prompt_text =
f"{self.user_info}@{self.machine_info}:~{os.getcwd()}$ \n\n"
        except FileNotFoundError:
            prompt_text =
f"{self.user_info}@{self.machine_info}:~{os.getcwd()}$ \n{new_dir}: No
such file or directory \n\n"
        self.output_text.insert(tk.END, f"{prompt_text}")
        self.output_text.see(tk.END)



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

OUTPUT:



Conclusion:

This Python code creates a basic shell interface using Tkinter, providing users with the ability to execute shell commands, change directories, clear the output, and exit the shell. It leverages the subprocess module to execute commands and updates the scrolled text widget to display the output interactively. The code demonstrates the integration of GUI components with command-line functionality, offering a simple yet functional shell-like experience within a graphical user interface.